

机器翻译

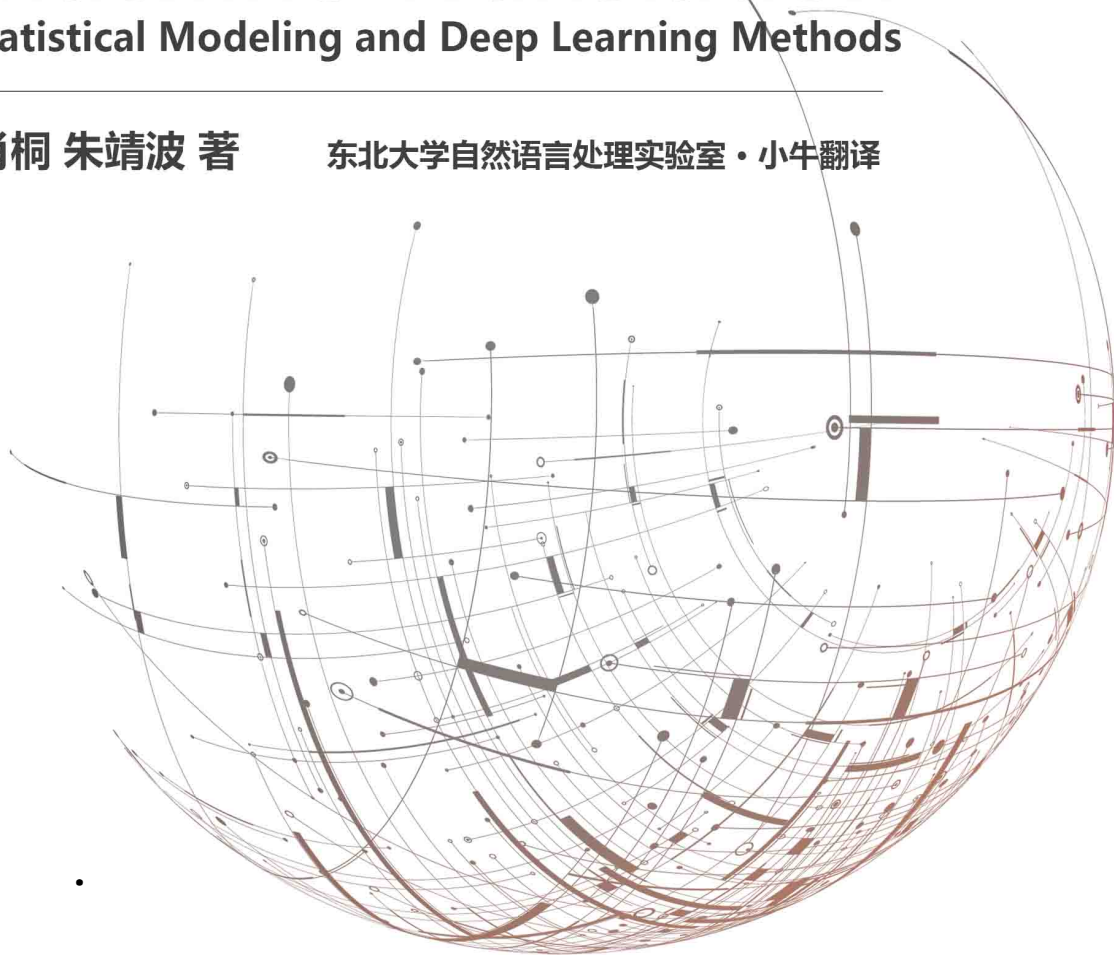
统计建模与深度学习方法

Machine Translation

Statistical Modeling and Deep Learning Methods

肖桐 朱靖波 著

东北大学自然语言处理实验室 · 小牛翻译



Copyright © 2020 肖桐 朱靖波

东北大学自然语言处理实验室 · 小牛翻译

顾问：姚天顺 王宝库

[HTTPS://OPENSOURCE.NIUTRANS.COM/MTBOOK/INDEX.HTML](https://opensource.niutrans.com/mtbook/index.html)

[HTTPS://GITHUB.COM/NIUTRANS/MTBOOK](https://github.com/NiUTrans/MTBook)

Licensed under the Creative Commons Attribution-NonCommercial 4.0 Unported License (the “License”). You may not use this file except in compliance with the License. You may obtain a copy of the License at <http://creativecommons.org/licenses/by-nc/4.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

May 30, 2020

在此感谢为本书做出贡献的小牛团队（部分）成员

曹润柘、曾信、孟霞、单韦乔、姜雨帆、王子扬、刘辉、许诺、李北、刘继强、张哲旻、周书含、周涛、李炎洋、林野、陈贺轩、刘晓倩、牛蕊、田丰宁、杜权、李垠桥、许晨、张裕浩、胡驰、冯凯、王泽洋、刘腾博、刘兴宇、徐萍、赵闯、高博、张春良、王会珍、张俐、杨木润、宁义明、李洋、秦浩、胡明涵

导 读

让计算机进行自然语言的翻译是人类长期的梦想，也是人工智能的终极目标之一。自上世纪九十年代起，机器翻译迈入了基于统计建模的时代，发展到今天，深度学习等机器学习方法已经在机器翻译中得到了大量的应用，取得了令人瞩目的进步。

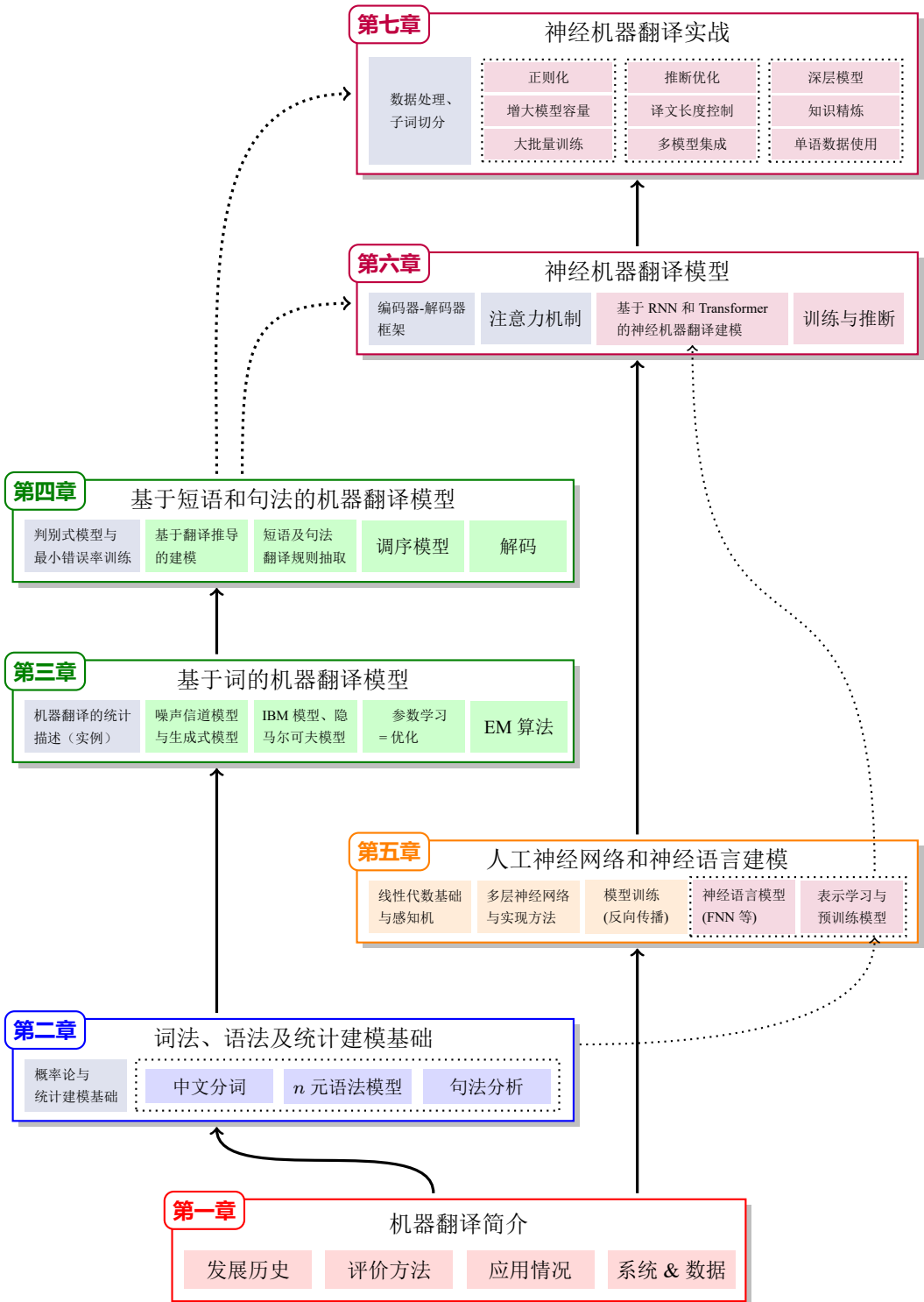
在这个时代背景下，对机器翻译的模型、方法和实现技术进行深入了解是自然语言处理领域研究者和实践者所渴望的。本书全面回顾了近三十年内机器翻译的技术发展历程，并围绕统计建模和深度学习两个主题对机器翻译的技术方法进行了全面介绍。在写作中，笔者力求用朴实的语言和简洁的实例阐述机器翻译的基本模型和方法，同时对相关的技术前沿进行讨论。本书可以供计算机相关专业高年级本科生及研究生学习之用，也可以作为自然语言处理，特别是机器翻译领域相关研究人员的参考资料。

本书共分为七个章节，章节的顺序参考了机器翻译技术发展的时间脉络，同时兼顾了机器翻译知识体系的内在逻辑。各章节的主要内容包括：

- 第一章：机器翻译简介
- 第二章：词法、语法及统计建模基础
- 第三章：基于词的机器翻译模型
- 第四章：基于短语和句法的机器翻译模型
- 第五章：人工神经网络和神经语言建模
- 第六章：神经机器翻译模型
- 第七章：神经机器翻译实战 —— 参加一次比赛

其中，第一章是对机器翻译的整体介绍。第二章和第五章是对统计建模和深度学习方法的介绍，分别建立了两个机器翻译范式的基础知识体系——统计机器翻译和神经机器翻译。统计机器翻译部分（第三、四章）涉及早期的基于单词的翻译模型，以及本世纪初流行的基于短语和句法的翻译模型。神经机器翻译（第六、七章）代表了当今机器翻译的前沿，内容主要涉及了基于端到端表示学习的机器翻译建模方法。特别地，第七章对一些最新的神经机器翻译方法进行了讨论，为相关科学问题的研究和实用系统的开发提供了可落地的思路。下图展示了本书各个章节及核心概念之间的关系。

用最简单的方式阐述机器翻译的基本思想是笔者所期望达到的目标。但是，书中不可避免会使用一些形式化定义和算法的抽象描述，因此，笔者尽所能通过图例进行解释（本书共 320 张插图）。不过，本书所包含的内容较为广泛，难免会有疏漏，望读者海涵，并指出不当之处。



本书各章节及核心概念关系图



Contents

I

机器翻译基础

1	机器翻译简介	17
1.1	机器翻译的概念	17
1.2	机器翻译简史	20
1.2.1	人工翻译	20
1.2.2	机器翻译的萌芽	21
1.2.3	机器翻译的受挫	22
1.2.4	机器翻译的快速成长	23
1.2.5	机器翻译的爆发	24
1.3	机器翻译现状	25
1.4	机器翻译方法	27
1.4.1	基于规则的机器翻译	27
1.4.2	基于实例的机器翻译	28
1.4.3	统计机器翻译	29
1.4.4	神经机器翻译	30
1.4.5	对比分析	31

1.5	翻译质量评价	32
1.5.1	人工评价	32
1.5.2	自动评价	33
1.6	机器翻译应用	36
1.7	开源项目与评测	38
1.7.1	开源机器翻译系统	38
1.7.2	常用数据集及公开评测任务	42
1.8	推荐学习资源	44
1.8.1	经典书籍	44
1.8.2	网络资源	45
1.8.3	专业组织和会议	45
2	词法、语法及统计建模基础	49
2.1	问题概述	50
2.2	概率论基础	51
2.2.1	随机变量和概率	52
2.2.2	联合概率、条件概率和边缘概率	53
2.2.3	链式法则	54
2.2.4	贝叶斯法则	55
2.2.5	KL 距离和熵	57
2.3	中文分词	59
2.3.1	基于词典的分词方法	60
2.3.2	基于统计的分词方法	61
2.4	n-gram 语言模型	66
2.4.1	建模	67
2.4.2	未登录词和平滑算法	69
2.5	句法分析（短语结构分析）	74
2.5.1	句子的句法树表示	74
2.5.2	上下文无关文法	76
2.5.3	规则和推导的概率	80
2.6	小结及深入阅读	82

3	基于词的机器翻译模型	87
3.1	词在翻译中的作用	87

3.2	一个简单的翻译系统.....	89
3.2.1	如何进行翻译?	89
3.2.2	基本框架	91
3.2.3	单词翻译概率	92
3.2.4	句子级翻译模型	95
3.2.5	解码	99
3.3	基于词的翻译建模	101
3.3.1	噪声信道模型	101
3.3.2	统计机器翻译的三个基本问题	104
3.4	IBM 模型 1-2.....	107
3.4.1	IBM 模型 1	108
3.4.2	IBM 模型 2	109
3.4.3	解码及计算优化	110
3.4.4	训练	112
3.5	IBM 模型 3-5 及隐马尔可夫模型	118
3.5.1	基于产出率的翻译模型	118
3.5.2	IBM 模型 3	120
3.5.3	IBM 模型 4	122
3.5.4	IBM 模型 5	124
3.5.5	隐马尔可夫模型	125
3.5.6	解码和训练	127
3.6	问题分析.....	128
3.6.1	词对齐及对称化	128
3.6.2	Deficiency	129
3.6.3	句子长度	130
3.6.4	其他问题	130
3.7	小结及深入阅读	131
4	基于短语和句法的机器翻译模型	133
4.1	翻译中的结构信息	133
4.1.1	更大粒度的翻译单元	134
4.1.2	句子的结构信息	136
4.2	基于短语的翻译模型.....	138
4.2.1	机器翻译中的短语	138
4.2.2	数学建模及判别式模型	141
4.2.3	短语抽取	144
4.2.4	调序	148

4.2.5	特征	151
4.2.6	最小错误率训练	152
4.2.7	栈解码	155
4.3	基于层次短语的模型.....	160
4.3.1	同步上下文无关文法	163
4.3.2	层次短语规则抽取	166
4.3.3	翻译模型及特征	168
4.3.4	CKY 解码	169
4.3.5	立方剪枝	172
4.4	基于语言学句法的模型.....	175
4.4.1	基于句法的翻译模型分类	177
4.4.2	基于树结构的文法	179
4.4.3	树到串翻译规则抽取	184
4.4.4	树到树翻译规则抽取	192
4.4.5	句法翻译模型的特征	195
4.4.6	基于超图的推导空间表示	196
4.4.7	基于树的解码 vs 基于串的解码	199
4.5	小结及深入阅读	203

III

神经机器翻译

5	人工神经网络和神经语言建模.....	207
5.1	深度学习与人工神经网络.....	208
5.1.1	发展简史	208
5.1.2	为什么需要深度学习	211
5.2	神经网络基础	212
5.2.1	线性代数基础	212
5.2.2	人工神经元和感知机	218
5.2.3	多层神经网络	222
5.2.4	函数拟合能力	226
5.3	神经网络的张量实现.....	229
5.3.1	张量及其计算	229
5.3.2	张量的物理存储形式	233
5.3.3	使用开源框架实现张量计算	234
5.3.4	前向传播与计算图	237
5.3.5	神经网络实例	238

5.4	神经网络的参数训练	239
5.4.1	损失函数	240
5.4.2	基于梯度的参数优化	241
5.4.3	参数更新的并行化策略	249
5.4.4	梯度消失、梯度爆炸和稳定性训练	251
5.4.5	过拟合	254
5.4.6	反向传播	255
5.5	神经语言模型	261
5.5.1	基于神经网络的语言建模	261
5.5.2	单词表示模型	267
5.5.3	句子表示模型及预训练	269
5.6	小结及深入阅读	274
6	神经机器翻译模型	275
6.1	神经机器翻译的发展简史	275
6.1.1	神经机器翻译的起源	277
6.1.2	神经机器翻译的品质	279
6.1.3	神经机器翻译的优势	282
6.2	编码器-解码器框架	283
6.2.1	框架结构	284
6.2.2	表示学习	285
6.2.3	简单的运行实例	286
6.2.4	机器翻译范式的对比	287
6.3	基于循环神经网络的翻译模型及注意力机制	288
6.3.1	建模	289
6.3.2	输入（词嵌入）及输出（Softmax）	292
6.3.3	循环神经网络结构	295
6.3.4	注意力机制	300
6.3.5	训练	307
6.3.6	推断	313
6.3.7	实例-GNMT	317
6.4	Transformer	318
6.4.1	自注意力模型	319
6.4.2	Transformer 架构	321
6.4.3	位置编码	323
6.4.4	基于点乘的注意力机制	325
6.4.5	掩码操作	327

6.4.6	多头注意力	328
6.4.7	残差网络和层正则化	329
6.4.8	前馈全连接网络子层	331
6.4.9	训练	332
6.4.10	推断	334
6.5	序列到序列问题及应用	335
6.5.1	自动问答	335
6.5.2	自动文摘	335
6.5.3	文言文翻译	336
6.5.4	对联生成	337
6.5.5	古诗生成	338
6.6	小结及深入阅读	338
7	神经机器翻译实战	341
7.1	神经机器翻译并不简单	341
7.1.1	影响神经机器翻译性能的因素	342
7.1.2	搭建神经机器翻译系统的步骤	343
7.1.3	架构选择	344
7.2	数据处理	344
7.2.1	分词	345
7.2.2	标准化	346
7.2.3	数据清洗	347
7.2.4	子词切分	349
7.3	建模与训练	354
7.3.1	正则化	354
7.3.2	增大模型容量	360
7.3.3	大批量训练	363
7.4	推断	366
7.4.1	推断优化	366
7.4.2	译文长度控制	375
7.4.3	多模型集成	378
7.5	进阶技术	382
7.5.1	深层模型	382
7.5.2	单语数据的使用	390
7.5.3	知识精炼	395
7.5.4	双向训练	399
7.6	小结及深入阅读	402

A	附录 A	407
A.1	基准数据集	407
A.2	平行语料	408
A.3	相关工具	409
A.3.1	数据预处理工具	409
A.3.2	评价工具	410
B	附录 B	411
B.1	IBM 模型 3 训练方法	411
B.2	IBM 模型 4 训练方法	413
B.3	IBM 模型 5 训练方法	415

机器翻译基础

1	机器翻译简介	17
1.1	机器翻译的概念	
1.2	机器翻译简史	
1.3	机器翻译现状	
1.4	机器翻译方法	
1.5	翻译质量评价	
1.6	机器翻译应用	
1.7	开源项目与评测	
1.8	推荐学习资源	
2	词法、语法及统计建模基础	49
2.1	问题概述	
2.2	概率论基础	
2.3	中文分词	
2.4	n -gram 语言模型	
2.5	句法分析（短语结构分析）	
2.6	小结及深入阅读	

1. 机器翻译简介

1.1 机器翻译的概念

从广义上来讲，“翻译”是指把一个事物转化为另一个事物的过程。这个概念多使用在对序列的转化上，比如，计算机程序的编译、自然语言文字翻译、蛋白质生物合成等。在程序编译中，高级语言编写的程序经过一系列的处理后转化为可执行的目标程序，这是一种从高级程序语言到低级程序语言的“翻译”。在人类语言的翻译中，一种语言文字通过人脑转化为另一种语言表达，这是一种自然语言的“翻译”。在蛋白质合成的第一步，RNA 分子序列转化为特定的氨基酸序列，这是一种生物学遗传信息的“翻译”。甚至说给上联对出下联、给一幅图片写出图片的主题等都可以被看作是“翻译”的过程。

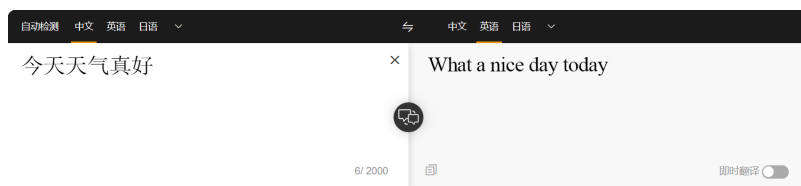


图 1.1: 通过计算机将中文翻译为英文

这里更加关注人类语言之间的翻译问题，即自然语言的翻译。如图1.1所示，通过计算机可以将一段中文文字自动转化为英文文字，中文被称为**源语言**（Source Language），英文被称为**目标语言**（Target Language）。

一直以来，文字的翻译往往是由人工完成。让计算机像人一样进行翻译似乎还是电影中的桥段，因为人们很难想象语言的多样性和复杂性可以用计算机语言进行描述。但是时至今日，人工智能技术的发展已经大大超越了人类传统的认知，用计算机进行自动翻译也不再是一种梦想，它已经深入到人们生活的很多方面，并发挥着重要作用。而这种由计算机进行自动翻译的过程也被称作**机器翻译**（Machine Translation）。类似地，自动翻译、智能翻译、多语言自动转换等概念也是指同样的事情。如果将今天的机器翻译和人工翻译进行对比，可以发现机器翻译系统所生成的译文还并不完美，甚至有时翻译质量非常差，但是它的生成速度快且成本低廉，更为重要的是机器翻译系统可以从大量数据中不断学习和进化。

人工翻译尽管精度很高，但是费时费力。当需要翻译大量的文本且精度要求不那么高时，比如海量数据的浏览型任务，机器翻译的优势就体现出来了。对于人工作业无法完成的事情，使用机器翻译可能只需花费几个小时甚至几分钟就能完成。这就类似于拿着锄头耕地种庄稼和使用现代化机器作业之间的区别。

实现机器翻译往往需要多个学科知识的融合，如数学、语言学、计算机科学、心理学等等。而最终呈现给使用者的是一套软件系统——机器翻译系统。通俗来讲，机器翻译系统就是一个可以在计算机上运行的软件工具，与人们使用的其他软件一样，只不过机器翻译系统是由“不可见的程序”组成。虽然这个系统非常复杂，但是呈现出来的展示形式却很简单，比如输入是待翻译的句子或文本，输出是译文句子或文本。

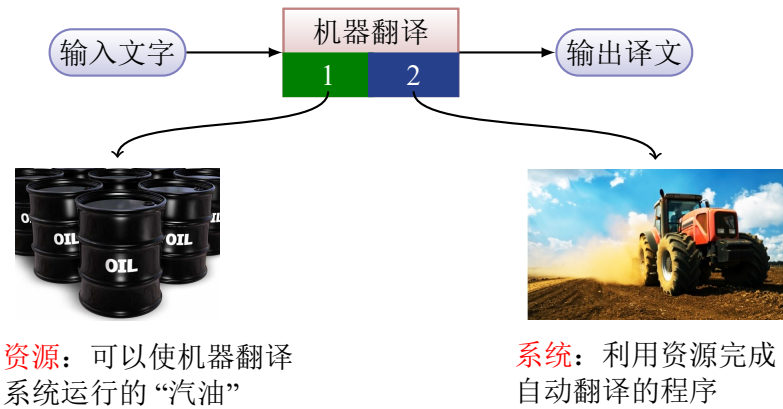


图 1.2: 机器翻译系统的组成

用机器进行翻译的想法可以追溯到电子计算机产生之前，发展过程中也经历了多个范式的变迁，现代机器翻译系统大多是基于数据驱动的方法——从数据中自动学习翻译知识，并运用这些知识对新的文本进行翻译。如图1.2所示，机器翻译系统通常由两部分组成：

- **资源**：如果把机器翻译系统比作一辆汽车，资源就好比是可以使汽车运行的“汽油”，它包括很多内容，如翻译规则、双（单）语数据、知识库等翻译知识，且

这些“知识”都是计算机可读的。值得一提的是,如果没有翻译资源的支持,任何机器翻译系统都无法运行起来。

- **系统:** 机器翻译算法的程序实现被称作系统,也就是机器翻译研究人员开发的软件。无论是翻译规则、翻译模板还是统计模型中的参数都需要通过机器翻译系统进行读取和使用。

构建一个强大的机器翻译系统需要“资源”和“系统”两方面共同作用。在资源方面,随着语料库语言学的发展,已经有大量高质量的双语和单语数据(称为语料)被整理并且电子化存储,研发机器翻译系统所需要的语料基础已经具备。特别是像英语、汉语等世界主流语种,相关语料资源已经非常丰富,这也大大加速了相关研究的进展。当然,对于一些稀缺资源语种或者特殊的领域,语料库仍然匮乏,但是这些并不影响机器翻译领域整体的发展速度。在现有语料库的基础上,很多研究者可以把精力集中在“系统”上。但是,机器翻译并非易事,有以下几方面挑战:

- **自然语言翻译问题的复杂性极高。**语言是人类进化的最高成就之一,自然语言具有高度的概括性、灵活性、多样性,这些都很难用几个简单的模型和算法进行描述。因此,翻译问题的数学建模和计算机程序实现难度很大。虽然近几年AlphaGo等人工智能系统在围棋等领域取得了令人瞩目的成绩,但是,相比翻译来说,围棋等棋类任务仍然“简单”。正如不同人对同一句话的理解不尽相同,一个句子往往不存在绝对的标准译文,其潜在的译文几乎是不可穷尽的。甚至人类译员在翻译一个句子、一个单词的时候,都要考虑整个篇章的上下文语境。这些难点都不是传统棋类任务所具有的。
- **计算机的“理解”与人类的“理解”存在鸿沟。**人类一直希望把自己翻译时所使用的知识描述出来,并用计算机程序进行实现,例如早期基于规则的机器翻译方法就源自这个思想。但是,经过实践发现,人和计算机在“理解”自然语言上存在着明显差异。首先,人类的语言能力是经过长时间在多种外部环境因素共同作用下形成的,这种能力很难直接准确地表达。况且人类的语言知识本身就很难描述,更不用说让计算机来理解;其次,人和机器翻译系统理解语言的目的不一样。人理解和使用语言是为了进行生活和工作,而机器翻译系统更多的是为了对某些数学上定义的目标函数进行优化。也就是说,机器翻译系统关注的是翻译这个单一目标,而并不是像人一样进行复杂的活动;此外,人和计算机的运行方式有着本质区别。人类语言能力的生物学机理与机器翻译系统所使用的计算模型本质上是不同的,机器翻译系统使用的是其自身能够理解的“知识”,比如,统计学上的词语表示。这种“知识”并不需要人来理解,当然从系统开发的角度,计算机也并不需要理解人是如何思考的。
- **单一的方法无法解决多样的翻译问题。**首先,语种的多样性会导致任意两种语言之间的翻译实际上都是不同的翻译任务。比如,世界上存在的语言多达几千种,如果选择任意两种语言进行互译就产生上百万种翻译方向。虽然已经有研究者

尝试用同一个框架甚至同一个翻译系统进行全语种的翻译，但是这类系统离真正可用还有很远的距离；其次，不同的领域，不同的应用场景对翻译也有不同的需求。比如，文学作品的翻译和新闻的翻译就有不同、口译和笔译也有不同，类似的情况不胜枚举。机器翻译要适用于多样的需求，这些又进一步增加了计算机建模的难度；再次，对于机器翻译来说，充足的高质量数据是必要的，但是不同语种、不同领域、不同应用场景所拥有的数据量有明显差异，甚至很多语种几乎没有可用的数据，这时开发机器翻译系统的难度可想而知。值得注意的是，现在的机器翻译还无法像人类一样在学习少量样例的情况下进行举一反三，因此数据稀缺情况下的机器翻译也给研究者带来了很大的挑战。

显然，实现机器翻译并不简单，甚至有人把机器翻译看作是实现人工智能的终极目标。幸运的是，今天的机器翻译无论从技术方法上还是从应用上都有了巨大的飞跃，很多问题在不断被求解。如果你看到过十年前机器翻译的结果，再对比今天的结果，一定会感叹翻译质量的今非昔比，很多译文已经非常准确且流畅。从当今机器翻译的前沿技术看，近三十年机器翻译的进步更多的得益于基于数据驱动方法和统计建模方法的使用。特别是近些年深度学习等基于表示学习的端到端方法使得机器翻译的水平达到了新高度。因此，本书将会对基于统计建模和深度学习方法的机器翻译模型、方法和系统实现进行全面介绍和分析，希望这些内容可以对相关内容的学习和科研工作提供参考。

1.2 机器翻译简史

虽然翻译这个概念在人类历史中已经存在了上千年，但机器翻译发展至今只有七十余年的历史。纵观机器翻译的发展，历程曲折又耐人寻味，可以说了解机器翻译的历史对我们深入理解相关技术方法会有很好的启发，甚至对我们了解整个自然语言处理领域的发展也有启示作用。

1.2.1 人工翻译

人类形成语言文字的过程中逐渐形成了翻译的概念。一个著名的标志性证据是罗塞塔石碑（Rosetta Stone），如图1.3所示。这个石碑制作于公元前 196 年，据说是可供考证的最久远的记载平行文字的历史遗迹。石碑由上至下刻有同一段埃及国王诏书的三种语言版本，最上面是古埃及象形文，中间是埃及草书，最下面是古希腊文。可以明显看出石碑上中下雕刻的文字的纹理是不同的。尽管用不同的语言文字描述同一件事在今天看来很常见，但是这在生产力低下的两千年前是很罕见的。很多人认为罗塞塔石碑是标志翻译或人工翻译的一个起点。目前罗塞塔石碑保存于大英博物馆，并成为该馆最具代表性的镇馆之宝之一。

随后，更多的翻译工作在文化和知识传播中开展。其中一个典型代表是宗教文献的翻译。在人类的历史长河中，宗教是人类意识形态的一个重要载体。为了宣传

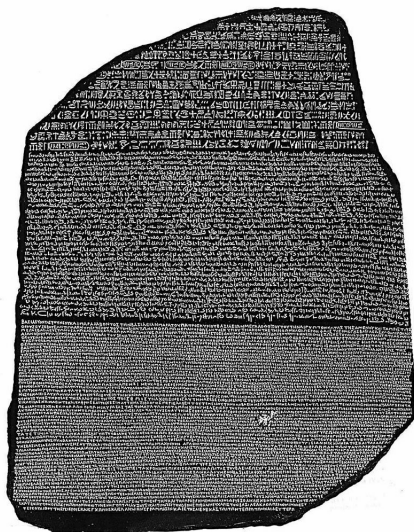


图 1.3: 罗塞塔石碑

教义，人们编写了大量的宗教文献。在西方，一项最早被记录的翻译活动是将旧约圣经（希伯来文及埃兰文）翻译为希腊文版本。迄今为止人类历史上翻译版本最多的书就是圣经。在中国唐代，有一位世界性的重量级文化人物——玄奘，他不仅是佛学家、旅行家，还是翻译家。玄奘西行求法归来后把全部的心血和智慧奉献给了译经事业，在助手们的帮助下，共翻译佛教经论 74 部，1335 卷，每卷万字左右，合计 1335 万字，占去整个唐代译经总数的一半以上，树立了我国古代翻译思想的光辉典范。

翻译在人类历史长河中起到了重要的作用。一方面，由于语言文字、文化和地理位置的差异性，使得翻译成为一个重要的需求；另一方面，翻译也加速了不同文明的融会贯通，促进了世界的发展。今天，翻译已经成为重要的行业之一，包括各个高校也都设立了翻译及相关专业，相关人才不断涌现。据《2019 年中国语言服务行业发展报告》统计：全球语言服务产值预计将首次接近 500 亿美元；中国涉及语言服务的在营企业 360,000 余家，语言服务为主营业务的在营企业近万家，总产值超过 300 亿元，年增长 3% 以上；全国开设外语类专业的高校数量多达上千所，其中设立有翻译硕士（MTI）和翻译本科（BTI）专业的院校分别有 250 余所和 280 余所，MTI 累计招生数达 6 万余人 [366]。当然，面对着巨大的需求，如何使用机器辅助翻译等技术手段提高人工翻译效率，也是人工翻译和机器翻译领域需要共同探索的方向。

1.2.2 机器翻译的萌芽

人工翻译已经存在了上千年，而机器翻译又起源于什么时候呢？机器翻译跌宕起伏的发展史可以分为萌芽期、受挫期、快速成长期和爆发期四个阶段。

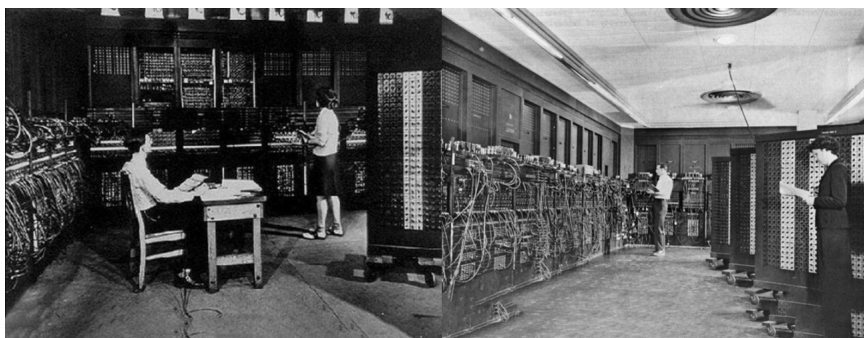


图 1.4: 世界上第一台通用电子数字计算机“埃尼阿克”(ENIAC)

世界上第一台通用电子数字计算机在 1946 年研制成功(图1.4¹)。但在上世纪 30 年代使用计算模型进行自动翻译的思想就开始萌芽, 当时法国科学家 G. B. Artsouni 提出了用机器来进行翻译的想法。

第二次世界大战使得数学和密码学相当发达, 由于战争的需要, 在那个时代消息传递变的更为隐秘, 对文字进行加密和解密成为重要的军事需求。因此, 有人提出是否能用密码学的技术或方法解决人类语言的翻译任务, 比如把汉语看成英语的一个加密文本, 汉语翻译成英语就类似于解密的过程。当然这只是最初的想法。第一次提出机器翻译这个概念是在 1949 年, 当时 W. Weaver 撰写了一篇名为《翻译》的备忘录, 正式开创了机器翻译(Machine Translation)的概念, 这个概念一直沿用至今。当然, 在那个年代进行机器翻译研究还有很多条件不具备, 包括使用加密解密技术进行自动翻译的很多尝试很快也被验证是不可行的。不过, 这些早期的探索为后来机器翻译的发展提供了思想的火种。

1.2.3 机器翻译的受挫

随着电子计算机的发展, 研究者开始尝试使用计算机来进行自动的翻译。但是事情并不总是一帆风顺, 怀疑论者对机器翻译一直存有质疑, 并很容易找出一些机器翻译无法解决的问题。自然地, 人们也期望能够客观地评估一下机器翻译的可行性。当时美国基金资助组织委任自动语言处理咨询会承担了这项任务。

经过近两年的调查与分析, 该委员会于 1966 年 11 月公布了一个题为《语言与机器》的报告(图1.5), 即 ALPAC 报告。该报告全面否定了机器翻译的可行性, 为机器翻译的研究泼了一盆冷水。

随后美国政府终止了对机器翻译研究的支持, 这导致整个产业界和学术界对机器翻译都开始回避。大家觉得机器翻译像伪科学, 无论是发表论文还是申请项目都很难得到支持。没有了政府的支持, 企业也无法进行大规模投入, 机器翻译的研究就此受挫。

从历史上看, 包括机器翻译在内很多人工智能领域在那个年代并不受“待见”, 其

¹<https://baike.baidu.com/item/ENIAC>

主要原因在于当时的技术水平还比较低，而大家又对机器翻译等技术的期望过高。最后发现，当时的机器翻译水平无法满足实际需要，因此转而排斥它。但是，也正是这一盆冷水，让人们可以更加冷静的思考机器翻译的发展方向，为后来的爆发蓄力。

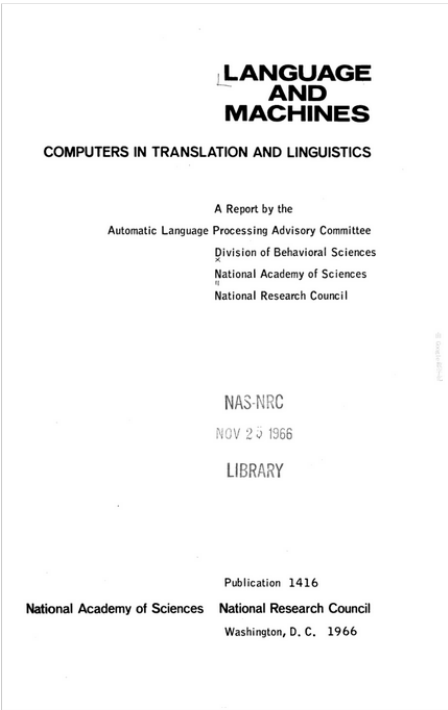


图 1.5: ALPAC 报告

1.2.4 机器翻译的快速成长

事物发展都是螺旋式上升的，机器翻译也是一样。上世纪 70 年代中后期，特别是 80 年代到 90 年代初，国家之间往来日益密切，而不同语言之间形成的交流障碍愈发严重，传统的人工作业方式已经远远不能满足需求。与此同时，语料库语言学的发展也为机器翻译提供了新的思路。其中，随着传统纸质文字资料不断电子化，计算机可读的语料越来越多，这使得人们可以用计算机对语言规律进行统计分析。另一方面，随着可用数据越来越多，用数学模型描述这些数据中的规律并进行推理逐渐成为可能。这也衍生出一类数学建模方法——**数据驱动**（Data-Driven）的方法。这类方法也成为了随后出现的统计机器翻译的基础。

传统的机器翻译方法，都需要人来书写规则，虽然对少部分句子具有较高的翻译精度，但这类方法对翻译现象的覆盖度有限，而且对规则或者模板中的噪声非常敏感，系统健壮性差。而基于数据驱动的方法不依赖于人写的规则，机器翻译的建模、训练和推断都可以自动地从数据中学习。这使得整个机器翻译的范式发生了翻天覆地的变化，比如，基于实例的方法和统计机器翻译就是在此期间兴起的。此外，这样的方法使得机器翻译系统的开发代价大大地降低。从上世纪 90 年代到本世纪初，统

计机器翻译发展迅猛，很快成为了当时机器翻译研究与应用的代表性方法。一个标志性的事件是谷歌推出了一个在线的免费自动翻译服务，也就是大家熟知的谷歌翻译。这使得机器翻译这种“高大上”的技术快速进入人们的生活，而不再是束之高阁的科研想法。随着机器翻译不断走向实用，机器翻译的应用也越来越多，这反过来进一步促进了机器翻译的研究进程。比如，在 2005 — 2015 年间，统计机器翻译这个主题几乎统治了 ACL 等自然语言处理相关方向顶级会议的论文，可见其在当时的影响力。

1.2.5 机器翻译的爆发

2005 年拉开了统计机器翻译发展十年黄金时期的序幕。在这一时期，各种基于统计机器翻译模型层出不穷，经典的基于短语的模型和基于句法的模型也先后被提出。在 2013 年以后，机器学习的进步带来了机器翻译技术的进一步提升。特别是基于神经网络的深度学习方法在机器视觉、语音识别中被成功应用，带来性能的飞跃式提升。很快，相关模型和方法也被用于机器翻译。对于机器翻译来说，深度学习的成功也是一种必然，原因如下：

- 第一，端到端学习不依赖于过多的先验假设。在统计机器翻译时代，模型设计或多或少会对翻译的过程进行假设，称为隐藏结构假设。比如基于短语的模型假设：源语言和目标语言都会被切分成短语序列，这些短语之间存在某种对齐关系。这种假设既有优点也有缺点：一方面，该假设有助于模型融入人类的先验知识，比如假设中的短语就借鉴了语言学相关的概念；另一方面，假设越多模型受到的限制也越多。如果假设是正确的，模型可以很好地描述问题。但如果假设错误，那么模型就可能产生偏差。深度学习不依赖于先验知识，也不需要手工设计特征，模型直接从输入和输出的映射上进行学习（端到端学习），这样也在一定程度上避免了隐藏结构假设造成的偏差。
- 第二，神经网络的连续空间模型有更强的表示能力。机器翻译中的一个基本问题是：如何表示一个句子？统计机器翻译把句子的生成过程看作是短语或者规则的推导，这本质上是一个离散空间上的符号系统。深度学习把传统的基于离散化的表示变成了连续空间的表示。比如，用实数空间的分布式表示代替了离散化的词语表示，而整个句子可以被描述为一个实数向量。这使得翻译问题可以在连续空间上描述，进而大大缓解了传统离散空间模型维度灾难等问题。更重要的是，连续空间模型可以用梯度下降等方法进行优化，具有很好的数学性质并且易于实现。
- 第三，深度网络学习算法的发展和 GPU（Graphics Processing Unit）等并行计算设备为训练神经网络提供了可能。早期的基于神经网络的方法一直没有在机器翻译甚至自然语言处理领域得到大规模应用，其中一个重要的原因是这类方法需要大量的浮点运算，而且以前计算机的计算能力无法达到这个要求。随着

GPU 等并行计算设备的进步，训练大规模神经网络也变为了可能。现在已经可以在几亿、几十亿，甚至上百亿句对上训练机器翻译系统，系统研发的周期越来越短，进展日新月异。

今天，神经机器翻译已经成为新的范式，大有全面替代统计机器翻译之势。比如，从世界上著名的机器翻译比赛 WMT 和 CCMT 中就可以看出这个趋势。如图1.6所示，其中左图是 WMT 19 全球机器翻译比赛的参赛队伍的截图，这些参赛队伍基本上都在使用深度学习完成机器翻译的建模。而在 WMT 19 各个项目夺冠系统中（1.6右图），神经机器翻译也几乎一统天下。

值得一提的是，近些年神经机器翻译的快速发展也得益于产业界的关注。各大互联网企业和机器翻译技术研发机构都对神经机器翻译的模型和实践方法给予了很大贡献。比如，谷歌、微软、百度、搜狗、金山、腾讯、阿里、有道、讯飞、小牛翻译等企业凭借自身人才和基础设施方面的优势，先后推出了以神经机器翻译为内核的产品及服务，相关技术方法已经在大规模应用中得到验证，大大推动了机器翻译的产业化进程，而且这种趋势在不断加强，机器翻译的前景也更加宽广。

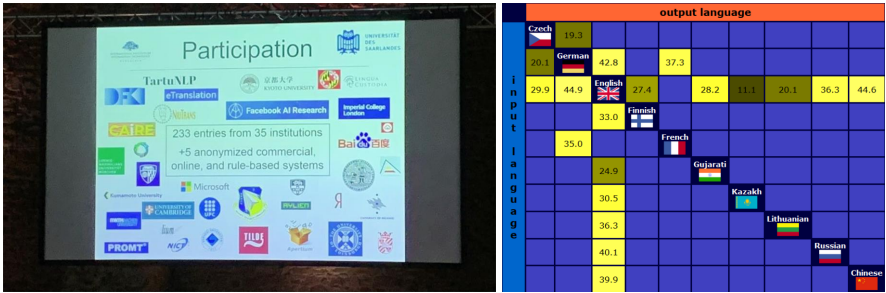


图 1.6: 国际机器翻译大赛 (左: WMT 19 参赛队伍; 右: WMT 19 各项的最好分数结果)

1.3 机器翻译现状

机器翻译技术发展到今天已经过无数次迭代，技术范式也经过若干次更替，近些年机器翻译的应用也如雨后春笋。但是大家都很好奇今天的机器翻译的质量究竟如何呢？乐观地说，在受限条件下，机器翻译的译文结果还是非常不错的，甚至可以接近人工翻译的结果。然而，在开放式翻译任务中，机器翻译的结果却并不理想。更严格来说，机器翻译的质量远没有达到人们所期望的完美的程度。对于有些人提到的“机器翻译代替人工翻译”也并不是事实。比如，在高精度同声传译任务中，机器翻译仍需要更多打磨；再比如，针对于小说的翻译，机器翻译还无法做到与人工翻译媲美；甚至有人尝试用机器翻译系统翻译中国古代诗词，这里更多的是娱乐的味道。但是毫无疑问的是，机器翻译可以帮助人类，甚至有朝一日可以代替一些低端的人工翻译工作。

图1.7展示了机器翻译和人工翻译质量的一个对比结果。在汉语到英语的新闻翻译任务中，如果对译文进行人工评价（五分制），那么机器翻译的译文得分为 3.9 分，

人工译文得分为 4.7 分（人的翻译也不是完美的）。可见，在这个任务中机器翻译表现不错，但是与人还有一定差距。如果换一种方式评价，把人的译文作为参考答案，用机器翻译的译文与其进行比对（百分制），会发现机器翻译的得分只有 47 分。当然，这个结果并不是说机器翻译的译文质量很差，它更多的是表明机器翻译系统可以生成一些与人工翻译不同的译文，机器翻译也具有一定的创造性。这也类似于，很多围棋选手都想向 AlphaGo 学习，因为智能围棋系统也可以走出一些人类从未走过的妙招。

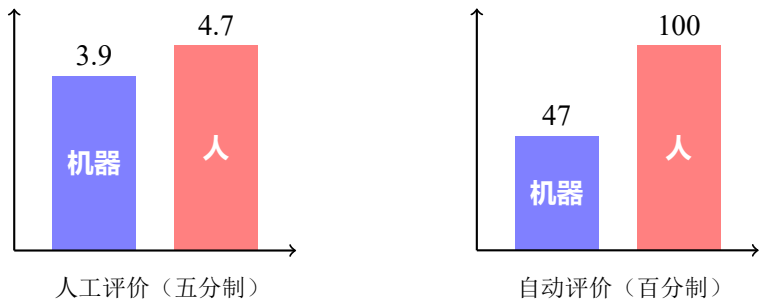


图 1.7: 机器翻译与人工翻译性能对比（汉英新闻领域翻译）

图1.8展示了一个真实的汉语到英语翻译实例。对比发现，机器翻译与人工翻译还是存在差距的，特别是在翻译一些具有感情色彩的词语时，机器翻译的译文缺一些味道。那么，机器翻译一点用都没有吗？显然不是。实际上，如果考虑翻译速度与翻译代价，机器翻译的价值是无可比拟的。还是同一个例子，翻译一篇短文如果人工翻译需要 30 分钟甚至更长时间，那么机器翻译仅仅需要两秒，换种情况思考，如果有 100 万篇这样的文档，其人工翻译的成本根本无法想象，消耗的时间更是难以计算，而计算机集群仅仅需要一天，而且只有电力的消耗。

源 文：从前有一个小岛，上面住着快乐、悲哀、知识和爱，还有其他各种情感。一天，情感们得知小岛快要下沉了。于是，大家都准备船只，离开小岛，只有爱决定留下来，她想坚持到最后一刻。过了几天，小岛真的要下沉了，爱想请人帮忙。

机器翻译：Once upon a time there was an island on which lived happiness,sorrow,knowledge,love and other emotions. One day, the emotions learned that the island was going to sink.As a result,everyone pre-
pared the boat and left the island. Only Love decided to stay.She wanted to stick to it until the last
moment. After a few days, the island was really going to sink and love wanted help.

人工翻译：Once upon a time, there was a small island where lived all kinds of emotions like JOY,SADNESS,
KNOWLEDGE, and LOVE.One day, these emotions found that the island was sinking, so one by
one they prepared the boat and planned to leave. None but LOVE chose to stay there. She was deter-
-mined to persist till the last moment.A few days later, almost the whole island sunk into the sea,
and LOVE had to seek for help.

源 文：这时，富裕乘着一艘大船经过。爱说：“富裕，你能带我走吗？”富裕答道：“不，我的船中有许多金银财宝，没有你的位置。”爱看见虚荣在一艘华丽的小船上，说：“虚荣，帮帮我吧！”“我帮不了你。你全身都湿透了，会弄坏我这漂亮的小船的。”悲哀来了，爱向他求助：“悲哀，让我跟你去吧！”“哦，……爱，我实在太悲哀了，想自己一个人呆一会儿！”悲哀答道。快乐走近爱的身边，但是她太快乐了，竟然没有听见爱在叫她！

机器翻译:	At this time, Richness <u>passed by</u> in a big ship. Love said, "Rich, can you take me away?" Richness replied, "No, <u>there are many treasures</u> of gold and silver in my ship, and there is no place for you."
	Love saw vanity in a magnificent boat and said, "Vanity, help me!" "I can't help you. You are <u>soak -ed to the skin</u> and will damage my beautiful boat." When sorrow came, love asked him for help: "sorrow, let me go with you!" "Oh,...love, I am so sad that I want to be alone for a while!" Sadly rep -lied. Happiness <u>approached</u> love, but she was too happy to hear love calling her!
人工翻译:	At that moment, WEALTH <u>was passing by</u> in a big boat. Love said, "WEALTH, can you take me with you?" WEALTH answered, "no, <u>there is a lot</u> of gold and silver in my boat. There is no place for you." Love saw VANITY in a beautiful boat and said, "VANITY, help me!" "I can't help you. You are <u>all wet</u> , and will break my pretty boat." Then SADNESS came. Love asked for help, "SAD -NESS, let me go with you!" "Oh,...LOVE, I am so sad that I want to be alone for a while!" "Repli -ed SADNESS. JOY <u>came close to</u> love, but she was so happy that she did not hear him call her!
源	文: 突然, 一个声音传来: "过来, 爱, 我带你走。" 这是位长者。爱大喜过望, 竟忘了问他他的名字。登上陆地后, 长者独自走开了。爱对长者感激不尽, 问另一位长者知识: "帮我的那个人是谁?" "他是时间。" 知识老人回答。"时间?" 爱问道, "他为什么要帮我?" 知识老人笑道: "因为只有时间才能理解爱有多么伟大。"
机器翻译:	Suddenly, a voice <u>came</u> : "Come here, love, I'll take you away." This is an elder. Love was <u>overjoy -ed and</u> forgot to ask his name. After landing on land, the elder walked away alone. Love was very grateful to the elder and asked another elder knowledge, " <u>Who is the person who helped me?</u> " "He is time." The <u>old intellectual</u> replied. "Time?" Love asked, "Why did he help me?" The old intellec -tual laughed, "Because only time can understand how great love is."
人工翻译:	Suddenly, a voice <u>said</u> , "come, LOVE, I'll take you." This is an elder. LOVE was <u>so over that</u> she forgot to ask his name. After landing on land, the elder walked away alone. LOVE was so grateful to the elder that she asked KNOWLEDGE, another elder, " <u>who help -ed me?</u> " "He is TIME." The <u>old man</u> replied. "TIME?" LOVE asked. "why did he help me?" The old man smiled and said, "Because only time can understand how great love is."

图 1.8: 机器翻译与人工翻译实例结果对比

1.4 机器翻译方法

为了对机器翻译技术有一个整体的认识, 这里对一些主要的机器翻译框架进行简要介绍。

1.4.1 基于规则的机器翻译

早期的机器翻译研究都是以基于规则的方法为主, 特别是在上世纪 70 年代, 以基于规则方法为代表的专家系统是人工智能中最具代表性的研究领域。它的主要思想是以词典和人工书写的规则库作为翻译知识, 用一系列规则的组合完成翻译。

图1.9展示了一个使用规则进行翻译的实例。这里, 利用一个简单的汉译英规则库完成对句子“我对你感到满意”的翻译。当翻译“我”时, 从规则库中找到规则 1, 该规则表示遇到单词“我”就翻译为“I”; 类似地, 也可以从规则库中找到规则 4, 该规则表示翻译调序, 即将单词“you”放到“be satisfied with”后面。可以看到, 这些规则的使用和进行翻译时所使用的思想非常类似, 可以说基于规则方法实际上在试图描述人类进行翻译的思维过程。

但是, 基于规则的机器翻译也存在问题。首先, 书写规则需要消耗大量人力, 规



图 1.9: 基于规则的机器翻译的示例图（左：规则库；右：规则匹配结果）

则库的维护代价极高；其次，规则很难涵盖所有的语言现象；再有，自然语言存在大量的歧义现象，规则之间也会存在冲突，这也导致规则数量不可能无限制增长。

1.4.2 基于实例的机器翻译

基于规则的方法更多地被使用在受限翻译场景中，比如受限词汇集的翻译。针对基于规则的方法存在的问题，基于实例的机器翻译于上世纪 80 年代中期被提出 [209]。该方法的基本思想是在双语句库中找到与待翻译句子相似的实例，之后对实例的译文进行修改，如替换、增加、删除等一系列操作，从而得到最终译文。这个过程可以类比人类学习并运用语言的过程：人会先学习一些翻译实例或者模板，当遇到新的句子时，会用以前的实例和模板作对比，之后得到新的句子的翻译结果。这也是一种举一反三的思想。

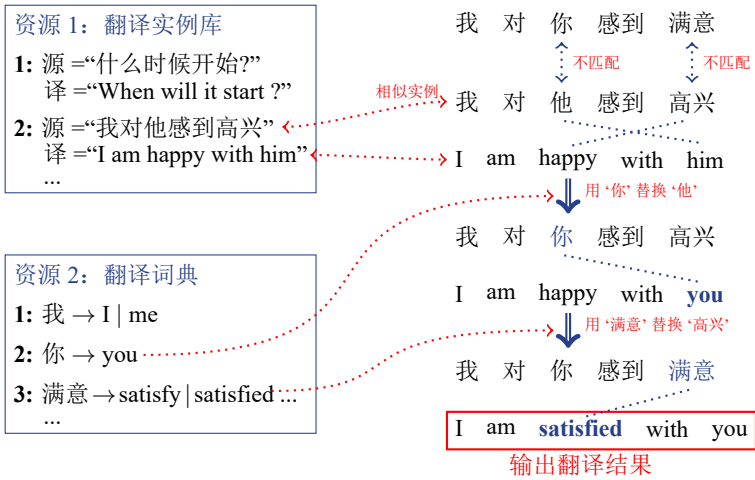


图 1.10: 基于实例的机器翻译的示例图（左：实例库；右：翻译结果）

图1.10展示了一个基于实例的机器翻译过程。它利用简单的翻译实例库与翻译词典完成对句子“我对你感到满意”的翻译。首先，使用待翻译句子的源语言端在翻

译实例库中进行比较，根据相似度大小找到相似的实例“我对他感到高兴”。然后，标记实例中不匹配的部分，即“你”和“他”，“满意”和“高兴”。再查询翻译词典得到词“你”和“满意”所对应的翻译结果“you”和“satisfied”，用这两个词分别替换实例中的“him”和“happy”，从而得到最终译文。

当然，基于实例的机器翻译也并不完美。首先，这种方法对翻译实例的精确度要求非常高，一个实例的错误可能会导致一个句型都无法翻译正确；其次，实例维护较为困难，实例库的构建通常需要单词级对齐的标注，而保证词对齐的质量是非常困难的工作，这也大大增加了实例库维护的难度；再次，尽管可以通过实例或者模板进行翻译，但是其覆盖度仍然有限。在实际应用中，很多句子无法找到可以匹配的实例或者模板。

1.4.3 统计机器翻译

统计机器翻译兴起于上世纪 90 年代 [24, 153] 它利用统计模型从单/双语语料中自动学习翻译知识。具体来说，可以使用单语语料学习语言模型，使用双语平行语料学习翻译模型，并使用这些统计模型完成对翻译过程的建模。整个过程不需要人工编写规则，也不需要从实例中构建翻译模板。无论是词还是短语，甚至是句法结构，统计机器翻译系统都可以自动学习。人更多的是定义翻译所需的特征和基本翻译单元的形式，而翻译知识都保存在模型的参数中。

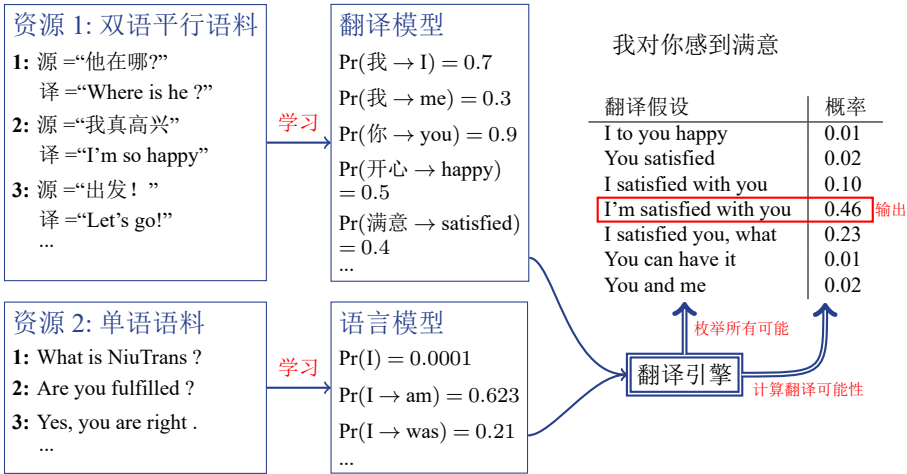


图 1.11: 统计机器翻译的示例图（左：语料资源；中：翻译模型与语言模型；右：翻译假设与翻译引擎）

图1.11展示了一个统计机器翻译系统运行的简单实例。整个系统需要两个模型：翻译模型和语言模型。其中，翻译模型从双语平行语料中学习翻译知识，得到短语表，其中包含各种词汇的翻译及其概率，这样可以度量源语言和目标语言片段之间互为翻译的可能性大小；语言模型从单语语料中学习目标语的词序列生成规律，来衡量目标语言译文的流畅性。最后，将这两种模型联合使用，翻译引擎来搜索尽可能多的翻译结果，并计算不同翻译结果的可能性大小，最后将概率最大的译文作为

最终结果输出。这个过程并没有显性使用人工翻译规则和模板，译文的生成仅仅依赖翻译模型和语言模型中的统计参数。

由于没有对翻译过程进行过多的限制，统计机器翻译有很灵活的译文生成方式，因此系统可以处理更加多样的句子。但是这种方法也带来了一些问题：首先，虽然并不需要人工定义翻译规则或模板，但统计机器翻译系统仍然需要人工定义翻译特征。提升翻译品质往往需要大量的特征工程，这导致人工特征设计的好坏会对系统产生决定性影响；其次，统计机器翻译的模块较多，系统研发比较复杂；再次，随着训练数据增多，统计机器翻译的模型（比如短语翻译表）会明显增大，在系统存储资源受限的情况下，这种模型不利于系统的正常使用。

1.4.4 神经机器翻译

随着机器学习技术的发展，基于深度学习的神经机器翻译逐渐兴起。自 2014 年开始，它在短短几年内已经在大部分任务上取得了明显的优势 [7, 276]。在神经机器翻译中，词串被表示成实数向量，即分布式向量表示。这样，翻译过程并不是在离散化的单词和短语上进行，而是在实数向量空间上计算，因此它对词序列表示的方式产生了本质的改变。通常，机器翻译可以被看作一个序列到另一个序列的转化。在神经机器翻译中，序列到序列的转化过程可以由**编码器-解码器**（Encoder-Decoder）框架实现。其中，编码器把源语言序列进行编码，并提取源语言中信息进行分布式表示，之后解码器再把这种信息转换为另一种语言的表达。

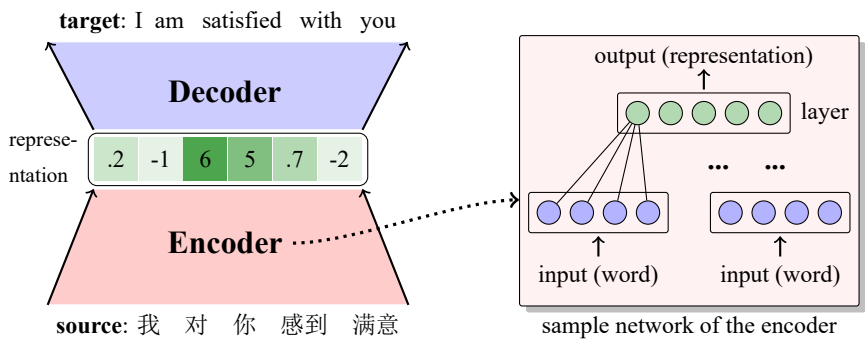


图 1.12: 神经机器翻译的示例图（左：编码器-解码器网络；右：编码器示例网络）

图1.12展示了一个神经机器翻译的实例。首先，通过编码器，源语言序列“我对你感到满意”经过多层神经网络编码生成一个向量表示，即图中的向量（0.2，-1，6，5，0.7，-2）。再将该向量作为输入送到解码器中，解码器把这个向量解码成目标语言序列。注意，目标语言序列的生成是逐词进行的（虽然图中展示的是解码器生成整个序列，但是在具体实现时是逐个单词生成目标语译文），产生某个词的时候依赖之前生成的目标语言的历史信息，直到产生句子结束符为止。

相比统计机器翻译，神经机器翻译的优势体现在其不需要特征工程，所有信息由神经网络自动从原始输入中提取。而且，相比离散化的表示，词和句子的分布式

连续空间表示可以为建模提供更为丰富的信息，同时可以使用相对成熟的基于梯度的方法优化模型。此外，神经网络的存储需求较小，天然适合小设备上的应用。但是，神经机器翻译也存在问题。首先，虽然脱离了特征工程，但神经网络的结构需要人工设计，即使设计好结构，系统的调优、超参数的设置等仍然依赖大量的实验；其次，神经机器翻译现在缺乏可解释性，其过程和人的认知差异很大，通过人的先验知识干预的程度差；再次，神经机器翻译对数据的依赖很大，数据规模、质量对性能都有很大影响，特别是在数据稀缺的情况下，充分训练神经网络很有挑战性。

1.4.5 对比分析

不同机器翻译方法有不同的特点。表1.1对比了这些方法，不难看出：

- 规则系统需要人工书写规则并维护，人工代价较高。统计和神经网络方法仅需要设计特征或者神经网络结构，对人工依赖较少（语言相关的）。
- 基于实例、统计和神经网络的方法都需要依赖语料库（数据），其中统计和神经网络方法具有一定的抗噪能力，因此也更适合大规模数据情况下的机器翻译系统研发。
- 基于规则和基于实例的方法在受限场景下有较好的精度，但是在开放领域的翻译上统计和神经网络方法更具优势。

表 1.1: 不同机器翻译方法的对比

	规则	实例	统计	神经
人工写规则	是	否	否	否
人工代价	高	一般	几乎没有	几乎没有
数据驱动	否	是	是	是
依赖数据质量	N/A	高	低	较低
抗噪声能力	低	低	高	较高
使用范围	受限领域	受限领域	通用领域	通用领域
翻译精度	高	较高	不确定	不确定

从现在机器翻译的研究和应用情况来看，基于统计建模的方法（统计机器翻译和神经机器翻译）是主流。这主要是由于它们的系统研发周期短，通过搜集一定量的数据即可实现快速原型。但是随着互联网等信息的不断开放，低成本的数据获取让神经机器翻译系统更快得以实现。因此最近神经机器翻译凭借其高质量的译文，受到越来越多研究人员和开发者的青睐。当然，对不同方法进行融合也是有价值的研究方向，也有很多有趣的探索，比如无指导机器翻译中还是会同时使用统计机器翻译和神经机器翻译方法，这也是一种典型的融合多种方法的思路。

1.5 翻译质量评价

机器翻译质量的评价对于机器翻译的发展具有至关重要的意义。首先，评价的结果可以用于指导研究人员不断改进机器翻译结果，并找到最具潜力的技术发展方向。同时，一个权威的翻译质量评价指标可以帮助用户更有效地使用机器翻译的结果。

一般来说，机器翻译的翻译**质量评价**（Quality Evaluation）是指在参考答案或者评价标准已知的情况下对译文进行打分。这类方法可以被称作有参考答案的评价，包括人工打分、BLEU 等自动评价方法都是典型的有参考答案评价。相对的，**无参考答案的评价**（Quality Estimation）是指在没有人工评价和参考答案的情况下，对译文质量进行评估。这类方法可以被看作是对机器翻译译文进行质量“预测”，这样用户可以选择性的使用机器翻译结果。这里主要讨论有参考答案的评价，因为这类方法是机器翻译系统研发所使用的主要评价方法。

1.5.1 人工评价

顾名思义，人工评价是指专家根据翻译结果好坏对译文进行评价。一般会根据句子的准确度和流利度对其进行打分，这样能够准确评定出句子是否准确翻译出原文的意思以及句子是否通顺。在对一个句子进行评定时，一般由多个专家匿名打分后进行综合评定。人工评价是最能准确反映句子翻译质量的评价方式，但是其缺点也十分明显：需要耗费人力物力，而且评价的周期长，不能及时得到有效的反馈。因此在实际系统开发中，纯人工评价不会过于频繁的被使用，它往往和自动评价一起配合，帮助系统研发人员准确的了解当前系统的状态。

人工评价的策略非常多。考虑不同的因素，往往会使用不同的评价方案，比如：

- 打分。常用的方法是对每个译文进行百分制或者五分制打分，分数越高表明译文越好。更粗糙的方法有三分制，甚至两分制打分。注意，打分越细致，评价者的工作量越大，因此五分制或者三分制评价更适合快速获得评价结果。
- 是否呈现源语言文本。评价者在进行人工评价时可仅被提供源语言文本或参考译文、或同时提供源语言文本和参考译文。从评价的角度，参考译文已经能够帮助评价者进行正确评价，但是源语言文本可以提供更多信息帮助评估译文的准确性。
- 评价者选择。理想情况下，评价者应同时具有源语言和目标语言的语言能力。但是，很多场景下双语能力的评价者很难招募，因此这时会考虑使用目标语为母语的评价者。配合参考译文，单语评价者也可以准确地进行评价。
- 多个系统评价。如果有多个不同系统的译文需要评价，可以直接使用每个系统单独打分的方法。但是，如果仅仅是想了解不同译文之间的相对好坏，也可以采用竞评的方式，即对于每个句子，对不同系统根据译文质量进行排序，这样做的效率会高于直接打分，而且评价准确性也能够得到保证。

- 数据选择。评价数据一般需要根据目标任务进行采集，为了避免和系统训练数据重复，往往会搜集最新的数据。而且，评价数据的规模越大，评价结果越科学。常用的做法是搜集一定量的评价数据，之后从中采样出所需的数据。由于不同的采样会得到不同的评价集合，这样的方法可以复用多次，得到不同的测试集。
- 面向应用的评价。除了人工直接打分，一种更有效的方法是把机器翻译的译文嵌入到下游应用中，通过机器翻译对下游应用的改善效果评估机器翻译译文质量。比如，可以把机器翻译放入译后编辑流程中，通过对比翻译效率的提升来评价译文质量。再比如，把机器翻译放入线上应用中，通过点击率或者用户反馈来评价机器翻译的品质。

简而言之，研究者可以根据实际情况选择不同的人工评价方案，人工评价也没有统一的标准。WMT 和 CCMT 机器翻译评测都有配套的人工评价方案，可以作为业界的参考标准。

1.5.2 自动评价

由于人工评价费事费力，同时具有一定的主观性，甚至不同人在不同时刻面对同一篇文章的理解都会不同，因此自动评价也是机器翻译系统研发人员所青睐的方法。自动评价的方式虽然不如人工评价准确，但是具有速度快，成本低、一致性高的优点。而且随着评价技术的不断发展，自动评价方式已经具有了比较好的指导性，可以帮助使用者快速了解当前机器翻译译文的质量。在机器翻译领域，自动评价已经成为了一个重要的分支，提出的自动评价方法不下几十种。这里无法对这些方法一一列举，为了便于后续章节的描述，这里仅对具有代表性的一些方法进行简要介绍。

BLEU

目前使用最广泛的自动评价指标是 BLEU。BLEU 是 Bilingual Evaluation Understudy 的缩写，最早由 IBM 在 2002 年提出 [226]。通过采用 n -gram 匹配的方式评定机器翻译结果和参考译文之间的相似度，即机器翻译的结果越接近人工参考译文就认定它的质量越高。 n -gram 是指 n 个连续单词组成的单元，称为 n 元语法单元。 n 越大表示评价时考虑的匹配片段越大。

BLEU 的计算首先考虑待评价译文中 n -gram 在参考答案中的匹配率，称为 n -gram 准确率 (n -gram Precision)。其计算方法如下：

$$P_n = \frac{\text{Count}_{\text{hit}}}{\text{Count}_{\text{output}}} \quad (1.1)$$

其中 $\text{Count}_{\text{hit}}$ 表示系统输出的译文中 n -gram 在参考答案中命中的次数， $\text{Count}_{\text{output}}$ 表示系统输出的译文中总共有多少 n -gram。为了避免同一个词被重复计算，BLEU

的定义中使用了截断的方式定义 $\text{Count}_{\text{hit}}$ 和 $\text{Count}_{\text{output}}$ 。例如：

■ **实例 1.1** Candidate: the the the the

Reference: The cat is standing on the ground ■

在引入截断方式之前，该译文的 1-gram 准确率为 $4/4 = 1$ ，这显然是不合理的。在引入截断的方式之后，“the”在译文中出现 4 次，在参考译文中出现 2 次，截断操作则是取二者的最小值，即 $\text{Count}_{\text{hit}} = 2$ ， $\text{Count}_{\text{output}} = 4$ ，该译文的 1-gram 准确率为 $2/4$ 。

译文整体的准确率等于各 n -gram 的加权平均：

$$P_{\text{avg}} = \exp\left(\sum_{n=1}^N w_n \cdot \log P_n\right) \quad (1.2)$$

但是，该方法更倾向于对短句子打出更高的分数。一个极端的例子是译文只有很少的几个词，但是都命中答案，准确率很高可显然不是好的译文。因此，BLEU 引入**短句惩罚因子**（Brevity Penalty, BP）的概念，对短句进行惩罚：

$$\text{BP} = \begin{cases} 1 & c > r \\ e^{(1-\frac{r}{c})} & c \leq r \end{cases} \quad (1.3)$$

其中 c 表示译文的句子长度， r 表示参考译文的句子长度。最终 BLEU 的计算公式为：

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{i=1}^N w_i \cdot \log P_i\right) \quad (1.4)$$

从机器翻译的发展来看，BLEU 的意义在于它给系统研发人员提供了一种简单、高效、可重复的自动评价手段，在研发机器翻译系统时不需要依赖人工评价。同时，BLEU 也有很多创新之处，包括引入 n -gram 的匹配，截断计数和短句惩罚等等，包括 NIST 等很多评价指标都是受到 BLEU 的启发。当然，BLEU 也并不完美，甚至经常被人诟病。比如，它需要依赖参考译文，而且评价结果有时与人工评价不一致，同时 BLEU 评价只是单纯地从匹配度的角度思考翻译质量的好坏，并没有真正考虑句子的语义是否翻译正确。但是，毫无疑问，BLEU 仍然是机器翻译中最常用的评价方法。在没有找到更好的替代方案之前，BLEU 还是机器翻译研究所使用的标准评价指标。

TER

TER 是 Translation Edit Rate 的缩写，是一种基于距离的评价方法，用来评定机器翻译结果的译后编辑的工作量 [261]。这里，距离被定义为将一个序列转换成另一个序列所需要的最少编辑操作次数。操作次数越多，距离越大，序列之间的相似性

越低；相反距离越小，表示一个句子越容易改写成另一个句子，序列之间的相似性越高。TER 使用的编辑操作包括：增加、删除、替换和移位。其中增加、删除、替换操作计算得到的距离被称为编辑距离，并根据错误率的形式给出评分：

$$\text{score} = \frac{\text{edit}(c, r)}{l} \quad (1.5)$$

其中 $\text{edit}(c, r)$ 是指机器翻译生成的候选译文 c 和参考译文 r 之间的距离， l 是归一化因子，通常为参考译文的长度。在距离计算中所有的操作的代价都为 1。在计算距离时，优先考虑移位操作，再计算编辑距离，也就是增加、删除和替换操作的次数。直到增加、移位操作无法减少编辑距离时，将编辑距离和移位操作的次数累加得到 TER 计算的距离。例如：

■ **实例 1.2** Candidate: cat is standing in the ground

Reference: The cat is standing on the ground ■

将 Candidate 转换为 Reference，需要进行一次增加操作，在句首增加“The”；一次替换操作，将“in”替换为“on”。所以 $\text{edit}(c, r) = 2$ ，归一化因子 l 为 Reference 的长度 7，所以该参考译文的 TER 错误率为 $2/7$ 。

与 BLEU 不同，基于距离的评价方法是一种典型的“错误率”的度量，类似的思想也广泛应用于语音识别等领域。在机器翻译中，除了 TER 外，还有 WER, PER 等十分相似的方法，只是在“错误”的定义上略有不同。需要注意的是，很多时候，研究者并不会单独使用 BLEU 或者 TER，而是将两种方法融合，比如，使用 BLEU 与 TER 相减后的值作为评价指标。

基于检测点的评价

BLEU、TER 等评价指标可以对译文的整体质量进行评估，但是缺乏对具体问题的细致评价。很多时候，研究人员需要知道系统是否能够处理特定的问题，而不是得到一个笼统的评价结果。基于监测点的方法正是基于此想法 [258]。基于检测点的评价的优点在于对机器翻译系统给出一个总体评价的同时针对系统在各个具体问题上的翻译能力进行评估，方便比较不同翻译模型的性能。这种方法也被多次用于机器翻译比赛的质量评测。

基于检测点的评价是根据事先定义好的语言学检测点对译文的相应部分进行打分。如下是几个英中翻译中的检测点实例：

■ **实例 1.3** They got up at six this morning.

他们今天早晨六点钟起床。

检测点：时间词的顺序。 ■

■ **实例 1.4** There are nine cows on the farm.

农场里有九头牛。

检测点：量词“头”

■ **实例 1.5** His house is on the south bank of the river.

他的房子在河的南岸。

We keep our money in a bank.

我们在一家银行存钱。

检测点：bank 的多义翻译

基于检测点的评价方法的意义在于，它并不是简单给出一个分数，而是帮助系统研发人员定位问题。因此这类方法更多地使用在对机器翻译的结果进行分析上，是对 BLEU 等整体评价指标的一种很好的补充。

1.6 机器翻译应用

机器翻译有着十分广泛的应用，下面看一下机器翻译在生活中的具体应用形式：

（一）网页翻译

进入信息爆炸的时代之后，互联网上海量的数据随处可得，然而由于国家和地区语言的不同，网络上的数据也呈现出多语言的特性。当人们在遇到包含不熟悉语言的网页时，无法及时有效地获取其中的信息。因此，对不同语言的网页进行翻译是必不可少的一步。由于网络上网页的数量数不胜数，依靠人工对网页进行翻译是不切实际的，相反，机器翻译十分适合这个任务。目前，市场上有很多浏览器提供网页翻译的服务，极大地简化了人们从网络上获取不同语言信息的难度。

（二）科技文献翻译

在专利等科技文献翻译中，往往需要将文献翻译为英语或者其他语言，比如摘要翻译。以往这种翻译工作通常由人工来完成。由于对翻译结果的质量要求较高，因此要求翻译人员具有相关背景知识，这导致译员资源稀缺。特别是，近几年国内专利申请数不断增加，这给人工翻译带来了很大的负担。相比于人工翻译，机器翻译可以在短时间内完成大量的专利翻译，同时结合术语词典和人工校对等方式，可以保证专利的翻译质量。同时，以专利为代表的科技文献往往具有很强的领域性，针对各类领域文本进行单独优化，机器翻译的品质可以大大提高。因此，机器翻译在专利翻译等行业有十分广泛的应用前景。

（三）视频字幕翻译

随着互联网的普及，人们可以通过互联网接触到大量境外影视作品。由于人们可能没有相应的外语能力，通常需要专业人员对字幕进行翻译（如图 1.13）。因此，这些境外视频的传播受限于字幕翻译的速度和准确度。现在的一些视频网站在使用语音识别为视频生成源语字幕的同时，通过机器翻译技术为各种语言的受众提供质量尚可的目标语言字幕，这种方式为人们提供了极大的便利。

（四）社交



图 1.13: 电影字幕

社交是人们的重要社会活动。人们可以通过各种各样的社交软件做到即时通讯，进行协作或者分享自己的观点。然而受限于语言问题，人们的社交范围往往不会超出自己所掌握的语种范围，很难方便地进行跨语言社交。随着机器翻译技术的发展，越来越多的社交软件开始支持自动翻译，用户可以轻易地将各种语言的内容翻译成自己的母语，方便了人们的交流，让语言问题不再成为社交的障碍。

（五）同声传译

在一些国际会议中，与会者来自许多不同的国家，为了保证会议的流畅，通常需要专业译员进行同声传译。同声传译需要在不打断演讲的同时，不间断地将讲话内容进行口译，对翻译人员的素质要求极高，成本高昂。现在，一些会议开始采用语音识别来将语音转换成文本，同时使用机器翻译技术进行翻译的方式，达到同步翻译的目的。这项技术已经得到了多个企业的关注，并在很多重要会议上进行尝试，取得了很好的反响。不过同声传译达到真正的使用还需一定时间的打磨，特别是会议场景下，准确进行语音识别和翻译仍然具有挑战性。

（六）医药领域翻译

在医药领域中，从药品研发、临床试验到药品注册，都有着大量的翻译需求。比如，在新药注册阶段，限定申报时间的同时，更是对翻译质量有着极高的要求。由于医药领域专业词汇量庞大、单词冗长复杂、术语准确且文体专业性强，翻译难度明显高于其他领域，人工翻译的方式代价大且很难满足效率的要求。为此，机器翻译近几年在医药领域取得广泛应用。在针对医药领域进行优化后，机器翻译质量可以很好地满足翻译的要求。

（七）中国传统语言文化的翻译

中国几千年的历史留下了极为宝贵的文化遗产，而其中，文言文作为古代书面语，具有言文分离、行文简练的特点，易于流传。言文分离的特点使得文言文和现在的标准汉语具有一定的区别。为了更好发扬中国传统文化，我们需要对文言文进行翻译。而文言文古奥难懂，人们需要具备一定的文言文知识背景才能准确翻译。机器翻译技术也可以帮助人们快速完成文言文的翻译。除此之外，机器翻译技术同样可以用于古诗生成和对联生成等任务。

（八）全球化

在经济全球化的今天，很多企业都有国际化的需求，企业员工或多或少地会遇到一些跨语言阅读和交流的情况，比如阅读进口产品的说明书，跨国公司之间的邮件、说明文件等等。相比于成本较高的人工翻译，机器翻译往往是一种很好的选择。在一些质量要求不高的翻译场景中，机器翻译可以得到应用。

（九）翻译机

出于商务、学术交流或者旅游的目的，人们在出国时会面临着跨语言交流的问题。近几年，随着出境人数的增加，不少企业推出了翻译机产品。通过结合机器翻译、语音识别和图像识别技术，翻译机实现了图像翻译和语音翻译的功能。用户可以很便捷地获取一些外语图像文字和语音信息，同时可以通过翻译机进行对话，降低跨语言交流门槛。

（十）翻译结果后编辑

翻译结果后编辑是指在机器翻译的结果之上，通过少量的人工编辑来进一步完善机器译文。在传统的人工翻译过程中，翻译人员完全依靠人工的方式进行翻译，这虽然保证了翻译质量，但是时间成本高。相对应的，机器翻译具有速度快和成本低的优势。在一些领域，目前的机器翻译质量已经可以很大程度上减小翻译人员的工作量，翻译人员可以在机器翻译的辅助下，花费相对较小的代价来完成翻译。

1.7 开源项目与评测

从实践的角度，机器翻译的发展主要可以归功于两方面的推动作用：开源系统和评测。开源系统通过代码共享的方式使得最新的研究成果可以快速传播，同时实验结果可以复现。而评测比赛，使得各个研究组织的成果可以进行科学的对比，共同推动机器翻译的发展与进步。此外，开源项目也促进了不同团队之间的协作，让研究人员在同一个平台上集中力量攻关。

1.7.1 开源机器翻译系统

下面列举一些优秀的开源机器翻译系统：

统计机器翻译开源系统

- NiuTrans.SMT。NiuTrans[322] 是由东北大学自然语言处理实验室自主研发的统计机器翻译系统，该系统可支持基于短语的模型、基于层次短语的模型以及基于句法的模型。由于使用 C++ 语言开发，所以该系统运行时间快，所占存储空间少。系统中内嵌有 n -gram 语言模型，故无需使用其他的系统即可对完成语言建模。网址：<http://opensource.niutrans.com/smt/index.html>
- Moses。Moses[150] 是统计机器翻译时代最著名的系统之一，（主要）由爱丁堡大学的机器翻译团队开发。最新的 Moses 系统支持很多的功能，例如，它

既支持基于短语的模型，也支持基于句法的模型。Moses 提供因子化翻译模型 (Factored Translation Model)，因此该模型可以很容易地对不同层次的信息进行建模。此外，它允许将混淆网络和字格作为输入，可缓解系统的 1-best 输出中的错误。Moses 还提供了很多有用的脚本和工具，被机器翻译研究者广泛使用。网址：<http://www.statmt.org/moses/>

- Joshua。Joshua[174] 是由约翰霍普金斯大学的语言和语音处理中心开发的层次短语翻译系统。由于 Joshua 是由 Java 语言开发，所以它在不同的平台上运行或开发时具有良好的可扩展性和可移植性。Joshua 也是使用非常广泛的开源器翻译系统之一。网址：<https://cwiki.apache.org/confluence/display/JOSHUA/>
- SilkRoad。SilkRoad 是由五个国内机构（中科院计算所、中科院软件所、中科院自动化所、厦门大学和哈尔滨工业大学）联合开发的基于短语的统计机器翻译系统。该系统是中国乃至亚洲地区第一个开源的统计机器翻译系统。SilkRoad 支持多种解码器和规则提取模块，这样可以组合成不同的系统，提供多样的选择。网址：<http://www.nlp.org.cn/project/project.php?projid=14>
- SAMT。SAMT[357] 是由卡内基梅隆大学机器翻译团队开发的语法增强的统计机器翻译系统。SAMT 在解码的时候使用目标树来生成翻译规则，而不严格遵守目标语言的语法。SAMT 的一个亮点是它提供了简单但高效的方式在机器翻译中使用句法信息。由于 SAMT 在 hadoop 中实现，它可受益于大数据集的分布式处理。网址：<http://www.cs.cmu.edu/zollmann/samt/>
- HiFST。HiFST[122] 是剑桥大学开发的统计机器翻译系统。该系统完全基于有限状态自动机实现，因此非常适合对搜索空间进行有效的表示。网址：<http://ucam-smt.github.io/>
- cdec。cdec[67] 是一个强大的解码器，是由 Chris Dyer 和他的合作者们一起开发。cdec 的主要功能是它使用了翻译模型的一个统一的内部表示，并为结构预测问题的各种模型和算法提供了实现框架。所以，cdec 也可以在被用来做一个对齐系统或者一个更通用的学习框架。此外，由于使用 C++ 语言编写，cdec 的运行速度较快。网址：<http://cdec-decoder.org/index.php?title=MainPage>
- Phrasal。Phrasal[28] 是由斯坦福大学自然语言处理小组开发的系统。除了传统的基于短语的模型，Phrasal 还支持基于非层次短语的模型，这种模型将基于短语的翻译延伸到非连续的短语翻译，增加了模型的泛化能力。网址：<http://nlp.stanford.edu/phrasal/>
- Jane。Jane[290] 是一个基于短语和基于层次短语的机器翻译系统，由亚琛工业大学的人类语言技术与模式识别小组开发。Jane 提供了系统融合模块，因此可以非常方便的对多个系统进行融合。网址：<https://www-i6.informatik.rwth-aachen.de/jane/>

th-aachen.de/jane/

- GIZA++. GIZA++[221] 是 Franz Och 研发的用于训练 IBM 模型 1-5 和 HMM 单词对齐模型的工具包。在早期, GIZA++ 是所有统计机器翻译系统中词对齐的标配工具。网址: <https://github.com/moses-smt/giza-pp>
- FastAlign。FastAlign[66] 是一个快速, 无监督的词对齐工具, 由卡内基梅隆大学开发。网址: https://github.com/clab/fast_align

神经机器翻译开源系统

- GroundHog。GroundHog[7] 基于 Theano[239] 框架, 由蒙特利尔大学 LISA 实验室使用 Python 语言编写的一个框架, 旨在提供灵活而高效的方式来实现复杂的循环神经网络模型。它提供了包括 LSTM 在内的多种模型。Bahdanau 等人 在此框架上又编写了 GroundHog 神经机器翻译系统。该系统也作为了很多论文的基线系统。网址: <https://github.com/lisa-groundhog/GroundHog>
- Nematus。Nematus[249] 是英国爱丁堡大学开发的, 基于 Theano 框架的神经机器翻译系统。该系统使用 GRU 作为隐层单元, 支持多层网络。Nematus 编码端有正向和反向的编码方式, 可以同时提取源语句子中的上下文信息。该系统的一个优点是, 它可以支持输入端有多个特征的输入 (例如词的词性等)。网址: <https://github.com/EdinburghNLP/nematus>
- ZophRNN。ZophRNN[360] 是由南加州大学的 Barret Zoph 等人使用 C++ 语言开发的系统。Zoph 既可以训练序列表示模型 (如语言模型), 也可以训练序列到序列的模型 (如神经机器翻译模型)。当训练神经机器翻译系统时, ZophRNN 也支持多源输入。网址: https://github.com/isi-nlp/Zoph_RNN
- Fairseq。Fairseq[224] 是由 Facebook 开发的, 基于 PyTorch 框架的用以解决序列到序列问题的工具包, 其中包括基于卷积神经网络、基于循环神经网络、基于 Transformer 的模型等。Fairseq 是当今使用最广泛的神经机器翻译开源系统之一。网址: <https://github.com/facebookresearch/fairseq>
- Tensor2Tensor。Tensor2Tensor[287] 是由谷歌推出的, 基于 TensorFlow 框架的开源系统。该系统基于 Transformer 模型, 因此可以支持大多数序列到序列任务。得益于 Transformer 的网络结构, 系统的训练速度较快。现在, Tensor2Tensor 也是机器翻译领域广泛使用的开源系统之一。网址: <https://github.com/tensorflow/tensor2tensor>
- OpenNMT。OpenNMT[143] 系统是由哈佛大学自然语言处理研究组开源的, 基于 Torch 框架的神经机器翻译系统。OpenNMT 系统的早期版本使用 Lua 语言编写, 现在也扩展到了 TensorFlow 和 PyTorch, 设计简单易用, 易于扩展, 同

时保持效率和翻译精度。网址: <https://github.com/OpenNMT/OpenNMT>

- 斯坦福神经机器翻译开源代码库。斯坦福大学自然语言处理组 (Stanford NLP) 发布了一篇教程, 介绍了该研究组在神经机器翻译上的研究信息, 同时实现了多种翻译模型 [187]。网址: <https://nlp.stanford.edu/projects/nmt/>
- THUMT。清华大学 NLP 团队实现的神经机器翻译系统, 支持 Transformer 等模型 [344]。该系统主要基于 TensorFlow 和 Theano 实现, 其中 Theano 版本包含了 RNNsearch 模型, 训练方式包括 MLE (Maximum Likelihood Estimate), MRT (Minimum Risk Training), SST (Semi-Supervised Training)。TensorFlow 版本实现了 Seq2Seq, RNNsearch, Transformer 三种基本模型。网址: <https://github.com/THUNLP-MT/THUMT>
- NiuTrans.NMT。由小牛翻译团队基于 NiuTensor 实现的神经机器翻译系统。支持循环神经网络、Transformer 等结构, 并支持语言建模、序列标注、机器翻译等任务。支持机器翻译 GPU 与 CPU 训练及解码。其小巧易用, 为开发人员提供快速二次开发基础。此外, NiuTrans.NMT 已经得到了大规模应用, 形成了支持 187 种语言翻译的小牛翻译系统。网址: <http://opensource.niutrans.com/niutensor/index.html>
- MARIAN。主要由微软翻译团队搭建 [133], 其使用 C++ 实现的用于 GPU/CPU 训练和解码的引擎, 支持多 GPU 训练和批量解码, 最小限度依赖第三方库, 静态编译一次之后, 复制其二进制文件就能在其他平台使用。网址: <https://marian-nmt.github.io/>
- Sockeye。由 Awslabs 开发的神经机器翻译框架 [107]。其中支持 RNNsearch、Transformer、CNN 等翻译模型, 同时提供了从图片翻译到文字的模块以及 WMT 德英新闻翻译、领域适应任务、多语言零资源翻译任务的教程。网址: <https://awslabs.github.io/sockeye/>
- CytonMT。由 NICT 开发的一种用 C++ 实现的神经机器翻译开源工具包 [302]。主要支持 Transformer 模型, 并支持一些常用的训练方法以及解码方法。网址: <https://github.com/arthurxlu/cytonMt>
- OpenSeq2Seq。由 NVIDIA 团队开发的 [157] 基于 TensorFlow 的模块化架构, 用于序列到序列的模型, 允许从可用组件中组装新模型, 支持混合精度训练, 利用 NVIDIA Volta Turing GPU 中的 Tensor 核心, 基于 Horovod 的快速分布式训练, 支持多 GPU, 多节点多模式。网址: <https://nvidia.github.io/OpenSeq2Seq/html/index.html>
- NMTPyTorch。由勒芒大学语言实验室发布的基于序列到序列框架的神经网络翻译系统 [26], NMTPyTorch 的核心部分依赖于 Numpy, PyTorch 和 tqdm。其允许训练各种端到端神经体系结构, 包括但不限于神经机器翻译、图像字幕和

自动语音识别系统。网址：<https://github.com/lium-lst/nmtpytorch>

1.7.2 常用数据集及公开评测任务

机器翻译相关评测主要有两种组织形式，一种是由政府及国家相关机构组织，权威性高。如由美国国家标准技术研究所组织的 NIST 评测、日本国家科学咨询系统中心主办的 NACSIS Test Collections for IR (NTCIR) PatentMT、日本科学振兴机构 (Japan Science and Technology Agency, 简称 JST) 等组织联合举办的 Workshop on Asian Translation (WAT) 以及国内由中文信息学会主办的全国机器翻译大会 (China Conference on Machine Translation, 简称 CCMT); 另一种是由相关学术机构组织，具有领域针对性的特点，如倾向新闻领域的 Conference on Machine Translation (WMT) 以及面向口语的 International Workshop on Spoken Language Translation (IWSLT)。下面将针对上述评测进行简要介绍。

- CCMT (全国机器翻译大会)，前身为 CWMT (全国机器翻译研讨会) 是国内机器翻译领域的旗舰会议，自 2005 年起已经组织多次机器翻译评测，对国内机器翻译相关技术的发展产生了深远影响。该评测主要针对汉语、英语以及国内的少数民族语言 (蒙古语、藏语、维吾尔语等) 进行评测，领域包括新闻、口语、政府文件等，不同语言方向对应的领域也有所不同。评价方式不同届略有不同，主要采用自动评价的方式，自 CWMT 2013 起则针对某些领域增设人工评价。自动评价的指标一般包括 BLEU-SBP、BLEU-NIST、TER、METEOR、NIST、GTM、mWER、mPER 以及 ICT 等，其中以 BLEU-SBP 为主，汉语为目标语的翻译采用基于字符的评价方式，面向英语的翻译采用基于词的评价方式。每年该评测吸引国内外近数十家企业及科研机构参赛，业内认可度极高。关于 CCMT 的更多信息可参考中文信息学会机器翻译专业委员会相关页面：<http://sc.cipsc.org.cn/mt/index.php/CWMT.html>。
- WMT 由 Special Interest Group for Machine Translation (SIGMT) 主办，会议自 2006 年起每年召开一次，是一个涉及机器翻译多种任务的综合性会议，包括多领域翻译评测任务、质量评价任务以及其他与机器翻译的相关任务 (如文档对齐评测等)。现在 WMT 已经成为机器翻译领域的旗舰评测会议，很多研究工作都以 WMT 评测结果作为基准。WMT 评测涉及的语言范围较广，包括英语、德语、芬兰语、捷克语、罗马尼亚语等十多种语言，翻译方向一般以英语为核心，探索英语与其他语言之间的翻译性能，领域包括新闻、信息技术、生物医学。最近，也增加了无指导机器翻译等热门问题。WMT 在评价方面类似于 CCMT，也采用人工评价与自动评价相结合的方式，自动评价的指标一般为 BLEU、TER 等。此外，WMT 公开了所有评测数据，因此也经常被机器翻译相关人员所使用。更多 WMT 的机器翻译评测相关信息可参考 SIGMT 官网：<http://www.sigmt.org/>。

- NIST 机器翻译评测开始于 2001 年，是早期机器翻译公开评测中颇具代表性的任务，现在 WMT 和 CCMT 很多任务的设置也大量参考了当年 NIST 评测的内容。NIST 评测由美国国家标准技术研究所主办，作为美国国防高级计划署（DARPA）中 TIDES 计划的重要组成部分。早期，NIST 评测主要评价阿拉伯语和汉语等语言到英语的翻译效果，评价方法一般采用人工评价与自动评价相结合的方式。人工评价采用 5 分制评价。自动评价使用多种方式，包括 BLEU, METEOR, TER 以及 HyTER。此外 NIST 从 2016 年起开始对稀缺语言资源技术进行评估，其中机器翻译作为其重要组成部分共同参与评测，评测指标主要为 BLEU。除对机器翻译系统进行评测之外，NIST 在 2008 和 2010 年对于机器翻译的自动评价方法（MetricsMaTr）也进行了评估，以鼓励更多研究人员对现有评价方法进行改进或提出更加贴合人工评价的方法。同时 NIST 评测所提供的数据集由于数据质量较高受到众多科研人员喜爱，如 MT04, MT06 等（汉英）平行语料经常被科研人员在实验中使用。不过，近几年 NIST 评测已经停止。更多 NIST 的机器翻译评测相关信息可参考官网：<https://www.nist.gov/programs-projects/machine-translation>。
- 从 2004 年开始举办的 IWSLT 也是颇具特色的机器翻译评测，它主要关注口语相关的机器翻译任务，测试数据包括 TED talks 的多语言字幕以及 QED 教育讲座影片字幕等，语言涉及英语、法语、德语、捷克语、汉语、阿拉伯语等众多语言。此外在 IWSLT 2016 中还加入了对于日常对话的翻译评测，尝试将微软 Skype 中一种语言的对话翻译成其他语言。评价方式采用自动评价的模式，评价标准和 WMT 类似，一般为 BLEU 等指标。另外，IWSLT 除了对文本到文本的翻译评测外，还有自动语音识别以及语音转另一种语言的文本的评测。更多 IWSLT 的机器翻译评测相关信息可参考 IWSLT 2019 官网：<https://workshop2019.iwslt.org/>。
- 日本举办的机器翻译评测 WAT 是亚洲范围内的重要评测之一，由日本科学振兴机构（JST）、情报通信研究机构（NICT）等多家机构共同组织，旨在为亚洲各国之间交流融合提供便宜之处。语言方向主要包括亚洲主流语言（汉语、韩语、印地语等）以及英语对日语的翻译，领域丰富多样，包括学术论文、专利、新闻、食谱等。评价方式包括自动评价（BLEU、RIBES 以及 AMFM 等）以及人工评价，其特点在于对于测试语料以段落为单位进行评价，考察其上下文关联的翻译效果。更多 WAT 的机器翻译评测相关信息可参考官网：<http://lotus.kuee.kyoto-u.ac.jp/WAT/>。
- NTCIR 计划是由日本国家科学咨询系统中心策划主办的，旨在建立一个用在自然语言处理以及信息检索相关任务上的日文标准测试集。在 NTCIR-9 的和 NTCIR-10 中开设的 Patent Machine Translation（PatentMT）任务主要针对专利领域进行翻译测试，其目的在于促进机器翻译在专利领域的发展和应用。在

NTCIR-9 中, 评测方式采取人工评价与自动评价相结合, 以人工评价为主导。人工评价主要根据准确度和流畅度进行评估, 自动评价采用 BLEU、NIST 等方式进行。NTCIR-10 评价方式在此基础上增加了专利审查评估、时间评估以及多语种评估, 分别考察机器翻译系统在专利领域翻译的实用性、耗时情况以及不同语种的翻译效果等。更多 NTCIR 评测相关信息可参考官网: <http://research.nii.ac.jp/ntcir/index-en.html>。

以上评测数据大多可以从评测网站上下载, 此外部分数据也可以从 LDC (Linguistic Data Consortium) 上申请, 网址为 <https://www.ldc.upenn.edu/>。ELRA (European Language Resources Association) 上也有一些免费的语料库供研究使用, 其官网为 <http://www.elra.info/>。更多机器翻译的语料信息可参看附录 A。

从机器翻译发展的角度看, 这些评测任务给相关研究提供了基准数据集, 使得不同的系统都可以在同一个环境下进行比较和分析, 进而建立了机器翻译研究所需的实验基础。此外, 公开评测也使得研究者可以第一时间了解机器翻译研究的最新成果, 比如, 有多篇 ACL 会议最佳论文的灵感就来自当年参加机器翻译评测任务的系统。

1.8 推荐学习资源

1.8.1 经典书籍

首先, 推荐一本书《Statistical Machine Translation》[147], 其作者是机器翻译领域著名学者 Philipp Koehn 教授。该书是机器翻译领域内的经典之作, 介绍了统计机器翻译技术的进展。该书从语言学和概率学两个方面介绍了统计机器翻译的构成要素, 然后介绍了统计机器翻译的主要模型: 基于词、基于短语和基于树的模型, 以及机器翻译评价、语言建模、判别式训练等方法。此外, 作者在该书的最新版本中增加了神经机器翻译的章节, 方便研究人员全面了解机器翻译的最新发展趋势 [148]。

《Foundations of Statistical Natural Language Processing》[192] 中文译名《统计自然语言处理基础》, 作者是自然语言处理领域的权威 Chris Manning 教授和 Hinrich Schütze 教授。该书对统计自然语言处理方法进行了全面介绍。书中讲解了统计自然语言处理所需的语言学和概率论基础知识, 介绍了机器翻译评价、语言建模、判别式训练以及整合语言学信息等基础方法。其中也包含了构建自然语言处理工具所需的基本理论和算法, 提供了对数学和语言学基础内容广泛而严格的覆盖, 以及统计方法的详细讨论。

《统计自然语言处理》[364] 由中国科学院自动化所宗成庆教授所著。该书中系统介绍了统计自然语言处理的基本概念、理论方法和最新研究进展, 既有对基础知识和理论模型的介绍, 也有对相关问题的研究背景、实现方法和技术现状的详细阐述。可供从事自然语言处理、机器翻译等研究的相关人员参考。

Ian Goodfellow、Yoshua Bengio、Aaron Courville 三位机器学习领域的学者所写

的《Deep Learning》[90]也是值得一读的参考书。其讲解了有关深度学习常用的方法，其中很多都会在深度学习模型设计和使用中用到。同时在该书的应用一章中也简单讲解了神经机器翻译的任务定义和发展过程。

《Neural Network Methods for Natural Language Processing》[88]是Yoav Goldberg编写的面向自然语言处理的深度学习参考书。相比《Deep Learning》，该书聚焦在自然语言处理中的深度学习方法，内容更加易读，非常适合刚入门自然语言处理及深度学习应用的人员参考。

《机器学习》[362]由南京大学周志华教授所著，作为机器学习领域入门教材，该书尽可能地涵盖了机器学习基础知识的各个方面，试图尽可能少地使用数学知识介绍机器学习方法与思想。

《统计学习方法》[365]由李航博士所著，该书对机器学习的有监督和无监督等方法进行了全面而系统的介绍。可以作为梳理机器学习的知识体系，同时了解相关基础概念的参考读物。

《神经网络与深度学习》[367]由复旦大学邱锡鹏教授所著，全面地介绍了神经网络和深度学习的基本概念和常用技术，同时涉及了许多深度学习的前沿方法。该书适合初学者阅读，同时又不失为一本面向专业人士的参考书。

1.8.2 网络资源

TensorFlow 官网提供了一个有关神经机器翻译的教程，介绍了从数据处理开始如何利用 TensorFlow 工具从零搭建一个神经机器翻译系统以及如何解码，其地址为https://www.tensorflow.org/tutorials/text/nmt_with_attention。此外谷歌和 Facebook 也分别提供了基于序列到序列机器翻译模型的高级教程。谷歌的版本是基于 TensorFlow 实现，网址为：<https://github.com/tensorflow/nmt>。Facebook 的教程主要是基于 PyTorch 实现，网址为：https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html。网站上也包含一些综述论文，其中详细的介绍了神经机器翻译的发展历程，问题定义以及目前遇到的问题。

<http://www.statmt.org>是一个介绍机器翻译研究的网站，该网站包含了对统计机器翻译研究的一些介绍资料，一些自然语言处理的会议，常用工具以及语料库。<http://www.mt-archive.info>与<https://www.aclweb.org/anthology>网站上有许多介绍机器翻译和自然语言处理的论文，通过这些网站还可以了解到自然语言处理领域的一些重要期刊和会议。

1.8.3 专业组织和会议

许多自然语言处理的相关学术组织会定期举办学术会议。与机器翻译相关的会议有：

- ACL, 全称 Annual Conference of the Association for Computational Linguistics, 是自然语言处理领域最高级别的会议。每年举办一次，主题涵盖计算语言学的所

有方向。

- NAACL, 全称 Annual Conference of the North American Chapter of the Association for Computational Linguistics, 为 ACL 北美分会, 在自然语言处理领域也属于顶级会议, 每年会选择在一个北美城市召开会议。
- EMNLP, 全称 Conference on Empirical Methods in Natural Language Processing, 自然语言处理另一个顶级会议之一, 由 ACL 当中对语言数据和经验方法有特殊兴趣的团体主办, 始于 1996 年。会议比较偏重于方法和经验性结果。
- COLING, 全称 International Conference on Computational Linguistics, 自然语言处理老牌顶级会议之一。该会议始于 1965 年, 是由 ICCL 国际计算语言学委员会主办。会议简称为 COLING, 是谐音瑞典著名作家 Albert Engström 小说中的虚构人物 Kolingen。COLING 每两年举办一次。
- EACL, 全称 Conference of the European Chapter of the Association for Computational Linguistics, 为 ACL 欧洲分会, 虽然在欧洲召开, 会议也吸引了全世界的大量学者投稿并参会。
- AACL, 全称 Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, 为 ACL 亚太地区分会。2020 年会议首次召开, 是亚洲地区自然语言处理领域最具影响力的会议之一。
- WMT, 全称 Conference on Machine Translation。机器翻译领域一年一度研讨会, 是国际公认的顶级机器翻译赛事之一。
- AMTA, 全称 Conference of the Association for Machine Translation in the Americas。AMTA 会议汇聚了学术界、产业界和政府的研究人员、开发人员和用户, 让工业界和学术界进行交流。
- CCL, 全称 China National Conference on Computational Linguistics, 中文为中国计算语言学大会。中国计算语言学大会创办于 1991 年, 由中国中文信息学会计算语言学专业委员会负责组织。经过 20 余年的发展, 中国计算语言学大会已成为国内自然语言处理领域权威性最高、规模和影响最大的学术会议。作为中国中文信息学会(国内一级学会)的旗舰会议, CCL 聚焦于中国境内各类语言的智能计算和信息处理, 为研讨和传播计算语言学最新学术和技术成果提供了最广泛的高层次交流平台。
- NLPCC, 全称 CCF International Conference on Natural Language Processing and Chinese Computing。NLPCC 是由中国计算机学会(CCF)主办的 CCF 中文信息技术专业委员会年度学术会议, 专注于自然语言处理及中文处理领域的研究和应用创新。会议自 2012 年开始举办, 主要活动有主题演讲、论文报告、技术测评等多种形式。

- CCMT, 全称 China Conference on Machine Translation, 中国机器翻译研讨会, 由中国中文信息学会主办, 旨在为国内外机器翻译界同行提供一个平台, 促进中国机器翻译事业。CCMT 不仅是国内机器翻译领域最具影响力、最权威的学术和评测活动, 而且也代表着汉语与民族语言翻译技术的最高水准, 对民族语言技术发展具有重要意义。

除了会议之外,《中文信息学报》、《Computational Linguistics》、《Machine Translation》、《Transactions of the Association for Computational Linguistics》、《IEEE/ACM Transactions on Audio, Speech, and Language Processing》、《ACM Transactions on Asian and Low Resource Language Information Processing》、《Natural Language Engineering》等期刊也发表了许多与机器翻译相关的重要论文。



2. 词法、语法及统计建模基础

机器翻译并非是一个孤立的系统，它依赖于很多模块，并且需要很多学科知识的融合。现在的机器翻译系统大多使用统计模型对翻译问题进行建模，同时也会用到一些自然语言处理工具来对不同语言的文字进行分析。因此，在正式开始机器翻译内容的介绍之前，本章将会对相关的基础知识进行概述，包括：概率论与统计建模基础、语言分析、语言建模等。

概率论与统计建模是机器翻译方法的基础。这里会对机器翻译所涉及的基本数学概念进行简要描述，确保后续使用到的数学工具是完备的。本章会重点关注如何利用统计建模的方式对自然语言处理问题进行描述，这种手段在统计机器翻译和神经机器翻译中会被使用。

语言分析部分将以汉语为例介绍词法和句法分析的基本概念。它们都是自然语言处理中的经典问题，而且在机器翻译中也会经常被使用。同样，本章会介绍这两个任务的定义和求解问题的思路。

语言建模是机器翻译中最常用的一种技术，它主要用于句子的生成和流畅度评价。本章会以传统统计语言模型为例，对语言建模的相关概念进行介绍。但是，这里并不深入探讨语言模型技术，在后面的章节中还会单独对神经网络语言模型等前沿技术进行讨论。

2.1 问题概述

很多时候机器翻译系统被看作是孤立的“黑盒”系统（图 2.1 (a)）。将一段文本作为输入送入机器翻译系统之后，系统输出翻译好的译文。但是真实的机器翻译系统非常复杂，因为系统看到的输入和输出实际上只是一些符号串，这些符号并没有任何意义，因此需要进一步对这些符号串进行处理才能更好的使用它们。比如，需要定义翻译中最基本的单元是什么？符号串是否具有结构信息？如何用数学工具刻画这些基本单元和结构？

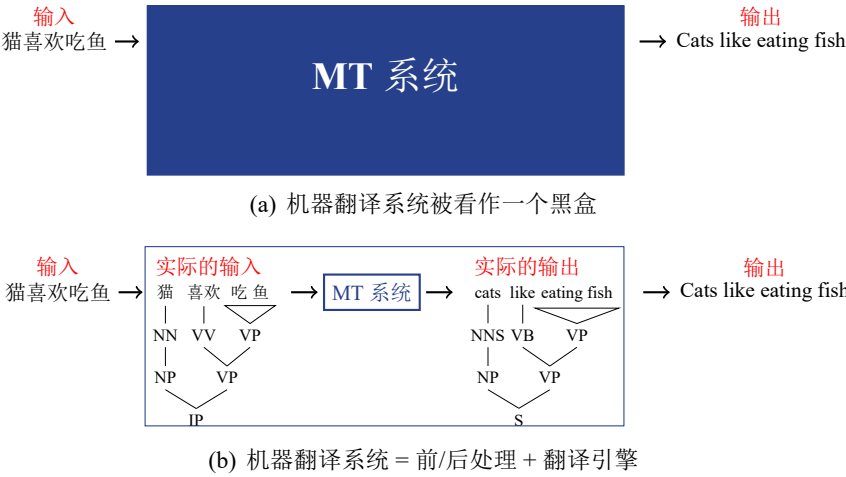


图 2.1: 机器翻译系统的结构

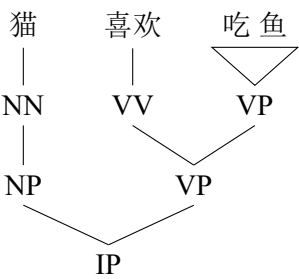


图 2.2: 中文句子“猫喜欢吃鱼”的分析结果（分词和句法分析）

图2.1 (b) 展示了一个机器翻译系统的输入和输出形式。可以看到，输入的中文词串“猫喜欢吃鱼”被加工成一个新的结构（图2.2）。直觉上，这个结构有些奇怪，因为上面多了很多新的符号，而且还有一些线将不同符号进行连接。实际上这就是语言分析中对句子常用的结构表示——短语结构树。从原始的词串转化为图2.2 的样子，有两个步骤：

- **分词 (Segmentation)**：这个过程会把词串进行切分，切割成最小的单元。因为只有知道了什么是待处理字符串的最小单元，机器翻译系统才能对其进行表示、

分析和生成。

- **句法分析 (Parsing)**：这个过程会对分词的结果进行进一步分析，得到句子的句法结构。这种结构是对句子的进一步抽象，比如，NP+VP 就可以表示由名词短语 (NP) 和动词短语 (VP) 构成的主谓结构。利用这些信息，机器翻译可以更加准确地对语言的结构进行分析和生成。

类似地，机器翻译输出的结果也可以包含同样的信息。甚至系统输出英文译文之后，还有一个额外的步骤来把部分英文单词的大小写恢复出来，比如，上例中句首单词 *Cats* 的首字母要大写。

一般来说，在送入机器翻译系统前需要对文字序列进行处理和加工，这个过程被称为**预处理 (Pre-processing)**。同理，在机器翻译模型输出译文后进行的处理被称作**后处理 (Post-processing)**。这两个过程对机器翻译性能影响很大，比如，在神经机器翻译里，不同的分词策略可能会造成翻译性能的天差地别。

值得注意的是，有些观点认为，不论是分词还是句法分析，对于机器翻译来说并不要求符合人的认知和语言学约束。换句话说，机器翻译所使用的“单词”和“结构”本身并不是为了符合人类的解释，它们更直接目的是为了进行翻译。从系统开发的角度，有时候即使是一些与人类的语言习惯有差别的处理，仍然会带来性能的提升，比如在神经机器翻译中，在传统分词的基础上进一步使用双字节编码 (Byte Pair Encoding, BPE) 子词切分会使得机器翻译性能大幅提高。当然，自然语言处理中语言学信息的使用一直是学界关注的焦点。甚至关于语言学结构对机器翻译是否有作用这个问题也有争论。但是不能否认的是，无论是语言学的知识，还是计算机自己学习到的知识，对机器翻译都是有价值的。在后续章节会看到，这两种类型的知识对机器翻译帮助很大¹。

剩下的问题是如何进行句子的切分和结构的分析。思路有很多，一种常用的方法是对问题进行概率化，用统计模型来描述问题并求解之。比如，一个句子切分的好坏，并不是非零即一的判断，而是要估计出这种切分的可能性大小，最终选择可能性最大的结果进行输出。这也是一种典型的用统计建模的方式来描述自然语言处理问题的方法。

本章将会对上述问题及求解问题的方法进行介绍。首先，会用一个例子给出统计建模的基本思路，之后会应用这种方法进行中文分词、语言建模和句法分析。

2.2 概率论基础

为了便于后续内容的介绍，首先对本书中使用的概率和统计学概念进行说明。

¹笔者并不认同语言学结构对机器翻译的帮助有限，相反机器翻译需要更多的人类先验知识的指导。当然，这个问题不是这里讨论的重点。

2.2.1 随机变量和概率

在自然界中，很多**事件**（Event）是否会发生是不确定的。例如，明天会下雨、掷一枚硬币是正面朝上、扔一个骰子的点数是 5…… 这类事件可能会发生也可能不会发生。通过大量的重复试验，能发现其具有某种规律性的事件叫做**随机事件**。

随机变量（Random Variable）是对随机事件发生可能状态的描述，是随机事件的数量表征。设 $\Omega = \{\omega\}$ 为一个随机试验的样本空间， $X = X(\omega)$ 就是定义在样本空间 Ω 上的单值实数函数，即 $X = X(\omega)$ 为随机变量，记为 X 。随机变量是一种能随机选取数值的变量，常用大写的英文字母或希腊字母表示，其取值通常用小写字母来表示。例如，用 A 表示一个随机变量，用 a 表示变量 A 的一个取值。根据随机变量可以选取的值的某些性质，可以将其划分为离散变量和连续变量。

离散变量是指在其取值区间内可以被一一列举，总数有限并且可计算的数值变量。例如，用随机变量 X 代表某次投骰子出现的点数，点数只可能取 1~6 这 6 个整数， X 就是一个离散变量。

连续变量是在其取值区间内连续取值，无法被一一列举，具有无限个取值的变量。例如，图书馆的开馆时间是 8:30-22:00，用 X 代表某人进入图书馆的时间，时间的取值范围是 [8:30, 22:00] 这个时间区间， X 就是一个连续变量。

概率（Probability）是度量随机事件呈现其每个可能状态的可能性的数值，本质上它是一个测度函数 [368][155]。概率的大小表征了随机事件在一次试验中发生的可能性大小。用 $P(\cdot)$ 表示一个随机事件的可能性，即事件发生的概率。比如 $P(\text{太阳从东方升起})$ 表示“太阳从东方升起的可能性”，同理， $P(A = B)$ 表示的就是“ $A = B$ ”这件事的可能性。

在实际问题中，往往需要得到随机变量的概率值。但是，真实的概率值可能是无法准确知道的，这时就需要对概率进行**估计**，得到的结果是概率的**估计值**（Estimate）。在概率论中，一个很简单的方法是利用相对频度作为概率的估计值。如果 $\{x_1, x_2, \dots, x_n\}$ 是一个试验的样本空间，在相同情况下重复试验 N 次，观察到样本 $x_i (1 \leq i \leq n)$ 的次数为 $n(x_i)$ ，那么 x_i 在这 N 次试验中的相对频率是 $\frac{n(x_i)}{N}$ 。当 N 越来越大时，相对概率也就越来越接近真实概率 $P(x_i)$ ，即 $\lim_{N \rightarrow \infty} \frac{n(x_i)}{N} = P(x_i)$ 。实际上，很多概率模型都等同于相对频度估计，比如，对于一个服从多项式分布的变量的极大似然估计就可以用相对频度估计实现。

表 2.1: 离散变量 A 的概率分布

A	$a_1 = 1$	$a_2 = 2$	$a_3 = 3$	$a_4 = 4$	$a_5 = 5$	$a_6 = 6$
P_i	$P_1 = \frac{4}{25}$	$P_2 = \frac{3}{25}$	$P_3 = \frac{4}{25}$	$P_4 = \frac{6}{25}$	$P_5 = \frac{3}{25}$	$P_6 = \frac{1}{25}$

概率函数是用函数形式给出离散变量每个取值发生的概率，其实就是将变量的概率分布转化为数学表达形式。如果把 A 看做一个离散变量， a 看做变量 A 的一个取值，那么 $P(A)$ 被称作变量 A 的概率函数， $P(A = a)$ 被称作 $A = a$ 的概率值，简记

为 $P(a)$ 。例如，在相同条件下掷一个骰子 50 次，用 A 表示投骰子出现的点数这个离散变量， a_i 表示点数的取值， P_i 表示 $A = a_i$ 的概率值。表2.1为 A 的概率分布，给出了 A 的所有取值及其概率。

除此之外，概率函数 $P(\cdot)$ 还具有非负性、归一性等特点。非负性是指，所有的概率函数 $P(\cdot)$ 都必须大于等于 0 的数值，概率函数中不可能出现负数，即 $\forall x, P(x) \geq 0$ 。归一性，又称规范性，简单的说就是所有可能发生的事件的概率总和为 1，即 $\sum_x P(x) = 1$ 。

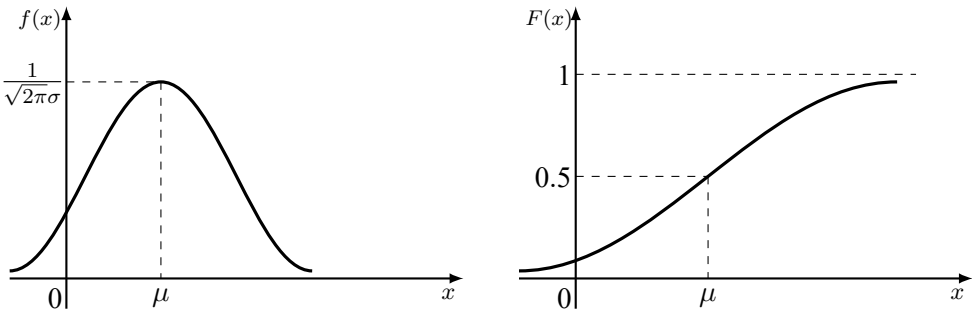


图 2.3: 一个概率密度函数 (左) 与其对应的分布函数 (右)

对于离散变量 A , $P(A = a)$ 是个确定的值, 可以表示事件 $A = a$ 的可能性大小; 对于连续变量, 求在某个定点处的概率是无意义的, 只能求其落在某个取值区间内的概率。因此, 用**概率分布函数** $F(x)$ 和**概率密度函数** $f(x)$ 来统一描述随机变量取值的分布情况 (如图2.3)。概率分布函数 $F(x)$ 表示取值小于等于某个值的概率, 是概率的累加 (或积分) 形式。假设 A 是一个随机变量, a 是任意实数, 将函数 $F(a) = P\{A \leq a\}$, $-\infty < a < \infty$ 定义为 A 的分布函数。通过分布函数, 可以清晰地表示任何随机变量的概率。

概率密度函数反映了变量在某个区间内的概率变化快慢, 概率密度函数的值是概率的变化率, 该连续变量的概率也就是对概率密度函数求积分得到的结果。设 $f(x) \geq 0$ 是连续变量 X 的概率密度函数, X 的分布函数就可以用如下公式定义:

$$F(x) = \int_{-\infty}^x f(x)dx \tag{2.1}$$

2.2.2 联合概率、条件概率和边缘概率

联合概率 (Joint Probability) 是指多个事件共同发生, 每个随机变量满足各自条件的概率, 表示为 $P(AB)$ 或 $P(A \cap B)$ 。**条件概率** (Conditional Probability) 是指 A 、 B 为任意的两个事件, 在事件 A 已出现的前提下, 事件 B 出现的概率, 使用 $P(B | A)$ 表示。

贝叶斯法则 (见2.2.4小节) 是条件概率计算时的重要依据, 条件概率可以表示

为

$$\begin{aligned}
 P(B|A) &= \frac{P(A \cap B)}{P(A)} \\
 &= \frac{P(A)P(B|A)}{P(A)} \\
 &= \frac{P(B)P(A|B)}{P(A)}
 \end{aligned} \tag{2.2}$$

边缘概率 (marginal probability) 是和联合概率对应的, 它指的是 $P(X = a)$ 或 $P(Y = b)$, 即仅与单个随机变量有关的概率。对于离散随机变量 X 和 Y , 如果知道 $P(X, Y)$, 则边缘概率 $P(X)$ 可以通过求和的方式得到。对于 $\forall x \in X$, 有

$$P(X = x) = \sum_y P(X = x, Y = y) \tag{2.3}$$

对于连续变量, 边缘概率 $P(X)$ 需要通过积分得到, 如下式所示

$$P(X = x) = \int P(x, y) dy \tag{2.4}$$

为了更好地区分条件概率、边缘概率和联合概率, 这里用一个图形面积的计算来举例说明。如图2.4所示, 矩形 A 代表事件 X 发生所对应的所有可能状态, 矩形 B 代表事件 Y 发生所对应的所有可能状态, 矩形 C 代表 A 和 B 的交集, 则

- 边缘概率: 矩形 A 或者矩形 B 的面积;
- 联合概率: 矩形 C 的面积;
- 条件概率: 联合概率/对应的边缘概率, 如: $P(A|B) = \text{矩形 } C \text{ 的面积} / \text{矩形 } B \text{ 的面积}$ 。

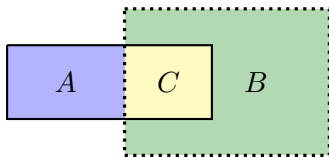


图 2.4: A 、 B 、 C 事件所对应概率的图形化表示

2.2.3 链式法则

条件概率公式 $P(A|B) = P(AB)/P(B)$ 反映了事件 B 发生的条件下事件 A 发生的概率。如果将其推广到三个事件 A 、 B 、 C , 为了计算 $P(A, B, C)$, 我们可以运用

两次 $P(A | B) = P(AB)/P(B)$ ，计算过程如下：

$$\begin{aligned} P(A, B, C) &= P(A | B, C)P(B, C) \\ &= P(A | B, C)P(B | C)P(C) \end{aligned} \quad (2.5)$$

推广到 n 个事件，可以得到了链式法则的公式

$$P(x_1, x_2, \dots, x_n) = P(x_1) \prod_{i=2}^n P(x_i | x_1, x_2, \dots, x_{i-1}) \quad (2.6)$$

下面的例子有助于更好的理解链式法则，如图2.5所示， A 、 B 、 C 、 D 、 E 分别代表五个事件，其中， A 只和 B 有关， C 只和 B 、 D 有关， E 只和 C 有关， B 和 D 不依赖其他任何事件。则 $P(A, B, C, D, E)$ 的表达式如下式：

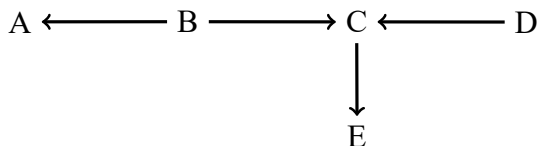


图 2.5: 事件 A 、 B 、 C 、 D 、 E 之间的关系图

$$\begin{aligned} &P(A, B, C, D, E) \\ &= P(E | A, B, C, D) \cdot P(A, B, C, D) \\ &= P(E | A, B, C, D) \cdot P(D | A, B, C) \cdot P(A, B, C) \\ &= P(E | A, B, C, D) \cdot P(D | A, B, C) \cdot P(C | A, B) \cdot P(A, B) \\ &= P(E | A, B, C, D) \cdot P(D | A, B, C) \cdot P(C | A, B) \cdot P(B | A) \cdot P(A) \end{aligned} \quad (2.7)$$

根据图2.5 易知 E 只和 C 有关，所以 $P(E | A, B, C, D) = P(E | C)$ ； D 不依赖于其他事件，所以 $P(D | A, B, C) = P(D)$ ； C 只和 B 、 D 有关，所以 $P(C | A, B) = P(C | B)$ ； B 不依赖于其他事件，所以 $P(B | A) = P(B)$ 。最终化简可得：

$$P(A, B, C, D, E) = P(E | C) \cdot P(D) \cdot P(C | B) \cdot P(B) \cdot P(A | B) \quad (2.8)$$

2.2.4 贝叶斯法则

首先介绍一下全概率公式：**全概率公式**（Law of Total Probability）是概率论中重要的公式，它可以将一个复杂事件发生的概率分解成不同情况的小事件发生概率的和。这里先介绍一个概念——划分。集合 S 的一个划分事件为 $\{B_1, \dots, B_n\}$ 是指它们满足 $\bigcup_{i=1}^n B_i = S$ 且 $B_i B_j = \emptyset, i, j = 1, \dots, n, i \neq j$ 。此时事件 A 的全概率公式可以被描述为：

$$P(A) = \sum_{k=1}^n P(A | B_k)P(B_k) \quad (2.9)$$

举个例子，小张从家到公司有三条路分别为 a , b , c ，选择每条路的概率分别为 0.5, 0.3, 0.2。令：

- S_a : 小张选择 a 路去上班
- S_b : 小张选择 b 路去上班
- S_c : 小张选择 c 路去上班
- S : 小张去上班

显然， S_a , S_b , S_c 是 S 的划分。如果三条路不拥堵的概率分别为 $P(S'_a)=0.2$, $P(S'_b)=0.4$, $P(S'_c)=0.7$ ，那么事件 L ：小张上班没有遇到拥堵情况的概率就是：

$$\begin{aligned} P(L) &= P(L|S_a)P(S_a) + P(L|S_b)P(S_b) + P(L|S_c)P(S_c) \\ &= P(S'_a)P(S_a) + P(S'_b)P(S_b) + P(S'_c)P(S_c) \\ &= 0.36 \end{aligned} \quad (2.10)$$

贝叶斯法则 (Bayes' rule) 是概率论中的一个经典公式，通常用于已知 $P(A | B)$ 求 $P(B | A)$ 。可以表述为：设 $\{B_1, \dots, B_n\}$ 是 S 的一个划分， A 为事件，则对于 $i = 1, \dots, n$ ，有如下公式

$$\begin{aligned} P(B_i | A) &= \frac{P(AB_i)}{P(A)} \\ &= \frac{P(A | B_i)P(B_i)}{\sum_{k=1}^n P(A | B_k)P(B_k)} \end{aligned} \quad (2.11)$$

其中，等式右端的分母部分使用了全概率公式。由上式，也可以得到贝叶斯公式的另外两种写法：

$$\begin{aligned} P(B | A) &= \frac{P(A | B)P(B)}{P(A)} \\ &= \frac{P(A | B)P(B)}{P(A | B)P(B) + P(A | \bar{B})P(\bar{B})} \end{aligned} \quad (2.12)$$

贝叶斯公式常用于根据已知的结果来推断使之发生的各因素的可能性。

2.2.5 KL 距离和熵

信息熵

熵 (Entropy) 是热力学中的一个概念，同时也是对系统无序性的一种度量标准。在自然语言处理领域也会使用到信息熵这一概念，比如描述文字的信息量大小。一条信息的信息量可以被看作是这条信息的不确定性。如果需要确认一件非常不确定甚至一无所知的事情，那么需要理解大量的相关信息才能进行确认；同样的，如果对某件事已经非常确定，那么就不需要太多的信息就可以把它搞清楚。如下就是两个例子，

■ 实例 2.1 确定性和不确定性的事件

“太阳从东方升起”

“明天天气多云”

■

在这两句话中，“太阳从东方升起”是一件确定性事件（在地球上），几乎不需要查阅更多信息就可以确认，因此这件事的信息熵相对较低；而“明天天气多云”这件事，需要关注天气预报，才能大概率确定这件事，它的不确定性很高，因而它的信息熵也就相对较高。因此，信息熵也是对事件不确定性的度量。进一步，定义**自信息** (Self-information) 为一个事件 X 的自信息的表达式为：

$$I(x) = -\log P(x) \quad (2.13)$$

其中， $P(x)$ 表示 x 发生的概率。自信息用来衡量单一事件发生时所包含的信息多少，当底数为 e 时，单位为 **nats**，其中 1nats 是通过观察概率为 $\frac{1}{e}$ 的事件而获得的信息量；当底数为 2 时，单位为 **bits** 或 **shannons**。 $I(x)$ 和 $P(x)$ 的函数关系如图2.6 所示。

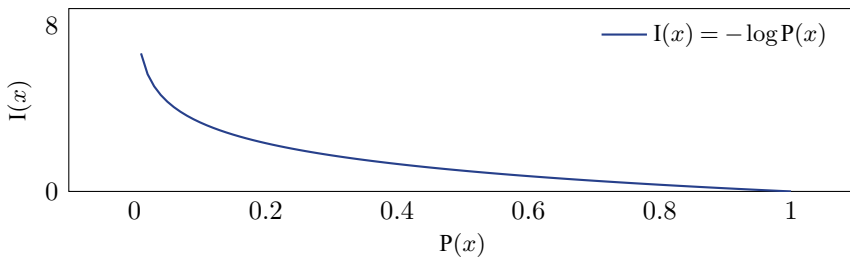


图 2.6: 自信息函数 $I(x)$ 关于 $P(x)$ 的曲线

自信息处理的是变量单一取值的情况。若量化整个概率分布中的不确定性或信息量，可以用信息熵，记为 $H(x)$ 。其公式如下：

$$\begin{aligned} H(x) &= \sum_{x \in X} [P(x)I(x)] \\ &= -\sum_{x \in X} [P(x) \log(P(x))] \end{aligned} \quad (2.14)$$

一个分布的信息熵也就是从该分布中得到的一个事件的期望信息量。比如， a 、 b 、 c 、 d 四支球队，四支队伍夺冠的概率分别是 P_1 、 P_2 、 P_3 、 P_4 ，某个人对比赛不感兴趣但是又想知道哪支球队夺冠，通过使用二分法 2 次就确定哪支球队夺冠了。但假设这四支球队中 c 的实力可以碾压其他球队，那么猜 1 次就可以确定。所以对于前面这种情况，哪支球队夺冠的信息量较高，信息熵也相对较高；对于后面这种情况，因为结果是容易猜到的，信息量和信息熵也就相对较低。因此可以得知：分布越尖锐熵越低；分布越均匀熵越高。

KL 距离

如果同一个随机变量 X 上有两个概率分布 $P(x)$ 和 $Q(x)$ ，那么可以使用 KL 距离 (“Kullback-Leibler” 散度) 来衡量这两个分布的不同，这种度量就是**相对熵** (Relative Entropy)。其公式如下：

$$\begin{aligned} D_{\text{KL}}(P \parallel Q) &= \sum_{x \in X} [P(x) \log \frac{P(x)}{Q(x)}] \\ &= \sum_{x \in X} [P(x)(\log P(x) - \log Q(x))] \end{aligned} \quad (2.15)$$

相对熵的意义在于：在一个事件空间里，概率分布 $P(x)$ 对应的每个事件的可能性。若用概率分布 $Q(x)$ 编码 $P(x)$ ，平均每个事件的信息量增加了多少。它衡量的是同一个事件空间里两个概率分布的差异。KL 距离有两条重要的性质：

- 非负性，即 $D_{\text{KL}}(P \parallel Q) \geq 0$ ，等号成立条件是 P 和 Q 相等。
- 不对称性，即 $D_{\text{KL}}(P \parallel Q) \neq D_{\text{KL}}(Q \parallel P)$ ，所以 KL 距离并不是常用的欧式空间中的距离。为了消除这种不确定性，有时也会使用 $D_{\text{KL}}(P \parallel Q) + D_{\text{KL}}(Q \parallel P)$ 作为度量两个分布差异性的函数。

交叉熵

交叉熵 (Cross-entropy) 是一个与 KL 距离密切相关的概念，它的公式是：

$$H(P, Q) = - \sum_{x \in X} [P(x) \log Q(x)] \quad (2.16)$$

结合相对熵公式可知，交叉熵是 KL 距离公式中的右半部分。因此，当概率分布 $P(x)$ 固定时，求关于 Q 的交叉熵的最小值等价于求 KL 距离的最小值。从实践的角度来说，交叉熵与 KL 距离的目的相同：都是用来描述两个分布的差异，由于交叉熵计算上更加直观方便，因此在机器翻译中被广泛应用。

2.3 中文分词

对于机器翻译系统而言，输入的是已经切分好的单词序列，而不是原始的字符串（图2.7）。比如，对于一个中文句子，单词之间是没有间隔的，因此需要把一个个的单词切分出来，这样机器翻译系统可以区分不同的翻译单元。甚至，可以对语言学上的单词进行进一步切分，得到词片段序列（比如：中国人 → 中国/人）。可以把上述过程看作是一种**分词**（Segmentation）过程，即：将一个输入的自然语言字符串切割成单元序列（token 序列），每个单元都对应可以处理的最小单位。



图 2.7: 一个简单的预处理流程

分词得到的单元序列可以是语言学上的词序列，也可以是根据其他方式定义的基本处理单元。在本章中，可以把分词得到的一个个单元称为**单词**（Word），或**词**，尽管这些单元可以不是语言学上的完整单词。而这个过程也被称作**词法分析**（Lexical Analysis）。除了汉语，词法分析在日语、泰语等单词之间无明确分割符的语言中有着广泛的应用，芬兰语、维吾尔语等一些形态学十分丰富的语言，也需要使用词法分析来解决复杂的词尾、词缀变化等形态学变化。

在机器翻译中，分词系统的好坏往往会决定译文的质量。分词的目的是定义系统处理的基本单元，那么什么叫做“词”呢？关于词的定义有很多，比如：

定义 2.3.1 词

- 语言里最小的可以独立运用的单位。
——《新华字典》
- 单词（word），含有语义内容或语用内容，且能被单独念出来的的最小单位。
——《维基百科》
- 語句中具有完整概念，能獨立自由運用的基本單位。
——《国语辞典》

从语言学的角度来看，人们普遍认为词是可以单独运用的、包含意义的基本单位。这样可以使用有限的词组合出无限的句子，这也正体现出自然语言的奇妙之处。不过，机器翻译并不仅仅局限在语言学定义的单词。比如，神经机器翻译中广泛使用的 BPE 子词切分方法（见第七章），可以被理解为将词的一部分也进行切开，也就是得到词片段送给机器翻译系统使用。比如，对如下英文字符串，可以得到如下切分结果：

Interesting → Interest/ing selection → se/lect/ion procession → pro/cess/ion
Interested → Interest/ed selecting → se/lect/ing processing → pro/cess/ing
Interests → Interest/s selected → se/lect/ed processed → pro/cess/ed

词法分析的重要性在自然语言处理领域已经有共识。如果切分的颗粒度很大，获得的单词的歧义也很小，比如“中华人民共和国”整体作为一个单词不存在歧义，而如果单独的一个单词“国”，可能会代表“中国”、“美国”等不同的国家，存在歧义。但是随着切分颗粒度的增大，特定单词出现的频度也随之降低，低频词容易和噪音混淆，系统很难进行学习。因此，处理这些问题并开发适合翻译任务的分词系统是机器翻译的第一步。

2.3.1 基于词典的分词方法

然而，计算机并不能像人类一样在概念上理解“词”，因此需要使用其他方式让计算机可以进行分词。一个最简单的方法就是给定一个词典，在这个词典中出现的汉字组合就是所定义的“词”。也就是，通过一个词典定义一个标准，符合这个标准定义的字符串都是合法的“词”。

在使用基于词典的分词方法时，只需预先加载词典到计算机中，扫描输入句子，查询每个词串是否出现在词典中。如图2.8所示，有一个包含六个词的词典，给定输入句子“确实现现在物价很高”后，分词系统自左至右遍历输入句子的每个字，发现词串“确实”在词典中出现，说明“确实”是一个“词”，进行分词操作并在切分该“词”之后重复这个过程。

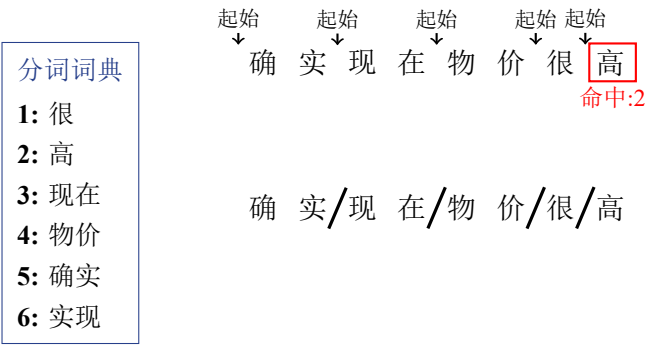


图 2.8: 基于词典进行分词的实例

但是，基于词典的分词方法很“硬”。这是因为自然语言非常灵活，经常出现歧义，用词典定义的合法单词之间有重叠的交叉型歧义就很难解决。图2.9就给出了上面例子中的交叉型歧义，从词典中查看，“实现”和“现在”都是合法的单词，但是在句子中二者有重叠，因此词典无法告诉我们哪个结果是正确的。

类似的例子在生活中也很常见。再比如“答辩结束的和尚未答辩的同学都请留在教室”一句中，正常的分词结果是“答辩/结束/的/和/和尚未/答辩/的/同学/都/请/留在/教

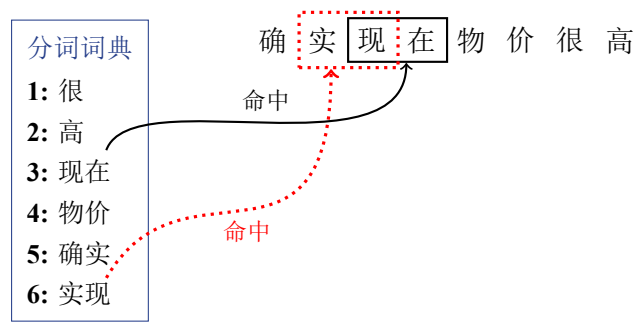


图 2.9: 交叉型分词歧义

室”，但是由于“尚未”、“和尚”都是常见词汇，使用基于词典的分词方法在这时很容易出现切分错误。

基于词典的分词方法是典型的基于规则的方法，完全依赖于人工给定的词典。在遇到歧义时，需要人工定义消除歧义的规则，比如，可以自左向右扫描每次匹配最长的单词，这是一种简单的启发式的消歧策略。图2.8中的例子实际上就是使用这种策略得到的分词结果。但是，启发式的消歧方法对人工的依赖程度很高，而且启发式规则也不能处理所有的情况。所以说简单的基于词典的方法还不能很好的解决分词问题。

2.3.2 基于统计的分词方法

既然基于词典的方法有很多问题，那么就需要一种更为有效的方法。在上文中提到，想要搭建一个分词系统，需要让计算机知道什么是“词”，那么可不可以给出已经切分好的分词数据，让计算机在这些数据中学习到规律呢？答案是肯定的，利用“数据”来让计算机明白“词”的定义，让计算机直接在数据中学到知识，这就常说的数据驱动的方法。这个过程也是一个典型的基于统计建模的学习过程。

统计模型的学习与推断

在分词任务中，数据驱动主要指用已经分词切分好的数据“喂”给系统，这个数据也被称作**标注数据**（Annotated Data）。在获得标注数据后，系统自动学习一个统计模型来描述分词的过程，而这个模型会把分词的“知识”作为参数保存在模型中。当送入一个新的需要分词的句子时，可以利用学习到的模型对所有可能的分词结果进行预测，并进行概率化的描述，最终选择概率最大的结果作为输出。这个方法就是基于统计的分词方法。具体来说，可以分为两个步骤：

- **训练**（Training）。利用标注数据，对统计模型的参数进行学习。
- **推断**（Inference）。利用学习到的模型和参数，对新的句子进行切分。

图2.10 给出了一个基于统计建模的汉语分词实例。左侧是标注数据，其中每个句子是已经经过人工标注的分词结果（单词用斜杠分开）。之后，建立一个统计模型，

记为 $P(\cdot)$ 。模型通过在标注数据上的学习来对问题进行描述，即学习 $P(\cdot)$ 。最后，对于新的未分词的句子，使用模型 $P(\cdot)$ 对每个可能的切分方式进行概率估计，之后选择概率最高的切分结果输出。

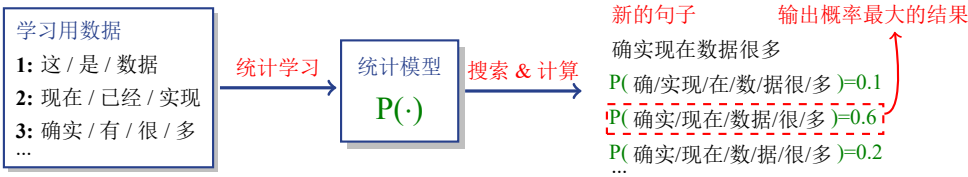


图 2.10: 基于统计的自动分词流程

掷骰子游戏

上述过程的核心在于从数据中学习一种对分词现象的统计描述，即学习函数 $P(\cdot)$ 。如何让计算机利用分词好的数据学习到分词的知识呢？可以先看一个有趣的实例（图2.11），用生活中比较常见的掷骰子来说，掷一个骰子，玩家猜一个数字，猜中就算赢，按照一般的常识，随便选一个数字，获胜的概率是一样的，即所有选择的获胜概率仅是 $1/6$ 。因此这个游戏玩家很难获胜，除非运气很好。假设进行一次游戏，玩家随便选了一个数字，比如是 1，投掷 30 次骰子，得到命中 $7/30 > 1/6$ ，还不错。

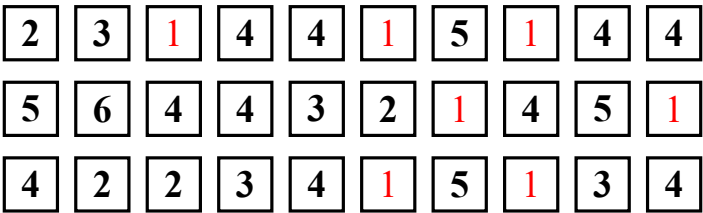


图 2.11: 骰子结果

似乎玩家的胜利只能来源于运气。不过，请注意，这里的假设“随便选一个数字”本身就是一个概率模型，它对骰子的六个面的出现做了均匀分布假设。

$$P(\text{“1”}) = P(\text{“2”}) = \dots = P(\text{“5”}) = P(\text{“6”}) = 1/6 \tag{2.17}$$

但是这个游戏没有人规定骰子是均匀的（有些被坑了的感觉）。如果骰子的六个面不均匀呢？我们可以用一种更加“聪明”的方式定义一种新的模型，即定义骰子的

每一个面都以一定的概率出现，而不是相同的概率。描述如下：

$$\begin{aligned} P(\text{“1”}) &= \theta_1 \\ P(\text{“2”}) &= \theta_2 \\ P(\text{“3”}) &= \theta_3 \\ P(\text{“4”}) &= \theta_4 \\ P(\text{“5”}) &= \theta_5 \\ P(\text{“6”}) &= 1 - \sum_{1 \leq i \leq 5} \theta_i \quad \triangleleft \text{归一性} \end{aligned}$$

(2.18)

这里， $\theta_1 \sim \theta_5$ 可以被看作是模型的参数，因此这个模型的自由度是 5。对于这样的模型，参数确定了，模型也就确定了。但是，新的问题来了，在定义骰子每个面的概率后，如何求出具体的值呢？一种常用的方法是，从大量实例中学习模型参数，这个方法也是常说的**参数估计**（Parameter Estimation）。可以将这个不均匀的骰子先实验性的掷很多次，这可以被看作是独立同分布的若干次采样，比如 X 次，发现“1”出现 X_1 次，“2”出现 X_2 次，以此类推，得到了各个面出现的次数。假设掷骰子中每个面出现的概率符合多项式分布，通过简单的概率论知识可以知道每个面出现概率的极大似然估计为：

$$P(\text{“}i\text{”}) = \frac{X_i}{X}$$

(2.19)

当 X 足够大的时， $\frac{X_i}{X}$ 可以无限逼近 $P(\text{“}i\text{”})$ 的真实值，因此可以通过大量的实验推算出掷骰子各个面的概率的准确估计值。回归到原始的问题，如果在正式开始游戏前，预先掷骰子 30 次，得到如图2.12的结果。

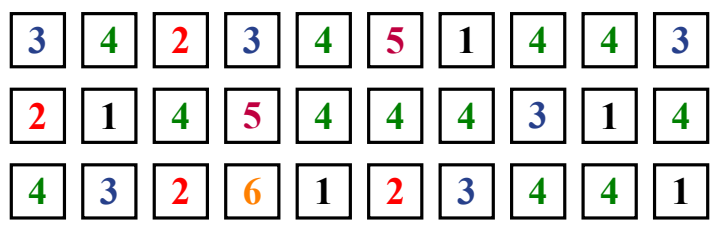


图 2.12: 实验性投掷骰子的结果

于是，我们看到了一个有倾向性的模型（图 2.13）：在这样的预先实验基础上，可以知道如果再次玩掷骰子游戏的话，选则数字“4”获胜的可能性是最大的。

通过上面这个掷骰子的游戏，可以得到一个道理：**上帝是不公平的**。因为在“公平”的世界中，没有任何一个模型可以学到有价值的事情。从机器学习的角度来看，所谓的“不公平”实际上这是客观事物中蕴含的一种**偏置**（Bias），也就是很多事情天然就有对某些情况有倾向。而图像处理、自然语言处理等问题中绝大多数都存在

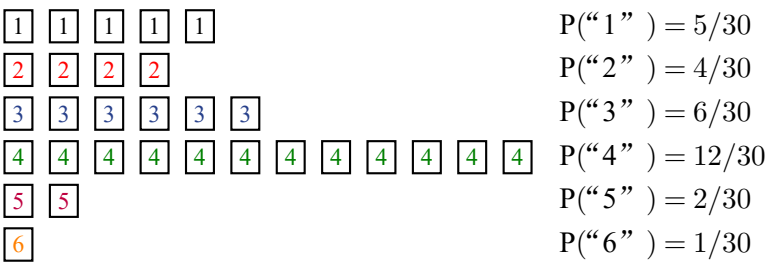


图 2.13: 投骰子模型

着偏置。比如，我们翻译一个英文单词的时候，它最可能的翻译结果往往就是那几个词。设计统计模型的目的正是要学习这种偏置，之后利用这种偏置对新的问题做出足够好的决策。

全概率分词方法

回到分词的问题上。与掷骰子游戏类似，分词系统的统计学原理也可以这么理解：假设有已经人工分词好的句子，其中每个单词的出现就好比掷一个巨大的骰子，与前面的例子中有所不同的是：

- 骰子有很多个面，每个面代表一个单词。
- 骰子是不均匀的，有些面会出现比较多次。

如果投掷这个新的骰子，可能会得到图2.14这样的结果，

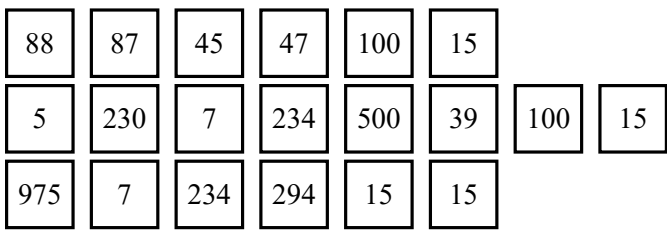


图 2.14: 投掷一个很多面骰子的结果

如果，把这些数字换成汉语中的词，比如

- 88 = 这
- 87 = 是
- 45 = 一

...

就可以得到图2.15所示的结果。

于是，在中文分词问题中，可以假设有一个不均匀的多面骰子，每个面对应一个单词。在获取人工分词标注数据后，可以统计每个单词出现的次数，进而利用极大似然估计推算出每个单词出现的概率的估计值。图2.16给出了一个实例。

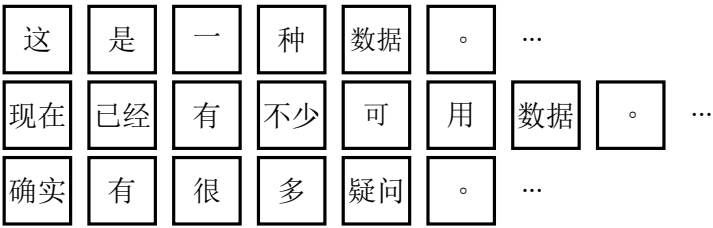


图 2.15: 掷骰子游戏中把数字换成汉字后的结果

总词数: $6 + 8 + 5 = 20$	更多数据-总词数: 100K ~ 1M
$P(\text{“很”}) = 1/20 = 0.05$	$P(\text{“很”}) = 0.000010$
$P(\text{“。”}) = 3/20 = 0.15$	$P(\text{“。”}) = 0.001812$
$P(\text{“确实”}) = 1/20 = 0.05$	$P(\text{“确实”}) = 0.000001$

图 2.16: 单词概率的估计结果

通过这个学习过程，我们得到了每个词出现的概率，即模型的参数。而我们原始的问题是如何计算这个整句分词结果的概率，比如， $P(\text{“确实/现在/数据/很/多”})=?$ 。这里可以使用“大题小做”的技巧：原始的问题很复杂，我们将其切分为小问题。这样，将复杂的分词问题简单化，基于独立性假设解决分词问题：假定所有词出现都是相互独立的。设 $w_1w_2w_3...w_m$ 表示一个由单词 $w_1, w_2, w_3, ..., w_m$ 组成的切分结果，于是有：

$$P(w_1w_2w_3...w_m) = P(w_1) \cdot P(w_2) \cdot \dots \cdot P(w_m) \tag{2.20}$$

以“确实现在数据很多”这个实例来说，如果把这句话按照“确实/现在/数据/很/多”这样的方式进行切分，这个句子切分的概率 $P(\text{“确实/现在/数据/很/多”})$ 可以通过每个词出现概率相乘的方式进行计算。

$$\begin{aligned} &P(\text{“确实/现在/数据/很/多”}) \\ &= P(\text{“确实”}) \cdot P(\text{“现在”}) \cdot P(\text{“数据”}) \cdot P(\text{“很”}) \cdot P(\text{“多”}) \\ &= 0.000001 \times 0.000022 \times 0.000009 \times 0.000010 \times 0.000078 \\ &= 1.5444 \times 10^{-25} \end{aligned} \tag{2.21}$$

这个假设也是自然语言处理中 1-gram 语言模型假设，即当前词的生成与任何历史都无关。当然，独立性假设并不能完美描述客观世界的问题，但是它大大化简了问题的复杂度。

最后再整体看一下分词系统的学习和使用过程。如图2.17所示，我们利用大量人工标注好的分词数据，通过统计学习方法获得一个统计模型 $P(\cdot)$ ，给定任意分词结果 $W = w_1w_2...w_m$ ，都能通过 $P(W) = P(w_1) \cdot P(w_2) \cdot \dots \cdot P(w_m)$ 计算这种切分的概率值。

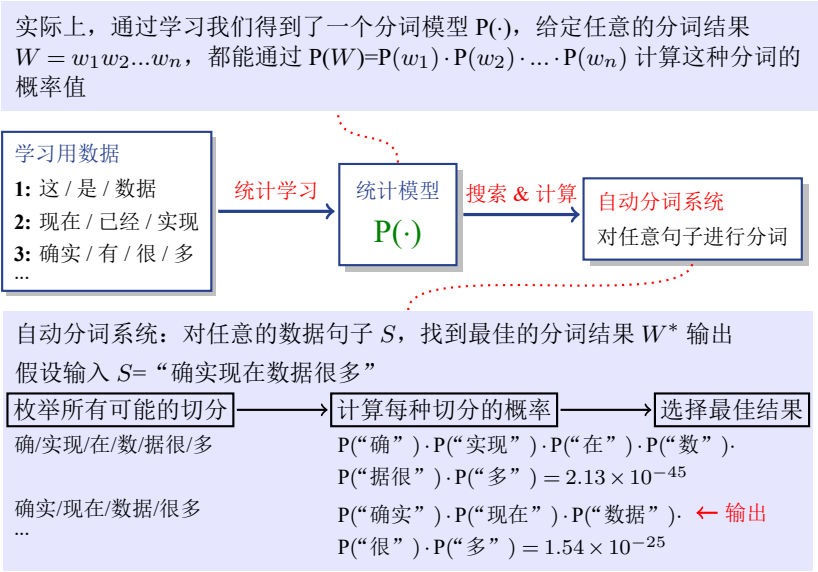


图 2.17: 基于 1-gram 语言模型的中文分词实例

经过充分训练的统计模型 $P(\cdot)$ 就是得到的分词模型。对于输入的新句子 S ，通过这个模型找到最佳的分词结果 W^* 输出。假设输入句子 S 是“确实现数据很多”，可以通过列举获得不同切分方式的概率，其中概率最高的切分方式，就是系统的目标输出。

这种分词方法也被称作基于 1-gram 语言模型的分词，或全概率分词，使用标注好的分词数据进行学习，获得分词模型。这种方法最大的优点是整个学习过程（模型训练过程）和推导过程（处理新句子进行切分的过程）都是全自动进行的。这种方法虽然简单，但是其效率很高，因此被广泛应用在工业界系统里。

当然，真正的分词系统还需要解决很多其他问题，比如使用动态规划等方法高效搜索最优解以及如何处理未见过的词等等，由于本节的重点是介绍中文分词的基础方法和统计建模思想，因此无法覆盖所有中文分词的技术内容，有兴趣的读者可以参考2.6节的相关文献做进一步深入研究。

2.4 n-gram 语言模型

在基于统计的汉语分词模型中，我们通过“大题小做”的技巧，利用独立性假设把整个句子的单词切分概率转化为每个单个词出现概率的乘积。这里，每个单词也被称作 1-gram（或 uni-gram），而 1-gram 概率的乘积实际上也是在度量词序列出现的可能性（记为 $P(w_1w_2...w_m)$ ）。这种计算整个单词序列概率 $P(w_1w_2...w_m)$ 的方法被称为统计语言模型。1-gram 语言模型是最简单的一种语言模型，它没有考虑任何的上下文。很自然的一个问题是：能否考虑上下文信息构建更强大的语言模型，进而得到更准确的分词结果。下面将进一步介绍更加通用的 n -gram 语言模型，它在机

器翻译及其他自然语言处理任务中有更加广泛的应用。

2.4.1 建模

语言模型 (Language Model) 的目的是描述文字序列出现的规律。这个对问题建模的过程被称作**语言建模** (Language Modeling)。如果使用统计建模的方式, 语言模型可以被定义为计算 $P(w_1w_2...w_m)$ 的问题, 也就是计算整个词序列 $w_1w_2...w_m$ 出现的可能性大小。具体定义如下,

定义 2.4.1 词汇表 V 上的语言模型是一个函数 $P(w_1w_2...w_m)$, 它表示 V^+ 上的一个概率分布。其中, 对于任何词串 $w_1w_2...w_m \in V^+$, 有 $P(w_1w_2...w_m) \geq 0$ 。而且对于所有的词串, 函数满足归一化条件 $\sum_{w_1w_2...w_m \in V^+} P(w_1w_2...w_m) = 1$ 。

直接求 $P(w_1w_2...w_m)$ 并不简单, 因为如果把整个词串 $w_1w_2...w_m$ 作为一个变量, 模型的参数量会非常大。 $w_1w_2...w_m$ 有 $|V|^m$ 种可能性, 这里 $|V|$ 表示词汇表大小。显然, 当 m 增大时, 模型的复杂度会急剧增加, 甚至都无法进行存储和计算。既然把 $w_1w_2...w_m$ 作为一个变量不好处理, 就可以考虑对这个序列的生成过程进行分解。使用链式法则, 很容易得到

$$P(w_1w_2...w_m) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2)...P(w_m|w_1w_2...w_{m-1}) \quad (2.22)$$

这样, $w_1w_2...w_m$ 的生成可以被看作是逐个生成每个单词的过程, 即首先生成 w_1 , 然后根据 w_1 再生成 w_2 , 然后根据 w_1w_2 再生成 w_3 , 以此类推, 直到根据所有前 $m-1$ 个词生成序列的最后一个单词 w_m 。这个模型把联合概率 $P(w_1w_2...w_m)$ 分解为多个条件概率的乘积, 虽然对生成序列的过程进行了分解, 但是模型的复杂度和以前是一样的, 比如, $P(w_m|w_1w_2...w_{m-1})$ 仍然不好计算。

换一个角度看, $P(w_m|w_1w_2...w_{m-1})$ 体现了一种基于“历史”的单词生成模型, 也就是把前面生成的所有单词作为“历史”, 并参考这个“历史”生成当前单词。但是这个“历史”的长度和整个序列长度是相关的, 也是一种长度变化的历史序列。为了化简问题, 一种简单的想法是使用定长历史, 比如, 每次只考虑前面 $n-1$ 个历史单词来生成当前单词, 这就是 n -gram 语言模型。这个模型的数学描述如下:

$$P(w_m|w_1w_2...w_{m-1}) \approx P(w_m|w_{m-n+1}...w_{m-1}) \quad (2.23)$$

这样, 整个序列 $w_1w_2...w_m$ 的生成概率可以被重新定义为:

链式法则	1-gram	2-gram	...	n -gram
$P(w_1w_2...w_m) =$	$P(w_1w_2...w_m) =$	$P(w_1w_2...w_m) =$...	$P(w_1w_2...w_m) =$
$P(w_1) \times$	$P(w_1) \times$	$P(w_1) \times$...	$P(w_1) \times$
$P(w_2 w_1) \times$	$P(w_2) \times$	$P(w_2 w_1) \times$...	$P(w_2 w_1) \times$
$P(w_3 w_1w_2) \times$	$P(w_3) \times$	$P(w_3 w_2) \times$...	$P(w_3 w_1w_2) \times$
$P(w_4 w_1w_2w_3) \times$	$P(w_4) \times$	$P(w_4 w_3) \times$...	$P(w_4 w_1w_2w_3) \times$
...
$P(w_m w_1...w_{m-1})$	$P(w_m)$	$P(w_m w_{m-1})$...	$P(w_m w_{m-n+1}...w_{m-1})$

可以看到，1-gram 语言模型只是 n -gram 语言模型的一种特殊形式。 n -gram 的优点在于，它所使用的历史信息是有限的，即 $n-1$ 个单词。这种性质也反映了经典的马尔可夫链的思想 [361][11]，有时也被称作马尔可夫假设或者马尔可夫属性。因此 n -gram 也可以被看作是变长序列上的一种马尔可夫模型，比如，2-gram 语言模型对应着 1 阶马尔可夫模型，3-gram 语言模型对应着 2 阶马尔可夫模型，以此类推。

那么，如何计算 $P(w_m|w_{m-n+1}...w_{m-1})$ 呢？有很多种选择，比如：

- **极大似然估计。**直接利用词序列在训练数据中出现的频度计算出 $P(w_m|w_{m-n+1}...w_{m-1})$

$$P(w_m|w_{m-n+1}...w_{m-1}) = \frac{\text{count}(w_{m-n+1}...w_m)}{\text{count}(w_{m-n+1}...w_{m-1})}$$

(2.24)

其中，count(·) 是在训练数据中统计频次的函数。

- **人工神经网络方法。**构建一个神经网络估计 $P(w_m|w_{m-n+1}...w_{m-1})$ 的值，比如，可以构建一个前馈神经网络来对 n -gram 进行建模。

极大似然估计方法和前面介绍的统计分词中的方法是一致的，它的核心是使用 n -gram 出现的频度进行参数估计，因此也是自然语言处理中一类经典的 n -gram 方法。基于人工神经网络的方法在近些年也非常受关注，它直接利用多层神经网络对问题的输入 ($w_{m-n+1}...w_{m-1}$) 和输出 $P(w_m|w_{m-n+1}...w_{m-1})$ 进行建模，而模型的参数通过网络中神经元之间连接的权重进行体现。严格意义上来说，基于人工神经网络的方法并不算基于 n -gram 的方法，或者说它并没有显性记录 n -gram 的生成概率，也不依赖 n -gram 的频度进行参数估计。为了保证内容的连贯性，本章将仍以传统 n -gram 语言模型为基础进行讨论，基于人工神经网络的方法将会在第五章和第六章进行详细介绍。

n -gram 语言模型的使用非常简单。可以像2.3.2节中一样，直接用它来对词序列出现的概率进行计算。比如，可以使用一个 2-gram 语言模型计算一个分词序列的概率：

$$\begin{aligned}
& P_{2\text{-gram}}(\text{“确实/现在/数据/很/多”}) \\
&= P(\text{“确实”}) \times P(\text{“现在”}|\text{“确实”}) \times P(\text{“数据”}|\text{“现在”}) \times \\
&\quad P(\text{“很”}|\text{“数据”}) \times P(\text{“多”}|\text{“很”})
\end{aligned} \tag{2.25}$$

以 n -gram 语言模型为代表的统计语言模型的应用非常广泛。除了分词，在文本生成、信息检索、摘要等自然语言处理任务中，语言模型都有举足轻重的地位。包括近些年非常受关注的预训练模型，本质上也是统计语言模型。这些技术都会在后续章节进行介绍。值得注意的是，统计语言模型为解决自然语言处理问题提供了一个非常好的建模思路，即：把整个序列生成的问题转化为逐个生成单词的问题。很快我们会看到，这种建模方式会被广泛地用于机器翻译建模，在统计机器翻译和神经机器翻译中都会有明显的体现。

2.4.2 未登录词和平滑算法

在式2.25所示的例子中，如果语料中从没有“确实”和“现在”两个词连续出现的情况，那么使用 2-gram 计算切分“确实/现在/数据/很/多”的概率时，会出现如下情况

$$\begin{aligned}
P(\text{“现在”}|\text{“确实”}) &= \frac{\text{count}(\text{“确实现在”})}{\text{count}(\text{“确实”})} \\
&= \frac{0}{\text{count}(\text{“确实”})} \\
&= 0
\end{aligned} \tag{2.26}$$

显然，这个结果是不能接受的。因为即使语料中没有“确实”和“现在”两个词连续出现，这种搭配也是客观存在的。这时简单的用极大似然估计得到概率却是 0，导致整个切分结果的概率为 0。更常见的问题是那些根本没有出现在词表中的词，称为**未登录词**（Out-of-Vocabulary Word, OOV Word），比如一些生僻词，可能模型训练阶段从来没有看到过，这时模型仍然会给出 0 概率。图2.18展示了一个真实语料库中词语出现频度的分布，可以看到绝大多数词都是低频词。

为了解决未登录词引起的零概率问题，常用的做法是对模型进行平滑处理，也就是给可能出现的情况一个非零的概率，使得模型不会对整个序列给出零概率。平滑可以用“劫富济贫”这一思想理解，在保证所有情况的概率和为 1 的前提下，使极低概率的部分可以从高概率的部分分配到一部分概率，从而达到平滑的目的。

语言模型使用的平滑算法有很多。在本节中，主要介绍三种平滑方法：加法平滑法、古德-图灵估计法和 Kneser-Ney 平滑。这些方法也可以被应用到其他任务的概率平滑操作中。

词汇出现总次数

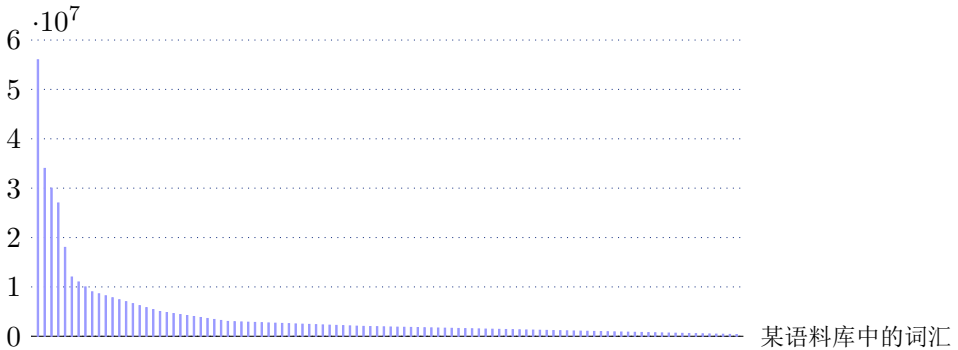


图 2.18: 词语频度的分布

加法平滑方法

加法平滑 (Additive Smoothing) 是一种简单的平滑技术。本小节首先介绍这一方法，希望通过它了解平滑算法的思想。通常情况下，系统研发者会利用采集到的语料库来模拟真实的全部语料库。当然，没有一个语料库能覆盖所有的语言现象。常见的一个问题是，使用的语料无法涵盖所有的词汇。因此，直接依据这样语料所获得的统计信息来获取语言模型就会产生偏差。假设依据某语料 C (从未出现“确实现在”二元语法)，评估一个已经分好词的句子 S = “确实/现在/物价/很/高” 的概率。当计算“确实/现在”的概率时， $P(S) = 0$ 。显然这个结果是不合理的。

加法平滑方法假设每个 n -gram 出现的次数比实际统计次数多 θ 次， $0 \leq \theta \leq 1$ 。这样，计算概率的时候分子部分不会为 0。重新计算 $P(\text{现在}|\text{确实})$ ，可以得到：

$$\begin{aligned} P(\text{“现在”}|\text{“确实”}) &= \frac{\theta + \text{count}(\text{“确实”}/\text{“现在”})}{\sum_w |V| (\theta + \text{count}(\text{“确实”}/w))} \\ &= \frac{\theta + \text{count}(\text{“确实”}/\text{“现在”})}{\theta |V| + \text{count}(\text{“确实”})} \end{aligned} \quad (2.27)$$

其中， V 表示所有词汇的词表， $|V|$ 为词表中单词的个数， w 为词典中的一个词。有时候，加法平滑方法会将 θ 取 1，这时称之为加一平滑或是拉普拉斯平滑。这种方法比较容易理解，也比较简单，因此也往往被用于对系统的快速原型中。

举一个例子。假设在一个英文文档中随机采样一些单词（词表大小 $|V| = 20$ ），各个单词出现的次数为：“look”: 4, “people”: 3, “am”: 2, “what”: 1, “want”: 1, “do”: 1。图2.19 给出了在平滑之前和平滑之后的概率分布。

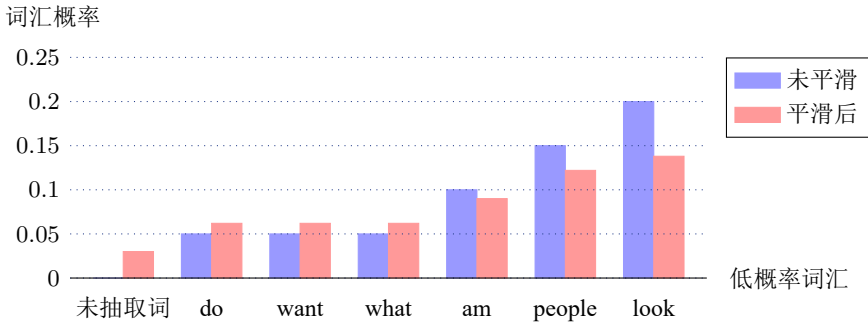


图 2.19: 无平滑和有平滑的概率分布

古德-图灵估计法

古德-图灵估计法 (Good-Turing Estimate) 是图灵 (Alan Turing) 和他的助手古德 (I.J. Good) 开发的, 作为他们在二战期间破解德国密码机 Enigma 所使用的方法的一部分, 在 1953 年古德将其发表, 这一方法也是很多平滑算法的核心, 其基本思路是: 把非零的 n 元语法单元的概率降低匀给一些低概率 n 元语法单元, 以减小最大似然估计与真实概率之间的偏离 [89][80]。

假定在语料库中出现 r 次的 n -gram 有 n_r 个, 特别的, 出现 0 次的 n -gram (即未登录词及词串) 出现的次数为 n_0 个。语料库中全部词语的个数为 N , 显然

$$N = \sum_{r=1}^{\infty} r n_r \quad (2.28)$$

这时, 出现 r 次的 n -gram 的相对频率为 r/N , 也就是不做平滑处理时的概率估计。为了解决零概率问题, 对于任何一个出现 r 次的 n -gram, 古德-图灵估计法利用出现 $r+1$ 次的 n -gram 统计量重新假设它出现 r^* 次, 这里

$$r^* = (r+1) \frac{n_{r+1}}{n_r} \quad (2.29)$$

基于这个公式, 就可以估计所有 0 次 n -gram 的频次 $n_0 r^* = (r+1) n_1 = n_1$ 。要把这个重新估计的统计数转化为概率, 需要进行归一化处理: 对于每个统计数为 r 的事件, 其概率为

$$P_r = \frac{r^*}{N} \quad (2.30)$$

其中

$$\begin{aligned}
 N &= \sum_{r=0}^{\infty} r^* n_r \\
 &= \sum_{r=0}^{\infty} (r+1) n_{r+1} \\
 &= \sum_{r=1}^{\infty} r n_r
 \end{aligned} \tag{2.31}$$

也就是说， N 仍然为这个整个样本分布最初的计数。样本中所有事件的概率之和为：

$$\begin{aligned}
 P(r > 0) &= \sum_{r>0} P_r \\
 &= 1 - \frac{n_1}{N} \\
 &< 1
 \end{aligned} \tag{2.32}$$

其中 n_1/N 就是分配给所有出现为 0 次事件的概率。古德-图灵方法最终通过出现 1 次的 n -gram 估计了出现为 0 次的事件概率，达到了平滑的效果。

这里使用一个例子来说明这个方法是如何对事件出现的可能性进行平滑的。仍然考虑在加法平滑法中统计单词的例子，根据古德-图灵方法进行修正如表2.2所示。

表 2.2: 单词出现频次及古德-图灵平滑结果

r	n_r	r^*	P_r
0	14	0.21	0.018
1	3	0.67	0.056
2	1	3	0.25
3	1	4	0.333
4	1	-	-

当 r 很大的时候经常会出现 $n_{r+1} = 0$ 的情况，而且这时 n_r 也会有噪音存在。通常，简单的古德-图灵方法可能无法很好的处理这种复杂的情况，不过古德-图灵方法仍然是其他一些平滑方法的基础。

Kneser-Ney 平滑方法

Kneser-Ney 平滑方法是由 R.Kneser 和 H.Ney 于 1995 年提出的用于计算 n 元语法概率分布的方法 [144][35]，并被广泛认为是最有效的平滑方法。这种平滑方法改进了 absolute discounting 中与高阶分布相结合的低阶分布的计算方法，使不同阶分布得到充分的利用。这种算法也综合利用了其他多种平滑算法的思想。

首先介绍一下 absolute discounting 平滑算法，公式如下所示：

$$P_{\text{AbsDiscount}}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w) \quad (2.33)$$

其中 d 表示被裁剪的值， λ 是一个正则化常数。可以看到第一项是经过减值调整过的 2-gram 的概率值，第二项则相当于一个带权重 λ 的 1-gram 的插值项。然而这种插值模型极易受到原始 1-gram 模型的干扰。

假设我们使用 2-gram 和 1-gram 的插值模型预测下面句子中下划线处的词

I cannot see without my reading _____

直觉上应该会猜测这个地方的词应该是“glasses”，但是在训练语料库中“Francisco”出现的频率非常高。如果在预测时仍然使用的是标准的 1-gram 模型，那么系统会高概率选择“Francisco”填入下划线处，这个结果明显是不合理的。当使用的是混合的插值模型时，如果“reading Francisco”这种二元语法并没有出现在语料中，就会导致 1-gram 对结果的影响变大，使得仍然会做出与标准 1-gram 模型相同的结果，犯下相同的错误。

观察语料中的 2-gram 发现，“Francisco”的前一个词仅可能是“San”，不会出现“reading”。这个分析提醒了我们，考虑前一个词的影响是有帮助的，比如仅在前一个词时“San”时，才给“Francisco”赋予一个较高的概率值。基于这种想法，改进原有的 1-gram 模型，创造一个新的 1-gram 模型 $P_{\text{continuation}}$ ，简写为 P_{cont} 。这个模型可以通过考虑前一个词的影响评估当前词作为第二个词出现的可能性。

为了评估 P_{cont} ，统计使用当前词作为第二个词所出现二元语法的种类，二元语法种类越多，这个词作为第二个词出现的可能性越高，呈正比：

$$P_{\text{cont}}(w_i) \propto |w_{i-1} : c(w_{i-1}w_i) > 0| \quad (2.34)$$

通过全部的二元语法的种类做归一化可得到评估的公式

$$P_{\text{cont}}(w_i) = \frac{|\{w_{i-1} : c(w_{i-1}w_i) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}w_j) > 0\}|} \quad (2.35)$$

基于分母的变化还有另一种形式

$$P_{\text{cont}}(w_i) = \frac{|\{w_{i-1} : c(w_{i-1}w_i) > 0\}|}{\sum_{w'} |\{w'_{i-1} : c(w'_{i-1}w'_i) > 0\}|} \quad (2.36)$$

结合基础的 absolute discounting 计算公式，从而得到了 Kneser-Ney 平滑方法的公式

$$P_{\text{KN}}(w_i|w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_{\text{cont}}(w_i) \quad (2.37)$$

其中

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} |\{w : c(w_{i-1}, w) > 0\}| \quad (2.38)$$

这里 $\max(\cdot)$ 保证了分子部分为不小 0 的数，原始 1-gram 更新成 P_{cont} 概率分布， λ 是正则化项。

为了更具普适性，不仅局限为 2-gram 和 1-gram 的插值模型，利用递归的方式可以得到更通用的 Kneser-Ney 平滑公式

$$P_{\text{KN}}(w_i | w_{i-n+1} \dots w_{i-1}) = \frac{\max(c_{\text{KN}}(w_{i-n+1} \dots w_{i-1}) - d, 0)}{c_{\text{KN}}(w_{i-n+1} \dots w_{i-1})} + \lambda(w_{i-n+1} \dots w_{i-1}) P_{\text{KN}}(w_i | w_{i-n+2} \dots w_{i-1}) \quad (2.39)$$

$$\lambda(w_{i-1}) = \frac{d}{c_{\text{KN}}(w_{i-n+1}^{i-1})} |\{w : c_{\text{KN}}(w_{i-n+1} \dots w_{i-1} w) > 0\}| \quad (2.40)$$

$$c_{\text{KN}}(\cdot) = \begin{cases} \text{count}(\cdot) & \text{for highest order} \\ \text{catcount}(\cdot) & \text{for lower order} \end{cases} \quad (2.41)$$

其中 $\text{catcount}(\cdot)$ 表示的是基于某个单个词作为第 n 个词的 n -gram 的种类数目。

Kneser-Ney 平滑是很多语言模型工具的基础 [298][106][270]。还有很多以此为基础衍生出来的算法，感兴趣的读者可以通过参考文献自行了解 [134][213][35]。

2.5 句法分析（短语结构分析）

通过前面两节的内容，读者已经了解什么叫做“词”、如何对分词问题进行统计建模。同时也了解了如何对词序列的生成进行概率描述。无论是分词还是语言模型都是句子浅层词串信息的一种表示。对于一个自然语言句子来说，它更深层次的结构信息可以通过句法信息来描述，而句法信息也是机器翻译和自然语言处理其他任务中常用的知识之一。

2.5.1 句子的句法树表示

句法（Syntax）是研究句子的每个组成部分和它们之间的组合方式。一般来说，句法和语言是相关的，比如，英文是主谓宾结构，而日语是主宾谓结构。因此不同的语言也会有不同的句法描述方式。自然语言处理领域最常用的两种句法分析形式是**短语结构分析**（Phrase Structure Parsing）和**依存分析**（Dependency Parsing）。图2.20展示了这两种的句法表示形式的实例。其中，左侧是短语结构树。它描述的是短语的结构功能，比如“吃”是动词（记为 VV），“鱼”是名词（记为 NN），“吃 鱼”组成动

词短语，这个短语再与“喜欢”这一动词组成新的动词短语。短语结构树的每个子树都是一个句法功能单元，比如，子树 VP(VV(吃) NN(鱼)) 就表示了“吃 鱼”这个动词短语的结构，其中子树根节点 VP 是句法功能标记。短语结构树利用嵌套的方式描述了语言学的功能。短语结构树中，每个词都有词性 (或词类)，不同的词或者短语可以组成名动结构、动宾结构等语言学短语结构。短语结构分析一般也被称为**成分分析**(Constituency Parsing) 或**完全分析** (Full Parsing) 。

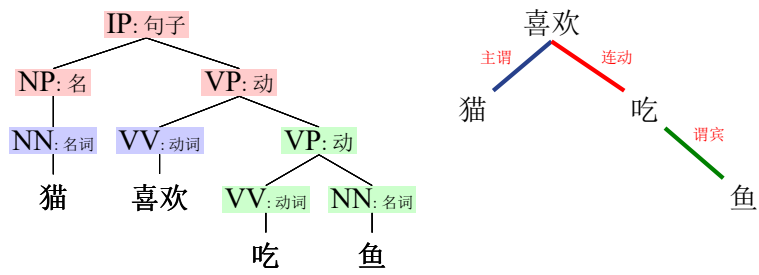


图 2.20: 短语结构树 (左) 和依存树 (右)

图2.20右侧展示的是另一种句法结构，被称作依存句法树。依存句法树表示了句子中单词和单词之间的依存关系。比如，从这个例子可以了解，“猫”依赖“喜欢”，“吃”依赖“喜欢”，“鱼”依赖“吃”。

短语结构树和依存句法树的结构和功能有很大不同。短语结构树的叶子节点是单词，中间节点是词性或者短语句法标记。在短语结构分析中，通常把单词称作**终结符** (Terminal)，把词性称为**预终结符** (Pre-terminal)，而把其他句法标记称为**非终结符** (Non-terminal)。依存句法树没有预终结符和非终结符，所有的节点都是句子中的单词，通过不同节点间的连线表示句子中各个单词之间的依存关系。每个依存关系实际上都是有方向的，头和尾分别指向“接受”和“发出”依存关系的词。依存关系也可以进行分类，图2.20中我们对每个依存关系的类型都进行了标记，这也被称作是有标记的依存分析。如果不生成这些标记，这样的句法分析被称作无标记的依存分析。

虽然短语结构树和依存树的句法表现形式有很大不同，但是它们在某些条件下能相互转化。比如，可以使用启发性规则将短语结构树自动转化为依存树。从应用的角度，依存分析由于形式更加简单，而且直接建模词语之间的依赖，因此在自然语言处理领域中受到很多关注。在机器翻译中，无论是哪种句法树结构，都已经被证明会对机器翻译系统产生帮助。特别是短语结构树，在机器翻译中的应用历史更长，研究更为深入，因此本节将会以短语结构分析为例介绍句法分析的相关概念。

而句法分析到底是什么呢？简单的理解，句法分析就是在小学语文课程中学习的句子成分的分析，以及对句子中各个成分内部、外部关系的判断。更规范一些的定义，可以参照百度百科维基百科的句法分析的解释。

定义 2.5.1 句法分析

句法分析 (Parsing) 就是指对句子中的词语语法功能进行分析。

——《百度百科》

在自然语言或者计算机语言中, 句法分析是利用形式化的文法规则对一个符号串进行分析的过程。

——《维基百科(译文)》

上面的定义中, 句法分析包含三个重要的概念:

- 形式化的文法: 描述语言结构的定义, 由文法规则组成。
- 符号串: 在本节中, 符号串就是指词串, 由前面提到的分词系统生成。
- 分析: 使用形式文法对符号串进行分析的具体方法, 在这里指实现分析的计算机算法。

以上三点是实现一个句法分析器的要素。本节的后半部分会对相关的概念和技术方法进行介绍。

2.5.2 上下文无关文法

句法树是对句子的一种抽象。这种树形结构表达了一种对句子结构的归纳过程, 比如, 从树的叶子开始, 把每一个树节点看作一次抽象, 最终形成一个根节点。那这个过程如何用计算机来实现呢? 这就需要使用到形式文法。

形式文法是分析自然语言的一种重要工具。根据乔姆斯基的定义 [208], 形式文法分为四种类型: 无限制文法 (0 型文法)、上下文相关文法 (1 型文法)、上下文无关文法 (2 型文法) 和正规文法 (3 型文法)。不同类型的文法有不同的应用, 比如, 正规文法可以用来描述有限状态自动机, 因此也会被使用在语言模型等系统中。对于短语结构分析问题, 常用的是**上下文无关文法** (Context-Free Grammar)。上下文无关文法的具体形式如下:

定义 2.5.2 上下文无关文法

一个上下文无关文法可以被视为一个系统 $G = \langle N, \Sigma, R, S \rangle$, 其中

- N 为一个非终结符集合
- Σ 为一个终结符集合
- R 为一个规则 (产生式) 集合, 每条规则 $r \in R$ 的形式为 $X \rightarrow Y_1 Y_2 \dots Y_n$, 其中 $X \in N, Y_i \in N \cup \Sigma$
- S 为一个起始符号集合且 $S \subseteq N$

举例说明, 假设有上下文无关文法 $G = \langle N, \Sigma, R, S \rangle$, 可以用它描述一个简单

中文句法结构。其中非终结符集合为不同的中文句法标记

$$N = \{NN, VV, NP, VP, IP\}$$

这里，NN 代表名词，VV 代表动词，NP 代表名词短语，VP 代表动词短语，IP 代表单句。进一步，把终结符集合定义为

$$\Sigma = \{\text{猫, 喜欢, 吃, 鱼}\}$$

再定义起始符集合为

$$S = \{IP\}$$

最后，文法的规则集定义图2.21所示（其中 r_i 为规则的编号）

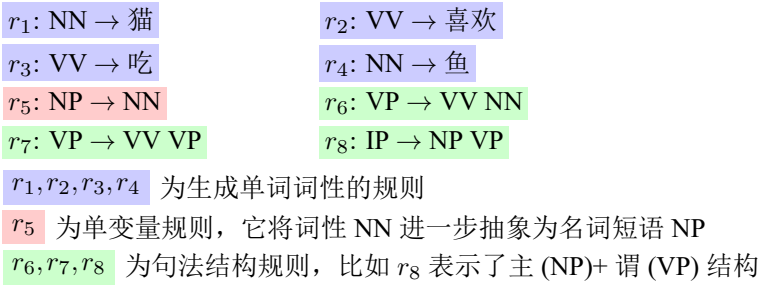


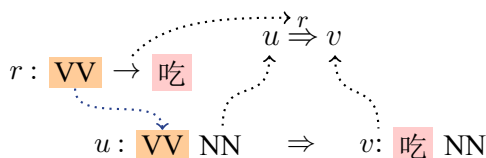
图 2.21: 一个示例文法的规则集

上面这个文法蕴含了不同“层次”的句法信息。比如，规则 r_1 、 r_2 、 r_3 和 r_4 表达了词性对单词的抽象；规则 r_6 、 r_7 和 r_8 是表达了短语结构的抽象，其中，规则 r_8 描述了汉语中名词短语 (主语)+ 动词短语 (谓语) 的结构。在实际应用中，像 r_8 这样的规则可以覆盖很大的片段（试想一下一个包含 50 个词的主谓结构的句子，可以使用 r_8 进行描述）。

上下文无关文法的规则是一种**产生式规则**（Production Rule），形如 $\alpha \rightarrow \beta$ ，它表示把规则左端的非终结符 α 替换为规则右端的符号序列 β 。通常， α 被称作规则的左部（Left-hand Side）， β 被称作规则的右部（Right-hand Side）。使用右部 β 替换左部 α 的过程也被称作规则的使用，而这个过程的逆过程称为规约。规则的使用可以如下定义：

定义 2.5.3 上下文无关文法规则的使用

一个符号序列 u 可以通过使用规则 r 替换其中的某个非终结符，并得到符号序列 v ，于是 v 是在 u 上使用 r 的结果，记为 $u \xrightarrow{r} v$ ：



给定起始非终结符，可以不断地使用规则，最终生成一个终结字符串，这个过程也被称为**推导**（Derivation）。形式化的定义为：

定义 2.5.4 推导

给定一个文法 $G = \langle N, \Sigma, R, S \rangle$ ，对于一个字符串序列 s_0, s_1, \dots, s_n 和规则序列 r_1, r_2, \dots, r_n ，满足

$$s_0 \xRightarrow{r_1} s_1 \xRightarrow{r_2} s_2 \xRightarrow{r_3} \dots \xRightarrow{r_n} s_n$$

且

- $\forall i \in [0, n], s_i \in (N \cup \Sigma)^*$ $\triangleleft s_i$ 为合法的字符串
- $\forall j \in [1, n], r_j \in R$ $\triangleleft r_j$ 为 G 的规则
- $s_0 \in S$ $\triangleleft s_0$ 为起始非终结符
- $s_n \in \Sigma^*$ $\triangleleft s_n$ 为终结符序列

则 $s_0 \xRightarrow{r_1} s_1 \xRightarrow{r_2} s_2 \xRightarrow{r_3} \dots \xRightarrow{r_n} s_n$ 为一个推导

比如，使用前面的示例文法，可以对“猫喜欢吃鱼”进行分析，并形成句法分析树（图2.22）。从起始非终结符 IP 开始，使用唯一拥有 IP 作为左部的规则 r_8 推导出 NP 和 VP，之后依次使用规则 r_5 、 r_1 、 r_7 、 r_2 、 r_6 、 r_3 、 r_4 ，得到了完整的句法树。

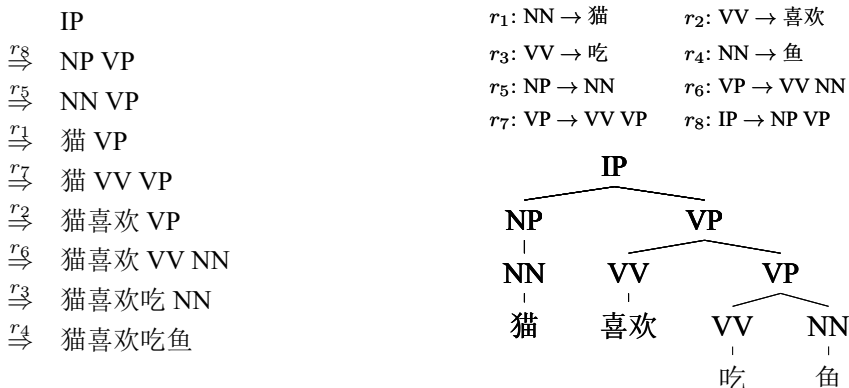


图 2.22: 上下文无关文法推导实例

通常，可以把推导简记为 $d = r_1 \circ r_2 \circ \dots \circ r_n$ ，其中 \circ 表示规则的组合。显然， d

也对应了树形结构，也就是句法分析结果。从这个角度看，推导就是描述句法分析树的一种方式。此外，规则的推导也把规则的使用过程与生成的字符串对应起来。一个推导所生成的字符串，也被称作文法所产生的一个**句子**（Sentence）。而一个文法所能生成的所有句子的集合是这个文法所对应的**语言**（Language）。

但是，句子和规则的推导并不是一一对应的。同一个句子，往往有很多推导的方式，这种现象被称为**歧义**（Ambiguity）。甚至同一棵句法树，也可以对应不同的推导。图2.23 给出同一棵句法树所对应的两种不同的规则推导。

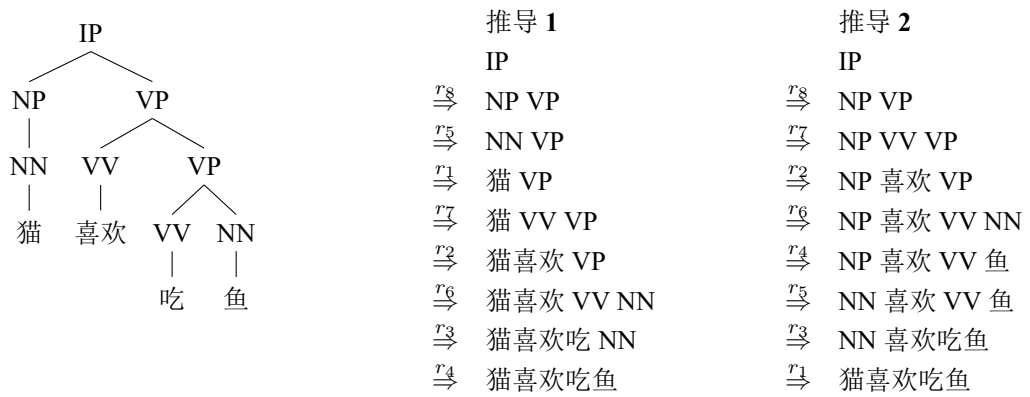


图 2.23: 同一棵句法树对应的不同规则推导

显然，规则顺序的不同会导致句法树的推导这一确定的过程变得不确定。因此，需要进行**消歧**（Disambiguation）。这里，可以使用启发式方法：要求规则使用都服从最左优先原则，这样得到的推导被称为**最左优先推导**（Left-most Derivation）。图2.23中的推导 1 就是符合最左优先原则的推导。

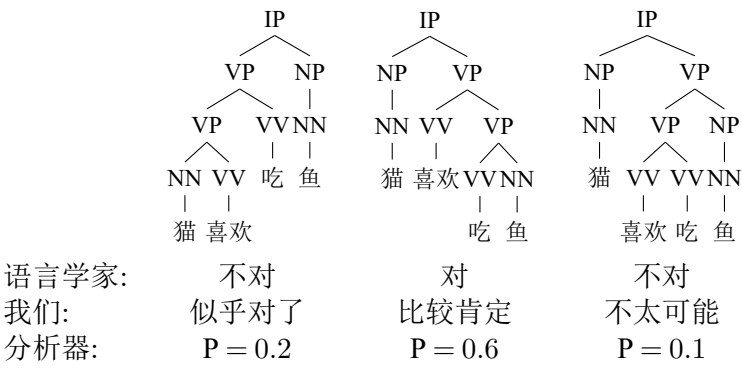


图 2.24: 如何选择最佳的句法分析结果 - 专家、普通人和句法分析器的视角

这样，对于一个上下文无关文法，每一棵句法树都有唯一的最左推导与之对应。于是，句法分析可以被描述为：对于一个句子找到能够生成它的最佳推导，这个推导所对应的句法树就是这个句子的句法分析结果。

不过问题又回来了，怎样才能知道什么样的推导或者句法树是“最佳”的呢？如图2.24所示，对于语言学专家，他们可以很确定的分辨出哪些句法树是正确的，哪些句法树是错误。甚至普通人也可以通过一些课本中学到的知识产生一些模糊的判断。而计算机如何进行判别呢？沿着前面介绍的统计建模的思想，计算机可以得出不同句法树出现的概率，进而选择概率最高的句法树作为输出，而这正是统计句法分析所做的事情。

在统计句法分析中，需要对每个推导进行统计建模，于是定义一个模型 $P(\cdot)$ ，对于任意的推导 d ，都可以用 $P(d)$ 计算出推导 d 的概率。这样，给定一个输入句子，我们可以对所有可能的推导用 $P(d)$ 计算其概率值，并选择概率最大的结果作为句法分析的结果输出（图2.25）。

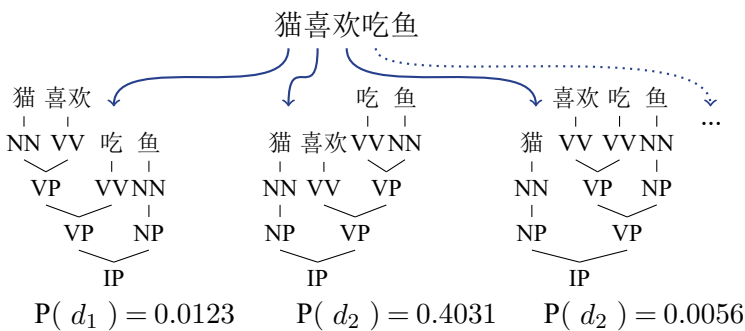


图 2.25: 不同推导（句法树）对应的概率值

2.5.3 规则和推导的概率

对句法树进行概率化，首先要对使用的规则进行概率化。为了达到这个目的，可以使用**概率上下文无关文法**（Probabilistic Context-Free Grammar），它是上下文无关文法的一种扩展。

定义 2.5.5 概率上下文无关文法

- 一个概率上下文无关文法可以被视为一个系统 $G = \langle N, \Sigma, R, S \rangle$ ，其中
- N 为一个非终结符集合
 - Σ 为一个终结符集合
 - R 为一个规则 (产生式) 集合，每条规则 $r \in R$ 的形式为 $p: X \rightarrow Y_1 Y_2 \dots Y_n$ ，其中 $X \in N, Y_i \in N \cup \Sigma$ ，每个 r 都对应一个概率 p ，表示其生成的可能性。
 - S 为一个起始符号集合且 $S \subseteq N$

概率上下文无关文法与传统上下文无关文法的区别在于，每条规则都会有一个

概率，描述规则生成的可能性。具体来说，规则 $P(\alpha \rightarrow \beta)$ 的概率可以被定义为：

$$P(\alpha \rightarrow \beta) = P(\beta|\alpha) \quad (2.42)$$

即，在给定规则左部的情况下生成规则右部的可能性。进一步，在上下文无关文法中，每条规则之间的使用都是相互独立的³。因此可以把 $P(d)$ 分解为规则概率的乘积：

$$\begin{aligned} P(d) &= P(r_1 \cdot r_2 \cdot \dots \cdot r_n) \\ &= P(r_1) \cdot P(r_2) \cdots P(r_n) \end{aligned} \quad (2.43)$$

这个模型可以很好的解释词串的生成过程。比如，对于规则集

$$\begin{aligned} r_3: & \quad VV \rightarrow \text{吃} \\ r_4: & \quad NN \rightarrow \text{鱼} \\ r_6: & \quad VP \rightarrow VV \, NN \end{aligned}$$

可以得到 $d_1 = r_3 \cdot r_4 \cdot r_6$ 的概率为

$$\begin{aligned} P(d_1) &= P(r_3) \cdot P(r_4) \cdot P(r_6) \\ &= P(\text{“}VV \rightarrow \text{吃}\text{”}) \cdot P(\text{“}NN \rightarrow \text{鱼}\text{”}) \cdot P(\text{“}VP \rightarrow VV \, NN\text{”}) \end{aligned} \quad (2.44)$$

这也对应了词串“吃 鱼”的生成过程。首先，从起始非终结符 VP 开始，使用规则 r_6 生成两个非终结符 VV 和 NN ；进一步，分别使用规则 r_3 和 r_4 从 VV 和 NN 进一步生成单词“吃”和“鱼”。整个过程的概率等于三条规则概率的乘积。

新的问题又来了，如何得到规则的概率呢？这里仍然可以从数据中学习文法规则的概率。假设有人工标注的数据，它包括很多人工标注句法树的句法，称之为**树库** (Treebank)。然后，对于规则 $r: \alpha \rightarrow \beta$ 可以使用极大似然估计：

$$P(r) = \frac{\text{规则 } r \text{ 在树库中出现的次数}}{\alpha \text{ 在树库中出现的次数}} \quad (2.45)$$

图2.26展示了通过这种方法计算规则概率的过程。与词法分析类似，可以统计树库中规则左部和右部同时出现的次数，除以规则左部出现的全部次数，所得的结果就是所求规则的概率。这种方法也是典型的相对频度估计。但是如果规则左部和右部同时出现的次数为 0 时是否代表这个规则概率是 0 呢？遇到这种情况，可以使用平滑方法对概率进行平滑处理，具体思路可参考2.4.2节内容。

³如果是上下文有关文法，规则会形如 $a\alpha b \rightarrow a\beta b$ ，这时 $\alpha \rightarrow \beta$ 的过程会依赖前后上下文 a 和 b

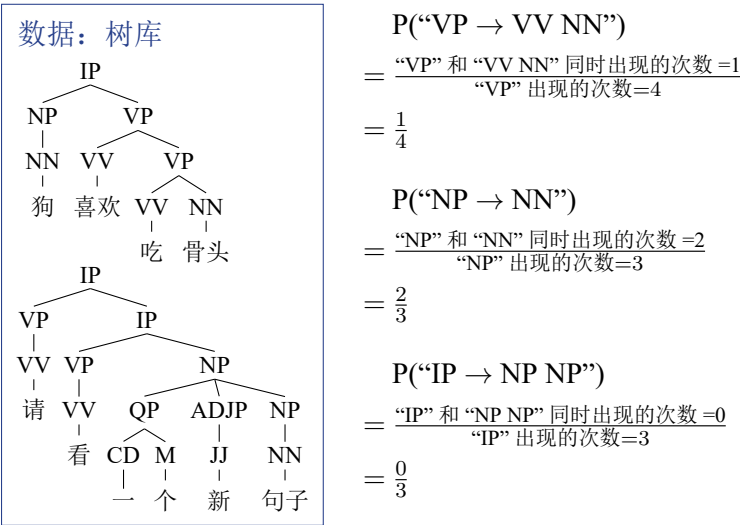


图 2.26: 上下文无关文法规则概率估计

图2.27展示了基于统计的句法分析的流程。首先，通过树库上的统计，获得各个规则的概率，这样就得到了一个上下文无关句法分析模型 $P(\cdot)$ 。对于任意句法分析结果 $d = r_1 \circ r_2 \circ \dots \circ r_n$ ，都能通过如下公式计算其概率值：

$$P(d) = \prod_{i=1}^n P(r_i) \tag{2.46}$$

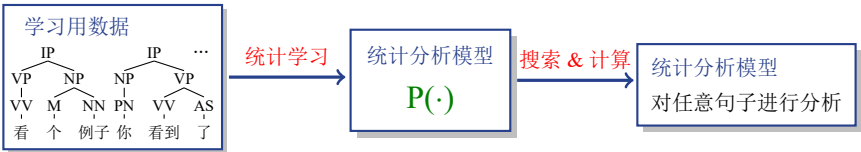


图 2.27: 统计句法分析的流程

在获取统计分析模型后，就可以使用模型对任意句子进行分析，计算每个句法分析树的概率，并输出概率最高的树作为句法分析的结果。

2.6 小结及深入阅读

本章重点介绍了如何对自然语言处理问题进行统计建模，并从数据中自动学习统计模型的参数，最终使用学习到的模型对新的问题进行处理。之后，本章将这种思想应用到三个自然语言处理任务中，包括：中文分词、语言建模、句法分析，它们也和机器翻译有着紧密的联系。通过系统化的建模，可以发现：经过适当的假设和化简，统计模型可以很好的描述复杂的自然语言处理问题。相关概念和方法也会在后续章节的内容中被广泛使用。

由于本章重点介绍如何用统计的思想对自然语言处理任务进行建模，因此并没有对具体的问题展开深入讨论。有几方面内容，读者可以继续关注：

- 在建模方面，本章介绍的三个任务均采用的是基于人工先验知识进行模型设计的思路。也就是，问题所表达的现象被“一步一步”生成出来。这是一种典型的生成式建模思想，它把要解决的问题看作一些观测结果的隐含变量（比如，句子是观测结果，分词结果是隐含在背后的变量），之后通过对隐含变量生成观测结果的过程进行建模，以达到对问题进行数学描述的目的。这类模型一般需要依赖一些独立性假设，假设的合理性对最终的性能有较大影响。相对于**生成模型**（Generative Model），另一类方法是**判别模型**（Discriminative Model），它直接描述了从隐含变量生成观测结果的过程，这样对问题的建模更加直接，同时这类模型可以更加灵活的引入不同的特征。判别模型在自然语言处理中也有广泛应用 [253][214]。在本书的第四章也会使用到判别式模型。
- 从现在自然语言处理的前沿看，基于端到端学习的深度学习方法在很多任务中都取得了领先的性能。但是，本章并没有涉及深度学习及相关方法，这是由于笔者认为：对问题的建模是自然语言处理的基础，对问题的本质刻画并不会因为方法的改变而改变。因此，本章的内容没有太多地陷入到更加复杂的模型和算法设计中，相反，我们希望关注对基本问题的理解和描述。不过，一些前沿方法仍可以作为参考，包括：基于条件随机场和双向长短时记忆模型的序列标注模型 [160][120][191]、神经语言模型 [14][204]、神经句法分析模型 [32][354]。
- 此外，本章并没有对模型的推断方法进行深入介绍。比如，对于一个句子如何有效的找到概率最大的分词结果？显然，简单枚举是不可行的。对于这类问题比较简单的解决方法是使用动态规划 [47]。如果使用动态规划的条件不满足，可以考虑使用更加复杂的搜索策略，并配合一定剪枝方法。实际上，无论是 n -gram 语言模型还是简单的上下文无关文法都有高效的推断方法。比如， n -gram 语言模型可以被视为概率有限状态自动机，因此可以直接使用成熟的自动机工具。对于更复杂的句法分析问题，可以考虑使用移进-规约方法来解决推断问题 [1]。

统计机器翻译

3	基于词的机器翻译模型	87
3.1	词在翻译中的作用	
3.2	一个简单的翻译系统	
3.3	基于词的翻译建模	
3.4	IBM 模型 1-2	
3.5	IBM 模型 3-5 及隐马尔可夫模型	
3.6	问题分析	
3.7	小结及深入阅读	
4	基于短语和句法的机器翻译模型	133
4.1	翻译中的结构信息	
4.2	基于短语的翻译模型	
4.3	基于层次短语的模型	
4.4	基于语言学句法的模型	
4.5	小结及深入阅读	



3. 基于词的机器翻译模型

使用概率化的方法对翻译问题进行建模是机器翻译发展中的重要里程碑。这种思想也影响了当今的统计机器翻译和神经机器翻译方法。虽然技术不断发展，传统的统计模型已经不再“新鲜”，但它对于今天机器翻译的研究仍然有着重要的启示作用。在了解前沿、展望未来的同时，我们更要冷静的思考前人给我们带来了什么。基于此，本章将介绍统计机器翻译的开山之作——IBM 模型，它提出了使用统计模型进行翻译的思想，并在建模中引入了单词对齐这一重要概念。IBM 模型由 Peter F. Brown 等人于上世纪九十年代初提出 [25]。客观的说，这项工作的视野和对问题的理解，已经超过当时很多人所能看到的东西，其衍生出来的一系列方法和新的问题还被后人花费将近 10 年的时间来进行研究与讨论。时至今日，IBM 模型中的一些思想仍然影响着很多研究工作。

3.1 词在翻译中的作用

在机器翻译中，我们希望得到一个源语言到目标语言的翻译。对于人类来说这个问题很简单，但是让计算机做这样的工作却很困难，因为我们需要把翻译“描述”成计算机可以计算的形式。这里面临的第一个问题是：如何对翻译进行建模？从计算机的角度来看，这就需要把自然语言的翻译问题转换为计算机可计算的问题。

那么，基于单词的统计机器翻译模型又是如何描述翻译问题的呢？Peter F. Brown 等人提出了一个观点 [25]：在翻译一个句子时，可以把其中的每个单词翻译成对应的目标语言单词，然后调整这些目标语言单词的顺序，最后得到整个句子的翻译结

果，而这个过程可以用统计模型来描述。尽管在人看来使用两个语言单词之间的对应进行翻译是很自然的事，但是对于计算机来说可是向前迈出了一大步。

先来看一个例子。图 3.1 展示了一个汉语翻译到英语的例子。首先，可以把源语句的单词“我”、“对”、“你”、“感到”和“满意”分别翻译为“I”、“with”、“you”、“am”和“satisfied”，然后调整单词的顺序，比如，“am”放在译文的第 2 个位置，“you”应该放在最后的位置等等，最后得到译文“I am satisfied with you”。

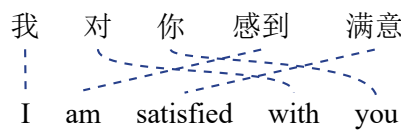


图 3.1: 汉语到英语翻译实例及两种语言单词之间的对应关系

上面的例子反映了人在做翻译时所使用的一些知识：首先，两种语言单词的顺序可能不一致，而且译文需要符合目标语的习惯，这也就是常说的翻译的**流畅度**问题 (Fluency)；其次，源语言单词需要准确的被翻译出来¹，也就是常说的翻译的**准确性**问题 (Accuracy) 和 **充分性**问题 (Adequacy)。为了达到以上目的，传统观点认为翻译过程需要包含三个步骤（图 3.2）

- **分析**：将源语言句子切分或者表示为能够处理的最小单元。在基于词的翻译模型中，最小的处理单元就是单词，因此在这里也可以简单地将分析理解为分词²。
- **转换**：把源语言句子中的每个单词翻译成目标语言单词。
- **生成**：基于转换的结果，将目标语译文变成通顺且合乎语法的句子。

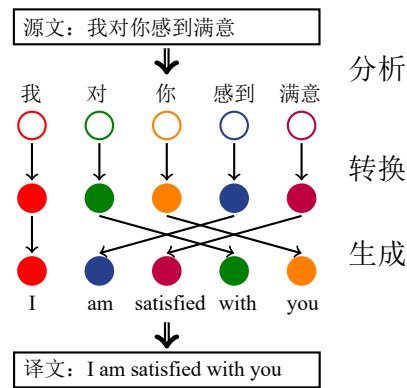


图 3.2: 翻译过程中的分析、转换和生成

¹当然，对于一些意译的情况或者虚词并不需要翻译。
²在后续章节中会看到，分析也包括对句子深层次结构的生成，但是这里为了突出基于单词的概念，因此把问题简化为最简单的情况。

对于今天的自然语言处理研究，“分析、转换和生成”依然是一个非常深刻的观点。包括机器翻译在内的很多自然语言处理问题都可以用这个过程来解释。比如，对于现在比较前沿的神经机器翻译方法，从大的框架来说，依然在做分析（编码器）、转换（编码-解码注意力）和生成（解码器），只不过这些过程隐含在神经网络的设计中。当然，这里并不会对“分析、转换和生成”的架构展开过多的讨论，随着后面技术内容讨论的深入，这个观念会有进一步体现。

3.2 一个简单的翻译系统

本节首先对比人工翻译和机器翻译过程的异同点，从中归纳出构建机器翻译系统的两个主要步骤：训练和解码。之后，会从学习翻译知识和运用翻译知识两个方面描述如何构建一个简单的机器翻译系统。

3.2.1 如何进行翻译？

人工翻译流程

当人翻译一个句子时，首先会快速地分析出句子的（单词）构成，然后根据以往的知识，得到每个词可能的翻译，最后利用对目标语的理解拼出来一个译文。尽管这个过程并不是严格来自心理学或者脑科学的相关结论，但至少可以帮助我们理解人在翻译时的思考方式。

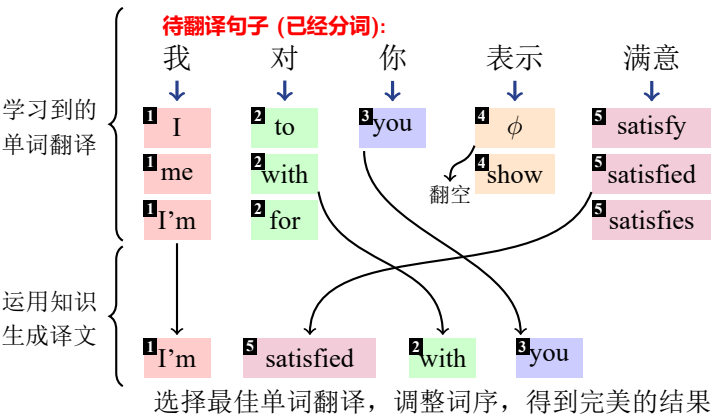


图 3.3: 人工翻译的过程

图3.3展示了人在翻译“我对你表示满意”时可能会思考的内容。具体来说，有如下两方面内容。

- **翻译知识的学习:** 对于输入的源语言句子，首先需要知道每个单词可能的翻译有什么，这些翻译被称为**翻译候选**（Translation Candidate）。比如，汉语单词“对”可能的译文有“to”、“with”和“for”等。对于人来说，可以通过阅读、背诵、做

题或者老师教等途径获得翻译知识，这些知识就包含了源语言与目标语言单词之间的对应关系。通常，也把这个过程称之为学习过程。

- **运用知识生成译文：**当翻译一个从未见过的句子时，可以运用学习到的翻译知识，得到新的句子中每个单词的译文，并处理常见的单词搭配、主谓一致等问题，比如，我们知道“satisfied”后面常常使用介词“with”构成搭配，基于这些知识可以快速生成译文。

当然，每个人进行翻译时所使用的方法和技巧都不相同，所谓人工翻译也没有固定的流程。但是，可以确定的是，人在进行翻译时也需要“学习”和“运用”翻译知识。对翻译知识“学习”和“运用”的好与坏，直接决定了人工翻译结果的质量。

机器翻译流程

人进行翻译的过程比较容易理解，那计算机是如何完成翻译的呢？虽然人工智能这个概念显得很神奇，但是计算机远没有人那么智能，有时甚至还很“笨”。一方面，它没有能力像人一样，在教室里和老师一起学习语言知识；另一方面，即使能列举出每个单词的候选译文，但是还是不知道这些译文怎么拼装成句的，甚至不知道哪些译文是对的。为了更加直观地理解机器在翻译时要解决的挑战，可以将问题归纳如下：

- 如何让计算机获得每个单词的译文，然后将这些单词的译文拼装成句？
- 如果可以形成整句的译文，如何让计算机知道不同译文的好坏？

对于第一个问题，可以给计算机一个翻译词典，这样计算机可以发挥计算方面的优势，尽可能多的把翻译结果拼装出来。比如，可以把每个翻译结果看作是对单词翻译的拼装，这可以被形象的比做贯穿多个单词的一条路径，计算机所做的就是尽可能多的生成这样的路径。图3.4中蓝色和红色的折线就分别表示了两条不同的译文选择路径，区别在于“满意”和“对”的翻译候选是不一样的，蓝色折线选择的是“satisfy”和“to”，而红色折线是“satisfied”和“with”。换句话说，不同的译文对应不同的路径（即使词序不同也会对应不同的路径）。

对于第二个问题，尽管机器能够找到很多译文选择路径，但它并不知道哪些路径是好的。说的再直白一些，简单的枚举路径实际上就是一个体力活，没有什么智能。因此计算机还需要再聪明一些，运用它的能够“掌握”的知识判断翻译结果的好与坏。这一步是最具挑战的，当然也有很多思路。在统计机器翻译中，这个问题被定义为：设计一种统计模型，它可以给每个译文一个可能性，而这个可能性越高表明译文越接近人工翻译。如图3.4所示，每个单词翻译候选的右侧黑色框里的数字就是单词的翻译概率，使用这些单词的翻译概率，可以得到整句译文的概率（用符号 P 表示）。这样，就用概率化的模型描述了每个翻译候选的可能性。基于这些翻译候选的可能性，机器翻译系统可以对所有的翻译路径进行打分，比如，图3.4中第一条路径的分数为 0.042，第二条是 0.006，以此类推。最后，系统可以选择分数最高的路径

作为源语言句子的最终译文。

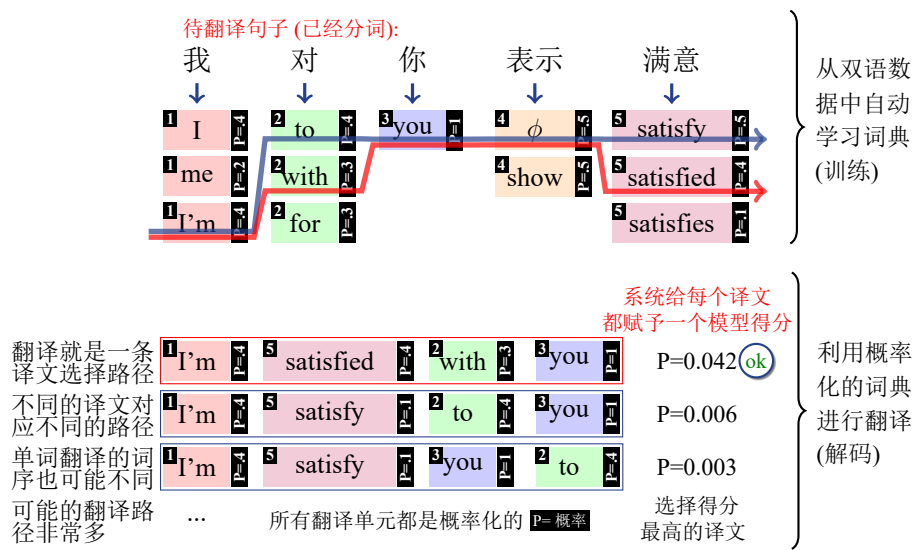


图 3.4: 机器翻译的过程 —— 把单词的译文进行拼装，并找到最优的拼装路径

人工翻译 vs. 机器翻译

人在翻译时的决策是非常确定并且快速的，但计算机处理这个问题时却充满了概率化的思想。当然它们也有类似的地方。首先，计算机使用统计模型的目的是把翻译知识变得可计算，并把这些“知识”储存在模型参数中，这个模型和人类大脑的作用是类似的³；其次，计算机对统计模型进行训练相当于人类对知识的学习，二者都可以被看作是理解、加工知识的过程；再有，计算机使用学习到的模型对新句子进行翻译的过程相当于人运用知识的过程。在统计机器翻译中，模型学习的过程被称为**训练**（Training），目的是从双语平行数据中自动学习翻译“知识”；而使用模型处理新句子的过程被称为**解码**（Decoding）或**推断**（Inference）。图3.4的右侧标注在翻译过程中训练和解码的作用。最终，统计机器翻译的核心由三部分构成 —— 建模、训练和解码。本章后续内容会围绕这三个问题展开讨论。

3.2.2 基本框架

为了对统计机器翻译有一个直观的认识，下面将介绍如何构建一个非常简单的基于单词的统计机器翻译系统，其中涉及到的很多思想来自 IBM 模型。这里，仍然使用数据驱动统计建模方法。图3.5展示了系统的主要流程，包括两个步骤：

- **训练**：从双语平行数据中学习翻译模型，记为 $P(t|s)$ ，其中 s 表示源语言句子， t 表示目标语句子。 $P(t|s)$ 表示把 s 翻译为 t 的概率。简言之，这一步需要从大

³这里并不是要把统计模型等同于生物学或者认知科学上的人脑，这里是指它们处理翻译问题时发挥的作用类似。

量的双语平行数据中学习到 $P(t|s)$ 的准确表达。

- **解码:** 当面对一个新的句子时, 需要使用学习到的模型进行推断。推断可以被视为一个搜索和计算的过程, 也就是, 尽可能搜索更多的翻译结果, 然后用训练好的模型对每个翻译结果进行打分, 最后选择得分最高的翻译结果作为输出。



图 3.5: 简单的统计机器翻译流程

接下来, 本节将介绍统计机器翻译模型训练和解码的方法。在模型学习中, 会分两小节进行描述——单词级翻译和句子级翻译。实现单词级翻译是实现句子级翻译的基础。换言之, 句子级翻译的统计模型是建立在单词翻译之上的。在解码中, 本节将介绍一个高效的搜索算法, 其中也使用到了剪枝和启发式搜索的思想。

3.2.3 单词翻译概率

什么是单词翻译概率?

单词翻译概率描述的是一个源语言单词与目标语言译文构成正确翻译的可能性, 这个概率越高表明单词翻译越可靠。使用单词翻译概率, 可以帮助机器翻译系统解决翻译时的“择词”问题, 即选择什么样的目标语译文是合适的。当人在翻译某个单词时, 可以利用积累的知识, 快速得到它的高质量候选译文。以汉译英为例, 当翻译“我”这个单词时, 可能直接会想到用“I”、“me”或“I’m”作为它的译文, 而几乎不会选择“you”、“satisfied”等含义相差太远的译文。这是为什么呢? 如果从统计学的角度来看, 无论是何种语料, 包括教材、新闻、小说等, 绝大部分情况下“我”都翻译成了“I”、“me”等, 几乎不会看到我被翻译成“you”或“satisfied”的情况。可以说“我”翻译成“I”、“me”等属于高频事件, 而翻译成“you”、“satisfied”等属于低频或小概率事件。因此人在翻译时也是选择在统计意义上概率更大的译文, 这也间接反映出统计模型可以在一定程度上描述人的翻译习惯和模式。

表3.1展示了汉语到英语的单词翻译实例及相应的翻译概率。可以看到, “我”翻译成“I”的概率最高, 为 0.5。这是符合人类对翻译的认知。此外, 这种概率化的模型避免了非 0 即 1 的判断, 所有的译文都是可能的, 只是概率不同。这也使得统计模型可以覆盖更多的翻译现象, 甚至捕捉到一些人所忽略的情况。

源语言	目标语言	翻译概率
我	I	0.50
	me	0.20
	I'm	0.10
	we	0.05
	am	0.10
...

表 3.1: 汉译英单词翻译概率

如何从一个双语平行数据中学习？

假设有一定数量的双语对照的平行数据，是否可以从中自动获得两种语言单词之间的翻译概率呢？回忆一下第二章中的掷骰子游戏，其中使用了相对频度估计方法来自动获得骰子不同面出现概率的估计值。其中，重复投掷骰子很多次，然后统计“1”到“6”各面出现的次数，再除以投掷的总次数，最后得到它们出现的概率的极大似然估计。这里，可以使用类似的方式计算单词翻译概率。但是，现在有的是句子一级对齐的数据，并不知道两种语言之间单词的对应关系。也就是，要从句子级对齐的平行数据中学习单词之间对齐的概率。这里，需要使用稍微“复杂”一些的模式来描述这个问题。

令 X 和 Y 分别表示源语言和目标语言的词汇表。对于任意源语言单词 $x \in X$ ，所有的目标语单词 $y \in Y$ 都可能是它的译文。给定一个互译的句对 (s, t) ，可以把 $P(x \leftrightarrow y; s, t)$ 定义为：在观测到 (s, t) 的前提下 x 和 y 互译的概率。其中 x 是属于句子 s 中的词，而 y 是属于句子 t 中的词。 $P(x \leftrightarrow y; s, t)$ 的计算公式描述如下：

$$\begin{aligned} P(x \leftrightarrow y; s, t) &\equiv P(x, y; s, t) \\ &= \frac{c(x, y; s, t)}{\sum_{x', y'} c(x', y'; s, t)} \end{aligned}$$

(3.1)

其中， \equiv 表示定义式。分子 $c(x, y; s, t)$ 表示 x 和 y 在句对 (s, t) 中共现的总次数，分母 $\sum_{x', y'} c(x', y'; s, t)$ 表示任意的源语言单词 x' 和任意的目标语言单词 y' 在 (s, t) 共同出现的总次数。

■ 实例 3.1 一个汉英互译的句对

s = 机器 翻译 就是 用 计算机 来 进行 翻译

t = machine translation is just translation by computer

■

看一个具体的例子，如例3.1所示，有一个汉英互译的句对 (s, t) 。假设， $x = \text{“翻译”}$ ， $y = \text{“translation”}$ ，现在要计算 x 和 y 共现的总次数。“翻译”和“translation”分别在 s 和 t 中出现了 2 次，因此 $c(\text{“翻译”}, \text{“translation”}; s, t)$ 等于 4。而对于 $\sum_{x', y'} c(x', y'; s, t)$ ，因为 x' 和 y' 分别表示的是 s 和 t 中的任意词，所以 $\sum_{x', y'} c(x', y'; s, t)$ 表示所有单词对的数量——即 s 的词数乘以 t 的词数。最后，“翻译”和“translation”的单词

翻译概率为:

$$\begin{aligned}
 P(\text{“翻译”}, \text{“translation”}; \mathbf{s}, \mathbf{t}) &= \frac{c(\text{“翻译”}, \text{“translation”}; \mathbf{s}, \mathbf{t})}{\sum_{x', y'} c(x', y'; \mathbf{s}, \mathbf{t})} \\
 &= \frac{4}{|\mathbf{s}| \times |\mathbf{t}|} \\
 &= \frac{4}{63}
 \end{aligned} \tag{3.2}$$

这里运算 $|\cdot|$ 表示句子长度。类似的, 可以得到“机器”和“translation”、“机器”和“look”的单词翻译概率:

$$P(\text{“机器”}, \text{“translation”}; \mathbf{s}, \mathbf{t}) = \frac{2}{63} \tag{3.3}$$

$$P(\text{“机器”}, \text{“look”}; \mathbf{s}, \mathbf{t}) = \frac{0}{63} \tag{3.4}$$

注意, 由于“look”没有出现在数据中, 因此 $P(\text{“机器”}, \text{“look”}; \mathbf{s}, \mathbf{t}) = 0$ 。这时, 可以使用第二章介绍的平滑算法赋予它一个非零的值, 以保证在后续的步骤中整个翻译模型不会出现零概率的情况。

如何从大量的双语平行数据中学习?

如果有更多的句子, 上面的方法同样适用。假设, 有 N 个互译句对 $(\mathbf{s}^{[1]}, \mathbf{t}^{[1]}), \dots, (\mathbf{s}^{[N]}, \mathbf{t}^{[N]})$ 。仍然可以使用基于相对频度的方法估计翻译概率 $P(x, y)$, 具体方法如下:

$$P(x, y) = \frac{\sum_{i=1}^N c(x, y; \mathbf{s}^{[i]}, \mathbf{t}^{[i]})}{\sum_{i=1}^N \sum_{x', y'} c(x', y'; \mathbf{s}^{[i]}, \mathbf{t}^{[i]})} \tag{3.5}$$

与公式3.1相比, 分子分母都多了一项累加符号 $\sum_{i=1}^N$, 它表示遍历语料库中所有的句对。换句话说, 当计算词的共现次数时, 需要对每个句对上的计数结果进行累加。从统计学习的角度, 使用更大规模的数据进行参数估计可以提高估计结果的可靠性。在自然语言处理任务中, 使用更大规模的数据往往是提高系统性能的捷径。计算单词的翻译概率也是一样, 在小规模的数据上看, 很多翻译现象的特征并不突出, 但是当使用的数据量增加到一定程度, 翻译的规律会很明显的体现出来。

■ 实例 3.2 两个汉英互译的句对

\mathbf{s}^1 = 机器 翻译 就是 用 计算机 来 进行 翻译

\mathbf{s}^1 = Machine translation is just translation by computer

\mathbf{s}^2 = 那 人工 翻译 呢 ?

\mathbf{t}^2 = So , what is human translation ?

举个例子来说明在多个句子上计算单词翻译概率的方法。例3.2展示了一个由两个句对构成的平行语料库。其中, $\mathbf{s}^{[1]}$ 和 $\mathbf{s}^{[2]}$ 分别表示第一个句对和第二个句对的源

语言句子, $\mathbf{t}^{[1]}$ 和 $\mathbf{t}^{[2]}$ 表示对应的目标语言句子。于是, “翻译” 和 “translation” 的翻译概率为

$$\begin{aligned}
 P(\text{“翻译”, “translation”}) &= \frac{c(\text{“翻译”, “translation”}; \mathbf{s}^{[1]}, \mathbf{t}^{[1]}) + c(\text{“翻译”, “translation”}; \mathbf{s}^{[2]}, \mathbf{t}^{[2]})}{\sum_{x', y'} c(x', y'; \mathbf{s}^{[1]}, \mathbf{t}^{[1]}) + \sum_{x', y'} c(x', y'; \mathbf{s}^{[2]}, \mathbf{t}^{[2]})} \\
 &= \frac{4 + 1}{|\mathbf{s}^{[1]}| \times |\mathbf{t}^{[1]}| + |\mathbf{s}^{[2]}| \times |\mathbf{t}^{[2]}|} \\
 &= \frac{4 + 1}{9 \times 7 + 5 \times 7} \\
 &= \frac{5}{98}
 \end{aligned} \tag{3.6}$$

公式3.6所展示的计算过程很简单, 分子是两个句对中 “翻译” 和 “translation” 共现次数的累计, 分母是两个句对的源语言单词和目标语言单词的组合数的累加。显然, 这个方法也很容易推广到处理更多句子的情况。

3.2.4 句子级翻译模型

下面继续回答如何获取句子级翻译概率的问题。如图3.6所示, 条件概率 $P(\mathbf{t}|\mathbf{s})$ 表示给出源语言句子 \mathbf{s} 的情况下译文为 \mathbf{t} 的概率。这也是整个句子级翻译模型的核心, 一方面需要从数据中学习这个模型的参数, 另一方面, 对于新输入的句子, 需要使用这个模型得到最佳的译文。下面介绍句子级翻译的建模方法。



图 3.6: $P(\mathbf{t}|\mathbf{s})$ 在句子级翻译中的作用

基础模型

计算句子级翻译概率并不简单。因为自然语言非常灵活, 任何数据无法覆盖足够多的句子, 因此, 无法像公式3.5一样直接用简单计数的方式对句子的翻译概率进行估计。这里, 采用一个退而求其次的方法: 找到一个函数 $g(\mathbf{s}, \mathbf{t}) \geq 0$ 来模拟翻译概率对译文可能性进行估计。可以定义一个新的函数 $g(\mathbf{s}, \mathbf{t})$, 令其满足: 给定 \mathbf{s} , 翻译结果 \mathbf{t} 出现的可能性越大, $g(\mathbf{s}, \mathbf{t})$ 的值越大; \mathbf{t} 出现的可能性越小, $g(\mathbf{s}, \mathbf{t})$ 的值越小。换句话说, $g(\mathbf{s}, \mathbf{t})$ 的单调性和翻译概率 $P(\mathbf{t}|\mathbf{s})$ 呈正相关。如果存在这样的函数 $g(\mathbf{s}, \mathbf{t})$, 可以利用 $g(\mathbf{s}, \mathbf{t})$ 近似表示 $P(\mathbf{t}|\mathbf{s})$, 如下:

$$P(\mathbf{t}|\mathbf{s}) \equiv \frac{g(\mathbf{s}, \mathbf{t})}{\sum_{\mathbf{t}'} g(\mathbf{s}, \mathbf{t}')} \tag{3.7}$$

公式3.7相当于在函数 $g(\cdot)$ 上做了归一化, 这样等式右端的结果具有一些概率的属性, 比如, $0 \leq \frac{g(\mathbf{s}, \mathbf{t})}{\sum_{\mathbf{t}'} g(\mathbf{s}, \mathbf{t}')} \leq 1$ 。具体来说, 对于源语言句子 \mathbf{s} , 枚举其所有的翻译

结果，并把所对应的函数 $g(\cdot)$ 相加作为分母，而分子是某个翻译结果 t 所对应的 $g(\cdot)$ 的值。

上述过程初步建立了句子级翻译模型，并没有直接求 $P(t|s)$ ，而是把问题转化为对 $g(\cdot)$ 的设计和计算上。但是，面临着两个新的问题：

- 如何定义函数 $g(s,t)$ ？即，在知道单词翻译概率的前提下，如何计算 $g(s,t)$ ；
- 公式3.7中分母 $\sum_{t'} g(s,t')$ 需要累加所有翻译结果的 $g(s,t')$ ，但枚举所有 t' 是不现实的。

当然，这里最核心的问题还是函数 $g(s,t)$ 的定义。而第二个问题其实不需要解决，因为机器翻译只关注于可能性最大的翻译结果，即 $g(s,t)$ 的计算结果最大时对应的译文。这个问题会在后面进行讨论。

回到设计 $g(s,t)$ 的问题上。这里，采用“大题小作”的方法，这个技巧在第二章已经进行了充分的介绍。具体来说，直接建模句子之间的对应比较困难，但可以利用单词之间的对应来描述句子之间的对应关系。这就用到了上一小节所介绍的单词翻译概率。

首先引入一个非常重要的概念——**词对齐**（Word Alignment），它是统计机器翻译中最核心的概念之一。词对齐描述了平行句对中单词之间的对应关系，它体现了一种观点：本质上句子之间的对应是由单词之间的对应表示的。当然，这个观点在神经机器翻译或者其他模型中可能会有不同的理解，但是翻译句子的过程中考虑词级的对应关系是符合我们对语言的认知的。图3.7 展示了一个句对 s 和 t ，单词的右下标数字表示了该词在句中的位置，而虚线表示的是句子 s 和 t 中的词对齐关系。比如，“满意”的右下标数字 5 表示在句子 s 中处于第 5 个位置，“satisfied”的右下标数字 3 表示在句子 t 中处于第 3 个位置，“满意”和“satisfied”之间的虚线表示两个单词之间是对齐的。为方便描述，用二元组 (j,i) 来描述词对齐，它表示源语言句子的第 j 个单词对应目标语言句子的第 i 个单词，即单词 s_j 和 t_i 对应。通常，也会把 (j,i) 称作一条**词对齐连接**。图3.7 中共有 5 条虚线，表示有 5 组单词之间的词对齐连接。可以把这些词对齐连接构成的集合作为词对齐的一种表示，记为 A ，即 $A = \{(1,1), (2,4), (3,5), (4,2), (5,3)\}$ 。

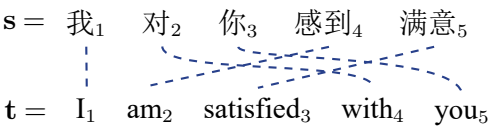


图 3.7: 汉英互译句对及词对齐连接（蓝色虚线）

对于句对 (s,t) ，假设可以得到最优词对齐 \hat{A} ，于是可以使用单词翻译概率计算

$g(s, t)$, 如下

$$g(s, t) = \prod_{(j,i) \in \hat{A}} P(s_j, t_i)$$

(3.8)

其中 $g(s, t)$ 被定义为句子 s 中的单词和句子 t 中的单词的翻译概率的乘积, 并且这两个单词之间必须有对齐连接。 $P(s_j, t_i)$ 表示具有对齐链接的源语言单词 s_j 和目标语言单词 t_i 的单词翻译概率。以图3.7中的句对为例, 其中“我”与“I”、“对”与“with”、“你”与“you”等相互对应, 可以把它们的翻译概率相乘得到 $g(s, t)$ 的计算结果, 如下:

$$g(s, t) = P(\text{“我”, “I”}) \times P(\text{“对”, “with”}) \times P(\text{“你”, “you”}) \times$$

$$P(\text{“感到”, “am”}) \times P(\text{“满意”, “satisfied”})$$

(3.9)

显然, 如果每个词对齐连接所对应的翻译概率变大, 那么整个句子翻译的得分也会提高。也就是说, 词对齐越准确, 翻译模型的打分越高, s 和 t 之间存在翻译关系的可能性越大。

生成流畅的译文

公式3.8定义的 $g(s, t)$ 存在的问题是没有考虑词序信息。这里用一个简单的例子说明这个问题。如图3.8所示, 源语言句子“我对你感到满意”有两个翻译结果, 第一个翻译结果是“I am satisfied with you”, 第二个是“I with you am satisfied”。虽然这两个译文包含的目标语单词是一样的, 但词序存在很大差异。比如, 它们都选择了“satisfied”作为源语单词“满意”的译文, 但是在第一个翻译结果中“satisfied”处于第3个位置, 而第二个结果中处于最后的位置。显然第一个翻译结果更符合英文的表达习惯, 翻译的质量更高。遗憾的是, 对于有明显差异的两个译文, 公式3.8计算得到的函数 $g(\cdot)$ 的值却是一样的。

					$\prod_{(j,i) \in \hat{A}} P(s_j, t_i)$
$s =$	我 ₁	对 ₂	你 ₃	感到 ₄	满意 ₅
$t' =$	I ₁	am ₂	satisfied ₃	with ₄	you ₅
$s =$	我 ₁	对 ₂	你 ₃	感到 ₄	满意 ₅
$t'' =$	I ₁	with ₂	you ₃	am ₄	satisfied ₅
					0.0023
					0.0023

图 3.8: 同一个源语言句子的不同译文所对应的 $g(\cdot)$ 得分

如何在 $g(s, t)$ 引入词序信息呢? 我们希望函数 $g(s, t)$ 对符合自然语言表达习惯的翻译结果给出更高的分数, 对于不符合的或不通顺的句子给出更低的分数。这里

很自然想到使用语言模型，因为语言模型可以度量一个句子出现的可能性。流畅的句子语言模型得分越高，反之越低。

这里可以使用第二章介绍的 n -gram 语言模型，它也是统计机器翻译中确保流畅翻译结果的重要手段之一。 n -gram 语言模型用概率化方法描述了句子的生成过程。以 2-gram 语言模型为例，可以使用如下公式计算一个词串的概率：

$$\begin{aligned} P_{lm}(t) &= P_{lm}(t_1...t_l) \\ &= P(t_1) \times P(t_2|t_1) \times P(t_3|t_2) \times ... \times P(t_l|t_{l-1}) \end{aligned}$$

(3.10)

其中， $t = t_1...t_l$ 表示由 l 个单词组成的句子， $P_{lm}(t)$ 表示语言模型给句子 t 的打分。具体而言， $P_{lm}(t)$ 被定义为 $P(t_i|t_{i-1})(i = 1, 2, ..., l)$ 的连乘⁴，其中 $P(t_i|t_{i-1})(i = 1, 2, ..., l)$ 表示前面一个单词为 t_{i-1} 时，当前单词为 t_i 的概率。语言模型的训练方法可以参看第二章相关内容。

回到建模问题上来。既然语言模型可以帮助系统度量每个译文的流畅度，那么可以使用它对翻译进行打分。一种简单的方法是把语言模型 $P_{lm}(t)$ 和公式3.8中的 $g(s, t)$ 相乘，这样就得到了一个新的 $g(s, t)$ ，它同时考虑了翻译准确性 ($\prod_{j,i \in \hat{A}} P(s_j, t_i)$) 和流畅度 ($P_{lm}(t)$)：

$$g(s, t) \equiv \prod_{j,i \in \hat{A}} P(s_j, t_i) \times P_{lm}(t)$$

(3.11)

如图3.9所示，语言模型 $P_{lm}(t)$ 分别给 t' 和 t'' 赋予 0.0107 和 0.0009 的概率，这表明句子 t' 更符合英文的表达，这与我们的期望是相吻合的。它们再分别乘以 $\prod_{j,i \in \hat{A}} P(s_j, t_i)$ 的值，就得到公式3.11定义的函数 $g(\cdot)$ 的值。显然句子 t' 的分数更高，同时它也是我们希望得到的翻译结果。至此，我们完成了对函数 $g(s, t)$ 的一个简单定义，把它带入公式3.7就得到了同时考虑准确性和流畅性的句子级统计翻译模型。

					$\prod_{(j,i) \in \hat{A}} P(s_j, t_i) \times P_{lm}(t)$
s = 我 ₁	对 ₂	你 ₃	感到 ₄	满意 ₅	0.0023 × 0.0107
t' = I ₁	am ₂	satisfied ₃	with ₄	you ₅	
s = 我 ₁	对 ₂	你 ₃	感到 ₄	满意 ₅	0.0023 × 0.0009
t'' = I ₁	with ₂	you ₃	am ₄	satisfied ₅	

图 3.9: 同一个源语言句子的不同译文所对应的语言模型得分和翻译模型得分

⁴为了确保数学表达的准确性，这书中定义 $P(t_1|t_0) \equiv P(t_1)$

3.2.5 解码

解码（Decoding）是指在得到翻译模型后，对于新输入的句子生成最佳译文的过程。具体来说，当给定任意的源语言句子 s ，解码系统要找到翻译概率最大的目标语译文 \hat{t} 。这个过程可以被形式化描述为：

$$\hat{t} = \operatorname{argmax}_t P(t|s) \tag{3.12}$$

其中 $\operatorname{argmax}_t P(t|s)$ 表示找到使 $P(t|s)$ 达到最大时的译文 t 。结合上一小节中关于 $P(t|s)$ 的定义，把公式3.7带入公式3.12得到：

$$\hat{t} = \operatorname{argmax}_t \frac{g(s,t)}{\sum_{t'} g(s,t')} \tag{3.13}$$

在公式3.13中，可以发现 $\sum_{t'} g(s,t')$ 是一个关于 s 的函数，当给定源语句 s 时，它是一个常数，而且 $g(\cdot) \geq 0$ ，因此 $\sum_{t'} g(s,t')$ 不影响对 \hat{t} 的求解，也不需要计算。基于此，公式3.13可以被化简为：

$$\hat{t} = \operatorname{argmax}_t g(s,t) \tag{3.14}$$

公式3.14定义了解码的目标，剩下的问题是实现 argmax ，以快速准确的找到最佳译文 \hat{t} 。但是，简单遍历所有可能的译文并计算 $g(s,t)$ 的值是不可行的，因为所有潜在译文构成的搜索空间是十分巨大的。为了理解机器翻译的搜索空间的规模，假设源语言句子 s 有 m 个词，每个词有 n 个可能的翻译候选。如果从左到右一步步翻译每个源语言单词，那么简单的顺序翻译会有 n^m 种组合。如果进一步考虑目标语单词的任意调序，每一种对翻译候选进行选择的结果又会对应 $m!$ 种不同的排序。因此，源语句句子 s 至少有 $n^m \cdot m!$ 个不同的译文。

$n^m \cdot m!$ 是什么样的概念呢？如表3.2所示，当 m 和 n 分别为 2 和 10 时，译文只有 200 个，不算多。但是当 m 和 n 分别为 20 和 10 时，即源语言句子的长度 20，每个词有 10 个候选译文，系统会面对 2.4329×10^{38} 个不同的译文，这几乎是不可计算的。

表 3.2: 机器翻译搜索空间大小的示例

句子长度 m	单词翻译候选数量 n	译文数量 $n^m \cdot m!$
1	1	1
1	10	10
2	10	200
10	10	3628800000000000
20	10	$2.43290200817664 \times 10^{38}$
20	30	$8.48300477127188 \times 10^{47}$

已经有工作证明机器翻译问题是 NP 难的 [145]。对于如此巨大的搜索空间，需要一种十分高效的搜索算法才能实现机器翻译的解码。这里介绍一种贪婪的解码算法，它把解码分成若干步骤，每步只翻译一个单词，并保留当前“最好”的结果，直至所有源语言单词都被翻译完毕。

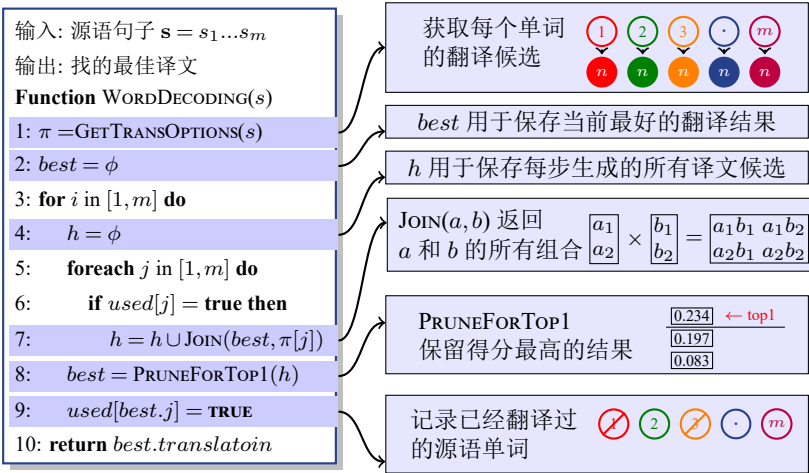
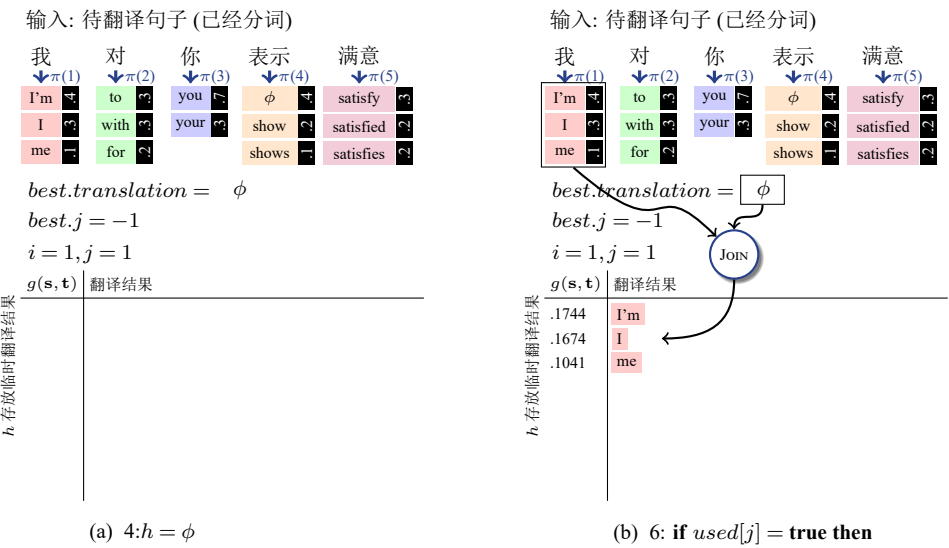


图 3.10: 贪婪的机器翻译解码算法的伪代码

图3.10给出了贪婪解码算法的伪代码。其中 π 保存所有源语单词的候选译文, $\pi[j]$ 表示第 j 个源语单词的翻译候选的集合, $best$ 保存当前最好的翻译结果, h 保存当前步生成的所有译文候选。算法的主体有两层循环, 在内层循环中如果第 j 个源语单词没有被翻译过, 则用 $best$ 和它的候选译文 $\pi[j]$ 生成新的翻译, 再存于 h 中, 即操作 $h = h \cup \text{Join}(best, \pi[j])$ 。外层循环再从 h 中选择得分最高的结果存于 $best$ 中, 即操作 $best = \text{PruneForTop1}(h)$; 同时标识相应的源语单词已翻译, 即 $used[best.j] = \text{true}$ 。



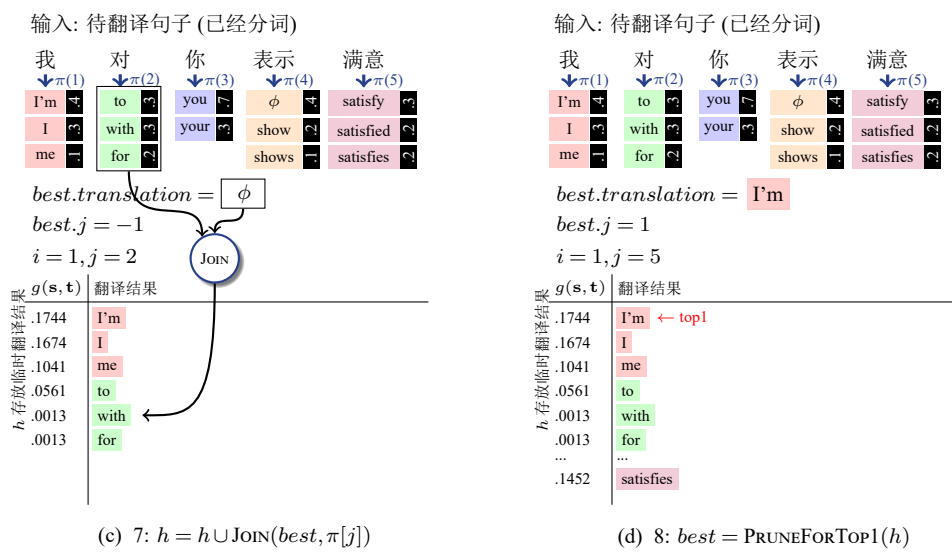


图 3.11: 贪婪的机器翻译解码过程实例

该算法的核心在于，系统一直维护一个当前最好的结果，之后每一步考虑扩展这个结果的所有可能，并计算模型得分，然后再保留扩展后的最好结果。注意，在每一步中，只有排名第一的结果才会被保留，其他结果都会被丢弃。这也体现了贪婪的思想。显然这个方法不能保证搜索到全局最优的结果，但是由于每次扩展只考虑一个最好的结果，因此该方法速度很快。图3.11给出了算法执行过程的简单示例。当然，机器翻译的解码方法有很多，这里仅仅使用简单的贪婪搜索方法来解决机器翻译的解码问题，在后续章节会对更加优秀的解码方法进行介绍。

3.3 基于词的翻译建模

在3.2节中，我们实现了一个简单的基于词的统计机器翻译模型，内容涉及建模、训练和解码。但是，还有很多问题还没有进行深入讨论，比如，如何处理空翻译？如何对调序问题进行建模？如何用更严密的数学模型描述翻译过程？如何对更加复杂的统计模型进行训练？等等。针对以上问题，本节将系统的介绍 IBM 统计机器翻译模型。作为经典的机器翻译模型，对 IBM 模型的学习将帮助我们建立对自然语言处理问题的系统化建模思想，特别是对问题的数学描述方法将会成为理解本书后续内容的基础工具。

3.3.1 噪声信道模型

首先，重新思考一下人类进行翻译的过程。对于给定的源语句 s ，人不会像计算机一样尝试很多的可能，而是快速准确的翻译出一个或者少数几个正确的译文。在人看来，除了正确的译文外，其他的翻译都是不正确的，或者说除了少数的译文人

甚至都不会考虑太多其他的可能性。但是，在统计机器翻译的世界里，没有译文是不可能的。换句话说，对于源语言句子 s ，所有目标语词串 t 都是可能的译文，只是可能性大小不同。即每对 (s, t) 都有一个概率值 $P(t|s)$ 来描述 s 翻译为 t 的好与坏（图3.12）。

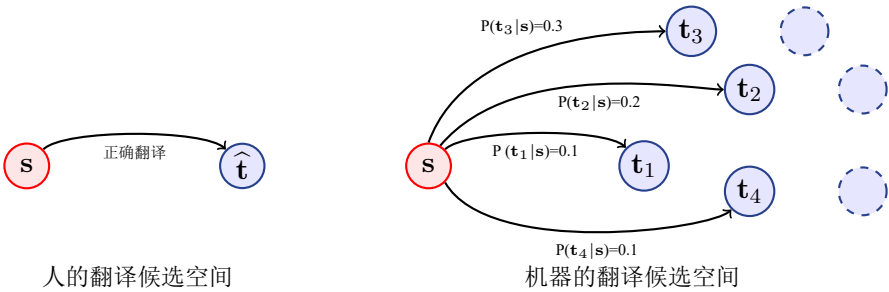


图 3.12: 不同翻译候选空间的对比：人（左）vs 机器翻译（右）

IBM 模型也是建立在如上统计模型之上。具体来说，IBM 模型的基础是**噪声信道模型**（Noise Channel Model），它是由 Shannon 在上世纪 40 年代末提出来的 [254]，并于上世纪 80 年代应用在语言识别领域，后来又被 Brown 等人用于统计机器翻译中 [24]。

在噪声信道模型中，源语言句子 s （信宿）被看作是由目标语言句子 t （信源）经过一个有噪声的信道得到的。如果知道了 s 和信道的性质，可以通过 $P(t|s)$ 得到信源的信息，这个过程如图3.13所示。

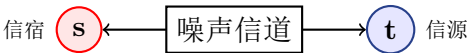


图 3.13: 噪声信道模型，其中 s 表示信宿， t 表示信源

举个例子。对于汉译英的翻译任务，汉语句子 s 可以被看作是英语句子 t 加入噪声通过信道后得到的结果。换句话说，英语句子经过噪声-信道传输时发生了变化，在信道的输出端呈现为汉语句子。于是需要根据观察到的汉语特征，通过概率 $P(t|s)$ 猜测最为可能的英语句子。这个找到最可能的目标语句（信源）的过程也被称为**解码**（Decoding）。直到今天，解码这个概念也被广泛的使用在机器翻译及相关任务中。这个过程也可以表述为：给定输入 s ，找到最可能的输出 t ，使得 $P(t|s)$ 达到最大：

$$\hat{t} = \underset{t}{\operatorname{argmax}} P(t|s) \tag{3.15}$$

公式3.15的核心内容之一是定义 $P(t|s)$ 。在 IBM 模型中，可以使用贝叶斯准则

对 $P(t|s)$ 进行如下变换：

$$\begin{aligned} P(t|s) &= \frac{P(s,t)}{P(s)} \\ &= \frac{P(s|t)P(t)}{P(s)} \end{aligned} \quad (3.16)$$

公式3.16把 s 到 t 的翻译概率转化为 $\frac{P(s|t)P(t)}{P(s)}$ ，它包括三个部分：

- 第一部分是由译文 t 到源语言句子 s 的翻译概率 $P(s|t)$ ，也被称为翻译模型。它表示给定目标语句 t 生成源语句 s 的概率，需要注意的是翻译的方向已经从 $P(t|s)$ 转向了 $P(s|t)$ ，但无须刻意的区分，可以简单地理解为翻译模型刻画了 s 和 t 的翻译对应程度；
- 第二部分是 $P(t)$ ，也被称为语言模型。它表示的是目标语言句子 t 出现的可能性；
- 第三部分是 $P(s)$ ，表示源语言句子 s 出现的可能性。因为 s 是输入的不变量，而且 $P(s) \geq 0$ ，所以省略分母部分 $P(s)$ 不会影响 $\frac{P(s|t)P(t)}{P(s)}$ 最大值的求解。

于是，机器翻译的目标可以被重新定义为：给定源语言句子 s ，寻找这样的目标语言译文 t ，它使得翻译模型 $P(s|t)$ 和语言模型 $P(t)$ 乘积最大：

$$\begin{aligned} \hat{t} &= \arg \max_t P(t|s) \\ &= \arg \max_t \frac{P(s|t)P(t)}{P(s)} \\ &= \arg \max_t P(s|t)P(t) \end{aligned} \quad (3.17)$$

公式3.17展示了 IBM 模型最基础的建模方式，它把模型分解为两项：（反向）翻译模型 $P(s|t)$ 和语言模型 $P(t)$ 。一个很自然的问题是：直接用 $P(t|s)$ 定义翻译问题不就可以了么，干嘛用 $P(s|t)$ 和 $P(t)$ 的联合模型？从理论上来说，正向翻译模型 $P(t|s)$ 和反向翻译模型 $P(s|t)$ 的数学建模可以是一样的，因为我们只需要在建模的过程中把两个语言调换即可。使用 $P(s|t)$ 和 $P(t)$ 的联合模型的意义在于引入了语言模型，它可以很好的对译文的流畅度进行评价，确保结果是通顺的目标语言句子。可以回忆一下3.2.4节中讨论的问题，如果只使用翻译模型可能会造成一个局面：译文的单词都和源语言单词对应的很好，但是由于语序的问题，读起来却不像人说的话。从这个角度说，引入语言模型是十分必要的。这个问题在 Brown 等人的论文中也有讨论 [24]，他们提到单纯使用 $P(s|t)$ 会把概率分配给一些翻译对应比较好但是不合法的目标语句，而且这部分概率可能会很大，影响模型的决策。这也正体现了 IBM 模型的创新之处，作者用数学技巧把 $P(t)$ 引入进来，保证了系统的输出是通顺的译文。语言模型也被广泛使用在语音识别等领域以保证结果的流畅性，甚至应用的历

史比机器翻译要长得多，这里的方法也有借鉴相关工作的味道。

实际上，在机器翻译中引入语言模型是一个很深刻的概念。在 IBM 模型之后相当长的时间里，语言模型一直是机器翻译各个部件中最重要的部分。即使现在机器翻译模型已经更新换代，对译文连贯性的建模也是所有系统中需要包含的内容（即使隐形体现）。

3.3.2 统计机器翻译的三个基本问题

公式3.17给出了统计机器翻译的数学描述。为了实现这个过程，面临着三个基本问题：

- **建模 (Modeling)**：如何建立 $P(s|t)$ 和 $P(t)$ 的数学模型。换句话说，需要用可计算的方式对翻译问题和语言建模问题进行描述，这也是最核心的问题。
- **训练 (Training)**：如何获得 $P(s|t)$ 和 $P(t)$ 所需的参数。即从数据中得到模型的最优参数。
- **解码 (Decoding)**：如何完成搜索最优解的过程。即完成 $\arg \max$ 。

为了理解以上的问题，可以先回忆一下3.2.3小节中的公式3.11，即 $g(s, t)$ 函数的定义，它用于评估一个译文的好坏。如图3.14所示， $g(s, t)$ 函数与公式3.17的建模方式非常一致，即 $g(s, t)$ 函数中红色部分描述译文 t 的可能性大小，对应翻译模型 $P(s|t)$ ；蓝色部分描述译文的平滑或流畅程度，对应语言模型 $P(t)$ 。尽管这种对应并不十分严格的，但也可以看出在处理机器翻译问题上，很多想法的本质是一样的。

$$g(s, t) = \underbrace{\prod_{(j,i) \in \hat{A}} P(s_j, t_i)}_{\substack{P(s|t) \\ \text{翻译模型}}} \times \underbrace{P_{lm}(t)}_{\substack{P(t) \\ \text{语言模型}}}$$

图 3.14: IBM 模型与公式3.11的对应关系

但 $g(s, t)$ 函数的建模很粗糙，因此下面将介绍 IBM 模型对问题更严谨的定义与建模。对于语言模型 $P(t)$ 和解码过程在前面的内容中都有介绍，所以本章的后半部分会重点介绍如何定义翻译模型 $P(s|t)$ 以及如何训练模型参数。

词对齐

IBM 模型的一个基本的假设是词对齐假设。**词对齐 (Word Alignment)** 描述了源语言句子和目标语句子之间单词级别的对应。具体来说，给定源语句子 $s = s_1 \dots s_m$ 和目标语译文 $t = t_1 \dots t_l$ ，IBM 模型假设词对齐具有如下两个性质。

- 一个源语言单词只能对应一个目标语单词。在图3.15表示的例子中，(a) 和 (c) 都满足该条件，尽管 (c) 中的“谢谢”和“你”都对应“thanks”，但并不违背这个约束。而 (b) 不满足约束，因为“谢谢”同时对应到了两个目标语单词上。这个约束条

件也导致这里的词对齐变成一种**非对称的词对齐**（Asymmetric Word Alignment），因为它只对源语言做了约束，但是目标语言没有。使用这样的约束的目的是为了减少建模的复杂度。在 IBM 模型之后的方法中也提出了双向词对齐，用于建模一个源语言单词对应到多个目标语单词的情况。

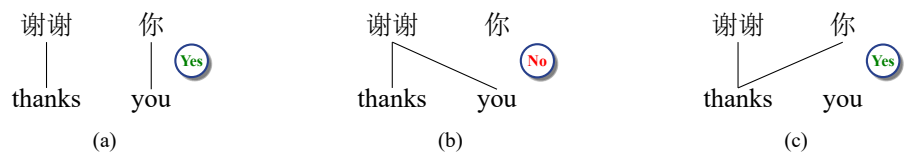


图 3.15: 不同词对齐对比

- 源语言单词可以翻译为空，这时它对应到一个虚拟或伪造的目标语单词 t_0 。在图3.16所示的例子中，“在”没有对应到“on the table”中的任意一个词，而是把它对应到 t_0 上。这样，所有的源语言单词都能找到一个目标语单词对应。这种设计也很好地引入了**空对齐**的思想，即源语言单词不对应任何真实存在的单词的情况。而这种空对齐的情况在翻译中是频繁出现的，比如虚词的翻译。

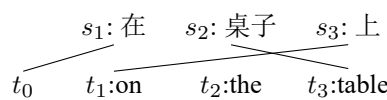


图 3.16: 词对齐实例（“在”对应到 t_0 ）

通常，把词对齐记为 \mathbf{a} ，它由 a_1 到 a_m 共 m 个词对齐连接组成，即 $\mathbf{a} = a_1 \dots a_m$ 。 a_j 表示第 j 个源语单词 s_j 对应的目标语单词的位置。在图3.16的例子中，词对齐关系可以记为 $a_1 = 0, a_2 = 3, a_3 = 1$ ，即第 1 个源语单词“在”对应到目标语译文的第 0 个位置，第 2 个源语单词“桌子”对应到目标语译文的第 3 个位置，第 3 个源语单词“上”对应到目标语译文的第 1 个位置。

基于词对齐的翻译模型

直接准确估计 $P(\mathbf{s}|\mathbf{t})$ 很难，训练数据只能覆盖整个样本空间非常小的一部分，绝大多数句子在训练数据中一次也没出现过。为了解决这个问题，IBM 模型假设：句子之间的对应可以由单词之间的对应进行表示。于是，句子翻译的概率可以被转化为词对齐生成的概率：

$$P(\mathbf{s}|\mathbf{t}) = \sum_{\mathbf{a}} P(\mathbf{s}, \mathbf{a}|\mathbf{t}) \tag{3.18}$$

公式3.18使用了简单的全概率公式把 $P(\mathbf{s}|\mathbf{t})$ 进行展开。通过访问 \mathbf{s} 和 \mathbf{t} 之间所有可能的词对齐 \mathbf{a} ，并把对应的对齐概率进行求和，得到了 \mathbf{t} 到 \mathbf{s} 的翻译概率。这里，可以把词对齐看作翻译的隐含变量，这样从 \mathbf{t} 到 \mathbf{s} 的生成就变为从 \mathbf{t} 同时生成 \mathbf{s} 和隐含变量 \mathbf{a} 的问题。引入隐含变量是生成式模型常用的手段，通过使用隐含变量，可

以把较为困难的端到端学习问题转化为分步学习问题。

举个例子说明公式3.18的实际意义。如图3.17所示，可以把从“谢谢 你”到“thank you”的翻译分解为9种可能的词对齐。因为源语言句子 s 有2个词，目标语言句子 t 加上空标记 t_0 共3个词，因此每个源语言单词有3个可能对齐的位置，整个句子共有 $3 \times 3 = 9$ 种可能的词对齐。

$$\begin{aligned}
 & P\left(\begin{array}{ccc} \text{谢谢} & \text{你} \\ t_0 & \text{thank} & \text{you} \end{array}\right) + P\left(\begin{array}{ccc} \text{谢谢} & \text{你} \\ t_0 & \text{thank} & \text{you} \end{array}\right) + P\left(\begin{array}{ccc} \text{谢谢} & \text{你} \\ t_0 & \text{thank} & \text{you} \end{array}\right) + \\
 & P\left(\begin{array}{ccc} \text{谢谢} & \text{你} \\ t_0 & \text{thank} & \text{you} \end{array}\right) + P\left(\begin{array}{ccc} \text{谢谢} & \text{你} \\ t_0 & \text{thank} & \text{you} \end{array}\right) + P\left(\begin{array}{ccc} \text{谢谢} & \text{你} \\ t_0 & \text{thank} & \text{you} \end{array}\right) + \\
 & P\left(\begin{array}{ccc} \text{谢谢} & \text{你} \\ t_0 & \text{thank} & \text{you} \end{array}\right) + P\left(\begin{array}{ccc} \text{谢谢} & \text{你} \\ t_0 & \text{thank} & \text{you} \end{array}\right) + P\left(\begin{array}{ccc} \text{谢谢} & \text{你} \\ t_0 & \text{thank} & \text{you} \end{array}\right) = P(s|t)
 \end{aligned}$$

图 3.17: 一个汉译英句对的所有词对齐可能

接下来的问题是如何定义 $P(s, a|t)$ —— 即定义词对齐的生成概率。但是，隐含变量 a 仍然很复杂，因此直接定义 $P(s, a|t)$ 也很困难，在 IBM 模型中，为了化简问题， $P(s, a|t)$ 被进一步分解。使用链式法则，可以得到：

$$P(s, a|t) = P(m|t) \prod_{j=1}^m P(a_j | a_1^{j-1}, s_1^{j-1}, m, t) P(s_j | a_1^j, s_1^{j-1}, m, t) \quad (3.19)$$

其中 s_j 和 a_j 分别表示第 j 个源语言单词及第 j 个源语言单词对应到目标位置， s_1^{j-1} 表示前 $j-1$ 个源语言单词（即 $s_1^{j-1} = s_1 \dots s_{j-1}$ ）， a_1^{j-1} 表示前 $j-1$ 个源语言的词对齐（即 $a_1^{j-1} = a_1 \dots a_{j-1}$ ）， m 表示源语句子的长度。公式3.19可以进一步被分解为四个部分，具体含义如下：

- 根据译文 t 选择源文 s 的长度 m ，用 $P(m|t)$ 表示；
- 当确定源语言句子的长度 m 后，循环每个位置 j 逐次生成每个源语言单词 s_j ，也就是 $\prod_{j=1}^m$ 计算的内容；
- 对于每个位置 j ，根据译文 t 、源文长度 m 、已经生成的源语言单词 s_1^{j-1} 和对齐 a_1^{j-1} ，生成第 j 个位置的对齐结果 a_j ，用 $P(a_j | a_1^{j-1}, s_1^{j-1}, m, t)$ 表示；
- 对于每个位置 j ，根据译文 t 、源文长度 m 、已经生成的源语言单词 s_1^{j-1} 和对齐 a_1^j ，生成第 j 个位置的源语言单词 s_j ，用 $P(s_j | a_1^j, s_1^{j-1}, m, t)$ 表示。

换句话说，当求 $P(s, a|t)$ 时，首先根据译文 t 确定源语言句子 s 的长度 m ；当知道源语言句子有多少个单词后，循环 m 次，依次生成第1个到第 m 个源语言单词；当生成第 j 个源语言单词时，要先确定它是由哪个目标语译文单词生成的，即确定

生成的源语言单词对应的译文单词的位置；当知道了目标语译文单词的位置，就能确定第 j 个位置的源语言单词。

需要注意的是公式3.19定义的模型并没有做任何化简和假设，也就是说公式的左右两端是严格相等的。在后面的内容中会看到，这种将一个整体进行拆分的方法可以有助于分步骤化简并处理问题。

基于词对齐的翻译实例

用前面图3.16中例子来对公式3.19进行说明。例子中，源语言句子“在 桌子 上”目标语译文“on the table”之间的词对齐为 $\mathbf{a} = \{1-0, 2-3, 3-1\}$ 。公式3.19的计算过程如下：

- 首先根据译文确定源文 \mathbf{s} 的单词数量 ($m = 3$)，即 $P(m = 3 | \text{"}t_0 \text{ on the table"})$ ；
- 再确定源语言单词 s_1 由谁生成的且生成的是什么。可以看到 s_1 由第 0 个目标语单词生成的，也就是 t_0 ，表示为 $P(a_1 = 0 | \phi, \phi, 3, \text{"}t_0 \text{ on the table"})$ ，其中 ϕ 表示空。当知道了 s_1 是由 t_0 生成的，就可以通过 t_0 生成源语言第一个单词“在”，即 $P(s_1 = \text{"在"} | \{1-0\}, \phi, 3, \text{"}t_0 \text{ on the table"})$ ；
- 类似于生成 s_1 ，依次确定源语言单词 s_2 和 s_3 由谁生成且生成的是什么；

最后得到基于词对齐 \mathbf{a} 的翻译概率为：

$$\begin{aligned}
 P(\mathbf{s}, \mathbf{a} | \mathbf{t}) &= P(m | \mathbf{t}) \prod_{j=1}^m P(a_j | a_1^{j-1}, s_1^{j-1}, m, \mathbf{t}) P(s_j | a_1^j, s_1^{j-1}, m, \mathbf{t}) \\
 &= P(m = 3 | \text{"}t_0 \text{ on the table"}) \times \\
 &\quad P(a_1 = 0 | \phi, \phi, 3, \text{"}t_0 \text{ on the table"}) \times \\
 &\quad P(f_1 = \text{"在"} | \{1-0\}, \phi, 3, \text{"}t_0 \text{ on the table"}) \times \\
 &\quad P(a_2 = 3 | \{1-0\}, \text{"在"}, 3, \text{"}t_0 \text{ on the table"}) \times \\
 &\quad P(f_2 = \text{"桌子"} | \{1-0, 2-3\}, \text{"在"}, 3, \text{"}t_0 \text{ on the table"}) \times \\
 &\quad P(a_3 = 1 | \{1-0, 2-3\}, \text{"在 桌子"}, 3, \text{"}t_0 \text{ on the table"}) \times \\
 &\quad P(f_3 = \text{"上"} | \{1-0, 2-3, 3-1\}, \text{"在 桌子"}, 3, \text{"}t_0 \text{ on the table"}) \quad (3.20)
 \end{aligned}$$

3.4 IBM 模型 1-2

公式3.18和公式3.19把翻译问题定义为对译文和词对齐同时进行生成的问题。其中有两个问题：首先，公式3.18的右端 ($\sum_{\mathbf{a}} P(\mathbf{s}, \mathbf{a} | \mathbf{t})$) 要求对所有的词对齐概率进行求和，但是词对齐的数量随着句子长度是呈指数增长，如何遍历所有的对齐 \mathbf{a} ？其次，公式3.19虽然对词对齐的问题进行了描述，但是模型中的很多参数仍然很复杂，如何计算 $P(m | \mathbf{t})$ 、 $P(a_j | a_1^{j-1}, s_1^{j-1}, m, \mathbf{t})$ 和 $P(s_j | a_1^j, s_1^{j-1}, m, \mathbf{t})$ ？针对这些问题，Brown 等人总共提出了 5 种解决方案，这也就是被后人所熟知的 5 个 IBM 翻译模型。第一个

问题可以通过一定的数学或者工程技巧进行求解；第二个问题可以通过一些假设进行化简，依据化简的层次和复杂度不同，可以分为 IBM 模型 1、IBM 模型 2、IBM 模型 3、IBM 模型 4 以及 IBM 模型 5。本节首先介绍较为简单的 IBM 模型 1-2。

3.4.1 IBM 模型 1

IBM 模型 1 对公式3.19中的三项进行了简化。具体方法如下：

- 假设 $P(m|\mathbf{t})$ 为常数 ε ，即源语言的长度的生成概率服从均匀分布，如下：

$$P(m|\mathbf{t}) \equiv \varepsilon \quad (3.21)$$

- 对齐概率 $P(a_j|a_1^{j-1}, s_1^{j-1}, m, \mathbf{t})$ 仅依赖于译文长度 l ，即每个词对齐连接的概率也服从均匀分布。换句话说，对于任何源语言位置 j 对齐到目标语言任何位置都是等概率的。比如译文为“on the table”，再加上 t_0 共 4 个位置，相应的，任意源语单词对齐到这 4 个位置的概率是一样的。具体描述如下：

$$P(a_j|a_1^{j-1}, s_1^{j-1}, m, \mathbf{t}) \equiv \frac{1}{l+1} \quad (3.22)$$

- 源语单词 s_j 的生成概率 $P(s_j|a_1^j, s_1^{j-1}, m, \mathbf{t})$ 仅依赖与其对齐的译文单词 t_{a_j} ，即词汇翻译概率 $f(s_j|t_{a_j})$ 。此时词汇翻译概率满足 $\sum_{s_j} f(s_j|t_{a_j}) = 1$ 。比如在图3.18表示的例子中，源语单词“上”出现的概率只和与它对齐的单词“on”有关系，与其他单词没有关系。

$$P(s_j|a_1^j, s_1^{j-1}, m, \mathbf{t}) \equiv f(s_j|t_{a_j}) \quad (3.23)$$

用一个简单的例子对公式3.23进行说明。比如，在图3.18中，“桌子”对齐到“table”，可被描述为 $f(s_2|t_{a_2}) = f(\text{“桌子”}|\text{“table”})$ ，表示给定“table”翻译为“桌子”的概率。通常， $f(s_2|t_{a_2})$ 被认为是一种概率词典，它反应了两种语言词汇一级的对应关系。

将上述三个假设和公式3.19代入公式3.18中，得到 $P(\mathbf{s}|\mathbf{t})$ 的表达式：

$$\begin{aligned} P(\mathbf{s}|\mathbf{t}) &= \sum_{\mathbf{a}} P(\mathbf{s}, \mathbf{a}|\mathbf{t}) \\ &= \sum_{\mathbf{a}} P(m|\mathbf{t}) \prod_{j=1}^m P(a_j|a_1^{j-1}, s_1^{j-1}, m, \mathbf{t}) P(s_j|a_1^j, s_1^{j-1}, m, \mathbf{t}) \\ &= \sum_{\mathbf{a}} \varepsilon \prod_{j=1}^m \frac{1}{l+1} f(s_j|t_{a_j}) \\ &= \sum_{\mathbf{a}} \frac{\varepsilon}{(l+1)^m} \prod_{j=1}^m f(s_j|t_{a_j}) \end{aligned} \quad (3.24)$$

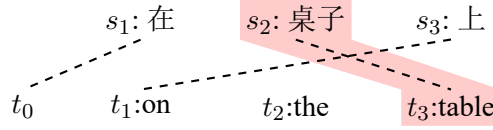


图 3.18: 汉译英双语句对齐及词对齐

在公式3.24中, 需要遍历所有的词对齐, 即 $\sum_{\mathbf{a}}$ 。但这种表示不够直观, 因此可以把这个过程重新表示为如下形式:

$$P(\mathbf{s}|\mathbf{t}) = \sum_{a_1=0}^l \cdots \sum_{a_m=0}^l \frac{\varepsilon}{(l+1)^m} \prod_{j=1}^m f(s_j|t_{a_j}) \quad (3.25)$$

公式3.25分为两个主要部分。第一部分: 遍历所有的对齐 \mathbf{a} 。其中 \mathbf{a} 由 $\{a_1, \dots, a_m\}$ 组成, 每个 $a_j \in \{a_1, \dots, a_m\}$ 从译文开始位置 (0) 循环到截止位置 (l)。如图3.19表示的例子, 描述的是源语单词 s_3 从译文的开始 t_0 遍历到结尾 t_3 , 即 a_3 的取值范围。第二部分: 对于每个 \mathbf{a} 累加对齐概率 $P(\mathbf{s}, \mathbf{a}|\mathbf{t}) = \frac{\varepsilon}{(l+1)^m} \prod_{j=1}^m f(s_j|t_{a_j})$ 。

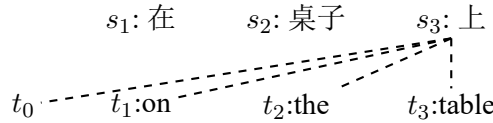


图 3.19: 公式3.25第一部分实例

这样就得到了 IBM 模型 1 中句子翻译概率的计算式。可以看出 IBM 模型 1 的假设把翻译模型化简成了非常简单的形式。对于给定的 \mathbf{s} , \mathbf{a} 和 \mathbf{t} , 只要知道 ε 和 $f(s_j|t_{a_j})$ 就可以计算出 $P(\mathbf{s}|\mathbf{t})$, 进而求出 $P(\mathbf{t}|\mathbf{s})$ 。

3.4.2 IBM 模型 2

IBM 模型 1 很好地化简了问题, 但是由于使用了很强的假设, 导致模型和实际情况有较大差异。其中一个比较严重的问题是假设词对齐的生成概率服从均匀分布。图3.20展示了一个简单的实例。尽管译文 \mathbf{t} 比 \mathbf{t}' 的质量更好, 但对于 IBM 模型 1 来说它们对应的翻译概率相同。这是因为当词对齐服从均匀分布时, 模型会忽略目标语言单词的位置信息。因此当单词翻译相同但顺序不同时, 翻译概率一样。同时, 由于源语言单词是由错误位置的目标语单词生成的, 不合理的对齐也会导致不合理的词汇翻译概率。

因此, IBM 模型 2 抛弃了对 $P(a_j|a_1^{j-1}, s_1^{j-1}, m, \mathbf{t})$ 服从均匀分布的假设。IBM 模型 2 认为词对齐是有倾向性的, 它要与源语单词的位置和目标语单词的位置有关。具体来说, 对齐位置 a_j 的生成概率与位置 j 、源语句子长度 m 和译文长度 l 有关, 形式化表述为:

$$P(a_j|a_1^{j-1}, s_1^{j-1}, m, \mathbf{t}) \equiv a(a_j|j, m, l) \quad (3.26)$$

$$\begin{array}{ccc}
 s = \text{在 桌子 上} & & s = \text{在 桌子 上} \\
 \downarrow & & \downarrow \\
 t = \text{on the table} & & t' = \text{table on the} \\
 \text{IBM 模型 1: } P(s|t) & = & P(s|t') \\
 \text{理想: } P(s|t) & > & P(s|t')
 \end{array}$$

图 3.20: 不同的译文导致不同 IBM 模型 1 得分的情况

这里还用图3.18中的例子来进行说明。在模型 1 中,“桌子”对齐到译文四个位置上的单词的概率是一样的。但在模型 2 中,“桌子”对齐到“table”被形式化为 $a(a_j|j,m,l) = a(3|2,3,3)$, 意思是对于源文位置 2 ($j=2$) 的词, 如果它的源语言和译文都是 3 个词 ($l=3, m=3$), 对齐到目标语译文位置 3 ($a_j=3$) 的概率是多少? 因为 $a(a_j|j,m,l)$ 也是模型需要学习的参数, 因此“桌子”对齐到不同目标语单词的概率也是不一样的。理想的情况下, 通过 $a(a_j|j,m,l)$, “桌子”对齐到“table”应该得到更高的概率。

IBM 模型 2 的其他假设均与模型 1 相同。把公式3.21、3.23和3.26重新带入公式3.19和3.18, 可以得到 IBM 模型 2 的数学描述:

$$\begin{aligned}
 P(s|t) &= \sum_{\mathbf{a}} P(s, \mathbf{a}|t) \\
 &= \sum_{a_1=0}^l \cdots \sum_{a_m=0}^l \varepsilon \prod_{j=1}^m a(a_j|j,m,l) f(s_j|t_{a_j})
 \end{aligned} \quad (3.27)$$

类似于模型 1, 模型 2 的表达式3.27也能被拆分为两部分进行理解。第一部分: 遍历所有的 \mathbf{a} ; 第二部分: 对于每个 \mathbf{a} 累加对齐概率 $P(s, \mathbf{a}|t)$, 即计算对齐概率 $a(a_j|j,m,l)$ 和词汇翻译概率 $f(s_j|t_{a_j})$ 对于所有源语言位置的乘积。

3.4.3 解码及计算优化

如果模型参数给定, 可以使用 IBM 模型 1-2 对新的句子进行翻译。比如, 可以使用3.2.5节描述的解码方法搜索最优译文。在搜索过程中, 只需要通过公式3.25和3.27计算每个译文候选的 IBM 模型翻译概率。但是, 公式3.25和3.27的高计算复杂度导致这些模型很难直接使用。以 IBM 模型 1 为例, 这里把公式3.25重写为:

$$P(s|t) = \frac{\varepsilon}{(l+1)^m} \underbrace{\sum_{a_1=0}^l \cdots \sum_{a_m=0}^l}_{(l+1)^m \text{次循环}} \underbrace{\prod_{j=1}^m f(s_j|t_{a_j})}_{m \text{次循环}} \quad (3.28)$$

可以看到, 遍历所有的词对齐需要 $(l+1)^m$ 次循环, 遍历所有源语言位置累计 $f(s_j|t_{a_j})$ 需要 m 次循环, 因此这个模型的计算复杂度为 $O((l+1)^m m)$ 。当 m 较大时, 计算这

样的模型几乎是不可能的。不过，经过仔细观察，可以发现公式右端的部分有另外一种计算方法，如下：

$$\sum_{a_1=0}^l \dots \sum_{a_m=0}^l \prod_{j=1}^m f(s_j|t_{a_j}) = \prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i) \quad (3.29)$$

公式3.29的技巧在于把若干个乘积的加法（等式左手端）转化为若干加法结果的乘积（等式右手端），这样省去了多次循环，把 $O((l+1)^m m)$ 的计算复杂度降为 $O((l+1)m)$ 。

⁵ 图3.21对这个过程进行了进一步解释。

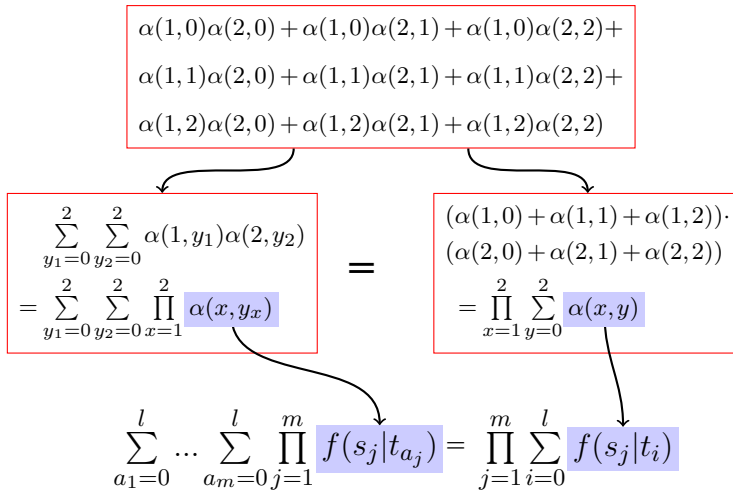


图 3.21: $\sum_{a_1=0}^l \dots \sum_{a_m=0}^l \prod_{j=1}^m f(s_j|t_{a_j}) = \prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i)$ 的实例

接着，利用公式3.29的方式，可以把公式3.25和3.27重写表示为：

$$\text{IBM 模型 1: } P(\mathbf{s}|\mathbf{t}) = \frac{\varepsilon}{(l+1)^m} \prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i) \quad (3.30)$$

$$\text{IBM 模型 2: } P(\mathbf{s}|\mathbf{t}) = \varepsilon \prod_{j=1}^m \sum_{i=0}^l a(i|j, m, l) f(s_j|t_i) \quad (3.31)$$

公式3.30和3.31是 IBM 模型 1-2 的最终表达式，在解码和训练中可以被直接使用。

⁵ 公式3.29相比公式3.28的另一个优点在于，公式3.29中乘法数量更少，因为现代计算机中乘法运算的强度要高于加法，因此公式3.29的计算机实现效率更高。

3.4.4 训练

在完成了建模和解码的基础上，剩下的问题是如何得到模型的参数。这也是整个统计机器翻译里最重要的内容。下面将会对 IBM 模型 1-2 的参数估计方法进行介绍。

目标函数

统计机器翻译模型的训练是一个典型的优化问题。简单来说，训练是指在给定数据集（训练集）上调整参数使得目标函数的值达到最大（或最小），此时得到的参数被称为是该模型在该目标函数下的最优解（图3.22）。

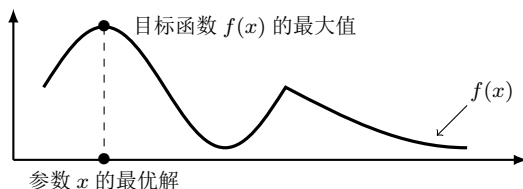


图 3.22: 一个目标函数的最优解

在 IBM 模型中，优化的目标函数被定义为 $P(s|t)$ 。也就是，对于给定的句对 (s, t) ，最大化翻译概率 $P(s|t)$ 。这里用符号 $P_\theta(s|t)$ 表示模型由参数 θ 决定，模型训练可以被描述为对目标函数 $P_\theta(s|t)$ 的优化过程：

$$\hat{\theta} = \arg \max_{\theta} P_\theta(s|t) \quad (3.32)$$

其中， $\arg \max_{\theta}$ 表示求最优参数的过程（或优化过程）。

公式3.32实际上也是一种基于极大似然的模型训练方法。这里，可以把 $P_\theta(s|t)$ 看作是模型对数据描述的一个似然函数，记做 $L(s, t; \theta)$ 。也就是，优化目标是对似然函数的优化： $\{\hat{\theta}\} = \{\arg \max_{\theta \in \Theta} L(s, t; \theta)\}$ ，其中 $\{\hat{\theta}\}$ 表示可能有多个结果， Θ 表示参数空间。

回到 IBM 模型的优化问题上。以 IBM 模型 1 为例，优化的目标是最大化翻译概率 $P(s|t)$ 。使用公式3.30，可以把这个目标表述为：

$$\begin{aligned} & \max \left(\frac{\varepsilon}{(l+1)^m} \prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i) \right) \\ \text{s.t. } & \text{任意单词 } t_y : \sum_{s_x} f(s_x|t_y) = 1 \end{aligned}$$

其中， $\max(\cdot)$ 表示最大化， $\frac{\varepsilon}{(l+1)^m} \prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i)$ 是目标函数， $f(s_j|t_i)$ 是模型的参数， $\sum_{s_x} f(s_x|t_y) = 1$ 是优化的约束条件，以保证翻译概率满足归一化的要求。需要注意的是 $\{f(s_x|t_y)\}$ 对应了很多参数，每个源语言单词和每个目标语单词的组合

都对应一个参数 $f(s_x|t_y)$ 。

优化

我们已经把 IBM 模型的参数训练问题定义为带约束的目标函数优化问题。由于目标函数是可微分函数，解决这类问题的一种常用手法是把带约束的优化问题转化为不带约束的优化问题。这里用到了**拉格朗日乘数法**（The Lagrange Multiplier Method），它的基本思想是把含有 n 个变量和 m 个约束条件的优化问题转化为含有 $n+m$ 个变量的无约束优化问题。

这里的目标是 $\max(\mathbf{P}_\theta(\mathbf{s}|\mathbf{t}))$ ，约束条件是对于任意的目标语单词 t_y 有 $\sum_{s_x} \mathbf{P}(s_x|t_y) = 1$ 。根据拉格朗日乘数法，可以把上述优化问题重新定义最大化如下拉格朗日函数：

$$L(f, \lambda) = \frac{\epsilon}{(l+1)^m} \prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i) - \sum_{t_y} \lambda_{t_y} \left(\sum_{s_x} f(s_x|t_y) - 1 \right) \quad (3.33)$$

$L(f, \lambda)$ 包含两部分， $\frac{\epsilon}{(l+1)^m} \prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i)$ 是原始的目标函数， $\sum_{t_y} \lambda_{t_y} (\sum_{s_x} f(s_x|t_y) - 1)$ 是原始的约束条件乘以拉格朗日乘数 λ_{t_y} ，拉格朗日乘数的数量和约束条件的数量相同。图3.23通过图例说明了 $L(f, \lambda)$ 各部分的意义。

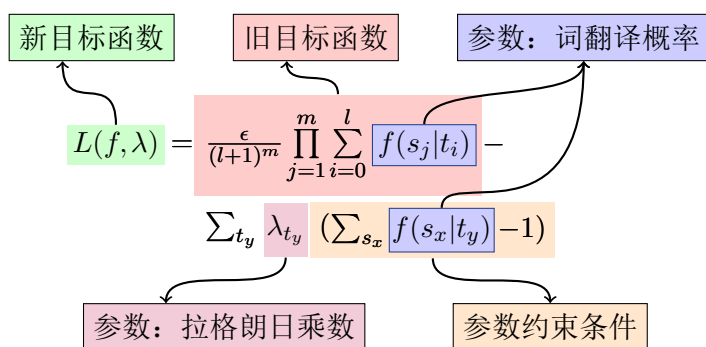


图 3.23: 拉格朗日乘数法 (IBM 模型 1)

因为 $L(f, \lambda)$ 是可微分函数，因此可以通过计算 $L(f, \lambda)$ 导数为零的点得到极值点。因为这个模型里仅有 $f(s_x|t_y)$ 一种类型的参数，只需要对如下导数进行计算。

$$\begin{aligned} \frac{\partial L(f, \lambda)}{\partial f(s_u|t_v)} &= \frac{\partial \left[\frac{\epsilon}{(l+1)^m} \prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i) \right]}{\partial f(s_u|t_v)} - \frac{\partial \left[\sum_{t_y} \lambda_{t_y} (\sum_{s_x} f(s_x|t_y) - 1) \right]}{\partial f(s_u|t_v)} \\ &= \frac{\epsilon}{(l+1)^m} \cdot \frac{\partial \left[\prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i) \right]}{\partial f(s_u|t_v)} - \lambda_{t_v} \end{aligned} \quad (3.34)$$

为了求 $\frac{\partial [\prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i)]}{\partial f(s_u|t_v)}$ ，这里引入一个辅助函数。令 $g(z) = \alpha z^\beta$ 为变量 z 的函数，显然， $\frac{\partial g(z)}{\partial z} = \alpha \beta z^{\beta-1} = \frac{\beta}{z} \alpha z^\beta = \frac{\beta}{z} g(z)$ 。这里可以把 $\prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i)$ 看做 $g(z) = \alpha z^\beta$ 的实例。首先，令 $z = \sum_{i=0}^l f(s_u|t_i)$ ，注意 s_u 为给定的源语单词。然后，把 β 定义为 $\sum_{i=0}^l f(s_u|t_i)$ 在 $\prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i)$ 中出现的次数，即源语句子中与 s_u 相同的单词的个数。

$$\beta = \sum_{j=1}^m \delta(s_j, s_u) \quad (3.35)$$

其中，当 $x = y$ 时， $\delta(x, y) = 1$ ，否则为 0。

根据 $\frac{\partial g(z)}{\partial z} = \frac{\beta}{z} g(z)$ ，可以得到

$$\frac{\partial g(z)}{\partial z} = \frac{\partial [\prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i)]}{\partial [\sum_{i=0}^l f(s_u|t_i)]} = \frac{\sum_{j=1}^m \delta(s_j, s_u)}{\sum_{i=0}^l f(s_u|t_i)} \prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i) \quad (3.36)$$

根据 $\frac{\partial g(z)}{\partial z}$ 和 $\frac{\partial z}{\partial f}$ 计算的结果，可以得到

$$\begin{aligned} \frac{\partial [\prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i)]}{\partial f(s_u|t_v)} &= \frac{\partial [\prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i)]}{\partial [\sum_{i=0}^l f(s_u|t_i)]} \cdot \frac{\partial [\sum_{i=0}^l f(s_u|t_i)]}{\partial f(s_u|t_v)} \\ &= \frac{\sum_{j=1}^m \delta(s_j, s_u)}{\sum_{i=0}^l f(s_u|t_i)} \prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i) \cdot \sum_{i=0}^l \delta(t_i, t_v) \quad (3.37) \end{aligned}$$

将 $\frac{\partial [\prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i)]}{\partial f(s_u|t_v)}$ 进一步代入 $\frac{\partial L(f, \lambda)}{\partial f(s_u|t_v)}$ ，得到 $L(f, \lambda)$ 的导数

$$\begin{aligned} &\frac{\partial L(f, \lambda)}{\partial f(s_u|t_v)} \\ &= \frac{\varepsilon}{(l+1)^m} \cdot \frac{\partial [\prod_{j=1}^m \sum_{i=0}^l f(s_j|t_{a_j})]}{\partial f(s_u|t_v)} - \lambda_{t_v} \\ &= \frac{\varepsilon}{(l+1)^m} \frac{\sum_{j=1}^m \delta(s_j, s_u) \cdot \sum_{i=0}^l \delta(t_i, t_v)}{\sum_{i=0}^l f(s_u|t_i)} \prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i) - \lambda_{t_v} \quad (3.38) \end{aligned}$$

令 $\frac{\partial L(f, \lambda)}{\partial f(s_u|t_v)} = 0$, 有

$$f(s_u|t_v) = \frac{\lambda_{t_v}^{-1} \varepsilon}{(l+1)^m} \cdot \frac{\sum_{j=1}^m \delta(s_j, s_u) \cdot \sum_{i=0}^l \delta(t_i, t_v)}{\sum_{i=0}^l f(s_u|t_i)} \prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i) \cdot f(s_u|t_v) \quad (3.39)$$

将上式稍作调整得到下式:

$$f(s_u|t_v) = \lambda_{t_v}^{-1} \frac{\varepsilon}{(l+1)^m} \prod_{j=1}^m \sum_{i=0}^l f(s_j|t_i) \sum_{j=1}^m \delta(s_j, s_u) \sum_{i=0}^l \delta(t_i, t_v) \frac{f(s_u|t_v)}{\sum_{i=0}^l f(s_u|t_i)} \quad (3.40)$$

可以看出, 这不是一个计算 $f(s_u|t_v)$ 的解析式, 因为等式右端仍含有 $f(s_u|t_v)$ 。不过它蕴含着一种非常经典的方法——**期望最大化** (Expectation Maximization) 方法, 简称 EM 方法 (或算法)。使用 EM 方法可以利用上式迭代地计算 $f(s_u|t_v)$, 使其最终收敛到最优值。EM 方法的思想是: 用当前的参数, 求似然函数的期望, 之后最大化这个期望同时得到新的一组参数的值。对于 IBM 模型来说, 其迭代过程就是反复使用公式3.40, 具体如图3.24所示。

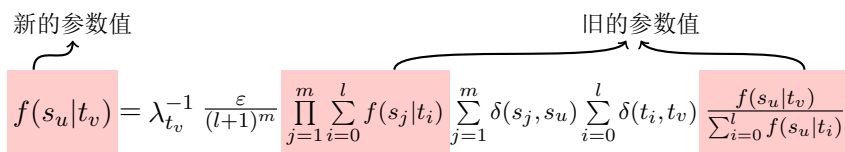


图 3.24: IBM 模型迭代过程示意图

为了化简 $f(s_u|t_v)$ 的计算, 在此对公式3.40进行了重新组织, 见图3.25。其中, 红色部分表示翻译概率 $P(\mathbf{s}|\mathbf{t})$; 蓝色部分表示 (s_u, t_v) 在句对 (\mathbf{s}, \mathbf{t}) 中配对的总次数, 即“ t_v 翻译为 s_u ”在所有对齐中出现的次数; 绿色部分表示 $f(s_u|t_v)$ 对于所有的 t_i 的相对值, 即“ t_v 翻译为 s_u ”在所有对齐中出现的相对概率; 蓝色与绿色部分相乘表示“ t_v 翻译为 s_u ”这个事件出现次数的期望的估计, 称之为**期望频次**(Expected Count)。

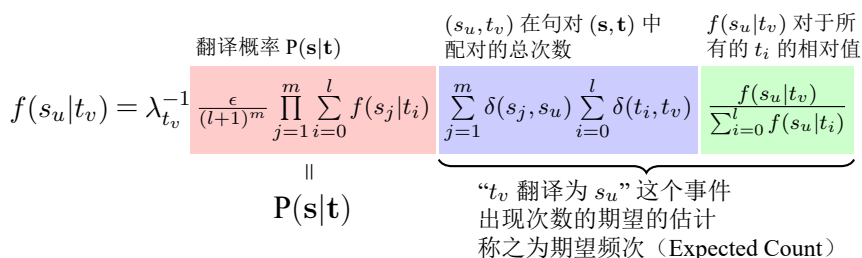


图 3.25: 公式3.40的解释

期望频次是事件在其分布下出现次数的期望。另 $c_{\mathbb{E}}(X)$ 为事件 X 的期望频次，其计算公式为：

$$c_{\mathbb{E}}(X) = \sum_i c(x_i) \cdot P(x_i) \quad (3.41)$$

其中 $c(x_i)$ 表示 X 取 x_i 时出现的次数， $P(x_i)$ 表示 $X = x_i$ 出现的概率。图3.26展示了事件 X 的期望频次的详细计算过程。其中 x_1 、 x_2 和 x_3 分别表示事件 X 出现 2 次、1 次和 5 次的情况。

x_i	$c(x_i)$	x_i	$c(x_i)$	$P(x_i)$	$c(x_i) \cdot P(x_i)$
x_1	2	x_1	2	0.1	0.2
x_2	1	x_2	1	0.3	0.3
x_3	5	x_3	5	0.2	1.0
$c_{\mathbb{E}}(X) = 8$		$c_{\mathbb{E}}(X) = 0.2 + 0.3 + 1.0 = 1.5$			

图 3.26: 频次（左）和期望频次（右）的计算过程

因为在 $P(\mathbf{s}|\mathbf{t})$ 中， t_v 翻译（连接）到 s_u 的期望频次为：

$$c_{\mathbb{E}}(s_u|t_v; \mathbf{s}, \mathbf{t}) \equiv \sum_{j=1}^m \delta(s_j, s_u) \sum_{i=0}^l \delta(t_i, t_v) \cdot \frac{f(s_u|t_v)}{\sum_{i=0}^l f(s_u|t_i)} \quad (3.42)$$

所以公式3.40可重写为：

$$f(s_u|t_v) = \lambda_{t_v}^{-1} \cdot P(\mathbf{s}|\mathbf{t}) \cdot c_{\mathbb{E}}(s_u|t_v; \mathbf{s}, \mathbf{t}) \quad (3.43)$$

在此如果令 $\lambda'_{t_v} = \frac{\lambda_{t_v}}{P(\mathbf{s}|\mathbf{t})}$ ，可得：

$$\begin{aligned} f(s_u|t_v) &= \lambda_{t_v}^{-1} \cdot P(\mathbf{s}|\mathbf{t}) \cdot c_{\mathbb{E}}(s_u|t_v; \mathbf{s}, \mathbf{t}) \\ &= (\lambda'_{t_v})^{-1} \cdot c_{\mathbb{E}}(s_u|t_v; \mathbf{s}, \mathbf{t}) \end{aligned} \quad (3.44)$$

又因为 IBM 模型对 $f(\cdot)$ 的约束如下：

$$\forall t_y : \sum_{s_x} f(s_x|t_y) = 1 \quad (3.45)$$

为了满足 $f(\cdot)$ 的概率归一化约束，易知 λ'_{t_v} 为：

$$\lambda'_{t_v} = \sum_{s_u} c_{\mathbb{E}}(s_u|t_v; \mathbf{s}, \mathbf{t}) \quad (3.46)$$

因此, $f(s_u|t_v)$ 的计算式可再一步变换成下式:

$$f(s_u|t_v) = \frac{c_{\mathbb{E}}(s_u|t_v; \mathbf{s}, \mathbf{t})}{\sum_{s_u} c_{\mathbb{E}}(s_u|t_v; \mathbf{s}, \mathbf{t})} \quad (3.47)$$

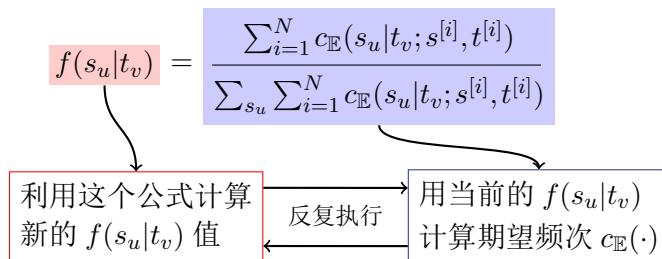


图 3.27: $f(s_u|t_v)$ 的计算公式和迭代过程

IBM 模型 1 的训练 (EM 算法)

输入: 平行语料 $(\mathbf{s}^{[1]}, \mathbf{t}^{[1]}), \dots, (\mathbf{s}^{[N]}, \mathbf{t}^{[N]})$

输出: 参数 $f(\cdot|\cdot)$ 的最优值

- 1: **Function** EM($\{(\mathbf{s}^{[1]}, \mathbf{t}^{[1]}), \dots, (\mathbf{s}^{[N]}, \mathbf{t}^{[N]})\}$)
- 2: Initialize $f(\cdot|\cdot)$ ▷ 比如给 $f(\cdot|\cdot)$ 一个均匀分布
- 3: Loop until $f(\cdot|\cdot)$ converges
- 4: **foreach** $k = 1$ to N **do**
- 5: $c_{\mathbb{E}}(s_u|t_v; \mathbf{s}^{[k]}, \mathbf{t}^{[k]}) = \sum_{j=1}^{|\mathbf{s}^{[k]}|} \delta(s_j, s_u) \sum_{i=0}^{|\mathbf{t}^{[k]}|} \delta(t_i, t_v) \cdot \frac{f(s_u|t_v)}{\sum_{i=0}^t f(s_u|t_i)}$
- 6: **foreach** t_v appears at least one of $\{\mathbf{t}^{[1]}, \dots, \mathbf{t}^{[N]}\}$ **do**
- 7: $\lambda'_{t_v} = \sum_{s_u} \sum_{k=1}^N c_{\mathbb{E}}(s_u|t_v; \mathbf{s}^{[k]}, \mathbf{t}^{[k]})$
- 8: **foreach** s_u appears at least one of $\{\mathbf{s}^{[1]}, \dots, \mathbf{s}^{[N]}\}$ **do**
- 9: $f(s_u|t_v) = \sum_{k=1}^N c_{\mathbb{E}}(s_u|t_v; \mathbf{s}^{[k]}, \mathbf{t}^{[k]}) \cdot (\lambda'_{t_v})^{-1}$
- 10: **return** $f(\cdot|\cdot)$

图 3.28: EM 算法流程图 (IBM 模型 1)

进一步, 假设有 N 个互译的句对 (称作平行语料): $\{(\mathbf{s}^{[1]}, \mathbf{t}^{[1]}), \dots, (\mathbf{s}^{[N]}, \mathbf{t}^{[N]})\}$, $f(s_u|t_v)$ 的期望频次为:

$$c_{\mathbb{E}}(s_u|t_v) = \sum_{i=1}^N c_{\mathbb{E}}(s_u|t_v; s^{[i]}, t^{[i]}) \quad (3.48)$$

于是有 $f(s_u|t_v)$ 的计算公式和迭代过程图3.27所示。完整的 EM 算法如图3.28所示。其中 E-Step 对应 4-5 行，目的是计算 $c_E(\cdot)$ ；M-Step 对应 6-9 行，目的是计算 $f(\cdot)$ 。

同样的，EM 算法可以直接用于训练 IBM 模型 2。对于句对 (\mathbf{s}, \mathbf{t}) ， $m = |\mathbf{s}|$ ， $l = |\mathbf{t}|$ ，E-Step 的计算公式如下，其中参数 $f(s_j|t_i)$ 与 IBM 模型 1 一样：

$$c_E(s_u|t_v; \mathbf{s}, \mathbf{t}) = \sum_{j=1}^m \sum_{i=0}^l \frac{f(s_u|t_v) a(i|j, m, l) \delta(s_j, s_u) \delta(t_i, t_v)}{\sum_{k=0}^l f(s_u|t_k) a(k|j, m, l)} \quad (3.49)$$

$$c_E(i|j, m, l; \mathbf{s}, \mathbf{t}) = \frac{f(s_j|t_i) a(i|j, m, l)}{\sum_{k=0}^l f(s_j|t_k) a(k, j, m, l)} \quad (3.50)$$

M-Step 的计算公式如下，其中参数 $a(i|j, m, l)$ 表示调序概率：

$$f(s_u|t_v) = \frac{\sum_{k=0}^K c_E(s_u|t_v; \mathbf{s}^{[k]}, \mathbf{t}^{[k]})}{\sum_{s_u} \sum_{k=0}^K c_E(s_u|t_v; \mathbf{s}^{[k]}, \mathbf{t}^{[k]})} \quad (3.51)$$

$$a(i|j, m, l) = \frac{\sum_{k=0}^K c_E(i|j; \mathbf{s}^{[k]}, \mathbf{t}^{[k]})}{\sum_i \sum_{k=0}^K c_E(i|j; \mathbf{s}^{[k]}, \mathbf{t}^{[k]})} \quad (3.52)$$

3.5 IBM 模型 3-5 及隐马尔可夫模型

本节在 IBM 模型 1-2 的基础上继续介绍 IBM 模型 3-5，这些模型采用了更细致的建模方式来描述翻译问题，包括引入产出率、单词的抽象等重要方法。此外，本节也会介绍隐马尔可夫模型，它和 IBM 模型有一定联系，但是从另一个视角看待翻译问题。

3.5.1 基于产出率的翻译模型

从前面的介绍可知，IBM 模型 1 和模型 2 把不同的源语言单词看作相互独立的单元来进行词对齐和翻译。换句话说，即使某个源语言短语中的两个单词都对齐到同一个目标语单词，它们之间也是相互独立的。这样模型 1 和模型 2 对于多个源语言单词对齐到同一个目标语单词的情况并不能很好地进行描述。

这里将会给出另一个翻译模型，能在一定程度上解决上面提到的问题。该模型把译文生成源文的过程分解为如下几个步骤：首先，确定每个目标语言单词生成源语言单词的个数，这里把它称为**产出率**或**繁衍率**（Fertility）；其次，决定译文中每个单词生成的源语言单词都是什么，即决定生成的第一个源语言单词是什么，生成的第二个源语言单词是什么，以此类推。这样每个目标语单词就对应了一个源语言单词列表；最后把各组源语言单词列表中的每个单词都放置到合适的位置上，完成目标语言译文到源语言句子的生成。

对于句对 (\mathbf{s}, \mathbf{t}) ，令 φ 表示产出率，同时令 τ 表示每个目标语单词对应的源语言单词列表。图3.29描述了一个英文句子生成中文句子的过程。首先，对于每个英语单词 t_i 决定它的产出率 φ_i 。比如“Scientists”的产出率是 2，可表示为 $\varphi_1 = 2$ 。这表明它会生成 2 个中文单词；其次，确定英文句子中每个单词生成的中文单词列表。比如“Scientists”

生成“科学家”和“们”两个中文单词，可表示为 $\tau_1 = \{\tau_{11} = \text{“科学家”}, \tau_{12} = \text{“们”}\}$ 。这里用特殊的空标记 NULL 表示翻译对空的情况；最后，把生成的所有中文单词放在合适的位置。比如“科学家”和“们”分别放在 s 的位置 1 和位置 2。可以用符号 π 记录生成的单词在源语言句子 s 中的位置。比如“Scientists”生成的中文单词在 s 中的位置表示为 $\pi_1 = \{\pi_{11} = 1, \pi_{12} = 2\}$ 。

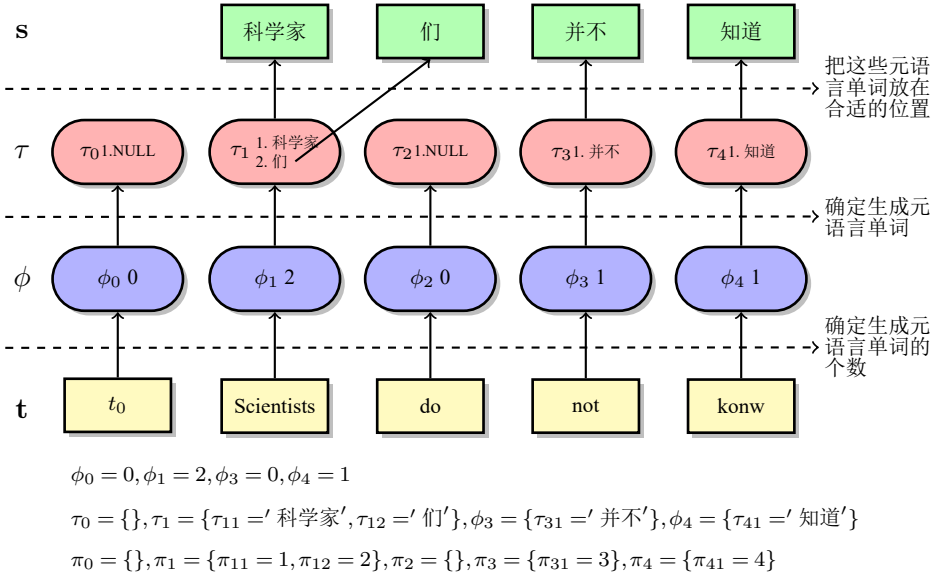


图 3.29: 基于产出率的翻译模型执行过程

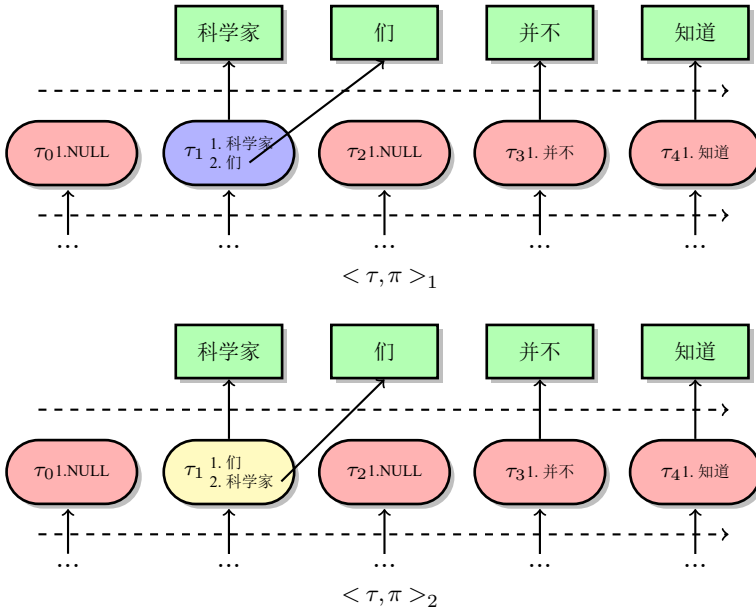
为了表述清晰，我们重新说明每个符号的含义。 s 、 t 、 m 和 l 分别表示源语言句子、目标语译文、源语言单词数量以及译文单词数量。 φ 、 τ 和 π 分别记录产出率、生成的源语言单词以及它们在源文中的位置。 φ_i 表示第 i 个译文单词 t_i 的产出率。 τ_i 和 π_i 分别表示 t_i 生成的源语言单词列表及其在源语言句子 s 中的位置列表。

可以看出，一组 τ 和 π （记为 $\langle \tau, \pi \rangle$ ）可以决定一个对齐 a 和一个源语句子 s 。相反的，一个对齐 a 和一个源语句子 s 可以对应多组 $\langle \tau, \pi \rangle$ 。如图 3.30 所示，不同的 $\langle \tau, \pi \rangle$ 对应同一个源语言句子和词对齐。它们的区别在于目标语单词“Scientists”生成的源语言单词“科学家”和“们”的顺序不同。这里把不同的 $\langle \tau, \pi \rangle$ 对应到的相同的源语句子 s 和对齐 a 记为 $\langle s, a \rangle$ 。因此计算 $P(s, a | t)$ 时需要把每个可能结果的概率加起来，如下：

$$P(s, a | t) = \sum_{\langle \tau, \pi \rangle \in \langle s, a \rangle} P(\tau, \pi | t) \quad (3.53)$$

不过 $\langle s, a \rangle$ 中有多少个元素呢？通过图 3.29 中的例子，可以推出 $\langle s, a \rangle$ 应该包含 $\prod_{i=0}^l \varphi_i!$ 个不同的二元组 $\langle \tau, \pi \rangle$ 。这是因为在给定源语言句子和词对齐时，对于每一个 τ_i 都有 $\varphi_i!$ 种排列。

进一步， $P(\tau, \pi | t)$ 可以被表示如图 3.31 的形式。其中 τ_{i1}^{k-1} 表示 $\tau_{i1} \tau_{i2} \cdots \tau_{i(k-1)}$ ， π_{i1}^{k-1} 表示 $\pi_{i1} \pi_{i2} \cdots \pi_{i(k-1)}$ 。可以把图 3.31 中的公式分为 5 个部分，并用不同的序号

图 3.30: 不同 τ 和 π 对应相同的源语言句子和词对齐的情况

和颜色进行标注。每部分的具体含义是：

- 对每个 $i \in [1, l]$ 的目标语单词的产出率建模（红色），即 φ_i 的概率。它依赖于 \mathbf{t} 和区间 $[1, i-1]$ 的目标语单词的产出率 φ_1^{i-1} 。⁶
- $i=0$ 时的产出率建模（蓝色），即空标记 t_0 的产出率的概率。它依赖于 \mathbf{t} 和区间 $[1, i-1]$ 的目标语单词的产出率 φ_1^l 。
- 词汇翻译建模（绿色），目标语言单词 t_i 生成第 k 个源语言单词 τ_{ik} 时的概率，依赖于 \mathbf{t} 、所有目标语言单词的产出率 φ_0^l 、区间 $i \in [1, l]$ 的目标语言单词生成的源语言单词 τ_1^{i-1} 和目标语单词 t_i 生成的前 k 个源语言单词 τ_{i1}^{k-1} 。
- 对于每个 $i \in [1, l]$ 的目标语言单词生成的源语言单词的**扭曲度**（Distortion）建模（黄色），即第 i 个译文单词生成的第 k 个源语言单词在源文中的位置 π_{ik} 的概率。其中 π_1^{i-1} 和 π_{i1}^{k-1} 分别表示区间 $[1, i-1]$ 的目标语言单词生成的源语言单词的扭曲度和第 i 译文单词生成的前 k 个源语言单词的扭曲度。
- $i=0$ 时的扭曲度建模（灰色），即空标记 t_0 生成的源语言单词在源语言句子中位置的概率。

3.5.2 IBM 模型 3

IBM 模型 3 通过一些假设对图3.31所表示的基本模型进行了化简。具体来说，对于每个 $i \in [1, l]$ ，假设 $P(\varphi_i | \varphi_1^{i-1}, \mathbf{t})$ 仅依赖于 φ_i 和 t_i ， $P(\pi_{ik} | \pi_{i1}^{k-1}, \pi_1^{i-1}, \tau_0^l, \varphi_0^l, \mathbf{t})$ 仅依

⁶这里约定，当 $i=1$ 时， φ_1^0 表示空。

依赖于 π_{ik} 、 i 、 m 和 l 。而对于所有的 $i \in [0, l]$ ，假设 $P(\tau_{ik} | \tau_{i1}^{k-1}, \tau_1^{i-1}, \phi_0^l, \mathbf{t})$ 仅依赖于 τ_{ik} 和 t_i 。形式化这些假设，可以得到：

$$\begin{aligned} P(\tau, \pi | \mathbf{t}) = & \prod_{i=1}^l P(\varphi_i | \varphi_1^{i-1}, \mathbf{t}) \times P(\varphi_0 | \varphi_1^l, \mathbf{t}) \times \\ & \prod_{i=0}^l \prod_{k=1}^{\varphi_i} P(\tau_{ik} | \tau_{i1}^{k-1}, \tau_1^{i-1}, \varphi_0^l, \mathbf{t}) \times \\ & \prod_{i=1}^l \prod_{k=1}^{\varphi_i} P(\pi_{ik} | \pi_{i1}^{k-1}, \pi_1^{i-1}, \tau_0^l, \varphi_0^l, \mathbf{t}) \times \\ & \prod_{k=1}^{\varphi_0} P(\pi_{0k} | \pi_{01}^{k-1}, \pi_1^l, \tau_0^l, \varphi_0^l, \mathbf{t}) \end{aligned}$$

图 3.31: $P(\tau, \pi | \mathbf{t})$ 的详细表达式

$$P(\varphi_i | \varphi_1^{i-1}, \mathbf{t}) = P(\varphi_i | t_i) \quad (3.54)$$

$$P(\tau_{ik} = s_j | \tau_{i1}^{k-1}, \tau_1^{i-1}, \varphi_0^l, \mathbf{t}) = t(s_j | t_i) \quad (3.55)$$

$$P(\pi_{ik} = j | \pi_{i1}^{k-1}, \pi_1^{i-1}, \tau_0^l, \varphi_0^l, \mathbf{t}) = d(j | i, m, l) \quad (3.56)$$

通常把 $d(j | i, m, l)$ 称为扭曲度函数。这里 $P(\varphi_i | \varphi_1^{i-1}, \mathbf{t}) = P(\varphi_i | t_i)$ 和 $P(\pi_{ik} = j | \pi_{i1}^{k-1}, \pi_1^{i-1}, \tau_0^l, \varphi_0^l, \mathbf{t}) = d(j | i, m, l)$ 仅对 $1 \leq i \leq l$ 成立。这样就完成了图3.31中第 1、3 和 4 部分的建模。

对于 $i = 0$ 的情况需要单独进行考虑。实际上， t_0 只是一个虚拟的单词。它要对应 \mathbf{s} 中原本为空对齐的单词。这里假设要等其他非空对应单词都被生成（放置）后，才考虑这些空对齐单词的生成（放置）。即非空对单词都被生成后，在那些还有空的位置上放置这些空对的源语单词。此外，在任何的空位置上放置空对的源语单词都是等概率的，即放置空对齐源语言单词服从均匀分布。这样在已经放置了 k 个空对齐源语言单词的时候，应该还有 $\varphi_0 - k$ 个空位置。如果第 i 个位置为空，那么 $P(\pi_{0k} = i | \pi_{01}^{k-1}, \pi_1^l, \tau_0^l, \varphi_0^l, \mathbf{t}) = \frac{1}{\varphi_0 - k}$ ，否则 $P(\pi_{0k} = i | \pi_{01}^{k-1}, \pi_1^l, \tau_0^l, \varphi_0^l, \mathbf{t}) = 0$ 。这样对于 t_0 所对应的 τ_0 ，就有

$$\prod_{k=1}^{\varphi_0} P(\pi_{0k} | \pi_{01}^{k-1}, \pi_1^l, \tau_0^l, \varphi_0^l, \mathbf{t}) = \frac{1}{\varphi_0!} \quad (3.57)$$

而上面提到的 t_0 所对应的这些空位置是如何生成的呢？即如何确定哪些位置是要放置空对齐的源语言单词。在 IBM 模型 3 中，假设在所有的非空对齐源语言单词都被生成出来后（共 $\varphi_1 + \varphi_2 + \dots + \varphi_l$ 个非空对源语单词），这些单词后面都以 p_1 概率随机地产生一个“槽”用来放置空对齐单词。这样， φ_0 就服从了一个二项分布。于是得到

$$P(\varphi_0 | \mathbf{t}) = \binom{\varphi_1 + \varphi_2 + \dots + \varphi_l}{\varphi_0} p_0^{\varphi_1 + \varphi_2 + \dots + \varphi_l - \varphi_0} p_1^{\varphi_0} \quad (3.58)$$

其中, $p_0 + p_1 = 1$ 。到此为止, 我们完成了图3.31中第2和5部分的建模。最终根据这些假设可以得到 $P(s|t)$ 的形式:

$$P(s|t) = \sum_{a_1=0}^l \cdots \sum_{a_m=0}^l \left[\binom{m-\varphi_0}{\varphi_0} p_0^{m-2\varphi_0} p_1^{\varphi_0} \prod_{i=1}^l \varphi_i! n(\varphi_i|t_i) \right. \\ \left. \times \prod_{j=1}^m t(s_j|t_{a_j}) \times \prod_{j=1, a_j \neq 0}^m d(j|a_j, m, l) \right] \quad (3.59)$$

其中, $n(\varphi_i|t_i) = P(\varphi_i|t_i)$ 表示产出率的分布。这里的约束条件为,

$$\sum_s t(s|t) = 1 \quad (3.60)$$

$$\sum_j d(j|i, m, l) = 1 \quad (3.61)$$

$$\sum_{\varphi} n(\varphi|t) = 1 \quad (3.62)$$

$$p_0 + p_1 = 1 \quad (3.63)$$

3.5.3 IBM 模型 4

IBM 模型 3 仍然存在问题, 比如, 它不能很好地处理一个目标语言单词生成多个源语言单词的情况。这个问题在模型 1 和模型 2 中也存在。如果一个目标语言单词对应多个源语言单词, 往往这些源语言单词构成短语或搭配。但是模型 1-3 把这些源语言单词看成独立的单元, 而实际上它们是一个整体。这就造成了在模型 1-3 中这些源语言单词可能会“分散”开。为了解决这个问题, 模型 4 对模型 3 进行了进一步修正。

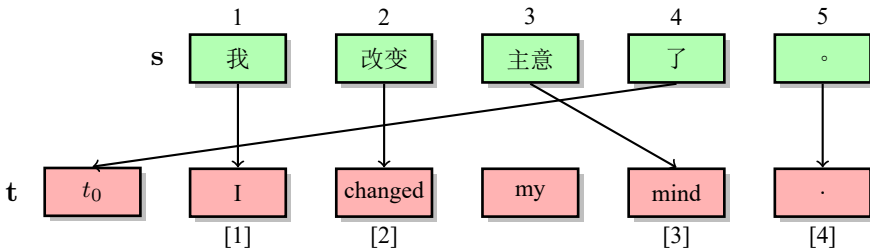


图 3.32: 词对齐的汉译英句对及独立单词 cept. 的位置

为了更清楚的阐述, 这里引入新的术语——**概念单元**或**概念** (Concept)。词对齐可以被看作概念之间的对应。这里的概念是指具有独立语法或语义功能的一组单词。依照 Brown 等人的表示方法 [25], 可以把概念记为 cept.。每个句子都可以被表示成

一系列的 cept.。这里要注意的是，源语言句子中的 cept. 数量不一定等于目标句子中的 cept. 数量。因为有些 cept. 可以为空，因此可以把那些空对的单词看作空 cept.。比如，在图3.32的实例中，“了”就对应一个空 cept.。

在 IBM 模型的词对齐框架下，目标语的 cept. 只能是那些非空对齐的目标语单词，而且每个 cept. 只能由一个目标语单词组成（通常把这类由一个单词组成的 cept. 称为独立单词 cept.）。这里用 $[i]$ 表示第 i 个独立单词 cept. 在目标语言句子中的位置。换句话说， $[i]$ 表示第 i 个非空对的目标语单词的位置。比如在本例中“mind”在 \mathbf{t} 中的位置表示为 $[3]$ 。

另外，可以用 \odot_i 表示位置为 $[i]$ 的目标语言单词对应的那些源语言单词位置的平均值，如果这个平均值不是整数则对它向上取整。比如在本例中，目标语句中第 4 个 cept. (“.”) 对应源语言句子中的第 5 个输出值。可表示为 $\odot_4 = 5$ 。

利用这些新引进的概念，模型 4 对模型 3 的扭曲度进行了修改。主要是把扭曲度分解为两类参数。对于 $[i]$ 对应的源语言单词列表 $(\tau_{[i]})$ 中的第一个单词 $(\tau_{[i]1})$ ，它的扭曲度用如下公式计算：

$$P(\pi_{[i]1} = j | \pi_1^{[i]-1}, \tau_0^l, \varphi_0^l, \mathbf{t}) = d_1(j - \odot_{i-1} | A(t_{[i-1]}), B(s_j)) \quad (3.64)$$

其中，译文的第 i 个单词生成的第 k 个源语单词在源语言句子中的位置用变量 π_{ik} 表示。而对于列表 $(\tau_{[i]})$ 中的其他的单词 $(\tau_{[i]k}, 1 < k \leq \varphi[i])$ 的扭曲度计算，进行如下计算

$$P(\pi_{[i]k} = j | \pi_{[i]1}^{k-1}, \pi_1^{[i]-1}, \tau_0^l, \varphi_0^l, \mathbf{t}) = d_{>1}(j - \pi_{[i]k-1} | B(s_j)) \quad (3.65)$$

这里的函数 $A(\cdot)$ 和函数 $B(\cdot)$ 分别把目标语言和源语言的单词影射到单词的词类。这么做的目的的一方面要减小参数空间的大小，另一方面是要减小数据的稀疏程度。词类信息通常可以通过外部工具得到，比如 Brown 聚类等。另一种简单的方法是把单词直接映射为它的词性。这样可以直接用现在已经非常成熟的词性标注工具解决问题。

从上面改进的扭曲度模型可以看出，对于 $t_{[i]}$ 生成的第一个源语言单词，要考虑中心 $\odot_{[i]}$ 和这个源语言单词之间的绝对距离。实际上也就要把 $t_{[i]}$ 生成的所有源语言单词看成一个整体并把它放置在合适的位置。这个过程要依据第一个源语言单词的词类和对应源语中心位置，和前一个非空对目标语言单词 $t_{[i-1]}$ 的词类。而对于 $t_{[i]}$ 生成的其他源语言单词，只需要考虑它与前一个刚放置完的源语言单词的相对位置 and 这个源语言单词的词类。

实际上，上述过程就要先用 $t_{[i]}$ 生成的第一个源语言单词代表整个 $t_{[i]}$ 生成的单词列表，并把第一个源语言单词放置在合适的位置。然后，相对于前一个刚生成的源语言单词，把列表中的其他单词放置在合适的地方。这样就可以在一定程度上保证由同一个目标语言单词生成的源语言单词之间可以相互影响，达到了改进的目的。

3.5.4 IBM 模型 5

模型 3 和模型 4 并不是“准确”的模型。这两个模型会把一部分概率分配给一些根本就不存在的句子。这个问题被称作 IBM 模型 3 和模型 4 的**缺陷**（Deficiency）。说的具体一些，模型 3 和模型 4 中并没有这样的约束：如果已经放置了某个源语言单词的位置不能再放置其他单词，也就是说句子的任何位置只能放置一个词，不能多也不能少。由于缺乏这个约束，模型 3 和模型 4 中在所有合法的词对齐上概率和并不等于 1。这部分缺失的概率被分配到其他不合法的词对齐上。举例来说，如图 3.33 所示，“吃早饭”和“have breakfast”之间的合法词对齐用直线表示。但是在模型 3 和模型 4 中，在它们上的概率和为 $0.9 < 1$ 。损失掉的概率被分配到像 5 和 6 这样的对齐上了（红色）。虽然 IBM 模型并不支持一对多的对齐，但是模型 3 和模型 4 把概率分配给这些“不合法”的词对齐上，因此也就产生所谓的 Deficiency 问题。

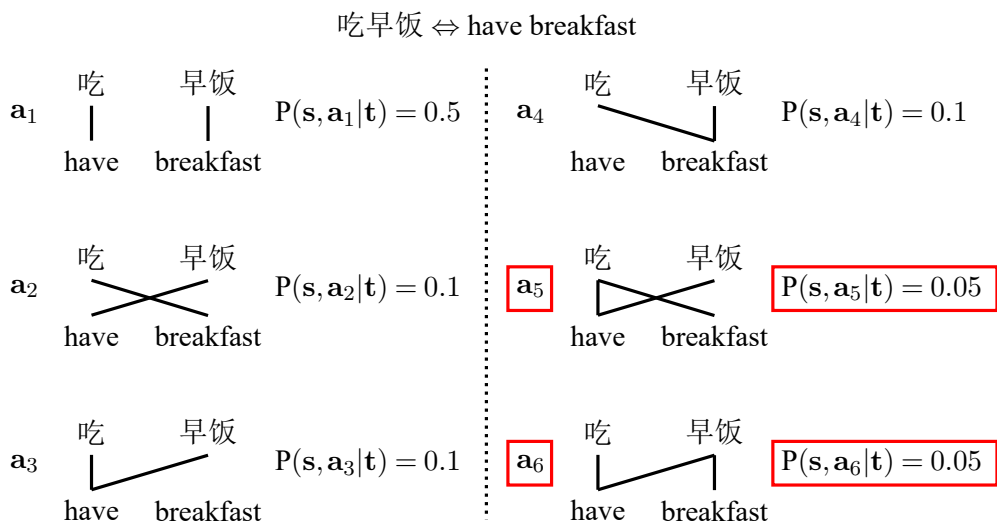


图 3.33: IBM 模型 3 的词对齐及概率分配

为了解决这个问题，模型 5 在模型中增加了额外的约束。基本想法是，在放置一个源语言单词的时候检查这个位置是否已经放置了单词，如果可以则把这个放置过程赋予一定的概率，否则把它作为不可能事件。依据这个想法，就需要在逐个放置源语言单词的时候判断源语言句子的哪些位置为空。这里引入一个变量 $v(j, \tau_1^{[i]-1}, \tau_{[i]1}^{k-1})$ ，它表示在放置 $\tau_{[i]k}$ 之前（ $\tau_1^{[i]-1}$ 和 $\tau_{[i]1}^{k-1}$ 已经被放置完了），从源语言句子的第一个位置到位置 j （包含 j ）为止还有多少个空位置。这里，把这个变量简写为 v_j 。于是，对于 $[i]$ 所对应的源语言单词列表 $(\tau_{[i]})$ 中的第一个单词 $(\tau_{[i]1})$ ，有：

$$P(\pi_{[i]1} = j | \pi_1^{[i]-1}, \tau_0^l, \varphi_0^l, t) = d_1(v_j | B(s_j), v_{\odot_{i-1}}, v_m - (\varphi_{[i]} - 1)) \cdot (1 - \delta(v_j, v_{j-1})) \quad (3.66)$$

对于其他单词 $(\tau_{[i]k}, 1 < k \leq \varphi_{[i]})$, 有:

$$\begin{aligned} & P(\pi_{[i]k} = j | \pi_{[i]1}^{k-1}, \pi_1^{[i]-1}, \tau_0^l, \varphi_0^l, \mathbf{t}) \\ &= d_{>1}(v_j - v_{\pi_{[i]k-1}} | B(s_j), v_m - v_{\pi_{[i]k-1}} - \varphi_{[i]} + k) \cdot (1 - \delta(v_j, v_{j-1})) \quad (3.67) \end{aligned}$$

这里, 因子 $1 - \delta(v_i, v_{i-1})$ 是用来判断第 i 个位置是不是为空。如果第 i 个位置为空则 $v_i = v_{i-1}$, 这样 $P(\pi_{[i]1} = i | \pi_1^{[i]-1}, \tau_0^l, \varphi_0^l, \mathbf{t}) = 0$ 。这样就从模型上避免了模型 3 和模型 4 中生成不存在的字符串的问题。这里还要注意的, 对于放置第一个单词的情况, 影响放置的因素有 v_i , $B(s_i)$ 和 v_{i-1} 。此外还要考虑在 i 位置放置了第一个单词以后它的右边是不是还有足够的位置留给剩下的 $k-1$ 个单词。参数 $v_m - (\varphi_{[i]} - 1)$ 正是为了考虑这个因素, 这里 v_m 表示整个源语言句子中还有多少空位置, $\varphi_{[i]} - 1$ 表示 i 位置右边至少还要留出的空格数。对于放置非第一个单词的情况, 主要是要考虑它和前一个放置位置的相对位置。这主要体现在参数 $v_i - v_{\pi_{[i]k-1}}$ 上。式 3.67 的其他部分都可以用上面的理论解释, 这里不再赘述。

实际上, 模型 5 和模型 4 的思想基本一致, 即, 先确定 $\tau_{[i]1}$ 的绝对位置, 然后再确定 $\tau_{[i]}$ 中剩余单词的相对位置。模型 5 消除了产生不存在的句子的可能性, 不过模型 5 的复杂性也大大增加了。

3.5.5 隐马尔可夫模型

IBM 模型可以得到双语句子间的词对齐, 因此也有很多工作在这个模型的基础上对词对齐方法进行改进。其中一个比较有代表性的工作是基于隐马尔可夫模型的方法 [292], 它可以被看作是 IBM 模型 2 的升级版本。

隐马尔可夫模型

隐马尔可夫模型 (Hidden Markov Model, HMM) 是经典的机器学习模型, 它在语音识别、自然语言处理等领域得到了非常广泛的应用。其本质是一个概率模型, 用来描述一个含有隐含参数的马尔可夫过程, 简单来说, 是用来描述一个系统, 它隐含状态的转移和可见状态的概率⁷。

我们用一个简单的例子来对这些概念进行说明。假设有三枚质地不同的硬币 A、B、C, 这三个硬币抛出正面的概率分别为 0.3、0.5、0.7。之后开始抛硬币, 随机从三个硬币里挑一个, 挑到每一个硬币的概率都是 1/3。不停的重复上述过程, 会得到一串硬币的正反序列, 如: 抛硬币 6 次, 得到: 正正反反正反。

这个正反序列叫做可见状态链, 由每个回合的可见状态构成。此外, HMM 模型还有一串隐含状态链, 在这里, 隐含状态链就是所用硬币的序列, 比如可能是: C B A B C A。同样的, HMM 模型还会描述系统隐藏状态的转移概率, 在本例子中, A 的下一个状态是 A、B、C 的概率都是 1/3。B、C 的下一个状态是 A、B、C 的转移概率

⁷<https://zh.wikipedia.org/zh-hans/隐马尔可夫模型>

也同样是 1/3。同样的，尽管可见状态链之间没有转移概率，但是隐含状态和可见状态之间存在着输出概率，即 A、B、C 抛出正面的输出概率为 0.3、0.5、0.7。图3.34描述了这个例子所对应的的隐马尔可夫模型示意图。

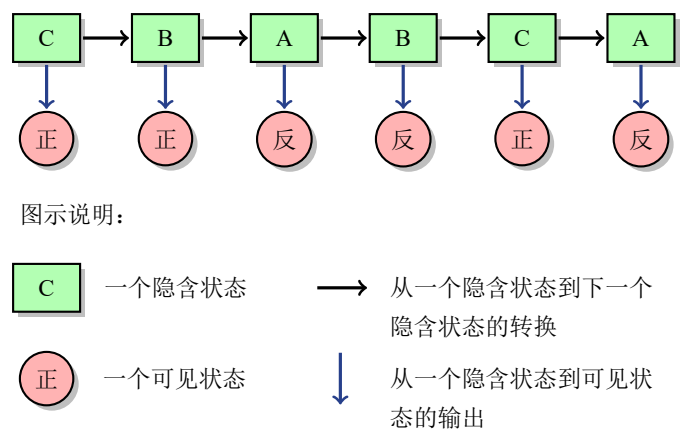


图 3.34: 抛硬币的隐马尔可夫模型实例

一般来说，HMM 包含下面三个问题 [192]：

- 估计：即给定模型（硬币种类和转移概率），根据可见状态链（抛硬币的结果），计算在该模型下得到这个结果的概率，这个问题的解决需要用到前后向算法。
- 参数学习：即给定硬币种类（隐含状态数量），根据多个可见状态链（抛硬币的结果），估计模型的参数（转移概率），同 IBM 模型的参数训练一样，这个问题的求解需要用到 EM 算法。
- 解码问题：即给定模型（硬币种类和转移概率）和可见状态链（抛硬币的结果），计算在可见状态链的情况下，最可能出现的对应的状态序列，这个问题的求解需要用到基于动态规划方法，在 HMM 中被称作维特比算法（Viterbi Algorithm）。

词对齐模型

IBM 模型把翻译问题定义为对译文和词对齐同时进行生成的问题，模型翻译质量的好坏与词对齐有着非常紧密的联系。IBM 模型 1 假设对齐概率仅依赖于译文长度，即对齐概率服从均匀分布；IBM 模型 2 假设对齐概率与源语言、目标语言的句子长度以及源语言位置和目标语言位置相关。IBM 模型 2 已经覆盖到了大部分的词对齐问题，但是该模型只考虑到了词语的绝对位置，并未考虑到相邻词语间的关系。图3.35展示了一个简单的实例，可以看到的是，汉语的每个词都被分配给了英语句子中的每一个单词，但是词语并不是任意分布在各个位置上的，而是倾向于生成簇。也就是说，如果源语言的两个词位置越近，它们的目标词在目标语言句子的位置也越近。

因此，基于 HMM 的词对齐模型抛弃了 IBM 模型 1-2 的绝对位置假设，将一阶

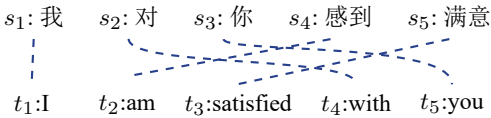


图 3.35: 汉译英句对及对齐

隐马尔可夫模型用于单词对齐问题。HMM 词对齐模型认为，词语与词语之间并不是毫无联系的，对齐概率应该取决于对齐位置的差异而不是本身词语所在的位置。具体来说，位置 j 的对齐概率 a_j 与前一个位置 $j-1$ 的对齐位置 a_{j-1} 和译文长度 l 有关，形式化的表述为：

$$P(a_j|a_1^{j-1}, s_1^{j-1}, m, \mathbf{t}) = P(a_j|a_{j-1}, l) \tag{3.68}$$

这里用图3.34的例子对公式进行说明。在 IBM 模型 1-2 中，词语的对齐都是与单词所在的绝对位置有关。但在 HMM 词对齐模型中，“你”对齐到“you”被形式化为 $P(a_j|a_{j-1}, l) = P(5|4, 5)$ ，意思是对于源文位置 3($j = 3$)的词，如果它的目标译文是 5 个词，上一个对齐位置是 4($a_2 = 4$)，对齐到目标语译文位置 5($a_j = 5$)的概率是多少？理想的情况下，通过 $P(a_j|a_{j-1}, l)$ ，“你”对齐到“you”应该得到更高的概率，并且由于源语词“对”和“你”距离很近，因此其对应的对齐位置“with”和“you”的距离也应该很近。

因此，把公式3.23和3.68重新带入公式 1.19 和 1.18, 可得 HMM 词对齐模型的数学描述：

$$P(\mathbf{s}|\mathbf{t}) = \sum_{\mathbf{a}} P(m|\mathbf{t}) \prod_{j=1}^m P(a_j|a_{j-1}, l) f(s_j|t_{a_j}) \tag{3.69}$$

此外，为了使得 HMM 的对齐概率 $P(a_j|a_{j-1}, l)$ 满足归一化的条件，这里还假设其对齐概率只取决于 $a_j - a_{j-1}$ ，即：

$$P(a_j|a_{j-1}, l) = \frac{\mu(a_j - a_{j-1})}{\sum_{i=1}^l \mu(i - a_{j-1})} \tag{3.70}$$

其中， $\mu(\cdot)$ 是隐马尔可夫模型的参数，可以通过训练得到。

3.5.6 解码和训练

和 IBM 模型 1-2 一样，IBM 模型 3-5 和隐马尔可夫模型的解码可以直接使用3.2.4 节所描述的方法。基本思路是对译文自左向右生成，每次扩展一个源语言单词的翻译，即把源语言单词的译文放到已经生成的译文的右侧。每次扩展可以选择不同的源语言单词或者同一个源语言单词的不同翻译候选，这样就可以得到多个不同的扩展译文。在这个过程中，同时计算翻译模型和语言模型的得分，对每个得到译文候

选打分。最终，保留一个或者多个译文。这个过程重复执行直至所有源语言单词被翻译完。

类似的，IBM 模型 3-5 和隐马尔可夫模型也都可以使用期望最大化（EM）方法进行模型训练。相关数学推导可参考附录B的内容。通常，可以使用这些模型获得双语句子间的词对齐结果，比如著名的 GIZA++ 工具。这时，往往会使用多个模型，把简单的模型训练后的参数作为初始值送给后面更加复杂的模型。比如，先用 IBM 模型 1 训练，之后把参数送给 IBM 模型 2，再训练，之后把参数送给隐马尔可夫模型等。值得注意的是，并不是所有的模型使用 EM 算法都能找到全局最优解。特别是 IBM 模型 3-5 的训练中使用一些剪枝和近似的方法，优化的真实目标函数会更加复杂。不过，IBM 模型 1 是一个**凸函数**（Convex Function），因此理论上使用 EM 方法是能找到全局最优解的。更实际的好处是，IBM 模型 1 训练的最终结果与参数的初始化过程无关。这也是为什么在使用 IBM 系列模型时，往往会使用 IBM 模型 1 作为起始模型的原因。

3.6 问题分析

IBM 模型是一个时代的经典，但也留下了一些值得思考的问题。这一方面体现了科学技术发展需要一步步前行，而非简单的一蹴而就。另一方面也体现了机器翻译问题的困难程度。下面对 IBM 存在的问题进行分析，同时给出一些解决问题的思路，希望通过这些讨论可以使我们对机器翻译问题有更深层次的理解。

3.6.1 词对齐及对称化

IBM 的五个模型都是基于一个词对齐的假设——一个源语言单词最多只能对齐到一个目标语言单词。这个约束大大简化了 IBM 模型的建模。最初，Brown 等人提出这个假设可能是因为在法英翻译中一对多的对齐情况并不多见，这个假设带来的问题也不是那么严重。但是，在像汉英翻译这样的任务中，一个汉语单词对应多个英语单词的翻译很常见，这时 IBM 模型的词对齐假设就表现出了明显的问题。比如在翻译“我会试一试。”→“I will have a try.”时，IBM 模型根本不能把单词“试一试”对齐到三个单词“have a try”，因而可能无法得到正确的翻译结果。

本质上说，IBM 模型词对齐的“不完整”问题是 IBM 模型本身的缺陷。解决这个问题有很多思路，第一种方法就是，反向训练后，合并源语言单词，然后再正向训练。这里用汉英翻译为例来解释这个方法。首先反向训练，就是把英语当作待翻译语言，而把汉语当作目标语言进行训练（参数估计）。这样可以得到一个词对齐结果（参数估计的中间结果）。在这个词对齐结果里面，一个汉语单词可对应多个英语单词。之后，扫描每个英语句子，如果有多个英语单词对应同一个汉语单词，就把这些英语单词合并成一个英语单词。处理完之后，再把汉语当作源语言而把英语当作目标语言进行训练。这样就可以把一个汉语单词对应到合并的英语单词上。虽然从模型上看，还是一个汉语单词对应一个英语“单词”，但实质上已经把这个汉语单

词对应到多个英语单词上了。训练完之后，再利用这些参数进行翻译（解码）时，就能把一个中文单词翻译成多个英文单词了。但是反向训练后再训练也存在一些问题。首先，合并英语单词会使数据变得更稀疏，训练不充分。其次，由于 IBM 模型的词对齐结果并不是高精度的，利用它的词对齐结果来合并一些英文单词可能造成严重的错误，比如：把本来很独立的几个单词合在了一起。因此，此方法也并不完美。具体使用时还要考虑实际需要和问题的严重程度来决定是否使用这个方法。

另一种方法是双向对齐之后进行词对齐**对称化**（Symmetrization）。这个方法可以在 IBM 词对齐的基础上获得对称的词对齐结果。思路很简单，用正向（汉语为源语言，英语为目标语言）和反向（汉语为目标语言，英语为源语言）同时训练。这样可以得到两个词对齐结果。然后利用一些启发性方法用这两个词对齐生成对称的结果（比如，取“并集”、“交集”等），这样就可以得到包含一对多和多对多的词对齐结果。比如，在基于短语的统计机器翻译中已经很成功地使用了这种词对齐信息进行短语的获取。直到今天，对称化仍然是很多自然语言处理系统中的一个关键步骤。

3.6.2 Deficiency

Deficiency 问题是指翻译模型会把一部分概率分配给一些根本不存在的源语言字符串。如果用 $P(\text{well}|\mathbf{t})$ 表示 $P(\mathbf{s}|\mathbf{t})$ 在所有的正确的（可以理解为语法上正确的） \mathbf{s} 上的和，即

$$P(\text{well}|\mathbf{t}) = \sum_{\mathbf{s} \text{ is well formed}} P(\mathbf{s}|\mathbf{t}) \quad (3.71)$$

类似地，用 $P(\text{ill}|\mathbf{t})$ 表示 $P(\mathbf{s}|\mathbf{t})$ 在所有的错误的（可以理解为语法上错误的） \mathbf{s} 上的和。如果 $P(\text{well}|\mathbf{t}) + P(\text{ill}|\mathbf{t}) < 1$ ，就把剩余的部分定义为 $P(\text{failure}|\mathbf{t})$ 。它的形式化定义为，

$$P(\text{failure}|\mathbf{t}) = 1 - P(\text{well}|\mathbf{t}) - P(\text{ill}|\mathbf{t}) \quad (3.72)$$

本质上，模型 3 和模型 4 就是对应 $P(\text{failure}|\mathbf{t}) > 0$ 的情况。这部分概率是模型损失掉的。有时候也把这类 Deficiency 问题称为 Technical Deficiency。还有一种 Deficiency 问题被称作 Spiritually Deficiency，它是指 $P(\text{well}|\mathbf{t}) + P(\text{ill}|\mathbf{t}) = 1$ 且 $P(\text{ill}|\mathbf{t}) > 0$ 的情况。模型 1 和模型 2 就有 Spiritually Deficiency 的问题。可以注意到，Technical Deficiency 只存在于模型 3 和模型 4 中，模型 1 和模型 2 并没有 Technical Deficiency 问题。根本原因是模型 1 和模型 2 的词对齐是从源语言出发对应到目标语言， \mathbf{t} 到 \mathbf{s} 的翻译过程实际上是从单词 s_1 开始到单词 s_m 结束，依次把每个源语言单词 s_j 对应到唯一的一个目标语言位置。显然，这个过程能够保证每个源语言单词仅对应一个目标语言单词。但是，模型 3 和模型 4 中对齐是从目标语言出发对应到源语言， \mathbf{t} 到 \mathbf{s} 的翻译过程从 t_1 开始 t_l 结束，依次把目标语言单词 t_i 生成的单词对应到某个源语言位置上。但是这个过程不能保证 t_i 中生成的单词所对应的位置没有被其他已经完成对齐的目标语

单词所生成的某个源语言单词对应过，因此也就产生了 Deficiency 问题。

这里还要强调的是，Technical Deficiency 是模型 3 和模型 4 是模型本身的缺陷造成的，如果有一个“更好”的模型就可以完全避免这个问题。而 Spiritually Deficiency 几乎是不能从模型上根本解决的，因为对于任意一种语言都不能枚举所有的句子 ($P(\text{ill}|\text{t})$ 实际上是得不到的)。

IBM 的模型 5 已经解决了 Technical Deficiency 问题。不过模型 5 过于复杂。实际上 Technical Deficiency 问题是不是需要解决，这一点在本节随后的内容中还要进行讨论。Spiritually Deficiency 的解决很困难，因为即使对于人来说也很难判断一个句子是不是“良好”的句子。当然可以考虑用语言模型来缓解这个问题，不过由于在翻译的时候源语言句子都是定义“良好”的句子， $P(\text{ill}|\text{t})$ 对 $P(\text{s}|\text{t})$ 的影响并不大。但用输入的源语言句子 s 的“良好性”并不能解决 Technical Deficiency，因为 Technical Deficiency 是模型的问题或者模型参数估计方法的问题。无论输入什么样的 s ，模型 3 和模型 4 的 Technical Deficiency 问题都存在。

3.6.3 句子长度

在 IBM 模型中， $P(\text{t})P(\text{s}|\text{t})$ 会随着目标语言句子长度的增加而减少，因为这种生成模型有多个概率化的因素组成，一般乘积项越多结果的值越小。这也就是说，IBM 模型会更倾向选择长度短一些的目标语言句子。显然这种对短句子的偏向性并不是我们所期望的。

这个问题在很多统计机器翻译系统中都存在，实际上也是一种**系统偏置** (System Bias) 的体现。为了消除这种偏置，可以通过在模型中增加一个短句子惩罚因子来抵消掉模型对短句子的倾向性。比如，可以定义一个惩罚因子，它的值随着长度的减少而增加。不过，简单引入这样的惩罚因子会导致模型并不符合一个严格的噪声信道模型。它对应一个判别式框架的翻译模型，这部分内容会在下一章进行介绍。

3.6.4 其他问题

模型 5 的意义是什么？模型 5 的提出是为了消除模型 3 和模型 4 的 Deficiency 问题。Deficiency 问题的本质是， $P(\text{s}, \mathbf{a}|\text{t})$ 在所有合理的对齐上概率和不为 1。但是，在统计机器翻译中更关心是哪个对齐 \mathbf{a} 使 $P(\text{s}, \mathbf{a}|\text{t})$ 达到最大，即使 $P(\text{s}, \mathbf{a}|\text{t})$ 不符合概率分布的定义，也并不影响我们寻找理想的对齐 \mathbf{a} 。从工程的角度说， $P(\text{s}, \mathbf{a}|\text{t})$ 不归一并不是一个十分严重的问题。遗憾的是，实际上到现在为止有太多对 IBM 模型 3 和模型 4 中的 Deficiency 问题进行过系统的实验和分析，但对于这个问题到底有多严重并没有定论。当然用模型 5 是可以解决这个问题。但是如果用一个非常复杂的模型去解决了一个并不产生严重后果的问题，那这个模型也就没有太大意义了（从实践的角度）。

概念 (cept.) 的意义是什么？经过前面的分析可知，IBM 模型的词对齐模型使用了 cept. 这个概念。但是，在 IBM 模型中使用的 cept. 最多只能对应一个目标语言

单词（模型并没有用到源语言 *ceptr* 的概念）。因此可以直接用单词代替 *ceptr*。这样，即使不引入 *ceptr* 的概念，也并不影响 IBM 模型的建模。实际上，*ceptr* 的引入确实可以帮助我们是从语法和语义的角度解释词对齐过程。不过，这个方法在 IBM 模型中的效果究竟如何还没有定论。

3.7 小结及深入阅读

本章对 IBM 系列模型进行了全面的介绍和讨论，从一个简单的基于单词的翻译模型开始，本章以建模、解码、训练多个维度对统计机器翻译进行了描述，期间也涉及了词对齐、优化等多个重要概念。IBM 模型共分为 5 个模型，对翻译问题的建模依次由浅入深，同时模型复杂度也依次增加。IBM 模型作为入门统计机器翻译的“必经之路”，其思想对今天的机器翻译仍然产生着影响。虽然单独使用 IBM 模型进行机器翻译现在已经不多见，甚至很多从事神经机器翻译等前沿研究的人对 IBM 模型已经逐渐淡忘，但是不能否认 IBM 模型标志着一个时代的开始。从某种意义上，当使用公式 $\hat{t} = \arg \max_t P(t|s)$ 描述机器翻译问题的时候，或多或少都在与 IBM 模型使用相似的思想。

当然，本书也无法涵盖 IBM 模型的所有内涵，很多内容需要感兴趣的读者继续研究和挖掘，有两个方向可以考虑：

- IBM 模型在提出后的十余年中，一直受到了学术界的关注。一个比较有代表性的成果是 GIZA++ (<https://github.com/moses-smt/giza-pp>)，它集成了 IBM 模型和隐马尔可夫模型，并实现了这些模型的训练。在随后相当长的一段时间里，GIZA++ 也是机器翻译研究的标配，用于获得双语平行数据上单词一级的对齐结果。此外，研究者也对 IBM 模型进行了大量的分析，为后人研究统计机器翻译提供了大量依据 [222]。虽然 IBM 模型很少被独立使用，甚至直接使用基于 IBM 模型的解码器也不多见，但是它通常会作为其他模型的一部分参与到对翻译的建模中。这部分工作会在下一章基于短语和句法的模型中进行讨论 [153]。此外，IBM 模型也给机器翻译提供了一种非常简便的计算双语词串对应好坏的方式，因此也被广泛用于度量双语词串对应的强度，是自然语言处理中的一种常用特征。
- 除了在机器翻译建模上的开创性工作，IBM 模型的另一项重要贡献是建立了统计词对齐的基础模型。在训练 IBM 模型的过程中，除了学习到模型参数，还可以得到双语数据上的词对齐结果。也就是说词对齐标注是 IBM 模型训练的间接产物。这也使得 IBM 模型成为了自动词对齐的重要方法。包括 GIZA++ 在内的很多工作，实际上更多的是被用于自动词对齐任务，而非简单的训练 IBM 模型参数。随着词对齐概念的不断深入，这个任务逐渐成为了自然语言处理中的重要分支，比如，对 IBM 模型的结果进行对称化 [221]，也可以直接使用判别式模型利用分类模型解决词对齐问题 [124]，甚至可以把对齐的思想用于短语

和句法结构的双语对应 [318]。除了 GIZA++，研究人员也开发了很多优秀的自动词对齐工具，比如，FastAlign (https://github.com/clab/fast_align)、Berkeley Aligner (<https://github.com/mhajiloo/berkeleyaligner>) 等，这些工具现在也有很广泛的应用。



4. 基于短语和句法的机器翻译模型

机器翻译的一个问题是要定义翻译的基本单元是什么。比如，可以像第三章介绍的那样，以单词为单位进行翻译，即把句子的翻译看作是单词之间对应关系的一种组合。基于单词的模型是符合人类对翻译问题的认知的，因为单词本身就是人类加工语言的一种基本单元。另一方面，在进行翻译时也可以使用一些更“复杂”的知识。比如，很多词语间的搭配需要根据语境的变化进行调整，而且对于句子结构的翻译往往需要更上层的知识，如句法知识。因此，在对单词翻译进行建模的基础上，需要探索其他类型的翻译知识，使得搭配和结构翻译等问题可以更好地被建模。

本章会介绍基于短语和基于句法的翻译模型。在过去二十年中，它们一直是机器翻译的主流方法。相比于基于单词的翻译模型，基于短语和基于句法的模型可以更好地对单词之间的依赖关系进行描述，同时可以对句子的上层结构进行有效的表示。这些方法也在相当长的一段时期内占据着机器翻译的统治地位。虽然近些年随着神经机器翻译的崛起，基于短语和基于句法的统计翻译模型有些“降温”，但是它仍然是机器翻译的主要框架之一，其中的思想和很多技术手段对今天的机器翻译研究仍然有很好的借鉴意义。

4.1 翻译中的结构信息

首先，回顾一下基于单词的统计翻译模型是如何完成翻译的。图4.1展示了一个实例。其中，左侧是一个单词的“翻译表”，它记录了源语言（中文）单词和目标语言（英文）单词之间的对应关系，以及这种对应的可能性大小（用 P 表示）。在翻译

时，会使用这些单词一级的对应，生成目标语言译文。比如，图4.1右侧就展示了一个基于词的模型生成的翻译结果，其中 **s** 和 **t** 分别表示源语言和目标语言句子，单词之间的连线表示两个句子中单词一级的对应。

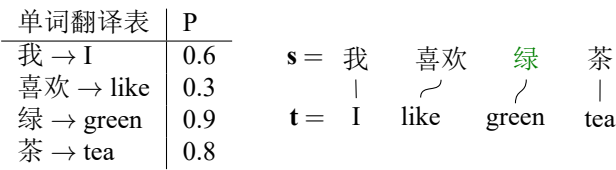


图 4.1: 基于单词的翻译实例

图4.1体现的是一个典型的基于单词对应关系的翻译方法。它非常适合**组合性翻译**（Compositional Translation）的情况，也就是通常说的直译。不过，自然语言作为人类创造的高级智能的载体，远比想象的复杂。比如，即使是同一个单词，词义也会根据不同的语境产生变化。

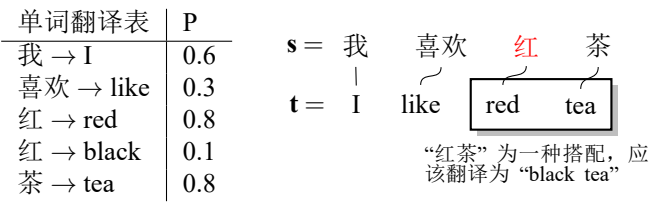


图 4.2: 基于单词的模型对搭配“红 茶”进行翻译

图4.2给出了一个新的例子。如果同样使用概率化的单词翻译对问题进行建模，对于输入的句子“我 喜欢 红 茶”，翻译概率最大的译文是“I like red tea”。显然，“red tea”并不是英文中“红 茶”的说法，正确的译文应该是“black tea”。

这里的问题在于，“black tea”不能通过“红”和“茶”这两个单词直译的结果组合而成，也就是，把“红”翻译为“red”并不符合“红 茶”这种特殊的搭配的翻译。即使在训练数据中“红”有很高的概率被翻译为“red”，在这个例子中，应该选择概率更低的译文“black”。那如何做到这一点呢？如果让人来做，这个事不难，因为所有人学习英语的时候都知道“红”和“茶”放在一起构成了一个短语，或者说一种搭配，这种搭配的译文是固定的，记住就好。同理，如果机器翻译系统也能学习并记住这样的搭配，显然可以做得更好。这也就形成了基于短语和句法的机器翻译建模的基本思路。

4.1.1 更大粒度的翻译单元

既然仅仅使用单词的直译不能覆盖所有的翻译现象，那就可以考虑在翻译中使用更大颗粒度的单元，这样能够对更大范围的搭配和依赖关系进行建模。一种非常简单的方法是把单词扩展为 *n*-gram（这里视为**短语**）。也就是，翻译的基本单元是一

个个连续的词串，而非一个个相互独立的单词。

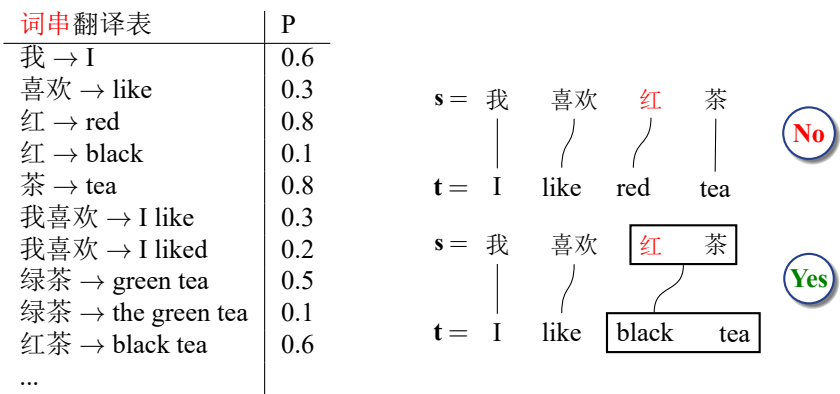


图 4.3: 基于短语（*n*-gram）的翻译的实例

图4.3展示了一个引入短语之后的翻译结果。其中的翻译表不仅包含源语言和目标语言单词之间的对应，同时也包括短语（*n*-gram）的翻译。这样，“红 茶”可以作为一个短语包含在翻译表中，它所对应译文是“black tea”。对于待翻译句子，可以使用单词翻译的组合得到“红 茶”的译文“red tea”，也可以直接使用短语翻译得到“black tea”。由于短语翻译“红 茶 → black tea”的概率更高，因此最终会输出正确的译文“black tea”。

一般来说，统计机器翻译的建模对应着一个两阶段的过程：首先，得到每个翻译单元所有可能的译文；然后，通过对这些译文的组合得到可能的句子翻译结果，并选择最佳的目标语言句子输出。如果基本的翻译单元被定义下来，机器翻译系统可以学习这些单元翻译所对应的翻译知识（对应训练过程），之后运用这些知识对新的句子进行翻译（对应解码过程）。

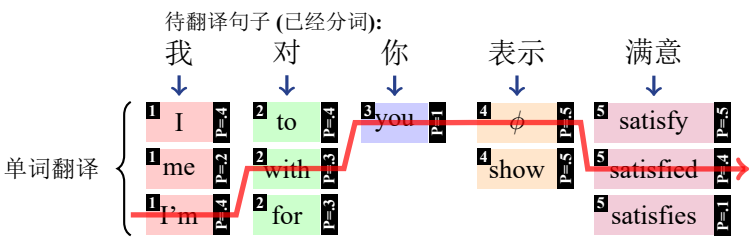


图 4.4: 基于单词的翻译被看作是一条“路径”

图4.4给出了基于单词的机器翻译过程的一个示例。首先，每个单词的候选译文都被列举出来，而机器翻译系统就是要找到覆盖所有源语言单词的一条路径，它所对应的译文概率是最高的。比如，图中的红色折线就代表了一条翻译路径，也就是一个单词译文的序列¹。

¹为了简化问题，这里没有描述单词译文的调序。对于调序的建模，可以把它当作是对目标语单词串

在引入短语翻译之后，并不需要对上述过程进行太大的修改。仍然可以把翻译当作是一条贯穿源语言所有单词译文的路径，只是这条路径中会包含短语，而非一个个单词。图4.5给出了一个实例，其中的蓝色折线表示包含短语的翻译路径。

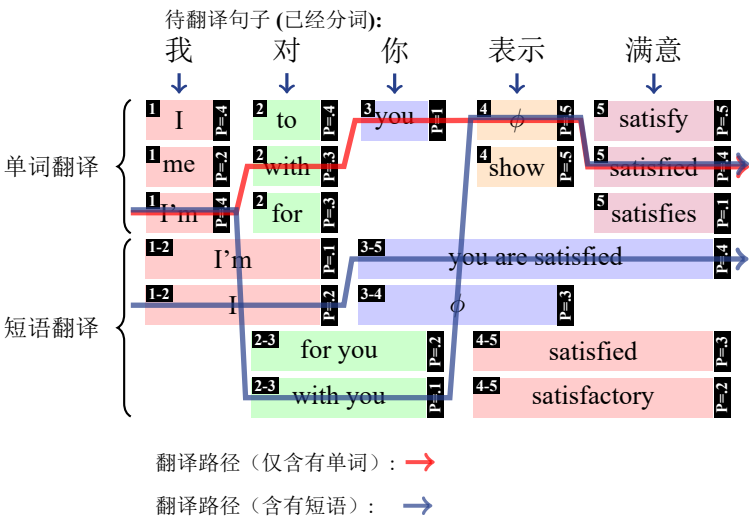


图 4.5: 翻译被看作是由单词和短语组成的“路径”

实际上，单词本身也是一种短语。从这个角度说，基于单词的翻译模型是包含在基于短语的翻译模型中的。而这里的所说的短语包括多个连续的单词，可以直接捕捉翻译中的一些局部依赖。而且，由于引入了更多样翻译单元，可选择的翻译路径数量也大大增加。本质上，引入更大颗粒度的翻译单元给建模增加了灵活性，同时增大了翻译假设空间。如果建模合理，更多的翻译路径会增加找到高质量译文的机会。在4.2节还将看到，基于短语的模型会从多个角度对翻译问题进行描述，包括基础数学建模、调序等等。

4.1.2 句子的结构信息

使用短语的优点在于可以捕捉具有完整意思的连续词串，因此能够对局部上下文信息进行建模。当单词之间的搭配和依赖关系出现在连续词串中时，短语可以很好地对其进行描述。但是，当单词之间距离很远时，使用短语的“效率”很低。同 n -gram 语言模型一样，当短语长度变长时，数据会变得非常稀疏。比如，很多实验已经证明，测试数据中超过 5 个的连续单词在训练数据中往往是很低频的现象，更长的短语甚至都很难在训练数据中找到。

当然，可以使用平滑算法对长短语的概率进行估计，但是使用过长的短语在实际系统研发中仍然不现实。图4.6展示了一个汉语到英语的翻译实例。源语言的两个短语（蓝色和红色高亮）在译文中产生了调序。但是，这两个短语在源语言句子中横

的排列，这个排列的好坏需要用额外的调序模型进行描述。详细内容见4.2.4节。

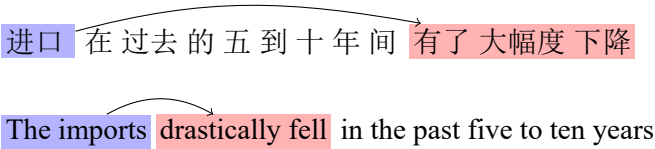


图 4.6: 汉英翻译中的长距离依赖

跨 11 个单词。如果直接使用这个 11 个单词构成的短语进行翻译，显然会有非常严重的数据稀疏问题，因为很难期望在训练数据中见到一模一样的短语。

如果仅仅使用连续词串不能处理所有的翻译问题，根本的原因在于句子的表层串很难描述片段之间大范围的依赖。一个新的思路是使用句子的结构信息进行建模。第二章已经介绍了句子的句法表示形式。对于每个句子，都可以用句法树描述它的结构。

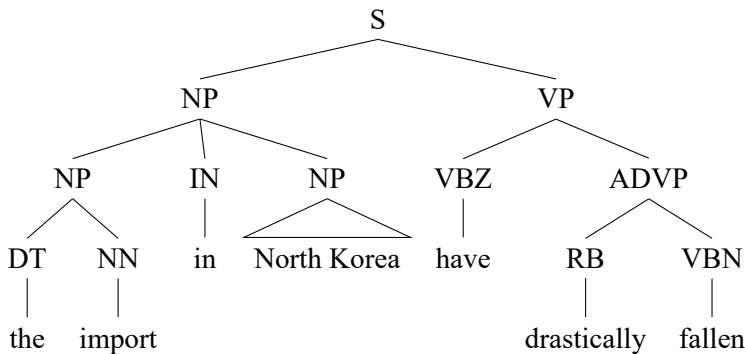


图 4.7: 一个英文句法树（短语结构树）

图4.7就展示了一棵英文句法树（短语结构树）。句法树描述了一种递归的结构，每个句法结构都可以用一个子树来描述，子树之间的组合可以构成更大的子树，最终完成整个句子的表示。相比线性的序列模型，树结构更容易处理大片段之间的关系。比如，两个在序列中距离“很远”的单词，在树结构中可能会“很近”。

句法树结构可以赋予机器翻译一种对语言进一步抽象的能力，这样，并不需要使用连续词串，而是通过句法结构来对大范围的译文生成和调序进行建模。图4.8是一个在翻译中融入源语言（中文）句法信息的实例。这个例子中，介词短语包含 15 个单词，因此，使用短语很难涵盖“在 ... 后”这样的片段。这时，系统会把“在 ... 后”错误的翻译为“In ...”。通过句法树，可以知道“在 ... 后”对应着一个完整的子树结构 PP（介词短语）。因此也很容易知道介词短语中“在 ... 后”是一个模板（红色），而“在”和“后”之间的部分构成从句部分（蓝色）。最终得到正确的译文“After ...”。

使用句法信息在机器翻译中不新鲜。在基于规则和模板的翻译模型中，就大量地使用了句法等结构信息。只是由于早期句法分析技术不成熟，系统的整体效果并

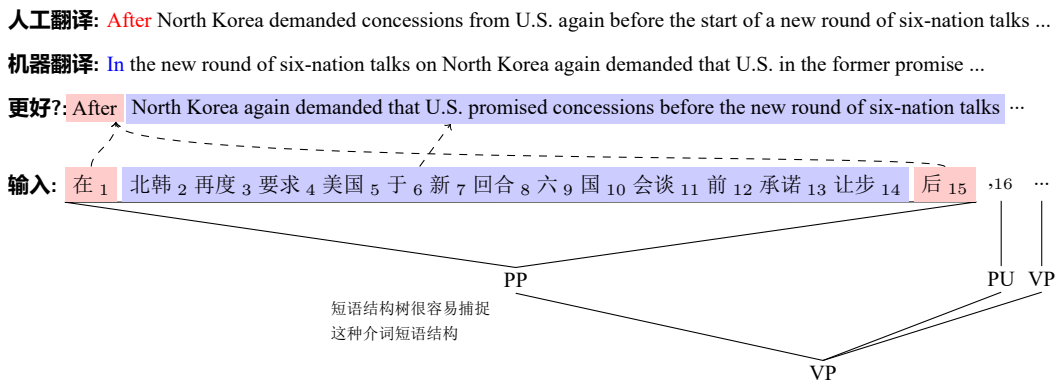


图 4.8: 使用句法结构进行机器翻译的实例

不突出。在统计机器翻译时代，句法可以很好地融合在统计建模中。通过概率化的文法设计，可以对翻译过程进行很好的描述。在本章的4.3节和4.4节中将会详细讨论句法信息在统计机器翻译中的应用。

4.2 基于短语的翻译模型

基于短语的翻译模型是统计机器翻译最具代表性的模型之一 [38, 153]。这类模型易于实现，而且性能突出。统计机器翻译中很多经典的方法都出自基于短语的模型，比如：统计调序模型、最小错误率训练等等。下面就来了解一下基于短语的机器翻译是如何工作的。

4.2.1 机器翻译中的短语

基于短语的机器翻译的基本假设是：双语句子的生成可以用短语之间的对应关系进行表示。图4.9展示了一个基于短语的翻译实例。可以看到，这里的翻译单元是连续的词串。比如，“进口”的译文“The imports have”就包含了三个单词，而“下降了”也是一个包含两个单词的源语言片段。

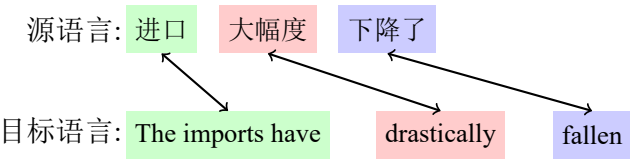


图 4.9: 基于短语的汉英翻译实例

不过，这里所说的短语并不是语言学上的短语，本身也没有任何语言学句法的结构约束。在基于短语的模型中，可以把短语简单地理解为一个词串。具体来说，有如下定义。

定义 4.2.1 短语

对于一个句子 $\mathbf{w} = w_1 \dots w_n$ ，任意子串 $w_i \dots w_j (i \leq j \text{ 且 } 0 \leq i, j \leq n)$ 都是句子 \mathbf{w} 的一个**短语**。

根据这个定义，对于一个由 n 个单词构成的句子，可以包含 $\frac{n(n-1)}{2}$ 个短语（子串）。进一步，可以把每个句子看作是由一系列短语构成的序列。组成这个句子的短语序列也可以被看作是句子的一个**短语切分**（Phrasal Segmentation）。

定义 4.2.2 句子的短语切分

如果一个句子 $\mathbf{w} = w_1 \dots w_n$ 可以被切分为 m 个子串，则称 \mathbf{w} 由 m 个短语组成，记为 $\mathbf{w} = p_1 \dots p_m$ ，其中 p_i 是 \mathbf{w} 的一个短语， $p_1 \dots p_m$ 也被称作句子 \mathbf{w} 的一个**短语切分**。

比如，对于一个句子，“机器 翻译 是 一 项 很 有 挑 战 的 问 题”，一种可能的短语切分为：

p_1 = 机器 翻译
 p_2 = 是 一 项
 p_3 = 很 有 挑 战 的
 p_4 = 问 题

进一步，把单语短语的概念推广到双语的情况：

定义 4.2.3 双语短语（或短语对）

对于源语和目标语句对 (\mathbf{s}, \mathbf{t}) ， \mathbf{s} 中的一个短语 \bar{s}_i 和 \mathbf{t} 中的一个短语 \bar{t}_j 可以构成一个双语短语对 (\bar{s}_i, \bar{t}_j) ，简称**短语对** (\bar{s}_i, \bar{t}_j) 。

也就是说，源语言句子中任意的短语和目标语言句子中任意的短语都构成一个双语短语。这里用 \leftrightarrow 表示互译关系。对于一个双语句对“进口 大幅度 下降 了 \leftrightarrow the imports have drastically fallen”，可以得到很多双语短语，比如：

大幅度 \leftrightarrow drastically
大幅度 下降 \leftrightarrow have drastically fallen
进口 大幅度 \leftrightarrow imports have drastically
大幅度 下降 了 \leftrightarrow drastically fallen
了 \leftrightarrow have drastically
...

接下来的问题是，如何使用双语短语描述双语句子的生成，即句子翻译的建模问题。在基于词的翻译模型里，可以用词对齐来描述双语句子的对应关系。类似的，

也可以使用双语短语描述句子的翻译。这里，借用形式文法中**推导**（Derivation）的概念。把生成双语句对的过程定义为一个基于短语的翻译推导：

定义 4.2.4 基于短语的翻译推导

对于源语言和目标语言句对 (s, t) ，分别有短语切分 $\{\bar{s}_i\}$ 和 $\{\bar{t}_j\}$ ，且 $\{\bar{s}_i\}$ 和 $\{\bar{t}_j\}$ 之间存在一一对应的关系。令 $\{\bar{a}_j\}$ 表示 $\{\bar{t}_j\}$ 中每个短语对应到源语言短语的编号，则称短语对 $\{(\bar{s}_{\bar{a}_j}, \bar{t}_j)\}$ 构成了 s 到 t 的**基于短语的翻译推导**（简称推导），记为 $d(\{(\bar{s}_{\bar{a}_j}, \bar{t}_j)\}, s, t)$ （简记为 $d(\{(\bar{s}_{\bar{a}_j}, \bar{t}_j)\})$ 或 d ）。

基于短语的翻译推导定义了一种从源语言短语序列到目标语言短语序列的对应，其中源语言短语序列是源语言句子的一种切分，同样的，目标语言短语序列是目标语言句子的一种切分。翻译推导提供了一种描述翻译过程的手段：对于一个源语言句子，可以找到从它出发的翻译推导，推导中短语的目标语部分就构成了译文。也就是，每个源语言句子 s 上的一个推导 d 都蕴含着一个目标语句子 t 。

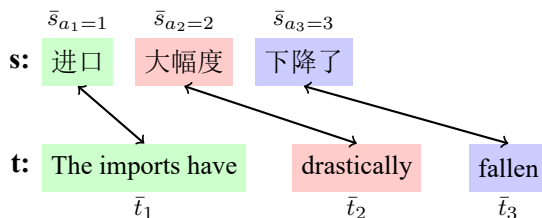


图 4.10: 三个双语短语 $\{(\bar{s}_{\bar{a}_1}, \bar{t}_1), (\bar{s}_{\bar{a}_2}, \bar{t}_2), (\bar{s}_{\bar{a}_3}, \bar{t}_3)\}$ 构成的翻译推导

图4.10给出了一个由三个双语短语 $\{(\bar{s}_{\bar{a}_1}, \bar{t}_1), (\bar{s}_{\bar{a}_2}, \bar{t}_2), (\bar{s}_{\bar{a}_3}, \bar{t}_3)\}$ 构成的汉英互译句对，其中短语对齐信息为 $\bar{a}_1 = 1, \bar{a}_2 = 2, \bar{a}_3 = 3$ 。这里，可以把这三个短语对的组合看作是翻译推导，形式化表示为如下公式。其中， \circ 表示短语的组合²。

$$d = (\bar{s}_{\bar{a}_1}, \bar{t}_1) \circ (\bar{s}_{\bar{a}_2}, \bar{t}_2) \circ (\bar{s}_{\bar{a}_3}, \bar{t}_3) \quad (4.1)$$

到此为止，就得到了一个基于短语的翻译模型。对于每个双语句对 (s, t) ，每个翻译推导 d 都对应了一个基于短语的翻译过程。而基于短语的机器翻译的目标就是对 d 进行描述。有四个基本问题：

- 如何用统计模型描述每个翻译推导的好坏 —— 即翻译的统计建模问题；
- 如何获得可使用的双语短语对 —— 即短语翻译获取问题；

²短语的组合是指将两个短语 a 和 b 进行拼接，形成新的短语 ab 。在机器翻译中，可以把双语短语的组合看作是对目标语短语的组合。比如，对于两个双语短语 $(\bar{s}_{\bar{a}_1}, \bar{t}_1), (\bar{s}_{\bar{a}_2}, \bar{t}_2)$ ，短语的组合表示将 \bar{t}_1 和 \bar{t}_2 进行组合，而源语言端作为输入已经给定，因此直接匹配源语言句子中相应的部分即可。根据两个短语在源语言中位置的不同，通常又分为顺序翻译、反序翻译、不连续翻译。关于这部分内容将会在后面4.2.4节调序模型的部分进行介绍。

- 如何对翻译中的调序问题进行建模——即调序问题；
- 如何找到输入句子 s 的最佳译文——即解码问题。

这四个问题也构成了基于短语的翻译模型的核心，下面对其逐一展开讨论。

4.2.2 数学建模及判别式模型

对于统计机器翻译，其目的是找到输入句子的可能性最大的译文：

$$\hat{t} = \arg \max_t P(t|s) \quad (4.2)$$

公式4.2中， s 是输入的源语言句子， t 是一个目标语译文。 $P(t|s)$ 被称为翻译模型，它描述了把 s 翻译为 t 的可能性。通过 $\arg \max P(t|s)$ 可以找到使 $P(t|s)$ 达到最大的 t 。

这里的第一个问题是如何定义 $P(t|s)$ 。直接描述 $P(t|s)$ 是非常困难的，因为 s 和 t 分别对应了巨大的样本空间，而在训练数据中能观测到的只是空间中的一小部分样本。直接用有限的训练数据描述这两个空间中样本的对应关系会面临着严重的数据稀疏问题。对于这个问题，常用的解决办法是把复杂的问题转化为容易计算的简单问题。

基于翻译推导的建模

基于短语的翻译模型假设 s 到 t 的翻译可以用翻译推导进行描述，这些翻译推导都是由双语短语组成。于是，两个句子之间的映射就可以被看作是一个个短语的映射。显然短语翻译的建模要比整个句子翻译的建模简单得多。从模型上看，可以把翻译推导 d 当作是从 s 到 t 翻译的一种隐含结构。这种结构定义了对问题的一种描述，即翻译由一系列短语组成。根据这个假设，可以把句子的翻译概率定义为：

$$P(t|s) = \sum_d P(d, t|s) \quad (4.3)$$

公式4.3中， $P(d, t|s)$ 表示翻译推导的概率。公式4.3把翻译问题转化为翻译推导的生成问题。但是，由于翻译推导的数量十分巨大³，公式4.3的右端需要对所有可能的推导进行枚举并求和，这几乎是无法计算的。

对于这个问题，常用的解决办法是利用一个化简的模型来近似完整的模型。如果把翻译推导的全体看作一个空间 D ，可以从 D 中选取一部分样本参与计算，而不是对整个 D 进行计算。比如，可以用最好的 n 个翻译推导来代表整个空间 D 。令

³如果把推导看作是一种树结构，推导的数量与词串的长度成指数关系。

$D_{n\text{-best}}$ 表示最好的 n 个翻译推导所构成的空间，于是可以定义：

$$P(t|s) \approx \sum_{d \in D_{n\text{-best}}} P(d, t|s) \quad (4.4)$$

进一步，把公式4.4带入公式4.2，可以得到翻译的目标为：

$$\hat{t} = \arg \max_t \sum_{d \in D_{n\text{-best}}} P(d, t|s) \quad (4.5)$$

另一种常用的方法是直接用 $P(d, t|s)$ 的最大值代表整个翻译推导的概率和。这种方法假设翻译概率是非常尖锐的，“最好”的推导会占有概率的主要部分。它被形式化为：

$$P(t|s) \approx \max P(d, t|s) \quad (4.6)$$

于是，翻译的目标可以被重新定义：

$$\hat{t} = \arg \max_t (\max P(d, t|s)) \quad (4.7)$$

值得注意的是，翻译推导中蕴含着译文的信息，因此每个翻译推导都与一个译文对应。因此可以把公式4.7所描述的问题重新定义为：

$$\hat{d} = \arg \max_d P(d, t|s) \quad (4.8)$$

也就是，给定一个输入句子 s ，找到从它出发的最优翻译推导 \hat{d} ，把这个翻译推导所对应的目标语词串看作最优的译文。假设函数 $t(\cdot)$ 可以返回一个推导的目标语词串，则最优译文也可以被看作是：

$$\hat{t} = t(\hat{d}) \quad (4.9)$$

注意，公式4.8-4.9和公式4.7本质上是一样的。它们也构成了统计机器翻译中最常用的方法——Viterbi 方法 [291]。在后面机器翻译的解码中还会看到它们的应用。而公式4.5也被称作 $n\text{-best}$ 方法，常常作为 Viterbi 方法的一种改进。

对数线性模型

对于如何定义 $P(d, t|s)$ 有很多种思路，比如，可以把 d 拆解为若干步骤，然后对这些步骤分别建模，最后形成描述 d 的**生成式模型**（Generative Model）。这种方法在第三章的 IBM 模型中也大量使用。但是，生成式模型的每一步推导需要有严格的概率解释，这也限制了研究人员从更多的角度对 d 进行描述。这里，可以使用另外一

种方法——**判别式模型**（Discriminative Model）来对 $P(d, \mathbf{t}|\mathbf{s})$ 进行描述 [220]。其模型形式如下：

$$P(d, \mathbf{t}|\mathbf{s}) = \frac{\exp(\text{score}(d, \mathbf{t}, \mathbf{s}))}{\sum_{d', \mathbf{t}'} \exp(\text{score}(d', \mathbf{t}', \mathbf{s}))} \quad (4.10)$$

$$\text{score}(d, \mathbf{t}, \mathbf{s}) = \sum_{i=1}^M \lambda_i \cdot h_i(d, \mathbf{t}, \mathbf{s}) \quad (4.11)$$

公式4.11是一种典型的**对数线性模型**（Log-linear Model）。所谓“对数线性”体现在对多个量求和后进行指数运算（ $\exp(\cdot)$ ），这相当于对多个因素进行乘法。公式4.10的右端是一种归一化操作。分子部分可以被看作是一种对翻译推导 d 的对数线性建模。具体来说，对于每个 d ，用 M 个特征对其进行描述，每个特征用函数 $h_i(d, \mathbf{t}, \mathbf{s})$ 表示。每个特征都对应一个权重 λ_i ，表示特征 i 的重要性。 $\sum_{i=1}^M \lambda_i \cdot h_i(d, \mathbf{t}, \mathbf{s})$ 表示了对这些特征的线性加权和，值越大表示模型得分越高，相应的 d 和 \mathbf{t} 的质量越高。公式4.10的分母部分实际上不需要计算，因为其值与求解最佳推导的过程无关。把公式4.10带入公式4.8得到：

$$\begin{aligned} \hat{d} &= \arg \max_d \frac{\exp(\text{score}(d, \mathbf{t}, \mathbf{s}))}{\sum_{d', \mathbf{t}'} \exp(\text{score}(d', \mathbf{t}', \mathbf{s}))} \\ &= \arg \max_d \exp(\text{score}(d, \mathbf{t}, \mathbf{s})) \end{aligned} \quad (4.12)$$

公式4.12中， $\exp(\text{score}(d, \mathbf{t}, \mathbf{s}))$ 表示指数化的模型得分，记为 $\text{mscore}(d, \mathbf{t}, \mathbf{s}) = \exp(\text{score}(d, \mathbf{t}, \mathbf{s}))$ 。于是，翻译问题就可以被描述为找到使函数 $\text{mscore}(d, \mathbf{t}, \mathbf{s})$ 达到最大的 d 。由于， $\exp(\text{score}(d, \mathbf{t}, \mathbf{s}))$ 和 $\text{score}(d, \mathbf{t}, \mathbf{s})$ 是单调一致的，因此有时也直接把 $\text{score}(d, \mathbf{t}, \mathbf{s})$ 当做模型得分。

判别式模型最大的好处在于它可以更灵活地引入特征。系统开发者可以设计任意的特征来描述翻译，特征的设计甚至都不需要统计上的解释，比如 0-1 特征、计数特征等。此外，判别式模型并不需要像生成式模型那样对问题进行具有统计学意义的“分解”，更不需要对每个步骤进行严格的数学推导。相反，它直接对问题的后验概率进行建模。由于不像生成式模型那样需要引入假设来对每个生成步骤进行化简，判别式模型对问题的刻画更加直接，因此也受到自然语言处理研究者的青睐。

搭建模型的基本流程

对于翻译的判别式建模，需要回答两个问题：

- 如何设计特征函数 $\{h_i(d, \mathbf{t}|\mathbf{s})\}$?
- 如何获得最好的特征权重 $\{\lambda_i\}$?

在基于短语的翻译模型中，通常包含三类特征：短语翻译特征、调序特征、语言模型相关的特征。这些特征都需要从训练数据中学习。

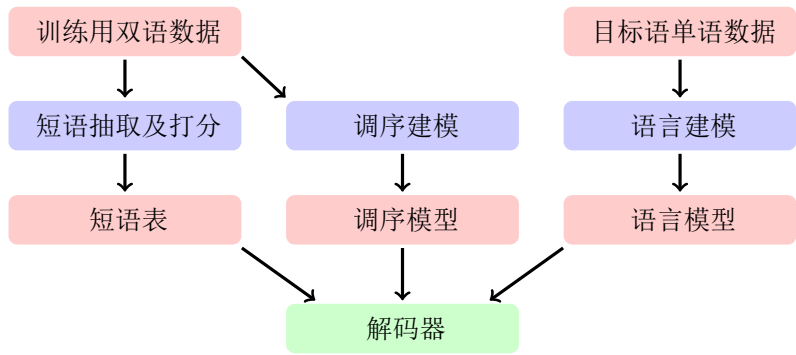


图 4.11: 基于短语的机器翻译的系统流程

图4.11展示了一个基于短语的机器翻译模型的搭建流程。其中的训练数据包括双语平行语料和目标语言单语语料。首先，需要从双语平行数据中学习短语的翻译，并形成短语翻译表；然后，再从双语平行数据中学习调序模型；最后，从目标语单语数据中学习语言模型。短语翻译表、调序模型、语言模型都会作为特征被送入判别式模型，由解码器完成对新句子的翻译。而这些特征的权重可以在额外的开发集上进行调优。关于短语翻译、调序模型和特征权重的学习，会在本章的4.2.3-4.2.6节进行介绍。

4.2.3 短语抽取

在基于短语的模型中，学习短语翻译是重要的步骤之一。获得短语翻译的方法有很多种，最常用的方法是从双语平行语料中进行**短语抽取**（Phrase Extraction）。前面已经介绍过短语的概念，句子中任意的连续子串都被称为短语。例如在图4.12中，用点阵的形式来表示双语之间的对应关系，那么图中任意一个矩形框都可以构成一个双语短语（或短语对），例如“什么都没”对应“learned nothing？”。

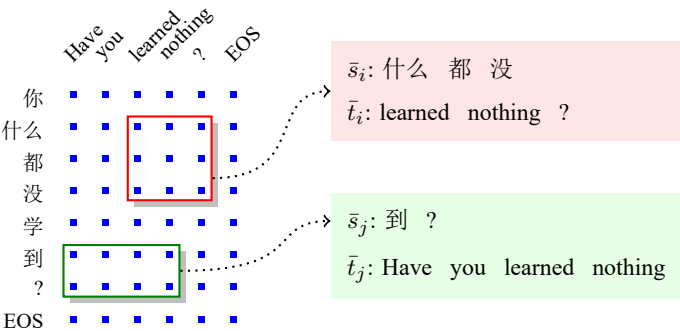


图 4.12: 无限制的短语抽取

按照上述抽取短语的方式可以找到所有可能的双语短语，但是这种不加限制的

抽取是非常十分低效的。一是可抽取的短语数量爆炸，二是抽取得到的大部分短语是没有意义的，如上面的例子中抽取到“到？”对应“Have you learned nothing？”这样的短语对在翻译中并没有什么意义。对于这个问题，一种解决方法是基于词对齐进行短语抽取，或者是抽取与词对齐相一致的短语。

与词对齐一致的短语

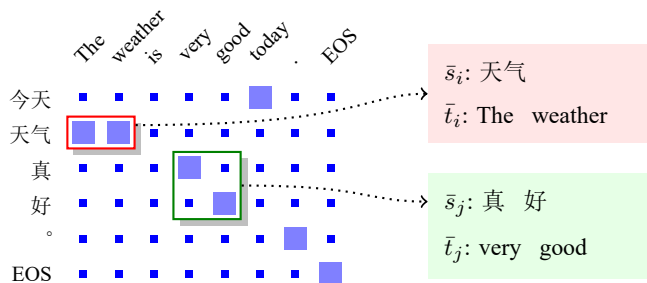


图 4.13: 与词对齐一致的短语抽取

图4.13中大蓝色方块代表词对齐。通过词对齐信息，可以很容易地获得双语短语“天气 \leftrightarrow The weather”。这里称其为与词对齐一致（兼容）的双语短语。具体定义如下：

定义 4.2.5 与词对齐一致（兼容）的双语短语

对于源语言句子 s 和目标语句子 t ，存在 s 和 t 之间的词对齐。如果有 (s, t) 中的双语短语 (\bar{s}, \bar{t}) ，且 \bar{s} 中所有单词仅对齐到 \bar{t} 中的单词，同时 \bar{t} 中所有单词仅对齐到 \bar{s} 中的单词，那么称 (\bar{s}, \bar{t}) 是与词对齐一致的（兼容的）双语短语。

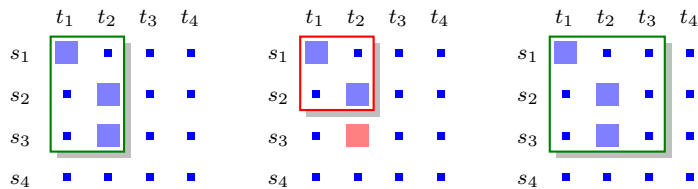


图 4.14: 词对齐一致性

如图4.14所示，左边的例子中的 t_1 和 t_2 严格地对应到 s_1 、 s_2 、 s_3 ，所以短语是与词对齐相一致的；中间的例子中的 t_2 对应到短语 s_1 和 s_2 的外面，所以短语是与词对齐不一致的；类似的，右边的例子也是与词对齐相一致的短语。

图4.15展示了与词对齐一致的短语抽取过程，首先判断抽取得到的双语短语是否与词对齐保持一致，若一致，则抽取出来。在实际抽取过程中，通常需要对短语的

最大长度进行限制，以免抽取过多的无用短语。比如，在实际系统中，最大短语长度一般是 5-7 个词。

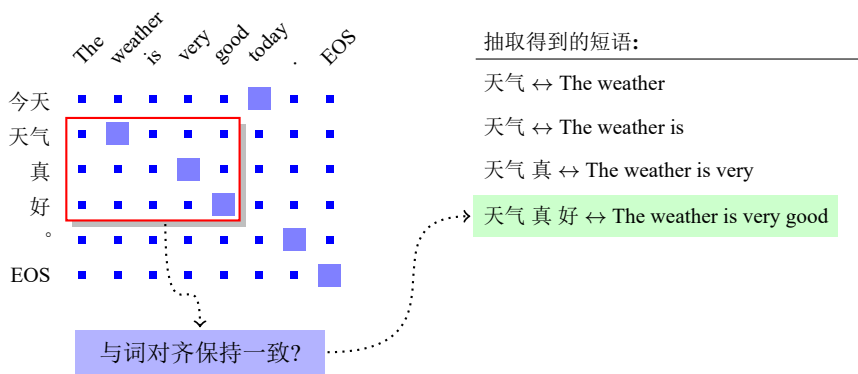


图 4.15: 与词对齐一致的短语抽取

获取词对齐

如何获得词对齐呢？上一章介绍的 IBM 模型本身就是一个词对齐模型，因此一种常用的方法是直接使用 IBM 模型生成词对齐。IBM 模型约定每个源语言单词必须对应、也只能对应到一个目标语单词。因此，IBM 模型得到的词对齐结果是不对称的。正常情况下词对齐可以是一个源语言单词对应多个目标语言单词，或者多对一，甚至多对多的情况。为了获得对称的词对齐，一种简单的方法是，分别进行正向翻译和反向翻译的词对齐，然后利用启发性方法生成对称的词对齐，例如，双向词对齐取交集、并集等。

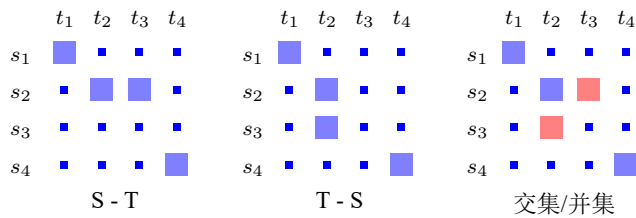


图 4.16: 词对齐的获取

如图4.16中，左边两个图就是正向和反向两种词对齐的结果。右边的图是融合双向词对齐的结果，取交集是蓝色的方框，取并集是红色的方框。当然，还可以设计更多的启发性规则生成词对齐 [149, 151]。

除此之外，一些外部工具也可以用来获取词对齐，如 Fastalign[66]、Berkeley Word Aligner[281] 等。词对齐的质量通常使用词对齐错误率（AER）来评价 [218]。但是词对齐并不是一个独立的系统，它一般会服务于其他任务。因此，也可以使用下游任

务来评价词对齐的好坏。比如，改进词对齐后观察机器翻译系统性能的变化。

度量双语短语质量

抽取双语短语之后，需要对每个双语短语的质量进行评价。这样，在使用这些双语短语时，可以更有效地估计整个句子翻译的好坏。在统计机器翻译中，一般用双语短语出现的可能性大小来度量双语短语的好坏。这里，使用相对频度估计对短语的翻译条件概率进行计算，公式如下：

$$P(\bar{t}|\bar{s}) = \frac{\text{count}(\bar{s}, \bar{t})}{\text{count}(\bar{s})} \quad (4.13)$$

给定一个双语句对 (\mathbf{s}, \mathbf{t}) ， $\text{count}(\bar{s})$ 表示短语 \bar{s} 在 \mathbf{s} 中出现的次数， $\text{count}(\bar{s}, \bar{t})$ 表示双语短语 (\bar{s}, \bar{t}) 在 (\mathbf{s}, \mathbf{t}) 中被抽取出来的次数。对于一个包含多个句子的语料库， $\text{count}(\bar{s})$ 和 $\text{count}(\bar{s}, \bar{t})$ 可以按句子进行累加。类似的，也可以用同样的方法，计算 \bar{t} 到 \bar{s} 的翻译概率，即 $P(\bar{s}|\bar{t})$ 。一般会同时使用 $P(\bar{t}|\bar{s})$ 和 $P(\bar{s}|\bar{t})$ 度量一个双语短语的好与坏。

当遇到低频短语时，短语翻译概率的估计可能会不准确。例如，短语 \bar{s} 和 \bar{t} 在语料中只出现了一次，且在一个句子中共现，那么 \bar{s} 到 \bar{t} 的翻译概率为 $P(\bar{t}|\bar{s}) = 1$ ，这显然是不合理的，因为 \bar{s} 和 \bar{t} 的出现完全可能是偶然事件。既然直接度量双语短语的好坏会面临数据稀疏问题，一个自然的想法就是把短语拆解成单词，利用双语短语中单词翻译的好坏间接度量双语短语的好坏。为了达到这个目的，可以使用**词汇化翻译概率**（Lexical Translation Probability）。前面借助词对齐信息完成了双语短语的抽取，因此，词对齐信息本身就包含了短语内部单词之间的对应关系。因此同样可以借助词对齐来计算词汇翻译概率，公式如下：

$$P_{\text{lex}}(\bar{t}|\bar{s}) = \prod_{j=1}^{|\bar{s}|} \frac{1}{|\{j|a(j,i)=1\}|} \sum_{\forall(j,i):a(j,i)=1} w(t_i|s_j) \quad (4.14)$$

它表达的意思是短语 \bar{s} 和 \bar{t} 存在词汇级的对应关系，其中 w 表示词汇翻译概率用来度量两个单词之间翻译的可能性大小（见第三章），作为两个词之间对应的强度。

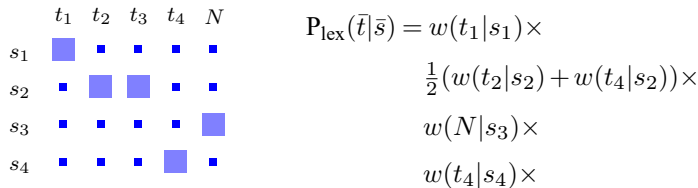


图 4.17: 词汇翻译概率实例

下面来看一个具体的例子，如图4.17所示。对于一个双语短语，将它们的词对齐

关系代入到上面的公式就会得到短语的词汇翻译概率。对于词汇翻译概率，可以使用 IBM 模型中的单词翻译表，也可以通过统计获得 [146]。如果一个单词的词对齐为空，则用 N 表示它翻译为空的概率。和短语翻译概率一样，可以使用双向的词汇化翻译概率来评价双语短语的好坏。

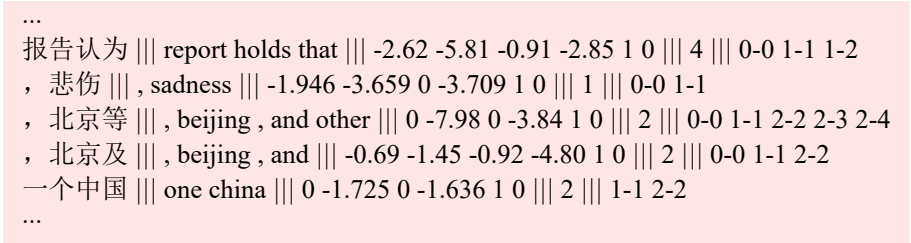


图 4.18: 短语表实例

经过上面的介绍，可以从双语平行语料中把双语短语抽取出来，同时得到相应的翻译概率（即特征），组成**短语表**（Phrase Table）。图4.18展示了一个真实短语表的片段。其中包括源语言短语和目标语言短语，用 ||| 进行分割。每个双语对应的得分，包括正向和反向的词汇翻译概率以及短语翻译概率，还包括词对齐信息（0-0、1-1）等其他信息。

4.2.4 调序

尽管已经知道了如何将一个源语言短语翻译成目标语言短语，但是想要获得一个高质量的译文，仅有互译的双语短语是远远不够的。

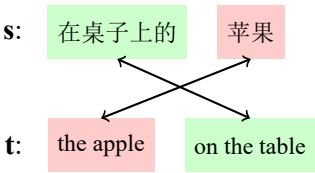


图 4.19: 基于短语翻译的调序

如图4.19所示，按照从左到右的顺序对一个句子“在 桌子 上 的 苹果”进行翻译，得到的译文“on the table the apple”的语序是不对的。虽然可以使用 n -gram 语言模型对语序进行建模，但是此处仍然需要用更加准确的方式描述目标语短语间的次序。一般，把这个问题称为短语调序，或者简称**调序**（Reordering）。通常，基于短语的调序模型会作为判别式模型的特征参与到翻译过程中来。接下来，会介绍 3 种不同的调序方法，分别是基于距离的调序、基于方向的调序（MSD 模型）以及基于分类的调序。

基于距离的调序

基于距离的调序是最简单的一种调序模型。很多时候，语言的翻译基本上都是顺序的，也就是，译文单词出现的顺序和源语言单词的顺序基本上是一致的。反过来说，如果译文和源语言单词（或短语）的顺序差别很大，就认为出现了调序。

基于距离的调序方法的核心思想就是度量当前翻译结果与顺序翻译之间的差距。对于译文中的第 i 个短语，令 $start_i$ 表示它所对应的源语言短语中第一个词所在的位置， end_i 是这个短语中最后一个词所在的位置。于是，这个短语（相对于前一个短语）的调序距离为：

$$dr = start_i - end_{i-1} - 1$$

(4.15)

在图4.20的例子中，“the apple”所对应的调序距离为4，“在桌子上的”所对应的调序距离为-5。显然，如果两个源语短语按顺序翻译，则 $start_i = end_{i-1} + 1$ ，这时调序距离为0。

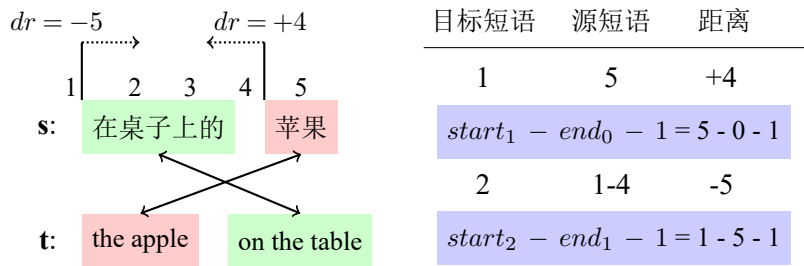


图 4.20: 基于距离的调序

如果把调序距离作为特征，一般会使用指数函数 $f(dr) = a^{|dr|}$ 作为特征函数（或者调序代价的函数），其中 a 是一个参数，控制调序距离对整个特征值的影响。调序距离 dr 的绝对值越大，调序代价越高。基于距离的调序模型比较适用于像法-英翻译这样的任务，因为两种语言的语序基本上是一致的。但是，对于汉-日翻译，由于句子结构存在很大差异（日语是谓词后置，而汉语中谓词放在宾语前），使用基于距离的调序会带来一些问题。因此，具体应用时应该根据语言之间的差异性有选择的使用该模型。

基于方向的调序

基于方向的调序模型是另一种常用的调序模型。该模型是一种典型的词汇化调序模型，因此调序的结果会根据不同短语有所不同。简单来说，在目标语言端连续的情况下，该模型会判断两个双语短语在源语言端的调序情况，包含三种调序类型：顺序的单调翻译（M）、与前一个短语交换位置（S）、非连续翻译（D）。因此，这个模型也被称作 MSD 调序模型 [93]。

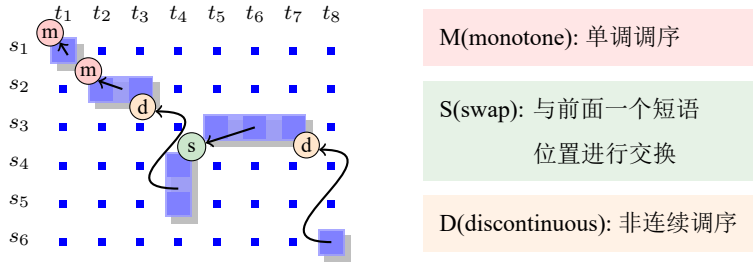


图 4.21: 词汇化调序模型的三种调序类型

图4.21展示了这三种调序类型，当两个短语对在源语言和目标语言中都是按顺序排列时，它们就是单调的（如：从左边数前两个短语）；如果对应的短语顺序在目标语中是反过来的，属于交换调序（如：从左边数第三和第四个短语）；如果两个短语之间还有其他的短语，就是非连续翻译（如：从右边数的前两个短语）。

对于每种调序类型，都可以定义一个调序概率，如下：

$$P(\mathbf{o}|\mathbf{s}, \mathbf{t}, \mathbf{a}) = \prod_{i=1}^K P(o_i|\bar{s}_{a_i}, \bar{t}_i, a_{i-1}, a_i) \tag{4.16}$$

其中， o_i 表示（目标语言）第 i 个短语的调序方向， $\mathbf{o} = \{o_i\}$ 表示短语序列的调序方向， K 表示短语的数量。短语之间的调序概率是由双语短语以及短语对齐决定的， o 表示调序的种类，可以取 M、S、D 中的任意一种。而整个句子调序的好坏就是把相邻的短语之间的调序概率相乘（对应取 \log 后的加法）。这样，公式4.16把调序的好坏定义为新的特征，对于 M、S、D 总共就有三个特征。除了当前短语和前一个短语的调序特征，还可以定义当前短语和后一个短语的调序特征，即将上述公式中的 a_{i-1} 换成 a_{i+1} 。于是，又可以得到三个特征。因此在 MSD 调序中总共可以有 6 个特征。

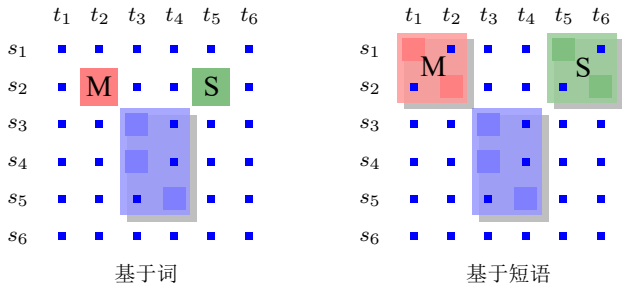


图 4.22: 调序类型的判断

具体实现时，通常使用词对齐对两个短语间的调序关系进行判断。图4.22展示了这个过程。先判断短语的左上角和右上角是否存在词对齐，再根据其位置对调序类型进行划分。每个短语对应的调序概率都可以用相对频率估计进行计算。而 MSD 调

序模型也相当于在短语表中的每个双语短语后添加 6 个特征。不过，调序模型一般并不会和短语表一起存储，因此在系统中通常会看到两个独立的模型文件，分别保存短语表和调序模型。

基于分类的调序

在 MSD 调序中，双语短语所对应的调序概率 $P(o_i | \bar{s}_{a_i}, \bar{t}_i, a_{i-1}, a_i)$ 是用极大似然估计方法进行计算的。但是，这种方法也会面临数据稀疏问题，同时对调序产生影响的细致特征也没有考虑进来。另一种有效的方法是直接用统计分类模型对调序进行建模，比如，可以使用最大熵、SVM 等分类器输出调序概率或者得分 [159, 223, 326]。对于基于分类的调序模型，有两方面问题需要考虑：

- 训练样本的生成。可以把 M、S、D 看作是类别标签，把所对应的短语及短语对齐信息看作是输入。这样就得到了大量分类器训练所需的样本；
- 分类特征设计。这部分是传统统计机器学习中的重要组成部分，好的特征会对分类结果产生很大影响。在调序模型中，一般直接使用单词作为特征，比如用短语的第一个单词和最后一个单词作为特征就可以达到很好的效果。

随着神经网络方法的兴起，也可以考虑使用多层神经网络构建调序模型 [169]。这时，可以把短语直接送入一个神经网络，之后由神经网络完成对特征的抽取和表示，并输出最终的调序模型得分。

4.2.5 特征

基于短语的模型使用判别式模型对翻译推导进行建模，给定双语句对 (s, t) ，每个翻译推导 d 都有一个模型得分，由 M 个特征线性加权得到，记为 $\text{score}(d, t, s) = \sum_{i=1}^M \lambda_i \cdot h_i(d, t, s)$ ，其中 λ_i 表示特征权重， $h_i(d, t, s)$ 表示特征函数（简记为 $h_i(d)$ ）。这些特征包含刚刚介绍过的短语翻译概率、调序模型得分等，除此之外，还包含语言模型等其他特征，它们共同组成了特征集合。这里列出了基于短语的模型中常用的特征：

- 短语翻译概率（取对数），包含正向翻译概率 $\log(P(\bar{t}|\bar{s}))$ 和反向翻译概率 $\log(P(\bar{s}|\bar{t}))$ ，它们是基于短语的模型中最主要的特征；
- 词汇化翻译概率（取对数），同样包含正向词汇化翻译概率 $\log(P_{\text{lex}}(\bar{t}|\bar{s}))$ 和反向词汇化翻译概率 $\log(P_{\text{lex}}(\bar{s}|\bar{t}))$ ，它们用来描述双语短语中单词间对应的好坏；
- n -gram 语言模型，用来度量译文的流畅程度，可以通过大规模目标端单语数据得到；
- 译文长度，避免模型倾向于短译文，同时让系统自动学习对译文长度的偏好；
- 翻译规则数量，为了避免模型仅使用少量特征构成翻译推导（规则数量少，短语翻译概率相乘的因子也会少，得分一般会大一些），同时让系统自动学习对

规则数量的偏好；

- 被翻译为空的源语言单词数量。注意，空翻译规则有时也被称作 **evil feature**，这类特征在一些数据上对 BLEU 有很好的提升作用，但会造成人工评价结果的下降，需要谨慎使用；
- 基于 MSD 的调序模型，包括与前一个短语的调序模型 $f_{M-pre}(d)$ 、 $f_{S-pre}(d)$ 、 $f_{D-pre}(d)$ 和与后一个短语的调序模型 $f_{M-fol}(d)$ 、 $f_{S-fol}(d)$ 、 $f_{D-fol}(d)$ ，共 6 个特征。

4.2.6 最小错误率训练

除了特征设计，统计机器翻译也需要找到每个特征所对应的最优权重 λ_i 。这也就是机器学习中所说的模型训练问题。不过，需要指出的是，统计机器翻译关于模型训练的定义与传统机器学习稍有不同。在统计机器翻译中，短语抽取和翻译概率的估计被看作是**模型训练**（Model Training），也就是说这里的模型训练是指特征函数的学习；而特征权重的训练，一般被称作**权重调优**（Weight Tuning），而这个过程才真正对应了传统机器学习（如分类任务）中的模型训练过程。在本章中，如果没有特殊说明，权重调优就是指特征权重的学习，模型训练是指短语抽取和特征函数的学习。

想要得到最优的特征权重，最简单的方法是枚举所有的特征权重可能的取值，然后评价每组权重所对应的翻译性能，最后选择最优的特征权重作为调优的结果。但是特征权重是一个实数值，因此可以考虑把实数权重进行量化，即把权重看作是在固定间隔上的取值，比如，每隔 0.01 取值。即使是这样，同时枚举多个特征的权重也是非常耗时的工作，当特征数量增多时这种方法的效率仍然很低。

这里介绍一种更加高效的特征权重调优方法——**最小错误率训练**（Minimum Error Rate Training, MERT）。最小错误率训练是统计机器翻译发展中代表性工作，也是从机器翻译中原创的重要技术方法之一 [216]。最小错误率训练假设：翻译结果相对于标准答案的错误是可度量的，进而可以通过降低错误数量的方式来找到最优的特征权重。假设有样本集合 $S = \{(s_1, \mathbf{r}_1), \dots, (s_N, \mathbf{r}_N)\}$ ， s_i 为样本中第 i 个源语言句子， \mathbf{r}_i 为相应的参考译文。注意， \mathbf{r}_i 可以包含多个参考译文。 S 通常被称为**调优集合**（Tuning Set）。对于 S 中的每个源语句子 s_i ，机器翻译模型会解码出 n -best 推导 $d_{ij}^* = \{\mathbf{d}_{ij}^*\}$ ，其中 d_{ij}^* 表示翻译源语言句子 s_i 时得到的第 j 个最好的推导。 $\{d_{ij}^*\}$ 可以被定义如下：

$$\{d_{ij}^*\} = \arg \max_{\{d_{ij}\}} \sum_{i=1}^M \lambda_i \cdot h_i(d, \mathbf{t}, \mathbf{s}) \quad (4.17)$$

对于每个样本都可以得到 n -best 推导集合，整个数据集上的推导集合被记为 $\mathbf{D}^* = \{\mathbf{d}_1^*, \dots, \mathbf{d}_s^*\}$ 。进一步，令所有样本的参考译文集合为 $\mathbf{R} = \{\mathbf{r}_1, \dots, \mathbf{r}_N\}$ 。最小错误率训练的目标就是降低 \mathbf{D}^* 相对于 \mathbf{R} 的错误。也就是，通过调整不同特征的权重

$\lambda = \{\lambda_i\}$, 让错误率最小, 形式化描述为:

$$\lambda^* = \arg \min_{\lambda} \text{Error}(\mathbf{D}^*, \mathbf{R}) \quad (4.18)$$

其中 $\text{Error}(\cdot)$ 是错误率函数。 $\text{Error}(\cdot)$ 的定义方式有很多, 一般来说 $\text{Error}(\cdot)$ 会与机器翻译的评价指标相关, 例如, 词错误率 (WER)、位置错误率 (PER)、BLEU 值、NIST 值等都可以用于 $\text{Error}(\cdot)$ 的定义。这里使用 1-BLEU 作为错误率函数, 即 $\text{Error}(\mathbf{D}^*, \mathbf{R}) = 1 - \text{BLEU}(\mathbf{D}^*, \mathbf{R})$ 。则公式 4.18 可改写为:

$$\begin{aligned} \lambda^* &= \arg \min_{\lambda} 1 - \text{BLEU}(\mathbf{D}^*, \mathbf{R}) \\ &= \arg \max_{\lambda} \text{BLEU}(\mathbf{D}^*, \mathbf{R}) \end{aligned} \quad (4.19)$$

需要注意的是, BLEU 本身是一个不可微分函数。因此, 无法使用梯度下降等方法对式 4.19 进行求解。那么如何能快速得到最优解? 这里会使用一种特殊的优化方法, 称作**线搜索** (Line Search), 它是 Powell 搜索的一种形式 [233]。这种方法也构成了最小错误率训练的核心。

首先, 重新看一下特征权重的搜索空间。按照前面的介绍, 如果要进行暴力搜索, 需要把特征权重的取值按小的间隔进行划分。这样, 所有特征权重的取值可以用图 4.23 的网格来表示。

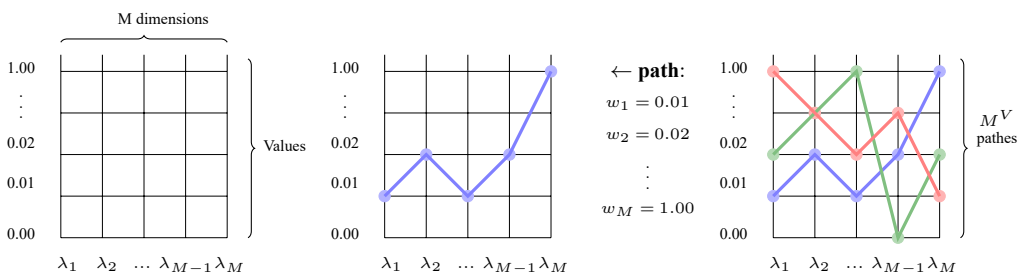


图 4.23: 特征权重的搜索空间表示

其中横坐标为所有的 M 个特征函数, 纵坐标为权重可能的取值。假设每个特征都有 V 种取值, 那么遍历所有特征权重取值的组合有 M^V 种。每组 $\lambda = \{\lambda_i\}$ 的取值实际上就是一个贯穿所有特征权重的折线, 如图 4.23 中间红线所展示的路径。当然, 可以通过枚举得到很多这样的折线 (图 4.23 右)。假设计算 BLEU 的时间开销为 B , 那么遍历所有路径的时间复杂度为 $O(M^V \cdot B)$, 由于 V 可能很大, 而且 B 往往也无法忽略, 因此这种计算方式的时间成本是极高的。

对全搜索的一种改进是使用局部搜索。循环处理每个特征, 每一次只调整一个特征权重的值, 找到使 BLEU 达到最大的权重。反复执行该过程, 直到模型达到稳定状态 (例如 BLEU 不再降低)。

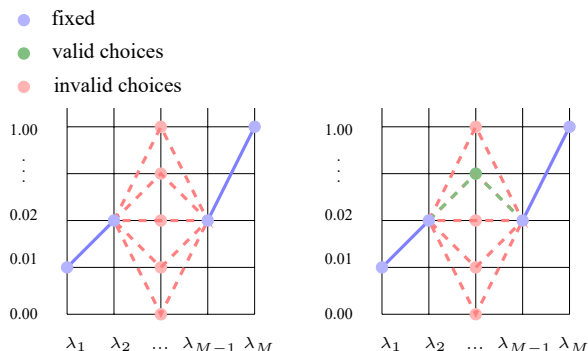


图 4.24: 格搜索（左侧：所有点都访问（蓝色）；右侧：避开无效点（绿色））

图4.24左侧展示了这种方法。其中红色部分为固定住的权重，相应的虚线部分为当前权重所有可能的取值，这样搜索一个特征权重的时间复杂度为 $O(V \cdot B)$ 。而整个算法的时间复杂度为 $O(L \cdot V \cdot B)$ ，其中 L 为循环访问特征的总次数。这种方法也被称作**格搜索**（Grid Search）。

格搜索的问题在于，每个特征都要访问 V 个点，且不说 V 个点无法对连续的特征权重进行表示，里面也会存在大量的无用访问。也就是说，这 V 个点中绝大多数点根本“不可能”成为最优的权重。可以把这样的点称为无效取值点。

能否避开这些无效的权重取值点呢？再重新看一下优化的目标 BLEU。实际上，当一个特征权重发生变化时，BLEU 的变化只会产生在系统 1-best 译文发生变化的时候。那么，可以只关注使 1-best 译文发生变化的取值点，而其他的取值点都不会对优化的目标函数产生变化。这也就构成了线搜索的思想。

假设对于每个输入的句子，翻译模型生成了两个推导 $\mathbf{d} = \{d_1, d_2\}$ ，每个推导 d 的得分 $\text{score}(d)$ 可以表示成关于第 i 个特征的权重 λ_i 的线性函数：

$$\begin{aligned}
 \text{score}(d) &= \sum_{k=1}^M \lambda_k \cdot h_k(d) \\
 &= h_i(d) \cdot \lambda_i + \sum_{k \neq i} \lambda_k \cdot h_k(d) \\
 &= a \cdot \lambda_i + b
 \end{aligned} \tag{4.20}$$

这里， $a = h_i(d)$ 是直线的斜率， $b = \sum_{k \neq i} \lambda_k \cdot h_k(d)$ 是截距。有了关于权重 λ_i 的直线表示，可以将 d_1 和 d_2 分别画成两条直线，如图4.25所示。在两条直线交叉点的左侧， d_2 是最优的翻译结果；在交叉点右侧， d_1 是最优的翻译结果。也就是说，只需知道交叉点左侧和右侧谁的 BLEU 值高， λ_i 的最优值就应该落在相应的范围，比如，这个例子中交叉点右侧（即 d_2 ）所对应的 BLEU 值更高，因此最优特征权重应该在交叉点右侧（ $\lambda_x \sim \lambda_i$ 任意取值都可以）。

这样，最优权重搜索的问题就被转化为找到最优推导 BLEU 值发生变化的点的

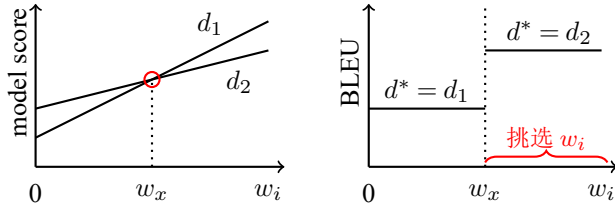


图 4.25: 推导得分关于权重的函数以及对应的 BLEU 值变化

问题。理论上，对于 n -best 翻译，交叉点计算最多需要 $\frac{n(n-1)}{2}$ 次。由于 n 一般不会过大，这个时间成本完全是可以接受的。此外，在实现时还有一些技巧，比如，并不需要在每个交叉点处对整个数据集进行 BLEU 计算，可以只对 BLEU 产生变化的部分（比如 n -gram 匹配的数量）进行调整，因此搜索的整体效率会进一步得到提高。相比于格搜索，线搜索可以确保在单个特征维度上的最优值，同时保证搜索的效率。

还有一些经验性的技巧用来完善基于线搜索的 MERT。例如：

- 随机生成特征权重的起始点；
- 搜索中，给权重加入一些微小的扰动，避免陷入局部最优；
- 随机选择特征优化的顺序；
- 使用先验知识来指导 MERT（对权重的取值范围进行约束）；
- 使用多轮迭代训练，最终对权重进行平均。

MERT 最大的优点在于可以用于目标函数不可微、甚至不连续的情况。对于优化线性模型，MERT 是一种很好的选择。但是，也有研究发现，简单使用 MERT 无法处理特征数量过多的情况。比如，用 MERT 优化 10000 个稀疏特征的权重时，优化效果可能会不理想，而且收敛速度慢。这时也可以考虑使用在线学习等技术对大量特征的权重进行调优，比较有代表性的方法包括 MIRA[42] 和 PRO[113]。由于篇幅所限，这里不对这些方法做深入讨论，感兴趣的读者可以参考 4.5 节的内容，对相关文献进行查阅。

4.2.7 栈解码

解码的目的是根据模型以及输入，找到模型得分最高的推导，即：

$$\hat{d} = \arg \max_d \text{score}(d, \mathbf{t}, \mathbf{s}) \quad (4.21)$$

然而想要找到得分最高的翻译推导并不是一件简单的事情。对于每一句源语言句子，可能的翻译结果是指数级的。而机器翻译解码也已经被证明是一个 NP 难问题 [145]。简单的暴力搜索显然不现实。因此，在机器翻译中会使用特殊的解码策略来

确保搜索的效率。本节将介绍基于栈的自左向右解码方法。它是基于短语的模型中的经典解码方法，非常适于处理语言生成的各种任务。

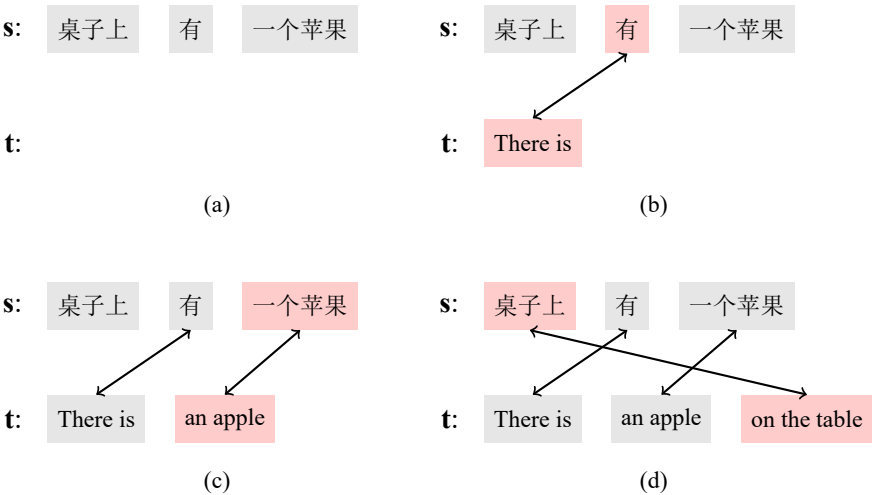


图 4.26: 翻译的基本流程

首先，看一下翻译一个句子的基本流程。如图4.26所示，首先需要得到译文句子的第一个单词。在基于短语的模型中，可以从源语言端找出生成句首译文的短语，之后把译文放到目标语言端，例如，源语言的“有”对应的译文是“**There is**”。这个过程可以重复执行，直到生成完整句子的译文。但是，有两点需要注意：

- 源语言的每个单词（短语）只能被翻译一次；
- 译文的生成需自左向右连续进行。

前者对应了一种**覆盖度模型**（Coverage Model）；后者定义了解码的方向，这样可以确保 n -gram 语言模型的计算是准确的。这样，就得到了一个简单的基于短语的机器翻译解码框架。每次从源语言句子中找到一个短语，作为译文最右侧的部分，重复执行直到整个译文被生成出来。

翻译候选匹配

在解码时，首先要知道每个源语言短语可能的译文都是什么。对于一个源语言短语，每个可能的译文也被称作**翻译候选**（Translation Candidate）。实现翻译候选的匹配很简单。只需要遍历输入的源语言句子中所有可能的短语，之后在短语表中找到相应的翻译即可。比如，图4.27展示了句子“桌子上 有 一个 苹果”的翻译候选匹配结果。可以看到，不同的短语会对应若干翻译候选。这些翻译候选会保存在所对应的跨度中。比如，“upon the table”是短语“桌子上有”的翻译候选，即对应源语言跨度 [0,3]。

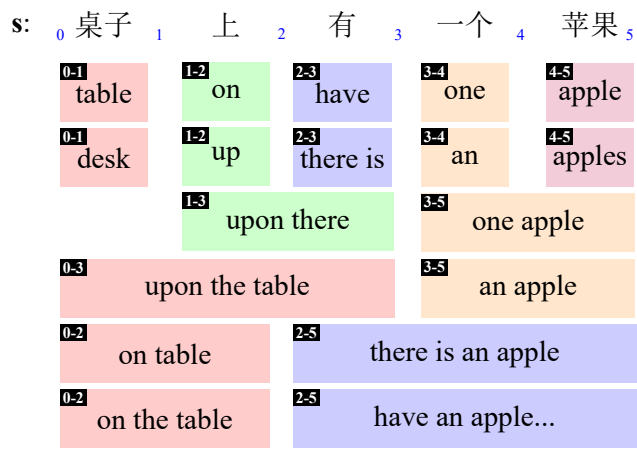


图 4.27: 一个句子匹配的短语翻译候选

翻译假设扩展

下一步，需要使用这些翻译候选生成完整的译文。在机器翻译中，一个很重要的概念是**翻译假设**（Translation Hypothesis）。它可以被当作是一个局部译文所对应的短语翻译推导。在解码开始时，只有一个空假设，也就是任何译文单词都没有被生成出来。接着，可以挑选翻译选项来扩展当前的翻译假设。

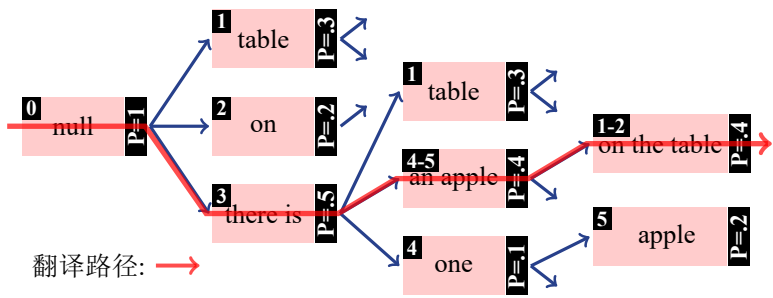


图 4.28: 翻译假设扩展

图4.28展示了翻译假设扩展的过程。在翻译假设扩展时，需要保证新加入的翻译候选放置在旧翻译假设译文的右侧，也就是要确保翻译自左向右的连续性。而且，同一个翻译假设可以使用不同的翻译候选进行扩展。例如，扩展第一个翻译假设时，可以选择“桌子”的翻译候选“table”；也可以选择“有”的翻译候选“there is”。扩展完之后需要记录输入句子中已翻译的短语，同时计算当前所有翻译假设的模型得分。这个过程相当于生成了一个图的结构，每个节点代表了一个翻译假设。当翻译假设覆盖了输入句子所有的短语，不能被继续扩展时，就生成了一个完整的翻译假设（译文）。最后需要找到得分最高的完整翻译假设，它对应了搜索图中的最优路径。

剪枝

假设扩展建立了解码算法的基本框架。但是，当句子变长时，这种方法还是面临着搜索空间爆炸的问题。对于这个问题，常用的解决办法是**剪枝**（Pruning），也就是在搜索图中排除掉一些节点。比如，可以使用**束剪枝**（Beam Pruning），确保每次翻译扩展时，最多生成 k 个新的翻译假设。这里 k 可以被看做是束的宽度。通过控制 k 的大小，可以在解码精度和速度之间进行平衡。这种基于束宽度进行剪枝的方法也被称作**直方图剪枝**（Histogram Pruning）。另一种思路是，每次扩展时只保留与最优翻译假设得分相差在 δ 之内的翻译假设。 δ 可以被看作是一种与最优翻译假设之间距离的阈值，超过这个阈值就被剪枝。这种方法也被称作**阈值剪枝**（Threshold Pruning）。

不过，即使引入束剪枝，解码过程中仍然会有很多冗余的翻译假设。有两种方法可以进一步加速解码：

- 对相同译文的翻译假设进行重新组合；
- 对低质量的翻译假设进行裁剪。

对翻译假设进行重新组合又被称作**假设重组**（Hypothesis Recombination）。其核心理念是，把代表同一个译文的不同翻译假设融合为一个翻译假设。如图 29 所示，对于给定的输入短语“一个 苹果”，系统可能将两个单词“一个”、“苹果”分别翻译成“an”和“apple”，也可能将这两个单词作为一个短语直接翻译成“an apple”。虽然这两个翻译假设得到的译文相同，并且覆盖了相同的源语言短语，但是却是两个不同的翻译假设，模型给它们的打分也是不一样的。这时，可以舍弃两个翻译假设中分数较低的那个，因为分数较低的翻译假设永远不可能成为最优路径的一部分。这也就相当于把两个翻译假设重组为一个假设。

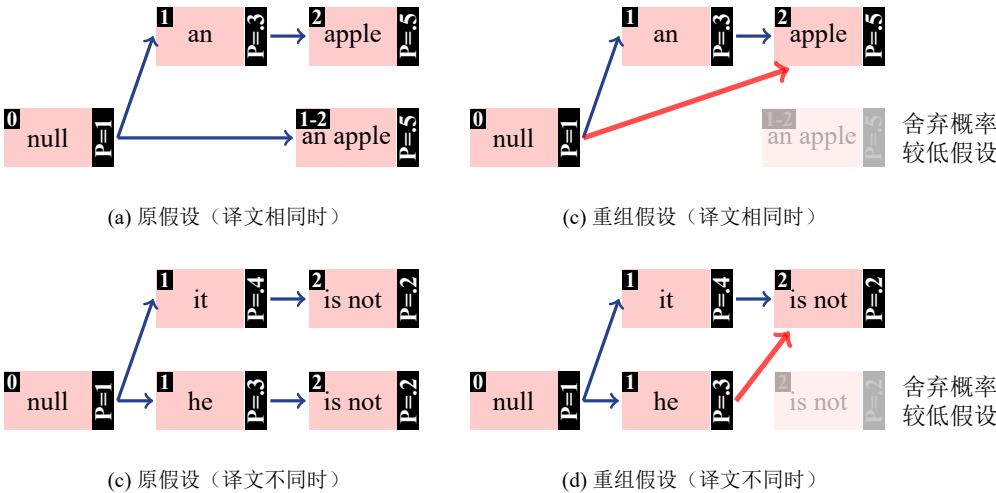


图 4.29: 假设重组示例

即使翻译假设对应的译文不同也可以进行假设重组。图4.29的下半部分给出了一个这样的实例。在两个翻译假设中，第一个单词分别被翻译成了“it”和“he”，紧接着它们后面的部分都被翻译成了“is not”。这两个翻译假设是非常相似的，因为它们译文的最后两个单词是相同的，而且翻译假设都覆盖了相同的源语言部分。这时，也可以对这两个翻译假设进行假设重组：如果得分较低的翻译假设和得分较高的翻译假设都使用相同的翻译候选进行扩展，且两个翻译假设都覆盖相同的源语言单词，分数低的翻译假设可以被剪枝掉。此外，还有两点需要注意：

- n -gram 语言模型将前 $n-1$ 单词作为历史信息，所以当两个假设最后 $n-1$ 个单词不相同，不能进行假设重组，因为后续的扩展可能会得到不同的语言模型得分，并影响最终的模型得分；
- 调序模型通常是用来判断当前输入的短语与前一个输入短语之间的调序代价。因此当两个翻译假设对应短语在源语言中的顺序不同时，也不能被重新组合。

然而在实际处理中，并不需要“删掉”分数低的翻译假设，而是将它们与分数高的翻译假设连在了一起。对于搜索最优翻译，这些连接可能并没有什么作用，但是如果需要分数最高的前两个或前三个翻译，就可能需要用到这些连接。

翻译假设的重组有效地减少了解码过程中相同或者相似翻译假设带来的冗余。因此这些方法在机器翻译中被广泛使用。包括本章后面将要介绍的基于句法的翻译模型解码中，也可以使用假设重组进行系统加速。

解码中的栈结构

当质量较差的翻译假设在扩展早期出现时，这些翻译假设需要被剪枝掉，这样可以忽略所有从它扩展出来的翻译假设，进而有效地减小搜索空间。但是这样做也存在一定的问题，首先，删除的翻译假设可能会在后续的扩展过程中被重新搜索出来。其次，过早地删除某些翻译假设可能会导致无法搜索到最优的翻译假设。所以最好的情况是尽早删除质量差的翻译假设，同时又不会对整个搜索结果产生过大影响。但是这个“质量”从哪个方面来衡量，也是一个需要思考的问题。理想的情况就是从早期的翻译假设中，挑选一些可比的翻译假设进行筛选。

目前比较通用的做法是将翻译假设进行整理，放进一种栈结构中。这里所说的“栈”是为了描述方便的一种说法。它实际上就是保存多个翻译假设的一种数据结构⁴。当放入栈的翻译假设超过一定阈值时（比如 200），可以删除掉模型得分低的翻译假设。一般，会使用多个栈来保存翻译假设，每个栈代表覆盖源语言单词数量相同的翻译假设。

比如，第一个堆栈包含了覆盖一个源语言单词的翻译假设，第二个堆栈包含了覆盖两个源语言单词的翻译假设，以此类推。利用覆盖源语言单词数进行栈的划分的原因在于：翻译相同数量的单词所对应的翻译假设一般是“可比的”，因此在同一

⁴虽然被称作栈，实际上使用一个堆进行实现。这样可以根据模型得分对翻译假设进行排序。

个栈里对它们进行剪枝带来的风险较小。

在基于栈的解码中，每次都会从所有的栈中弹出一个翻译假设，并选择一个或者若干个翻译假设进行扩展，之后把新得到的翻译假设重新压入解码栈中。这个过程不断执行，并可以配合束剪枝、假设重组等技术。最后在覆盖所有源语言单词的栈中得到整个句子的译文。图4.30展示了一个简单的栈解码过程。第一个栈（0号栈）用来存放空翻译假设。之后通过假设扩展，不断将翻译假设填入对应的栈中。

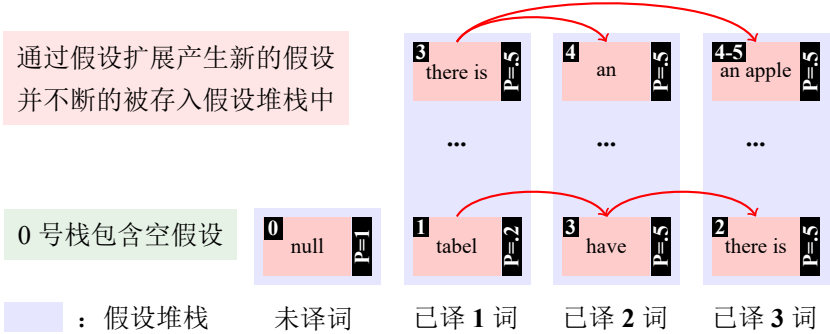


图 4.30: 栈解码示例

4.3 基于层次短语的模型

在机器翻译中，如果翻译需要局部上下文的信息，把短语作为翻译单元是一种理想的方案。但是，单词之间的关系并不总是“局部”的，很多时候需要距离更远一些的搭配。比较典型的例子是含有从句的情况。比如：

我 在 今天 早上 没有 吃 早饭 的 情况 下 还是 正常 **去 上班** 了。

这句话的主语“我”和谓语“去 上班”构成了主谓搭配，而二者之间的部分是状语。显然，用短语去捕捉这个搭配需要覆盖很长的词串，也就是整个“我 ... 去上班”的部分。如果把这样的短语考虑到建模中，会面临非常严重的数据稀疏问题，因为无法保证这么长的词串在训练数据中能够出现。

实际上，随着短语长度变长，短语在数据中会变得越来越低频，相关的统计特征也会越来越不可靠。表4.1就展示了不同长度的短语在训练数据中出现的频次。可以看到，长度超过 3 的短语已经非常低频了，更长的短语甚至在训练数据中一次也没有出现过。

显然，利用过长的短语来处理长距离的依赖并不是一种十分有效的方法。过于低频的长短语无法提供可靠的信息，而且使用长短语会导致模型体积急剧增加。

再来看一个翻译实例 [40]，图4.31是一个基于短语的机器翻译系统的翻译结果。这个例子中的调序有一些复杂，比如，“少数 国家 之一”和“与 北韩 有 邦交”的英文

表 4.1: 不同短语在训练数据中出现的频次

短语 (中文)	训练数据中出现的频次
包含	3341
包含 多个	213
包含 多个 词	12
包含 多个 词 的	8
包含 多个 词 的 短语	0
包含 多个 词 的 短语 太多	0

翻译都需要进行调序，分别是 “one of the few countries” 和 “have diplomatic relations with North Korea”。基于短语的系统可以很好地处理这些调序问题，因为它们仅仅使用了局部的信息。但是，系统却无法在这两个短语（1 和 2）之间进行正确的调序。

源语：澳洲 是 与 北韩 有 邦交 的 少数 国家 之一

短语系统：Australia is diplomatic relations with North Korea **1**
is one of the few countries **2**

参考译文：Australia is one of the few countries that have
diplomatic relations with North Korea

图 4.31: 基于短语的机器翻译实例 [40]

这个例子也在一定程度上说明了长距离的调序需要额外的机制才能得到更好地被处理。实际上，两个短语（1 和 2）之间的调序现象本身对应了一种结构，或者说模板。也就是汉语中的：

与 [什么东西] 有 [什么事]

可以翻译成：

have [什么事] with [什么东西]

这里 [什么东西] 和 [什么事] 表示模板中的变量，可以被其他词序列替换。通常，可以把这个模板形式化描述为：

⟨ 与 X_1 有 X_2 , have X_2 with X_1 ⟩

其中，逗号分隔了源语言和目标语言部分， X_1 和 X_2 表示模板中需要替换的内容，或者说变量。源语言中的变量和目标语言中的变量是一一对应的，比如，源语言中的

X_1 和目标语言中的 X_1 代表这两个变量可以“同时”被替换。假设给定短语对：

- 〈 北韩, North Korea 〉
- 〈 邦交, diplomatic relations 〉

可以使用第一个短语替换模板中的变量 X_1 ，得到：

〈 与 [北韩] 有 X_2 , have X_2 with [North Korea] 〉

其中，[·] 表示被替换的部分。可以看到，在源语言和目标语言中， X_1 被同时替换为相应的短语。进一步，可以用第二个短语替换 X_2 ，得到：

〈 与 北韩 有 [邦交], have [diplomatic relations] with North Korea 〉

至此，就得到了一个完整词串的译文。类似的，还可以写出其他的翻译模板，如下：

- 〈 X_1 是 X_2 , X_1 is X_2 〉
- 〈 X_1 之一, one of X_1 〉
- 〈 X_1 的 X_2 , X_2 that have X_1 〉

使用上面这种变量替换的方式，就可以得到一个完整句子的翻译。

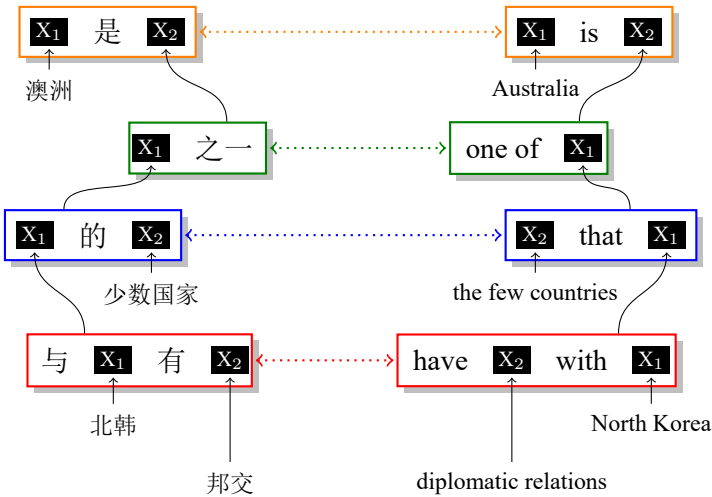


图 4.32: 使用短语和翻译模板进行双语句子的同步生成

这个过程如图4.32所示。其中，左右相连接的方框表示翻译模版的源语言和目标语言部分。可以看到，模版中两种语言中的变量会被同步替换，替换的内容可以是

其他模版生成的结果。这也就对应了一种层次结构，或者说互译的句对可以被双语的层次结构同步生成出来。

实际上，在翻译中使用这样的模版就构成了层次短语模型的基本思想。下面就一起来看看如何对翻译模版进行建模，以及如何自动学习并使用这些模版。

4.3.1 同步上下文无关文法

基于层次短语的模型（Hierarchical Phrase-based Model）是 David Chiang 于 2005 提出的统计机器翻译模型 [37, 38]。这个模型可以很好地解决短语系统对翻译中长距离调序建模不足的问题。基于层次短语的系统也在多项机器翻译比赛中取得了很好的成绩。这项工作也获得了自然处理领域顶级会议 ACL2015 的最佳论文奖。

层次短语模型的核心是把翻译问题归结为两种语言词串的同步生成问题。实际上，词串的生成问题是自然语言处理中的经典问题，早期的研究更多的是关注单语句子的生成，比如，如何使用句法树描述一个句子的生成过程。层次短语模型的创新之处是把传统单语词串的生成推广到双语词串的同步生成上。这使得机器翻译可以使用类似句法分析的方法进行求解。

文法定义

层次短语模型中一个重要的概念是**同步上下文无关文法**（Synchronous Context-free Grammar，简称 SCFG）。SCFG 可以被看作是对源语言和目标语言上下文无关文法的融合，它要求源语言和目标语言的产生式及产生式中的变量具有对应关系。具体定义如下：

定义 4.3.1 同步上下文无关文法

一个同步上下文无关文法由五部分构成 (N, T_s, T_t, I, R) ，其中：

1. N 是非终结符集合。
2. T_s 和 T_t 分别是源语言和目标语言的终结符集合。
3. $I \subseteq N$ 起始非终结符集合。
4. R 是规则集合，每条规则 $r \in R$ 有如下形式：

$$\text{LHS} \rightarrow \langle \alpha, \beta, \sim \rangle$$

其中， $\text{LHS} \in N$ 表示规则的左部，它是一个非终结符；规则的右部由三部分组成， $\alpha \in (N \cup T_s)^*$ 表示由源语言终结符和非终结符组成的串； $\beta \in (N \cup T_t)^*$ 表示由目标语言终结符和非终结符组成的串； \sim 表示 α 和 β 中非终结符的 1-1 对应关系。

根据这个定义，源语言和目标语言有不同的终结符集合（单词），但是它们会共享同一个非终结符集合（变量）。每个产生式包括源语言和目标语言两个部分，分别表示由规则左部生成的源语言和目标语言符号串。由于产生式会同时生成两种语言的符号串，因此这是一种“同步”生成，可以很好地描述翻译中两个词串之间的对应。

下面是一个简单的 SCFG 实例：

$$\begin{aligned} S &\rightarrow \langle NP_1 \text{ 希望 } VP_2, \quad NP_1 \text{ wish to } VP_2 \rangle \\ VP &\rightarrow \langle \text{对 } NP_1 \text{ 感到 } VP_2, \quad \text{be } VP_2 \text{ wish } NP_1 \rangle \\ NN &\rightarrow \langle \text{强大,} \quad \text{strong} \rangle \end{aligned}$$

这里的 S、NP、VP 等符号可以被看作是具有句法功能的标记，因此这个文法和传统句法分析中的 CFG 很像，只是 CFG 是单语语法，而 SCFG 是双语同步语法。当然，复杂的句法功能标记并不是必须的。比如，也可以使用更简单的文法形式：

$$\begin{aligned} X &\rightarrow \langle X_1 \text{ 希望 } X_2, \quad X_1 \text{ wish to } X_2 \rangle \\ X &\rightarrow \langle \text{对 } X_1 \text{ 感到 } X_2, \quad \text{be } X_2 \text{ wish } X_1 \rangle \\ X &\rightarrow \langle \text{强大,} \quad \text{strong} \rangle \end{aligned}$$

这个文法只有一种非终结符 X，因此所有的变量都可以使用任意的产生式进行推导。这就给翻译提供了更大的自由度，也就是说，规则可以被任意使用，进行自由组合。这也符合基于短语的模型中对短语进行灵活拼接的思想。基于此，层次短语系统也使用这种并不依赖语言学句法标记的文法。在本书的内容中，如果没有特殊说明，本章中把这种没有语言学句法标记的文法称作**基于层次短语的文法**（Hierarchical Phrase-based Grammar），或简称层次短语文法。

推导

下面是一个完整的层次短语文法：

$$\begin{aligned} r_1: \quad X &\rightarrow \langle \text{进口 } X_1, \quad \text{The imports } X_1 \rangle \\ r_2: \quad X &\rightarrow \langle X_1 \text{ 下降 } X_2, \quad X_2 \text{ } X_1 \text{ fallen} \rangle \\ r_3: \quad X &\rightarrow \langle \text{大幅度,} \quad \text{drastically} \rangle \\ r_4: \quad X &\rightarrow \langle \text{了,} \quad \text{have} \rangle \end{aligned}$$

其中，规则 r_1 和 r_2 是含有变量的规则，这些变量可以被其他规则的右部替换；规则 r_2 是调序规则；规则 r_3 和 r_4 是纯词汇化规则，表示单词或者短语的翻译。

对于一个双语句对：

源语： 进口 大幅度 下降 了
目标语： The imports have drastically fallen

可以进行如下的推导（假设起始符号是 X ）：

$$\begin{aligned}
 & \langle X_1, X_1 \rangle \\
 \xrightarrow{r_1} & \langle \text{进口 } X_2, \text{The imports } X_2 \rangle \\
 \xrightarrow{r_2} & \langle \text{进口 } X_3 \text{ 下降 } X_4, \text{The imports } X_4 \text{ } X_3 \text{ fallen} \rangle \\
 \xrightarrow{r_3} & \langle \text{进口 大幅度 下降 } X_4, \\
 & \text{The imports } X_4 \text{ drastically fallen} \rangle \\
 \xrightarrow{r_4} & \langle \text{进口 大幅度 下降 了,} \\
 & \text{The imports have drastically fallen} \rangle
 \end{aligned}$$

其中，每使用一次规则就会同步替换源语言和目标语言符号串中的一个非终结符。通常，可以把上面这个过程称作翻译**推导**（Derivation），记为：

$$d = r_1 \circ r_2 \circ r_3 \circ r_4 \quad (4.22)$$

在层次短语模型中，每个翻译推导都唯一的对应一个目标语译文。因此，可以用推导的概率 $P(d)$ 描述翻译的好坏。同基于短语的模型是一样的（见4.2.2节），层次短语翻译的目标是：求概率最高的翻译推导 $\hat{d} = \arg \max P(d)$ 。值得注意的是，基于推导的方法在句法分析中也十分常用。层次短语翻译实质上也是通过生成翻译规则的推导来对问题的表示空间进行建模。在4.4节还将看到，这种方法可以被扩展到语言学上基于句法的翻译模型中。而且这些模型都可以用一种被称作超图的结构来进行建模。从某种意义上讲，基于规则推导的方法将句法分析和机器翻译进行了形式上的统一。因此机器翻译也借用了许多句法分析的思想。

胶水规则

由于翻译现象非常复杂，在实际系统中往往需要把两个局部翻译线性拼接到一起。在层次短语模型中，这个问题通过引入**胶水规则**（Glue Rule）来处理，形式如下：

$$\begin{aligned}
 S & \rightarrow \langle S_1 X_2, S_1 X_2 \rangle \\
 S & \rightarrow \langle X_1, X_1 \rangle
 \end{aligned}$$

胶水规则引入了一个新的非终结符 S ， S 只能和 X 进行顺序拼接，或者 S 由 X 生成。如果把 S 看作文法的起始符，使用胶水规则后，相当于把句子划分为若干部分，每个部分都被归纳为 X 。之后，顺序地把这些 X 拼接到一起，得到最终的译文。比如，最极端的情况，整个句子会生成一个 X ，之后再归纳为 S ，这时并不需要进行胶水规则的顺序拼接；另一种极端的情况，每个单词都是独立的被翻译，被归纳为 X ，之后先把最左边的 X 归纳为 S ，再依次把剩下的 X 依次拼到一起。这样的

推导形式如下：

$$\begin{aligned} S &\rightarrow \langle S_1 X_2, S_1 X_2 \rangle \\ &\rightarrow \langle S_3 X_4 X_2, S_3 X_4 X_2 \rangle \\ &\rightarrow \dots \\ &\rightarrow \langle X_n \dots X_4 X_2, X_n \dots X_4 X_2 \rangle \end{aligned}$$

实际上，胶水规则在很大程度上模拟了基于短语的系统对字符串顺序翻译的操作，而且在实践中发现，这个步骤是十分必要的。特别是对法-英翻译这样的任务，由于语言的结构基本上是顺序翻译的，因此引入顺序拼接的操作符合翻译的整体规律。同时，这种拼接给翻译增加了灵活性，系统会更加健壮。

需要说明的是，使用同步文法进行翻译时由于单词的顺序是内嵌在翻译规则内的，因此这种模型并不依赖额外的调序模型。一旦文法确定下来，系统就可以进行翻译。

处理流程

层次短语系统的流程如图4.33所示。其核心是从双语数据中学习同步翻译文法，并进行翻译特征的学习，形成翻译模型（即规则 + 特征）。同时，要从目标语言数据中学习语言模型。最终，把翻译模型和语言模型一起送入解码器，在特征权重调优后，完成对新输入句子的翻译。

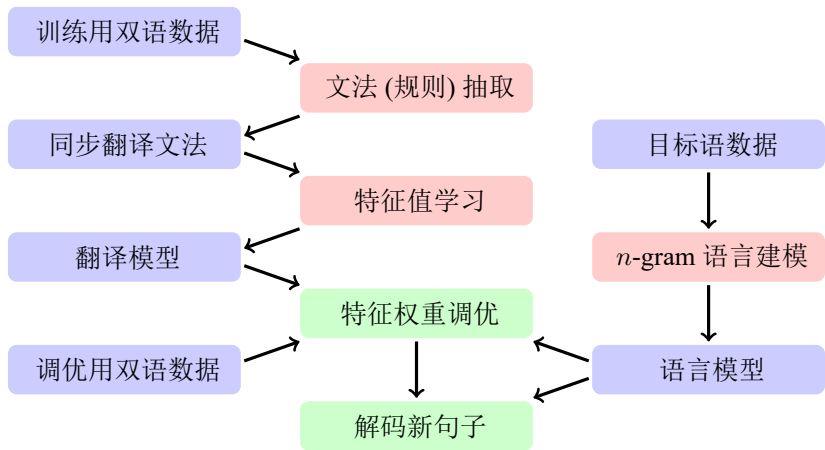


图 4.33: 层次短语系统的处理流程

4.3.2 层次短语规则抽取

层次短语系统所使用的文法包括两部分：1）不含变量的层次短语规则（短语翻译）；2）含有变量的层次短语规则。短语翻译的抽取直接复用基于短语的系统即可。

此处重点讨论如何抽取含有变量的层次短语规则。

在4.2.3节已经介绍了短语与词对齐相兼容的概念。这里，所有层次短语规则也是与词对齐相兼容（一致）的。

定义 4.3.2 与词对齐相兼容的层次短语规则

对于句对 (s,t) 和它们之间的词对齐 \mathbf{a} ，令 N 表示在句对 (s,t) 上与 \mathbf{a} 相兼容的双语短语集合。则：

- 1. 如果 $(x,y) \in N$ ，则 $X \rightarrow \langle x,y,\phi \rangle$ 是与词对齐相兼容的层次短语规则。
- 2. 对于 $(x,y) \in N$ ，存在 m 个双语短语 $(x_i,y_j) \in N$ ，同时存在 $(1,...,m)$ 上面的一个排序 $\sim = \pi_1,...,\pi_m$ ，且：

$$x = \alpha_0 x_1 \alpha_1 x_2 \dots \alpha_{m-1} x_m \alpha_m \tag{4.23}$$

$$y = \beta_0 y_{\pi_1} \beta_1 y_{\pi_2} \dots \beta_{m-1} y_{\pi_m} \beta_m \tag{4.24}$$

其中， $\alpha_0,...,\alpha_m$ 和 $\beta_0,...,\beta_m$ 表示源语言和目标语言的若干个词串（包含空串）。则 $X \rightarrow \langle x,y,\sim \rangle$ 是与词对齐相兼容的层次短语规则。这条规则包含 m 个变量，变量的对齐信息是 \sim 。

这个定义中，所有规则都是由双语短语生成。如果规则中含有变量，则变量部分也需要满足与词对齐相兼容的定义。按上述定义实现层次短语规则抽取也很简单。只需要对短语抽取系统进行改造：对于一个短语，可以通过挖“槽”的方式生成含有变量的规则。每个“槽”就代表一个变量。

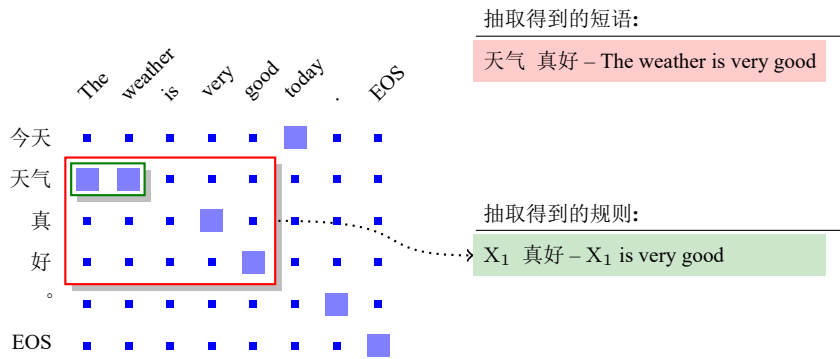


图 4.34: 通过双语短语抽取层次短语规则

图4.34展示了一个层次短语抽取的示意图。可以看到，在获取一个“大”短语的基础上（红色），直接在其内部挖掉另一个“小”短语（绿色），这样就生成了一个层次短语规则。

这种方式可以抽取大量的层次短语规则。但是，不加限制的抽取，会带来规则集合的过度膨胀，对解码系统造成很大负担。比如，如果考虑任意长度的短语会使得层次短语规则过大，一方面这些规则很难在测试数据上被匹配，另一方面抽取这样的“长”规则会使得抽取算法变慢，而且规则数量猛增之后难以存储。还有，如

果一个层次短语规则中含有过多的变量，也会导致解码算法变得更加复杂，不利于系统实现和调试。针对这些问题，在标准的层次短语系统中会考虑一些限制，包括：

- 抽取的规则最多可以跨越 10 个词；
- 规则的（源语言端）变量个数不能超过 2；
- 规则的（源语言端）变量不能连续出现。

在具体实现时还会考虑其他的限制，比如，限定规则的源语言端终结符数量的上限等。

4.3.3 翻译模型及特征

在层次短语模型中，每个翻译推导都有一个模型得分 $\text{score}(d, \mathbf{s}, \mathbf{t})$ 。 $\text{score}(d, \mathbf{s}, \mathbf{t})$ 是若干特征的线性加权之和： $\text{score}(d, \mathbf{t}, \mathbf{s}) = \sum_{i=1}^M \lambda_i \cdot h_i(d, \mathbf{t}, \mathbf{s})$ ，其中 λ_i 是特征权重， $h_i(d, \mathbf{t}, \mathbf{s})$ 是特征函数。层次短语模型的特征包括与规则相关的特征和语言模型特征，如下：

对于每一条翻译规则 $\text{LHS} \rightarrow \langle \alpha, \beta, \sim \rangle$ ，有：

- (h1-2) 短语翻译概率（取对数），即 $\log(P(\alpha | \beta))$ 和 $\log(P(\beta | \alpha))$ ，特征的计算与基于短语的模型完全一样；
- (h3-4) 词汇化翻译概率（取对数），即 $\log(P_{\text{lex}}(\alpha | \beta))$ 和 $\log(P(\beta | \alpha))$ ，特征的计算与基于短语的模型完全一样；
- (h5) 翻译规则数量，让模型自动学习对规则数量的偏好，同时避免使用过少规则造成分数偏高的现象；
- (h6) 胶水规则数量，让模型自动学习使用胶水规则的偏好；
- (h7) 短语规则数量，让模型自动学习使用纯短语规则的偏好。

这些特征可以被具体描述为：

$$h_i(d, \mathbf{t}, \mathbf{s}) = \sum_{r \in d} h_i(r) \quad (4.25)$$

公式4.25中， r 表示推导 d 中的一条规则， $h_i(r)$ 表示规则 r 上的第 i 个特征。可以看出，推导 d 的特征值就是所有包含在 d 中规则的特征值的和。进一步，可以定义

$$\text{rscore}(d, \mathbf{t}, \mathbf{s}) = \sum_{i=1}^7 \lambda_i \cdot h_i(d, \mathbf{t}, \mathbf{s}) \quad (4.26)$$

最终，模型得分被定义为：

$$\text{score}(d, \mathbf{t}, \mathbf{s}) = \text{rscore}(d, \mathbf{t}, \mathbf{s}) + \lambda_8 \log(P_{\text{lm}}(\mathbf{t})) + \lambda_9 |\mathbf{t}| \quad (4.27)$$

其中：

- $\log(P_{lm}(t))$ 表示语言模型得分；
- $|t|$ 表示译文的长度。

在定义特征函数之后，特征权重 $\{\lambda_i\}$ 可以通过最小错误率训练在开发集上进行调优。关于最小错误率训练可以参考4.2.6节的内容。

4.3.4 CKY 解码

层次短语模型解码的目标是找到模型得分最高的推导，即：

$$\hat{d} = \arg \max_d \text{score}(d, s, t) \tag{4.28}$$

\hat{d} 的目标语部分即最佳译文 \hat{t} 。令函数 $t(\cdot)$ 返回翻译推导的目标语词串，于是有：

$$\hat{t} = t(\hat{d}) \tag{4.29}$$

由于层次短语规则本质上就是 CFG 规则，因此公式4.28代表了一个典型的句法分析过程。需要做的是，用模型源语言端的 CFG 对输入句子进行分析，同时用模型目标语言端的 CFG 生成译文。基于 CFG 的句法分析是自然语言处理中的经典问题。一种广泛使用的方法是：首先把 CFG 转化为 ϵ -free 的**乔姆斯基范式**（Chomsky Normal Form）⁵，之后采用 CKY 方法进行分析。

CKY 是形式语言中一种常用的句法分析方法 [45, 137, 332]。它主要用于分析符合乔姆斯基范式的句子。由于乔姆斯基范式中每个规则最多包含两叉（或者说两个变量），因此 CKY 方法也可以被看作是基于二叉规则的一种分析方法。对于一个待分析的字符串，CKY 方法从小的“范围”开始，不断扩大分析的“范围”，最终完成对整个字符串的分析。在 CKY 方法中，一个重要的概念是**跨度**（Span），所谓跨度表示了一个符号串的范围。这里可以把跨度简单的理解为从一个起始位置到一个结束位置中间的部分。

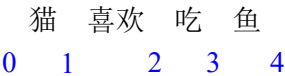


图 4.35: 一个单词串及位置索引

比如，如图4.35 所示，每个单词左右都有一个数字来表示序号。可以用序号的

⁵能够证明任意的 CFG 都可以被转换为乔姆斯基范式，即文法只包含形如 $A \rightarrow BC$ 或 $A \rightarrow a$ 的规则。这里，假设文法中不包含空串产生式 $A \rightarrow \epsilon$ ，其中 ϵ 表示空字符串。

范围来表示跨度，例如：

$span[0,1]$ = “猫”
 $span[2,4]$ = “吃 鱼”
 $span[0,4]$ = “猫 喜欢 吃 鱼”

CKY 方法是按跨度由小到大的次序执行的,这也对应了一种**自下而上的分析**(Top-down Parsing) 过程。对于每个跨度，检查：

- 是否有形如 $A \rightarrow a$ 的规则可以匹配；
- 是否有形如 $A \rightarrow BC$ 的规则可以匹配。

对于第一种情况，简单匹配字符串即可；对于第二种情况，需要把当前的跨度进一步分割为两部分，并检查左半部分是否已经被归纳为 B ，右半部分是否已经被归纳为 C 。如果可以匹配，会在这个跨度上保存匹配结果。后面，可以访问这个结果（也就是 A ）来生成更大跨度上的分析结果。

输入：符合乔姆斯基范式的待分析字符串和一个上下文无关文法（CFG）
输出：字符串全部可能的语法分析结果
参数： s 为输入字符串。 G 为输入 CFG。 J 为待分析字符串长度。

```
Function CKY-Algorithm( $s, G$ )  
  for  $j = 0$  to  $J - 1$   
     $span[j, j + 1].Add(A \rightarrow a \in G)$   
    for  $l = 1$  to  $J$            // length of span  
      for  $j = 0$  to  $J - l$      // beginning of span  
        for  $k = j$  to  $j + l$    // partition of span  
           $hypos = Compose(span[j, k], span[k, j + l])$   
           $span[j, j + l].Update(hypos)$   
  return  $span[0, J]$ 
```

图 4.36: CKY 算法

CKY 算法的伪代码如图4.36所示。整个算法的执行顺序是按跨度的长度(l)组织的。对于每个 $span[j, j + l]$ ，会在位置 k 进行切割。之后，判断 $span[j, k]$ 和 $span[k, j + l]$ 是否可以形成一个规则的右部。也就是判断 $span[j, k]$ 是否生成了 B ，同时判断 $span[k, j + l]$ 是否生成了 C ，如果文法中有规则 $A \rightarrow BC$ ，则把这个规则放入 $span[j, j + l]$ 。这个过程由 $Compose$ 函数完成。如果 $span[j, j + l]$ 可以匹配多条规则，所有生成的推导都会被记录在 $span[j, j + l]$ 所对应的一个列表里⁶。

图4.37展示了 CKY 方法的一个运行实例（输入词串是 `aabbc`）。算法在处理完最

⁶通常，这个列表会用优先队列实现。这样可以对推导按模型得分进行排序，方便后续的剪枝操作。

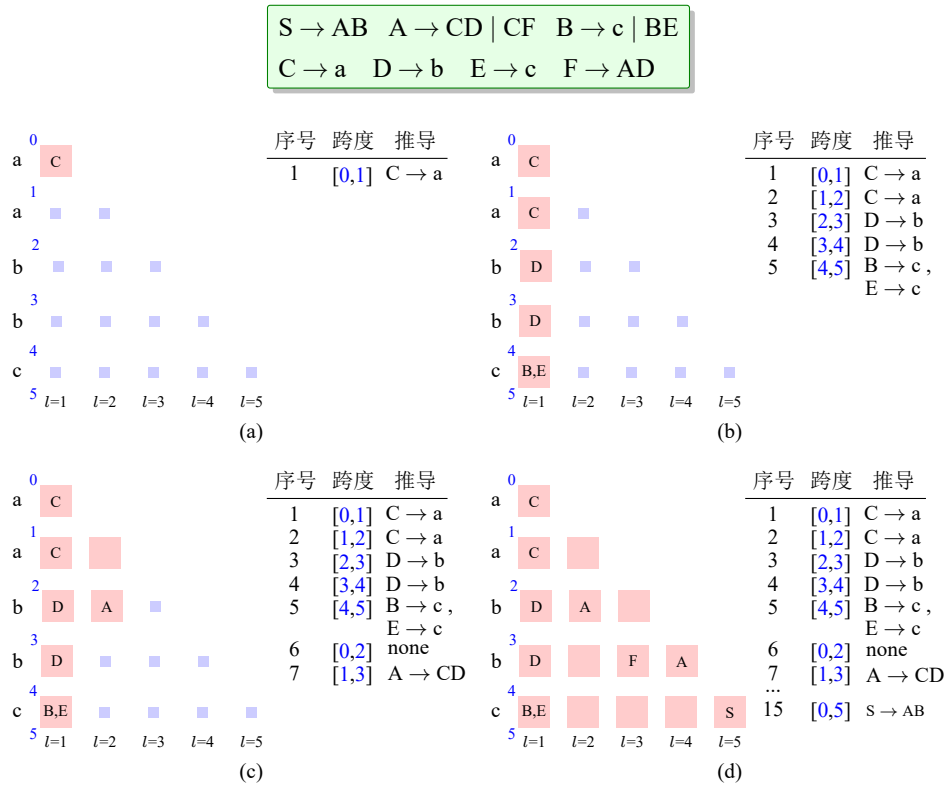


图 4.37: CKY 算法执行实例

后一个跨度后会得到覆盖整个词串的分析结果，即句法树的根结点 S。

不过，CKY 方法并不能直接用于层次短语模型。有两个问题：

- 层次短语模型的文法不符合乔姆斯基范式；
- 机器翻译中需要语言模型。由于当前词的语言模型得分需要前面的词做条件，因此机器翻译的解码过程并不是上下文无关的。

解决第一个问题有两个思路：

- 把层次短语文法转化为乔姆斯基范式，这样可以直接使用原始的 CKY 方法进行分析；
- 对 CKY 方法进行改造。解码的核心任务要知道每个跨度是否能匹配规则的源语言部分。实际上，层次短语模型的文法是一种特殊的文法。这种文法规则的源语言部分最多包含两个变量，而且变量不能连续。这样的规则会对应一种特定类型的模版，比如，对于包含两个变量的规则，它的源语言部分形如 $\alpha_0 X_1 \alpha_1 X_2 \alpha_2$ 。其中， α_0 、 α_1 和 α_2 表示终结串， X_1 和 X_2 是变量。显然，如果 α_0 、 α_1 和 α_2 确定下来那么 X_1 和 X_2 的位置也就确定了下来。因此，对于每一个词串，都可以很容易的生成这种模版，进而完成匹配。而 X_1 和 X_2 和原始

CKY 中匹配二叉规则本质上是一样的。由于这种方法并不需要对 CKY 方法进行过多调整，因此层次短语系统中广泛使用这种改造的 CKY 方法进行解码。

对于语言模型在解码中的集成问题，一种简单的办法是：在 CKY 分析的过程中，用语言模型对每个局部的翻译结果进行评价，并计算局部翻译（推导）的模型得分。注意，局部的语言模型得分可能是不准确的，比如，局部翻译片段最左边单词的概率计算需要依赖前面的单词。但是由于每个跨度下生成的翻译是局部的，当前跨度下看不到前面的译文。这时会用 1-gram 语言模型的得分代替真实的高阶语言模型得分。等这个局部翻译片段和其他片段组合之后，可以知道前文的内容，这时才会得出最终的语言模型得分。

另一种解决问题的思路是，先不加入语言模型，这样可以直接使用 CKY 方法进行分析。在得到最终的结果后，对最好的多个推导用含有语言模型的完整模型进行打分，选出最终的最优推导。

不过，在实践中发现，由于语言模型在机器翻译中起到至关重要的作用，因此对最终结果进行重排序会带来一定的性能损失。不过这种方法的优势在于速度快，而且容易实现。另外，在实践时，还需要考虑两方面问题：

- 剪枝：在 CKY 中，每个跨度都可以生成非常多的推导（局部翻译假设）。理论上，这些推导的数量会和跨度大小成指数关系。显然不可能保存如此大量的翻译推导。对于这个问题，常用的办法是只保留 top- k 个推导。也就是每个局部结果只保留最好的 k 个。这种方法也被称作**束剪枝**（Beam Pruning）。在极端情况下，当 $k=1$ 时，这个方法就变成了贪婪的方法；
- n -best 结果的生成： n -best 推导（译文）的生成是统计机器翻译必要的功能。比如，最小错误率训练中就需要最好的 n 个结果用于特征权重调优。在基于 CKY 的方法中，整个句子的翻译结果会被保存在最大跨度所对应的结构中。因此一种简单的 n -best 生成方法是从这个结构中取出排名最靠前的 n 个结果。另外，也可以考虑自上而下遍历 CKY 生成的推导空间，得到更好的 n -best 结果 [116]。

4.3.5 立方剪枝

相比于基于短语的模型，基于层次短语的模型引入了“变量”的概念。这样，可以根据变量周围的上下文信息对变量进行调序。变量的内容由其所对应的跨度上的翻译假设进行填充。图4.38展示了一个层次短语规则匹配词串的实例。可以看到，规则匹配词串之后，变量 X 的位置对应了一个跨度。这个跨度上所有标记为 X 的局部推导都可以作为变量的内容。

真实的情况会更加复杂。对于一个规则的源语言端，可能会有多个不同的目标

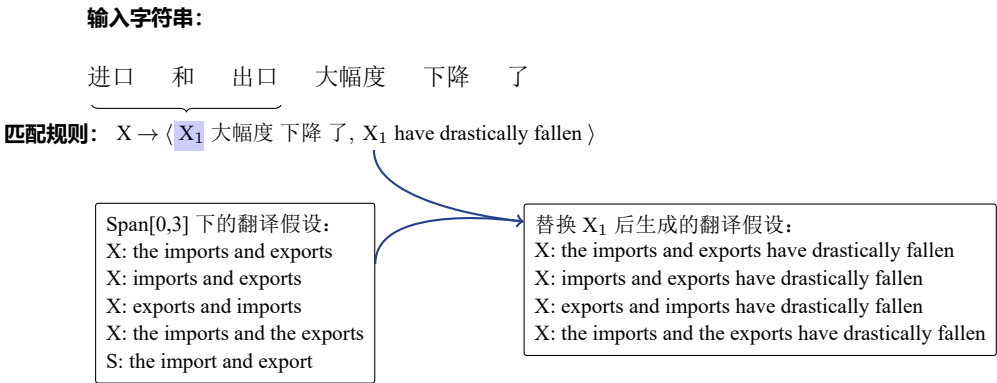


图 4.38: 层次短语规则匹配及译文生成

语言端与之对应。比如，如下规则的源语言端完全相同，但是译文不同：

- $X \rightarrow \langle X_1 \text{ 大幅度 下降 了}, X_1 \text{ have drastically fallen} \rangle$
- $X \rightarrow \langle X_1 \text{ 大幅度 下降 了}, X_1 \text{ have fallen drastically} \rangle$
- $X \rightarrow \langle X_1 \text{ 大幅度 下降 了}, X_1 \text{ has drastically fallen} \rangle$

这也就是说，当匹配规则的源语言部分“ X_1 大幅度下降了”时会有三个译文可以选择。而变量 X_1 部分又有很多不同的局部翻译结果。不同的规则译文和不同的变量译文都可以组合出一个局部翻译结果。图4.39展示了这种情况的实例。



图 4.39: 不同规则目标语端及变量译文的组合

假设有 n 个规则源语言端相同，规则中每个变量可以被替换为 m 个结果，对于只含有一个变量的规则，一共有 nm 种不同的组合。如果规则含有两个变量，这种

组合的数量是 nm^2 。由于翻译中会进行大量的规则匹配，如果每个匹配的源语言端都考虑所有 nm^2 种译文的组合，解码速度会很慢。

在层次短语系统中，会进一步对搜索空间剪枝。简言之，此时并不需要对所有 nm^2 种组合进行遍历，而是只考虑其中的一部分组合。这种方法也被称作**立方剪枝**（Cube Pruning）。所谓“立方”是指组合译文时的三个维度：规则的目标语端、第一个变量所对应的翻译候选、第二个变量所对应的翻译候选。立方剪枝假设所有的译文候选都经过排序，比如，按照短语翻译概率排序。这样，每个译文都对应一个坐标，比如， (i,j,k) 就表示第 i 个规则目标语端、第二个变量的第 j 个翻译候选、第三个变量的第 k 个翻译候选的组合。于是，可以把每种组合看作是一个三维空间中的一个点。在立方剪枝中，开始的时候会看到 $(0,0,0)$ 这个翻译假设，并把这个翻译假设放入一个优先队列中。之后每次从这个优先队里中弹出最好的结果，之后沿着三个维度分别将坐标加 1，比如，如果优先队列弹出 (i,j,k) ，则会生成 $(i+1,j,k)$ 、 $(i,j+1,k)$ 和 $(i,j,k+1)$ 这三个新的翻译假设。之后，计算出它们的模型得分，并压入优先队列。这个过程不断被执行，直到达到终止条件，比如，扩展次数达到一个上限。

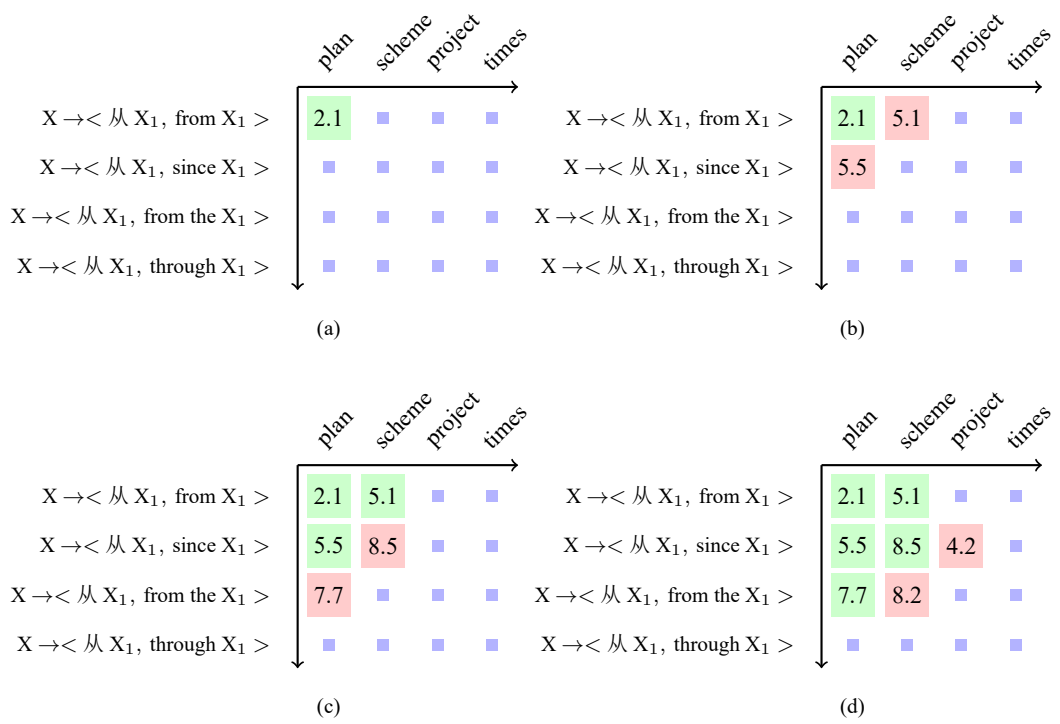


图 4.40: 立方剪枝执行过程（行表示规则，列表示变量可替换的内容）

图4.40展示了立方剪枝的过程（规则只含有一个变量的情况）。可以看到，每个步骤中，算法只会扩展当前最好结果周围的两个点（对应两个维度，横轴对应变

被替换的内容，纵轴对应规则的目标语端)。

理论上，立方剪枝最多访问 nm^2 个点。但是在实践中发现，如果终止条件设计的合理，搜索的代价基本上与 m 或者 n 呈线性关系。因此，立方剪枝可以大大提高解码速度。立方剪枝实际上是一种启发性的搜索方法。它把搜索空间表示为一个三维空间。它假设：如果空间中某个点的模型得分较高，那么它“周围”的点的得分也很可能较高。这也是对模型得分沿着空间中不同维度具有连续性的一种假设。这种方法也大量的使用在句法分析中，并取得了很好的效果。

4.4 基于语言学句法的模型

层次短语模型是一种典型的基于翻译文法的模型。它把翻译问题转化为语言分析问题。在翻译一个句子的时候，模型会生成一个树形结构，这样也就得到了句子结构的某种表示。图4.41展示了一个使用层次短语系统进行翻译时所生成的翻译推导 d ，以及这个推导所对应的树形结构（源语言）。这棵树体现了机器翻译的视角下的句子结构，尽管这个结构并不是人类语言学中的句法树。

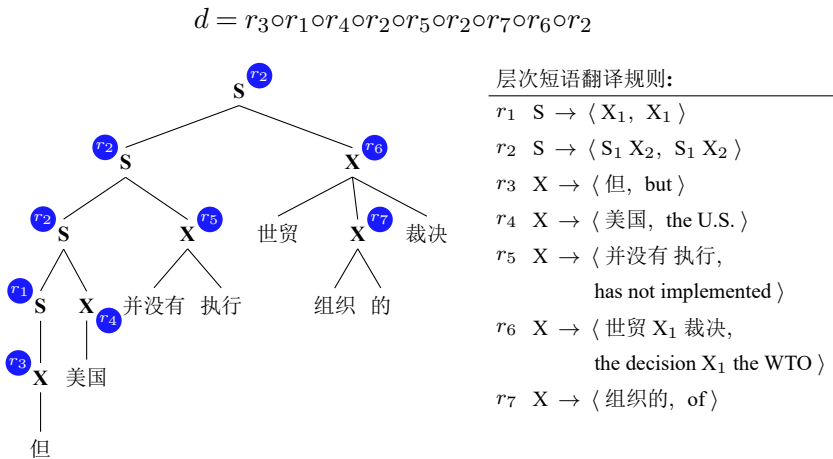


图 4.41: 层次短语模型所对应的翻译推导及树结构（源语言）

在翻译中使用树结构的好处在于，模型可以更加有效地对句子的层次结构进行抽象。而且树结构可以作为对序列结构的一种补充，比如，在句子中距离较远的两个单词，在树结构中可以很近。不过，层次短语模型也存在一些不足：

- 层次短语规则没有语言学句法标记，很多规则并不符合语言学认知，因此译文的生成和调序也不遵循语言学规律。比如，层次短语系统经常会把完整的句法结构打散，或者“破坏”句法成分进行组合；
- 层次短语系统中有大量的工程化约束条件。比如，规则的源语言部分不允许两个变量连续出现，而且变量个数也不能超过两个。这些约束在一定程度上限制了模型处理翻译问题的能力。

实际上，基于层次短语的方法可以被看作是介于基于短语的方法和基于语言学句法的方法之间的一种折中。它的优点在于，具备短语模型简单、灵活的优点，同时，由于同步翻译文法可以对句子的层次结构进行表示，因此也能够处理一些较长距离的调序问题。但是，另一方面，层次短语模型并不是一种“精细”的句法模型，当翻译需要复杂的结构信息时，这种模型可能会无能为力。

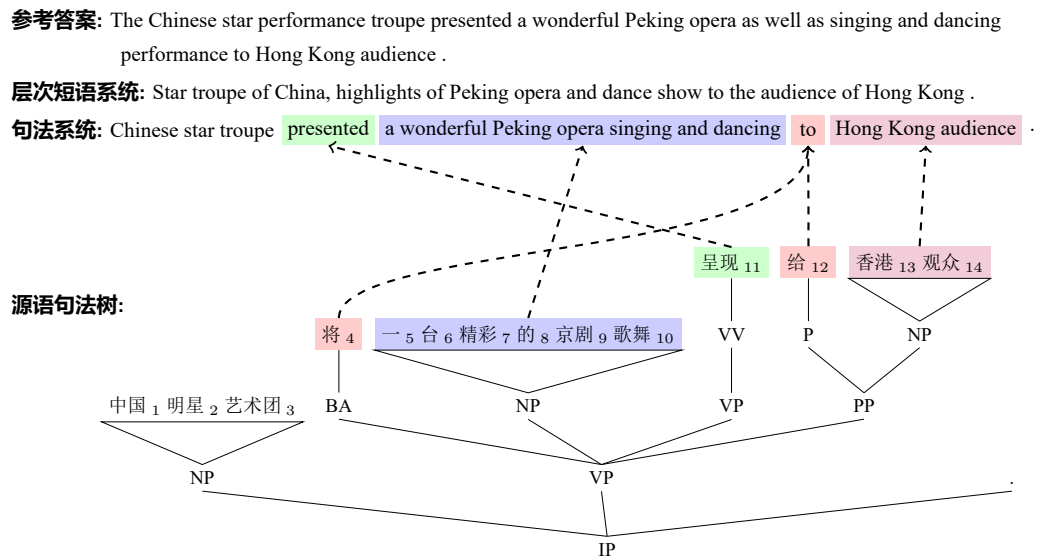


图 4.42: 含有复杂调序的翻译实例（汉语翻译到英语）

图4.42展示了一个翻译实例，对图中句子进行翻译需要通过复杂的调序才能生成正确译文。为了完成这样的翻译，需要对多个结构（超过两个）进行调序，但是这种情况在标准的层次短语系统中是不允许的。

从这个例子中可以发现，如果知道源语言的句法结构，翻译其实并不“难”。比如，语言学句法结构可以告诉模型句子的主要成分是什么，而调序实际上是在这些成分之间进行的。从这个角度说，语言学句法可以帮助模型进行更上层结构的表示和调序。

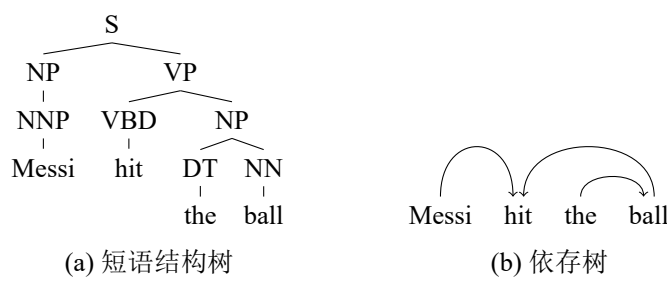


图 4.43: 短语结构树 vs 依存树

显然，使用语言学句法对机器翻译进行建模也是一种不错的选择。不过，语言学句法有很多种，因此首先需要确定使用何种形式的句法。比如，在自然语言处理

中经常使用的是短语结构分析和依存分析（图4.43）。二者的区别已经在第二章进行了讨论。

在机器翻译中，上述这两种句法信息都可以被使用。不过为了后续讨论的方便，这里仅介绍基于短语结构树的机器翻译建模。使用短语结构树的原因在于，它提供了较为丰富的句法信息，而且相关句法分析工具比较成熟。如果没有特殊说明，本章中所提到的句法树都是指短语结构树（或成分句法树），有时也会把句法树简称为树。此外，这里也假设所有句法树都可以由句法分析器自动生成⁷。

4.4.1 基于句法的翻译模型分类

可以说基于句法的翻译模型贯穿了现代统计机器翻译的发展历程。从概念上讲，不管是层次短语模型，还是语言学句法模型都是基于句法的模型。基于句法的机器翻译模型种类繁多，这里先对相关概念进行简要介绍，以避免后续论述中产生歧义。表4.2给出了基于句法的机器翻译中涉及的一些概念。

表 4.2: 基于句法的机器翻译中常用概念

术语	说明
翻译规则	翻译的最小单元（或步骤）
推导	由一系列规则组成的分析或翻译过程，推导可以被看作是规则的序列
规则表	翻译规则的存储表示形式，可以高效进行查询
层次短语模型	基于同步上下文无关文法的翻译模型，非终结符只有 S 和 X 两种，文法并不需要符合语言学句法约束
树到串模型	一类翻译模型，它使用源语语言学句法树，因此翻译可以被看作是从一颗句法树到词串的转换
串到树模型	一类翻译模型，它使用目标语语言学句法树，因此翻译可以被看作是从词串到句法树的转换
树到树模型	一类翻译模型，它同时使用源语和目标语语言学句法树，因此翻译可以被看作从句法树到句法树的转换
基于句法	使用语言学句法
基于树	（源语言）使用树结构（大多指句法树）
基于串	（源语言）使用词串，比如串到树翻译系统的解码器一般都是基于串的解码方法
基于森林	（源语言）使用句法森林，这里森林只是对多个句法树的一种压缩表示

⁷对于汉语、英语等大语种，句法分析器的选择有很多。不过，对于一些小语种，句法标注数据有限，句法分析可能并不成熟，这时在机器翻译中使用语言学句法信息会面临较大的挑战。

术语	说明
词汇化规则	含有终结符的规则
非词汇规则	不含有终结符的规则
句法软约束	不强制规则推导匹配语言学句法树，通常把句法信息作为特征使用
句法硬约束	要求推导必须符合语言学句法树，不符合的推导会被过滤掉

基于句法的翻译模型可以被分为两类：基于形式化文法的模型和语言学上基于句法的模型（图4.44）。基于形式化文法的模型的典型代表包括，吴德恺提出的基于反向转录文法的模型 [309] 和 David Chiang 提出的基于层次短语的模型 [38]。而语言学上基于句法的模型包括，句法树到串的模型 [117, 180]、串到句法树的模型 [81, 82]、句法树到句法树的模型 [39, 70, 181, 348] 等。

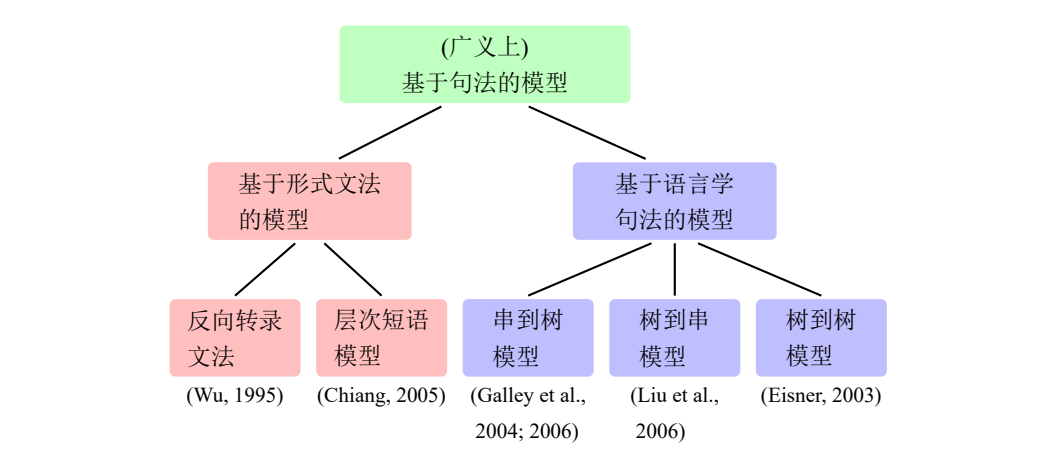


图 4.44: 基于句法的机器翻译模型的分类

通常来说，基于形式化文法的模型并不需要句法分析技术的支持。这类模型只是把翻译过程描述为一系列形式化文法规则的组合过程。而语言学上基于句法的模型则需要源语言和（或者）目标语言句法分析的支持，以获取更丰富的语言学信息来提高模型的翻译能力。这也是本节所关注的重点。当然，所谓分类也没有唯一的标准，比如，还可以把句法模型分为基于软约束的模型和基于硬约束的模型，或者分为基于树的模型和基于串模型。

表4.3进一步对比了不同模型的区别。其中，树到串和树到树模型都使用了源语言句法信息，串到树和树到树模型使用了目标语言句法信息。不过，这些模型都依赖句法分析器的输出，因此会对句法分析的错误比较敏感。相比之下，基于形式文法的模型并不依赖句法分析器，因此会更健壮一些。

表 4.3: 基于句法的机器翻译模型对比

模型	形式句法	语言学句法		
		树到串	串到树	树到树
源语句法	No	Yes	No	Yes
目标语句法	No	No	Yes	Yes
基于串的解码	Yes	No	Yes	Yes
基于树的解码	No	Yes	No	Yes
健壮性	High	Mid	Mid	Low

4.4.2 基于树结构的文法

基于句法的翻译模型的一个核心问题是要对树结构进行建模，进而完成树之间或者树和串之间的转换。在计算机领域中，所谓树就是由一些节点组成的层次关系的集合。计算机领域的树和自然世界中的树没有任何关系，只是借用了相似的概念，因为这种层次结构很像一个倒过来的树。在使用树时，经常会把树的层次结构转化为序列结构，称为树结构的**序列化**或者**线性化**（Linearization）。

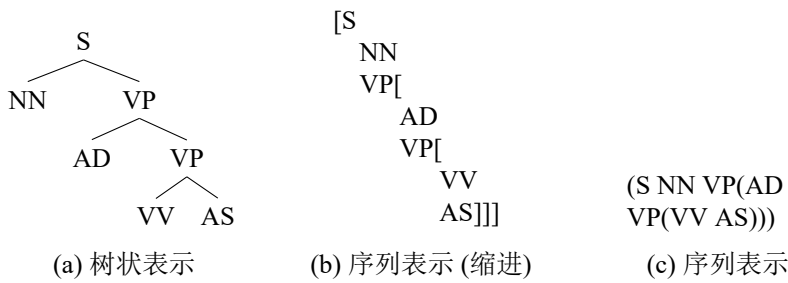


图 4.45: 树结构的不同表示形式

比如，使用树的先序遍历就可以得到一个树的序列表示。图4.45就对比了同一棵树的不同表示方式。实际上，树的序列表示是非常适合计算机进行读取和处理的。因此，本章也会使用树的序列化结果来表示句法结构。

在基于语言学句法的机器翻译中，两个句子间的转化仍然需要使用文法规则进行描述。有两种类型的规则：

- **树到串翻译规则**（Tree-to-String Translation Rule）：在树到串、串到树模型中使用；
- **树到树翻译规则**（Tree-to-Tree Translation Rule）：在树到树模型中使用。

树到串规则描述了一端是树结构而另一端是串的情况，因此树到串模型和串到树模型都可以使用这种形式的规则。树到树模型需要在两种语言上同时使用句法树结构，需要树到树翻译规则。

树到树翻译规则

虽然树到串翻译规则和树到树翻译规则蕴含了不同类型的翻译知识，但是它们都在描述一个结构（树/串）到另一个结构（树/串）的映射。这里采用了一种更加通用的文法——基于树结构的文法——将树到串翻译规则和树到树翻译规则进行统一。定义如下：

定义 4.4.1 基于树结构的文法

一个基于树结构的文法由七部分构成 $(N_s, N_t, T_s, T_t, I_s, I_t, R)$ ，其中

- 1. N_s 和 N_t 是源语言和目标语言非终结符集合；
- 2. T_s 和 T_t 是源语言和目标语言终结符集合；
- 3. $I_s \subseteq N_s$ 和 $I_t \subseteq N_t$ 是源语言和目标语言起始非终结符集合；
- 4. R 是规则集合，每条规则 $r \in R$ 有如下形式：

$$\langle \alpha_h, \beta_h \rangle \rightarrow \langle \alpha_r, \beta_r, \sim \rangle$$

其中，规则左部由非终结符 $\alpha_h \in N_s$ 和 $\beta_h \in N_t$ 构成；规则右部由三部分组成， α_r 表示由源语言终结符和非终结符组成的树结构； β_r 表示由目标语言终结符和非终结符组成的树结构； \sim 表示 α_r 和 β_r 中叶子非终结符的 1-1 对应关系。

基于树结构的规则非常适合于描述树结构到树结构的映射。比如，图4.46是一个汉语句法树结构到一个英语句法树结构的对应。其中的树结构可以被看作是完整句法树上的一个片段，称为**树片段**（Tree Fragment）。

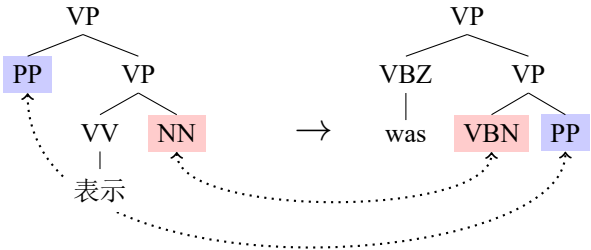


图 4.46: 汉语句法树到英语句法树的结构对应

树片段的叶子节点既可以是终结符（单词）也可以是非终结符。当叶子节点为非终结符时，表示这个非终结符会被进一步替换，因此它可以被看作是变量。而源语言树结构和目标语言树结构中的变量是一一对应的，对应关系用虚线表示。

这个双语映射关系可以被表示为一个基于树结构的文法规则，套用规则的定义

$\langle \alpha_h, \beta_h \rangle \rightarrow \langle \alpha_r, \beta_r, \sim \rangle$ 形式，可以知道：

$$\begin{aligned}\langle \alpha_h, \beta_h \rangle &= \langle \text{VP}, \text{VP} \rangle \\ \alpha_r &= \text{VP}(\text{PP}:x \text{ VP}(\text{VV}(\text{表示}) \text{ NN}:x)) \\ \beta_r &= \text{VP}(\text{VBZ}(\text{was}) \text{ VP}(\text{VBN}:x \text{ PP}:x)) \\ \sim &= \{1-2, 2-1\}\end{aligned}$$

这里， α_h 和 β_h 表示规则的左部，对应树片段的根节点； α_r 和 β_r 是两种语言的树结构（序列化表示），其中标记为 x 的非终结符是变量。 $\sim = \{1-2, 2-1\}$ 表示源语言的第一个变量对应目标语言的第二个变量，而源语言的第二个变量对应目标语言的第一个变量，这也反应出两种语言句法结构中的调序现象。有时候为了化简规则的形式，会把规则中变量的对应关系用下标进行表示。例如，上面的规则也可以被写为如下形式。

$$\langle \text{VP}, \text{VP} \rangle \rightarrow \langle \text{PP}_1 \text{ VP}(\text{VV}(\text{表示}) \text{ NN}_2)) \text{ VP}(\text{VBZ}(\text{was}) \text{ VP}(\text{VBN}_2 \text{ PP}_1)) \rangle$$

其中，两种语言中变量的对应关系为 $\text{PP}_1 \leftrightarrow \text{PP}_1$ ， $\text{NN}_2 \leftrightarrow \text{VBN}_2$ 。

基于树结构的翻译推导

规则中的变量预示着一一种替换操作，即变量可以被其他树结构替换。实际上，上面的树到树翻译规则就是一种**同步树替换文法规则**（Synchronous Tree Substitution Grammar Rule）。不论是源语言端还是目标语言端，都可以通过这种替换操作不断生成更大的树结构，也就是通过树片段的组合得到更大的树片段。图4.47就展示了树替换操作的一个实例。

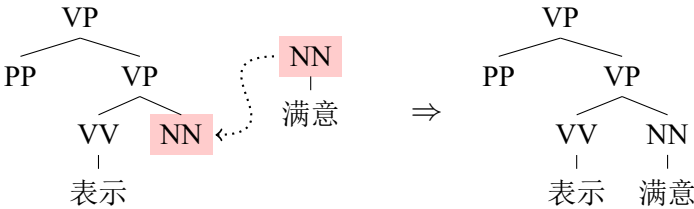


图 4.47: 树替换操作（将 NN 替换为一个树结构）

这种方法也可以被扩展到双语的情况。图4.48给出了一个使用基于树结构的同步文法生成双语句对的实例。其中，每条规则都同时对应源语言和目标语言的一个树片段（用矩形表示）。变量部分可以被替换，这个过程不断执行。最后，四条规则组合在一起形成了源语言和目标语言的句法树。这个过程也被称作规则的推导。

规则的推导对应了一种源语言和目标语言树结构的同步生成的过程。比如，使

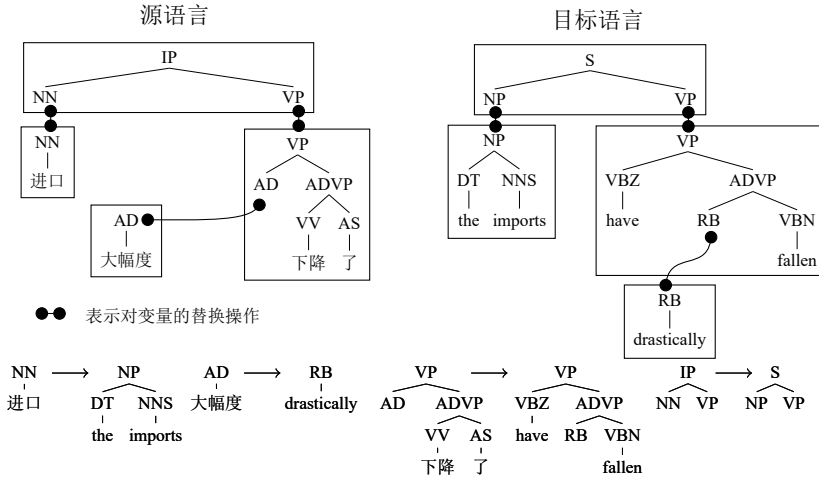


图 4.48: 使用基于树结构的同步文法生成汉英句对

用下面的规则集:

- $r_3: AD(\text{大幅度}) \rightarrow RB(\text{drastically})$
 $r_4: VV(\text{减少}) \rightarrow VBN(\text{fallen})$
 $r_6: AS(\text{了}) \rightarrow VBP(\text{have})$
 $r_7: NN(\text{进口}) \rightarrow NP(DT(\text{the}) NNS(\text{imports}))$
 $r_8: VP(AD_1 VP(VV_2 AS_3)) \rightarrow VP(VBP_3 ADVP(RB_1 VBN_2))$
 $r_9: IP(NN_1 VP_2) \rightarrow S(NP_1 VP_2)$

可以得到一个翻译推导:

$$\begin{aligned}
 & \langle IP^{[1]}, S^{[1]} \rangle \\
 & \xrightarrow[r_9]{IP^{[1]} \Leftrightarrow S^{[1]}} \langle IP(NN^{[2]} VP^{[3]}), S(NP^{[2]} VP^{[3]}) \rangle \\
 & \xrightarrow[r_7]{NN^{[2]} \Leftrightarrow NP^{[2]}} \langle IP(NN(\text{进口}) VP^{[3]}), S(NP(DT(\text{the}) NNS(\text{imports})) VP^{[3]}) \rangle \\
 & \xrightarrow[r_8]{VP^{[3]} \Leftrightarrow VP^{[3]}} \langle IP(NN(\text{进口}) VP(AD^{[4]} VP(VV^{[5]} AS^{[6]}))), \\
 & \quad S(NP(DT(\text{the}) NNS(\text{imports})) VP(VBP^{[6]} ADVP(RB^{[4]} VBN^{[5]}))) \rangle \\
 & \xrightarrow[r_3]{AD^{[4]} \Leftrightarrow RB^{[4]}} \langle IP(NN(\text{进口}) VP(AD(\text{大幅度}) VP(VV^{[5]} AS^{[6]}))), \\
 & \quad S(NP(DT(\text{the}) NNS(\text{imports})) VP(VBP^{[6]} ADVP(RB(\text{drastically}) VBN^{[5]}))) \rangle \\
 & \xrightarrow[r_4]{VV^{[5]} \Leftrightarrow VBN^{[5]}} \langle IP(NN(\text{进口}) VP(AD(\text{大幅度}) VP(VV(\text{减少}) AS^{[6]}))), \\
 & \quad S(NP(DT(\text{the}) NNS(\text{imports})) VP(VBP^{[6]} \\
 & \quad \quad ADVP(RB(\text{drastically}) VBN(\text{fallen})))) \rangle
 \end{aligned}$$

$$\frac{AS^{[6]} \Leftrightarrow VBP^{[6]}}{r_6} \rightarrow \langle IP(NN(进口) VP(AD(大幅度) VP(VV(减少) AS(了))))), \\ S(NP(DT(the) NNS(imports)) VP(VBP(have) \\ ADVP(RB(drastically) VBN(fallen)))) \rangle$$

其中，箭头 \rightarrow 表示推导之意。显然，可以把翻译看作是基于树结构的推导过程（记为 d ）。因此，与层次短语模型一样，基于语言学句法的机器翻译也是要找到最佳的推导 $\hat{d} = \arg \max P(d)$ 。

树到串翻译规则

基于树结构的文法可以很好的表示两个树片段之间的对应关系，即树到树翻译规则。那树到串翻译规则该如何表示呢？实际上，基于树结构的文法也同样适用于树到串模型。比如，图4.49是一个树片段到串的映射，它可以被看作是树到串规则的一种表示。

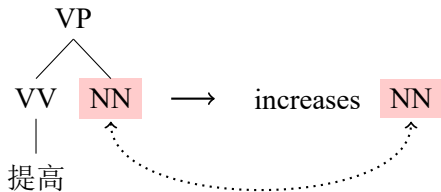


图 4.49: 树片段到串的映射

其中，源语言树片段中的叶子结点 NN 表示变量，它与右手端的变量 NN 对应。这里仍然可以使用基于树结构的规则对上面这个树到串的映射进行表示。参照规则形式 $\langle \alpha_h, \beta_h \rangle \rightarrow \langle \alpha_r, \beta_r, \sim \rangle$ ，有：

$$\begin{aligned} \alpha_h &= VP \\ \beta_h &= VP (= \alpha_h) \\ \alpha_r &= VP(VV(提高) NN:x) \\ \beta_r &= VP(increases NN:x) \\ \sim &= \{1-1\} \end{aligned}$$

这里，源语言部分是一个树片段，因此 α_h 和 α_r 很容易确定。对于目标语部分，可以把这个符号串当作是一个单层的树片段，根结点直接共享源语言树片段的根结点，叶子结点就是符号串本身。这样，也可以得到 β_h 和 β_r 。从某种意义上说，树到串翻译仍然体现了一种双语的树结构，只是目标语部分不是语言学句法驱动的，而是一种借用了源语言句法标记所形成的层次结构。

这里也可以把变量的对齐信息用下标表示，同时将左部两个相同的非终结符合并⁸，于是规则可以被写作：

$$VP \rightarrow \langle VP(VV(\text{提高}) NN_1), \text{increases } NN_1 \rangle$$

另外，在机器翻译领域，大家习惯把规则看作源语言结构（树/串）到目标语言结构（树/串）的一种映射，因此常常会把上面的规则记为：

$$VP(VV(\text{提高}) NN_1) \rightarrow \text{increases } NN_1$$

在后面的内容中也会使用这种形式来表示基于句法的翻译规则。

4.4.3 树到串翻译规则抽取

基于句法的机器翻译包括两个步骤：文法归纳和解码。其中，文法归纳是指从双语平行数据中自动学习翻译规则及规则所对应的特征；解码是指利用得到的文法对新的句子进行分析，并获取概率最高的翻译推导。

本节首先介绍树到串文法归纳的经典方法——GHKM方法[81, 82]。所谓GHKM是四位作者名字的首字母。GHKM方法的输入包括：

- 源语言句子及其句法树；
- 目标语言句子；
- 源语言句子和目标语言句子之间的词对齐。

它的输出是这个双语句对上的树到串翻译规则。GHKM不是一套单一的算法，它还包括很多技术手段用于增加规则的覆盖度和准确性。下面就具体看看GHKM是如何工作的。

树的切割与最小规则

获取树到串规则就是要找到源语言树片段与目标语言词串之间的对应关系。一棵句法树会有很多个树片段，那么哪些树片段可以和目标语言词串产生对应关系呢？

在GHKM方法中，源语言树片段和目标语言词串的对应是由词对齐决定的。GHKM假设：一个合法的树到串翻译规则，不应该违反词对齐。这个假设和双语短语抽取中的词对齐一致性约束是一样的（见4.2.3节）。简单来说，规则中两种语言互相对应的部分不应包含对齐到外部的词对齐连接。

为了说明这个问题，来看一个例子。图4.50包含了一棵句法树、一个词串和它们之间的词对齐结果。图中包含如下规则：

$$PP(P(\text{对}) NP(NN(\text{回答}))) \rightarrow \text{with the answer}$$

⁸在树到串规则中， α_h 和 β_h 是一样的。

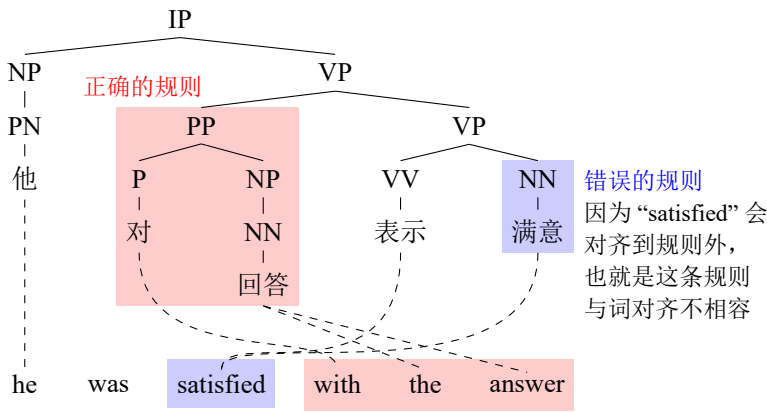


图 4.50: 树到串规则与词对齐兼容性示例

该规则是一条满足词对齐约束的规则（对应于图4.50中红色部分），因为不存在从规则的源语言或目标语言部分对齐到规则外部的情况。但是，如下的规则却是一条不合法的规则：

$$\text{NN(满意)} \rightarrow \text{satisfied}$$

这是因为，“satisfied”除了对齐到“满意”，还对齐到“表示”。也就是，这条规则会产生歧义，因为“satisfied”不应该只由“满意”生成。

为了能够获得与词对齐相兼容的规则，GHKM 引入了几个概念。首先，GHKM 方法中定义了 Span 和 Complement Span：

定义 4.4.2 Span

对于一个源语言句法树节点，它的 Span 是这个节点所对应到的目标语言第一个单词和最后一个单词所构成的索引范围。

定义 4.4.3 Complement Span

对于一个源语言句法树节点，它的 Complement Span 是除了它的祖先和子孙节点外的其他节点 Span 的并集。

Span 定义了每个节点覆盖的源语言片段所对应的目标语言片段。实际上，它表示了目标语言句子上的一个跨度，这个跨度代表了这个源语言句法树节点所能达到的最大范围。因此 Span 实际上是一个目标语单词索引的范围。Complement Span 是与 Span 相对应的一个概念，它定义了句法树中一个节点之外的部分对应到目标语的范围，但是这个范围并不必须是连续的。

有了 Span 和 Complement Span 的定义之后，可以进一步定义：

定义 4.4.4 可信节点 (Admissible Node)

对于源语言树节点 *node*，如果它的 *Span* 和 *Complement Span* 不相交，节点 *node* 就是一个可信节点，否则是一个不可信节点。

可信节点表示这个树节点 *node* 和树中的其他部分（不包括 *node* 的祖先和孩子）没有任何词对齐上的歧义。也就是说，这个节点可以完整的对应到目标语言句子的一个连续范围，不会出现在这个范围中的词对应到其他节点的情况。如果节点不是可信节点，则表示它会引起词对齐的歧义，因此不能作为树到串规则中源语言树片段的根节点或者变量部分。图4.51给出了一个可信节点的实例。

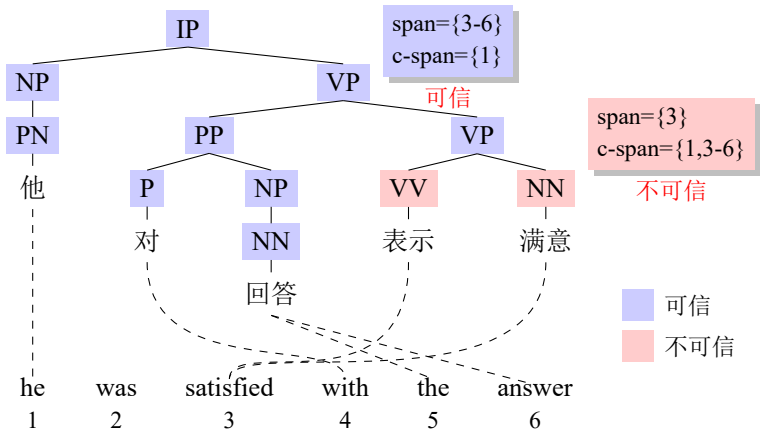


图 4.51: 标注了可信节点信息的句法树

进一步，可以定义树到串模型中合法的树片段：

定义 4.4.5 合法的树片段

如果一个树片段的根节点是可信节点，同时它的叶子节点中的非终结符系节点也是可信节点，那么这个树片段就是不产生词对齐歧义树片段，也被称为合法的树片段。

图4.52是一个基于可信节点得到的树到串规则：

$$VP(PP(P(\text{对}) NP(NN(\text{回答}))) VP_1) \rightarrow VP_1 \text{ with the answer}$$

其中，蓝色部分表示可以抽取到的规则，显然它的根节点和叶子非终结符节点都是可信节点。由于源语言树片段中包含一个变量（VP），因此需要对 VP 节点的 *Span* 所表示的目标语言范围进行泛化（红色方框部分）。

至此，对于任何一个树片段都能够使用上述方法判断它是否合法。如果合法，就可以抽取相应的树到串规则。但是，枚举句子中的所有树片段并不是一个很高效的方法，因为对于任何一个节点，以它为根的树片段数量随着其深度和宽度的增加呈指数增长。在 GHKM 方法中，为了避免低效的枚举操作，可以使用另一种方法抽取

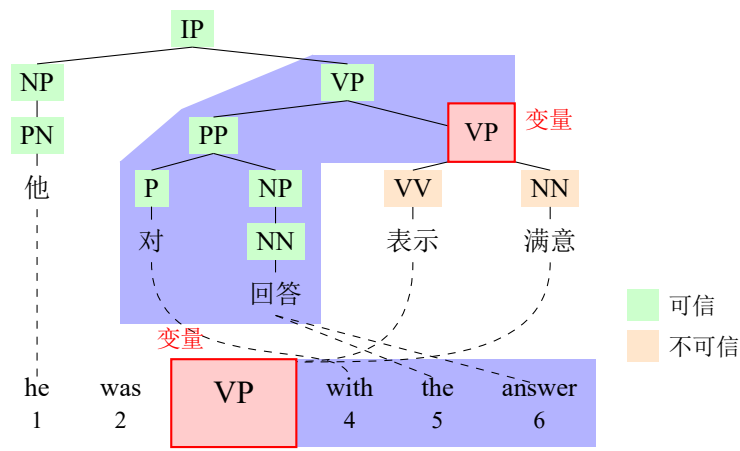


图 4.52: 根据可信结点得到的树到串翻译规则

规则。实际上，可信节点确定了哪些地方可以作为规则的边界（合法树片段的根节点或者叶子节点），可以把所有的可信节点看作是一个**边缘集合**（Frontier Set）。所谓边缘集合就是定义了哪些地方可以被“切割”，通过这种切割可以得到一个个合法的树片段，这些树片段无法再被切割为更小的合法树片段。图4.53给出了一个通过边缘集合定义的树切割。图右侧中的矩形框表示切割得到的树片段。

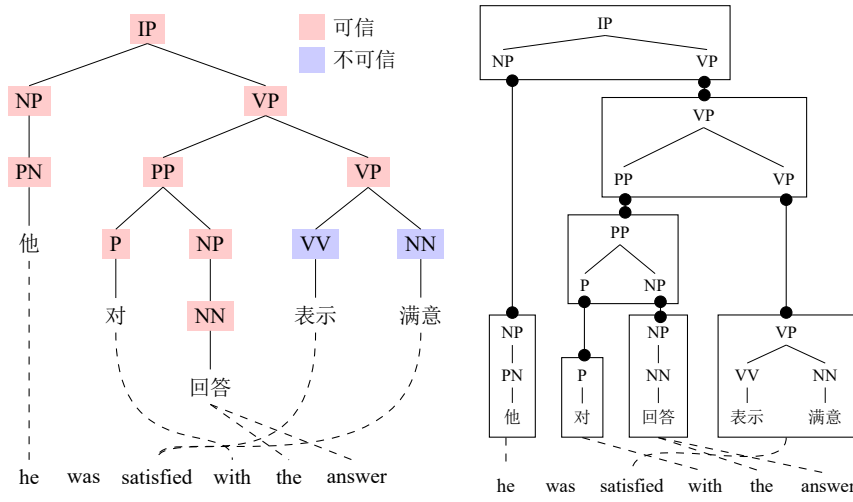


图 4.53: 根据边缘节点定义的树切割

需要注意的是，因为“NP→PN→ 他”对应着一个单目生成的过程，所以这里“NP(PN(他))”被看作是一个最小的树片段。当然，也可以把它当作两个树片段“NP(PN)”和“PN(他)”，不过这种单目产生式往往会导致解码时推导数量的膨胀。因此，这里约定把连续的单目生成看作是一个生成过程，它对应一个树片段，而不是多个。

将树进行切割之后，可以得到若干树片段，每个树片段都可以对应一个树到串规则。由于这些树片段不能被进一步切割，因此这样得到的规则也被称作**最小规则**（Minimal Rules）。它们就构成了树到串模型中最基本的翻译单元。图4.54展示了基于树切割得到的最小规则。其中左侧的每条规则都对应着右侧相同编号的树片段。

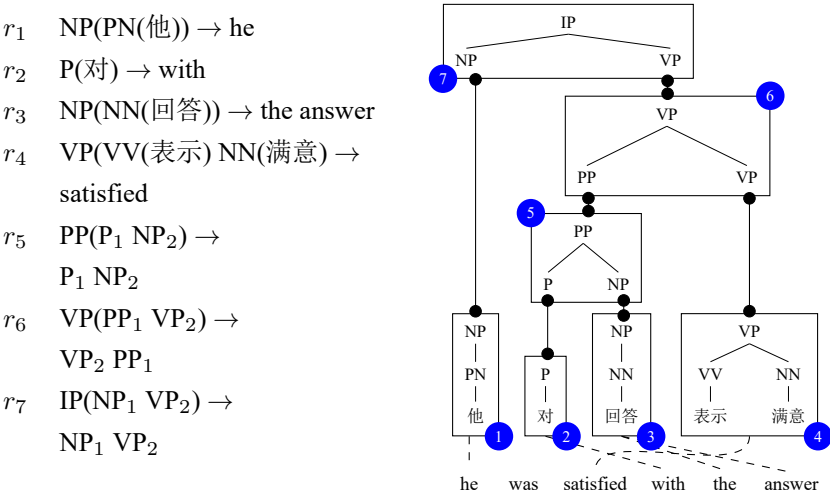


图 4.54: 基于树切割得到的最小规则实例

空对齐处理

空对齐是翻译中的常见现象。比如，一些虚词经常找不到在另一种语言中的对应，因此不会被翻译，这种情况也被称作空对齐。比如，在图4.54中目标语中的“was”就是一个空对齐单词。空对齐的使用可以大大增加翻译的灵活度。具体到树到串规则抽取任务，需要把空对齐考虑进来，这样能够覆盖更多的语言现象。

处理空对齐单词的手段非常简单。只需要把空对齐单词附着在它周围的规则上即可。也就是，检查每条最小规则，如果空对齐单词能够作为规则的一部分进行扩展，就可以生成一条新的规则。

图4.55展示了前面例子中“was”被附着在周围的规则上的结果。其中，含有红色“was”的规则是通过附着空对齐单词得到的新规则。比如，对于规则：

$$\text{NP(PN(他))} \rightarrow \text{he}$$

“was”紧挨着这个规则目标端的单词“he”，因此可以把“was”包含在规则的目标端，形成新的规则：

$$\text{NP(PN(他))} \rightarrow \text{he was}$$

- r_1 NP(PN(他)) \rightarrow he
 r_4 VP(VV(表示) NN(满意)) \rightarrow
 satisfied
 r_6 VP(PP₁ VP₂) \rightarrow VP₂ PP₁
 r_7 IP(NP₁ VP₂) \rightarrow NP₁ VP₂
 r_8 NP(PN(他)) \rightarrow he **was**
 r_9 VP(VV(表示) NN(满意)) \rightarrow
was satisfied
 r_{10} VP(PP₁ VP₂) \rightarrow
was VP₂ PP₁
 r_{11} IP(NP₁ VP₂) \rightarrow
 NP₁ **was** VP₂

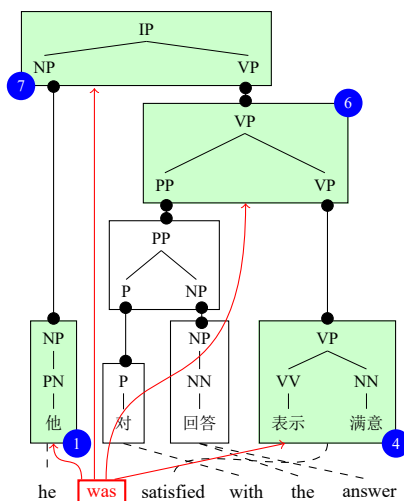


图 4.55: 树到串规则抽取中空对齐单词的处理 (绿色矩形)

通常，在规则抽取中考虑空对齐可以大大增加规则的覆盖度。

组合规则

最小规则是句法翻译模型中最小的翻译单元。但是，在翻译复杂句子的时候，往往需要更大范围的上下文信息，比如，本节开始图4.42中的例子，需要一条规则同时处理多个变量的调序，而这种规则很可能不是最小规则。为了得到“更大”的规则，一种方法是对最小规则进行组合。得到的规则称为 **composed- m** 规则，其中 m 表示这个规则是由 m 条最小规则组合而成。

- | | |
|---------------|---|
| r_1 | $\text{NP}(\text{PN}(\text{他})) \rightarrow \text{he}$ |
| r_5 | $\text{PP}(\text{P}_1 \text{ NP}_2) \rightarrow \text{P}_1 \text{ NP}_2$ |
| r_6 | $\text{VP}(\text{PP}_1 \text{ VP}_2) \rightarrow \text{VP}_2 \text{ PP}_1$ |
| r_7 | $\text{IP}(\text{NP}_1 \text{ VP}_2) \rightarrow \text{NP}_1 \text{ VP}_2$ |
| $r_{1,7}$ | $\text{IP}(\text{NP}(\text{PN}(\text{他})) \text{ VP}_1) \rightarrow$
he VP_1 |
| $r_{1,6,7}$ | $\text{IP}(\text{NP}(\text{PN}(\text{他})) \text{ VP}(\text{PP}_1 \text{ VP}_2)) \rightarrow$
$\text{he VP}_2 \text{ PP}_1$ |
| $r_{1,5,6,7}$ | $\text{IP}(\text{NP}(\text{PN}(\text{他})) \text{ VP}(\text{P}_1 \text{ NP}_2 \text{ VP}_3)) \rightarrow$
$\text{he VP}_3 \text{ P}_1 \text{ NP}_2$ |

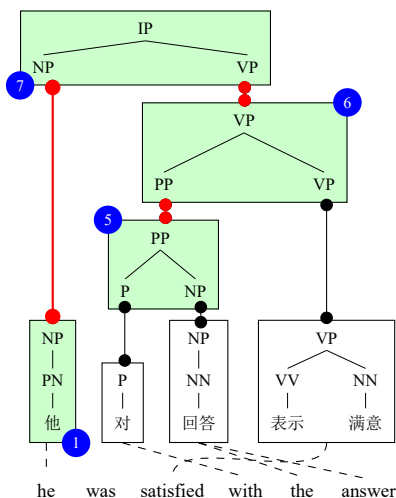


图 4.56: 对最小规则进行组合 (绿色矩形)

比规则组合的方法，SPMT 方法可以更有效的抽取包含短语的规则。

句法树二叉化

句法树是使用人类语言学知识归纳出来的一种解释句子结构的工具。比如,CTB[328]、PTB[197] 等语料就是常用的训练句法分析器的数据。

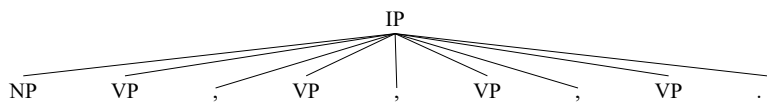
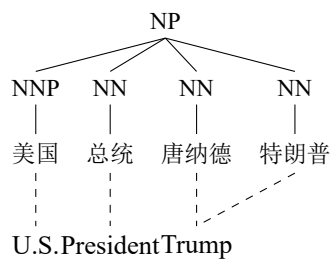


图 4.58: CTB 中含有多个分句的句法树结构

但是，这些数据的标注中会含有大量的扁平结构，如图4.58所示，多个分句可能会导致一个根节点下有很多个分支。这种扁平的结构会给规则抽取带来麻烦。



抽取到的规则：
NP(NNP₁ NN₂ NN(唐纳德) NN(特朗普))
→ NNP₁ NN₂ Trump
NP(NNP₁ NN(总统) NN(唐纳德) NN(特朗普))
→ NNP₁ President Trump
不能抽取到的规则：
NP(NN(唐纳德) NN(特朗普)) → Trump

图 4.59: 一个扁平的句法结构所对应的规则抽取结果

图4.59给出了一个实例，其中的名词短语（NP），包含四个词，都在同一层树结构中。由于“唐纳德 特朗普”并不是一个独立的句法结构，因此无法抽取类似于下面这样的规则：

$$NP(NN(唐纳德)) NN(特朗普)) \rightarrow Trump$$

对于这个问题，一种解决办法是把句法树变得更深，使局部的翻译片段更容易被抽取出来。常用的手段是树**二叉化**（Binarization）。比如，图4.60就是一个树二叉化的实例。二叉化生成了一些新的节点（记为 X-BAR），其中“唐纳德 特朗普”被作为一个独立的结构体现出来。这样，就能够抽取到规则：

$$NP\text{-}BAR(NN(唐纳德)) NN(特朗普)) \rightarrow Trump$$
$$NP\text{-}BAR(NN_1 NP\text{-}BAR_2) \rightarrow NN_1 NP\text{-}BAR_2$$

由于树二叉化可以帮助规则抽取得到更细颗粒度的规则，提高规则抽取的召回

率，因此成为了基于句法的机器翻译中的常用方法。二叉化方法也有很多不同的实现策略，比如：左二叉化 [342]、右二叉化 [315]、基于中心词的二叉化 [30, 142] 等。具体实现时可以根据实际情况进行选择。

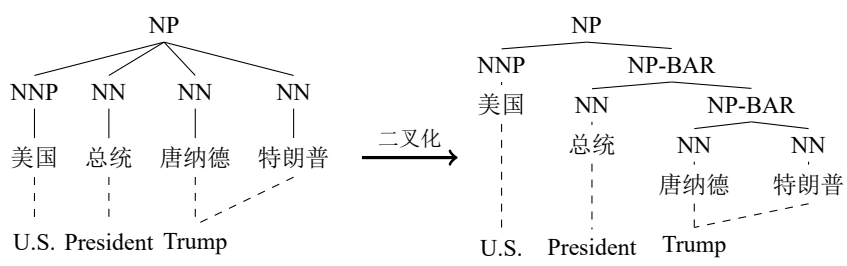


图 4.60: 句法树二叉化结果

4.4.4 树到树翻译规则抽取

树到串/串到树模型只在一个语言端使用句法树，而树到树模型可以同时利用源语言和目标语言句法信息，因此可以更细致地刻画两种语言结构的对应关系，进而更好地完成句法结构的调序和生成。树到树翻译中，需要两端都有树结构的规则，比如：

$$\langle \text{VP}, \text{VP} \rangle \rightarrow \langle \text{VP}(\text{PP}_1 \text{ VP}(\text{VV}(\text{表示}) \text{NN}_2)), \text{VP}(\text{VBZ}(\text{was}) \text{VP}(\text{VBN}_2 \text{ PP}_1)) \rangle$$

也可以把它写为如下形式：

$$\text{VP}(\text{PP}_1 \text{ VP}(\text{VV}(\text{表示}) \text{NN}_2)) \rightarrow \text{VP}(\text{VBZ}(\text{was}) \text{VP}(\text{VBN}_2 \text{ PP}_1))$$

其中，规则的左部是源语言句法树结构，右部是目标语言句法树结构，变量的下标表示对应关系。为了获取这样的规则，需要进行树到树规则抽取。最直接的办法是把 GHKM 方法推广到树到树翻译的情况。比如，可以利用双语结构的约束和词对齐，定义树的切割点，之后找到两种语言树结构的映射关系 [181]。

基于节点对齐的规则抽取

不过，GHKM 方法的问题在于过于依赖词对齐结果。在树到树翻译中，真正需要的是树结构（节点）之间的对应关系，而不是词对齐。特别是在两端都加入句法树结构约束的情况下，词对齐的错误可能会导致较为严重的规则抽取错误。图4.61就给出了一个实例，其中，中文的“了”被错误的对齐到了英文的“the”，导致很多高质量的规则无法被抽取出来。

换一个角度来看，词对齐实际上只是帮助模型找到两种语言句法树中节点的对

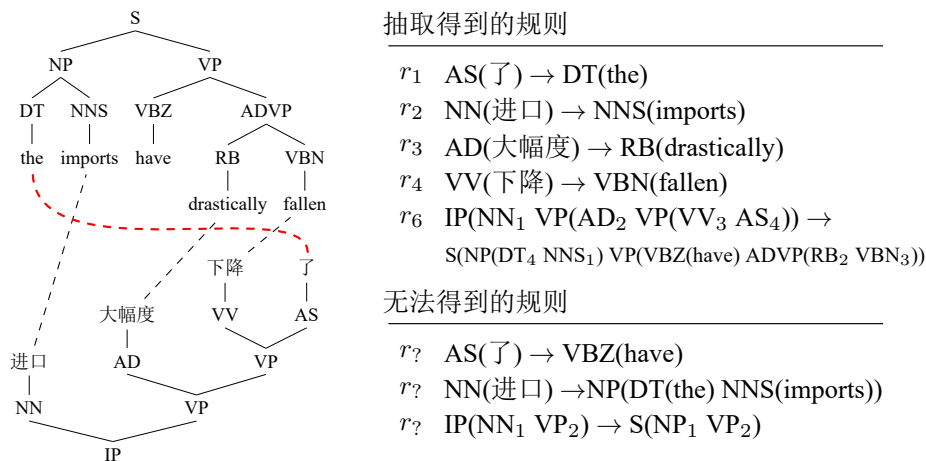


图 4.61: 基于词对齐的树到树规则抽取

应关系。如果能够直接得到句法树节点的对应，就可以避免掉词对齐的错误。也就是，可以直接使用节点对齐来进行树到树规则的抽取。首先，利用外部的节点对齐工具获得两棵句法树节点之间的对齐关系。之后，将每个对齐的节点看作是树片段的根节点，再进行规则抽取。图4.62展示了基于节点对齐的规则抽取结果。

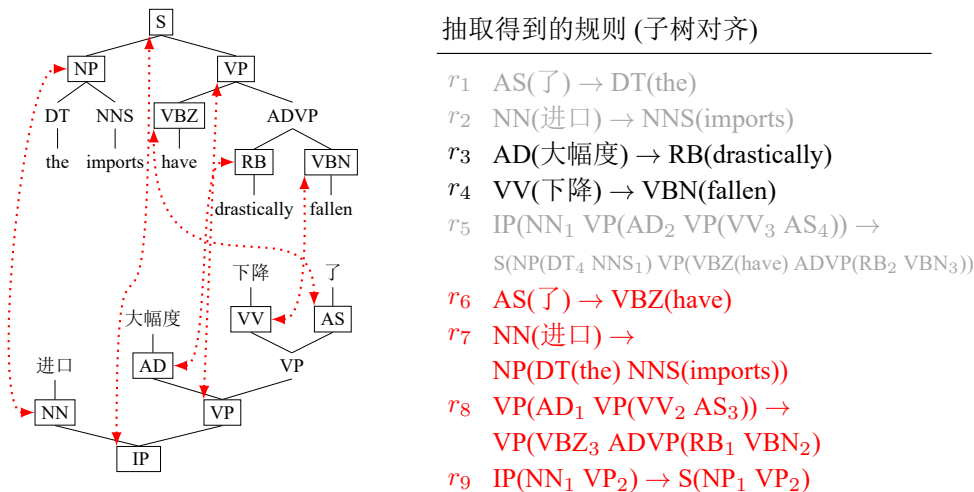
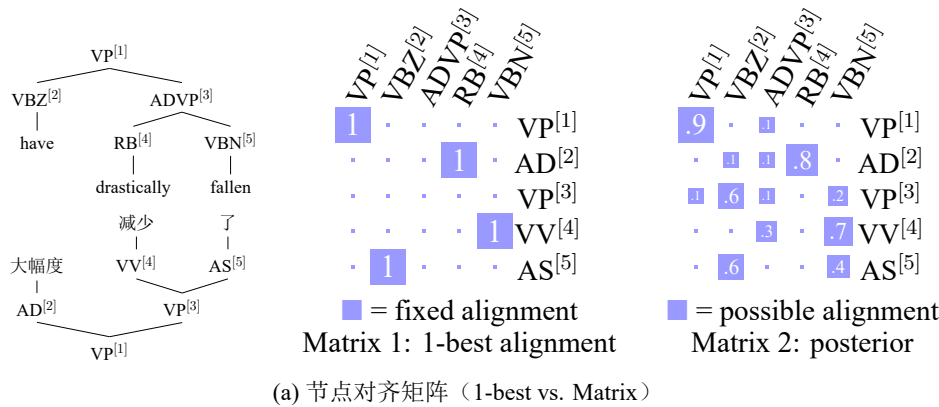


图 4.62: 基于节点对齐的树到树规则抽取

可以看到，节点对齐可以避免词对齐错误造成的影响。不过，节点对齐需要开发额外的工具。有很多方法可以参考，比如可以基于启发性规则 [94]、基于分类模型 [272]、基于无指导的方法 [318] 等。

基于对齐矩阵的规则抽取

同词对齐一样，节点对齐也会存在错误，这样就不可避免的造成规则抽取的错误。既然单一的对齐中含有错误，那能否让系统看到更多样的对齐结果，进而提高正确规则被抽取到的几率呢？答案是肯定的。实际上，在基于短语的模型中就有基于多个词对齐（如 *n*-best 词对齐）进行规则抽取的方法，这种方法可以在一定程度上提高短语的召回率。在树到树规则抽取中也可以使用多个节点对齐结果进行规则抽取。但是，简单使用多个对齐结果会使系统运行代价线性增长，而且即使是 *n*-best 对齐，也无法保证涵盖到正确的对齐结果。对于这个问题，另一种思路是使用对齐矩阵进行规则的“软”抽取。



Minimal Rules

Extracted from Matrix 1 (1-best)

- r_3 AD(大幅度) \rightarrow RB(drastically)
- r_4 VV(减少) \rightarrow VBN(fallen)
- r_6 AS(了) \rightarrow VBZ(have)
- r_8 VP(AD₁ VP(VV₂ AS₃)) \rightarrow
VP(VBZ₃ ADVP(RB₁ VBN₂))

Minimal Rules

Extracted from Matrix 2 (posterior)

- r_3 AD(大幅度) \rightarrow RB(drastically)
- r_4 VV(减少) \rightarrow VBN(fallen)
- r_6 AS(了) \rightarrow VBZ(have)
- r_8 VP(AD₁ VP(VV₂ AS₃)) \rightarrow
VP(VBZ₃ ADVP(RB₁ VBN₂))
- r_{10} VP(VV(减少) AS(了)) \rightarrow VBN(fallen)
- r_{11} VP(AD₁ VP₂) \rightarrow VP(VBZ₁ ADVP₂)

...

(b) 抽取得到的树到树翻译规则

图 4.63: 使用 1-best 节点对齐和概率化节点对齐矩阵的树到树规则抽取 [318]

所谓对齐矩阵，是描述两个句法树节点之间对应强度的数据结构。矩阵的每个单元中都是一个 0 到 1 之间的数字。规则抽取时，可以认为所有节点之间都存在对齐，这样可以抽取很多 *n*-best 对齐中无法覆盖的规则。图4.63展示了一个用对齐矩阵的进行规则抽取的实例。其中矩阵 1（Matrix 1）表示的标准的 1-best 节点对齐，矩阵 2（Matrix 2）表示的是一种概率化的对齐矩阵。可以看到使用矩阵 2 可以抽取

到更多样的规则。另外，值得注意的是，基于对齐矩阵的方法也同样适用于短语和层次短语规则的抽取。关于对齐矩阵的生成可以参考相关论文的内容 [183, 273, 274, 318]。

此外，在基于句法的规则抽取中，一般会对规则进行一些限制，以避免规则数量过大，系统无法处理。比如，可以限制树片段的深度、变量个数、规则组合的次数等等。这些限制往往需要根据具体任务进行设计和调整。

4.4.5 句法翻译模型的特征

基于语言学句法的翻译模型使用判别式模型对翻译推导进行建模（4.2.2节）。给定双语句对 (s, t) ，由 M 个特征经过线性加权，得到每个翻译推导 d 的得分，记为 $\text{score}(d, t, s) = \sum_{i=1}^M \lambda_i \cdot h_i(d, t, s)$ ，其中 λ_i 表示特征权重， $h_i(d, t, s)$ 表示特征函数。翻译的目标就是要找到使 $\text{score}(d, t, s)$ 达到最高的推导 d 。

这里，可以使用最小错误率训练对特征权重进行调优（4.2.6节）。而特征函数可参考如下定义：

基于短语的特征（对应于每条规则 $r: \langle \alpha_h, \beta_h \rangle \rightarrow \langle \alpha_r, \beta_r, \sim \rangle$ ）

- (h1-2) 短语翻译概率，即规则源语言和目标语言树覆盖的序列翻译概率。令函数 $\tau(\cdot)$ 返回一个树片段的叶子节点序列。对于规则：

$$\text{VP}(\text{PP}_1 \text{ VP}(\text{VV}(\text{表示}) \text{ NN}_2)) \rightarrow \text{VP}(\text{VBZ}(\text{was}) \text{ VP}(\text{VBN}_2 \text{ PP}_1))$$

可以得到：

$$\begin{aligned} \tau(\alpha_r) &= \text{PP 表示 NN} \\ \tau(\beta_r) &= \text{was VBN PP} \end{aligned}$$

于是，可以定义短语翻译概率为 $P(\tau(\alpha_r)|\tau(\beta_r))$ 和 $P(\tau(\beta_r)|\tau(\alpha_r))$ 。它们的计算方法与基于短语的系统是完全一样的⁹；

- (h3-4) 词汇化翻译概率，即 $P_{\text{lex}}(\tau(\alpha_r)|\tau(\beta_r))$ 和 $P_{\text{lex}}(\tau(\beta_r)|\tau(\alpha_r))$ 。这两个特征的计算方法与基于短语的系统也是一样的。

基于句法的特征（对应于每条规则 $r: \langle \alpha_h, \beta_h \rangle \rightarrow \langle \alpha_r, \beta_r, \sim \rangle$ ）

- (h5) 基于根节点句法标签的规则生成概率，即 $P(r|\text{root}(r))$ 。这里， $\text{root}(r)$ 是规则所对应的双语根节点 (α_h, β_h) ；
- (h6) 基于源语言端的规则生成概率，即 $P(r|\alpha_r)$ ，给定源语言端生成整个规则的概率；

⁹对于树到串规则， $\tau(\beta_r)$ 就是规则目标语言端的符号串。

- (h7) 基于目标语言端的规则生成概率，即 $P(r|\beta_r)$ ，给定目标语言端生成整个规则的概率。

其他特征（对应于整个推导 d ）

- (h8) 语言模型，即 $P_{lm}(t)$ ，用于度量译文的流畅度；
- (h9) 译文长度，即 $|t|$ ，用于避免模型过于倾向生成短译文（因为短译文语言模型分数高）；
- (h10) 翻译规则数量，学习对使用规则数量的偏好。比如，如果这个特征的权重较高，则表明系统更喜欢使用数量多的规则；
- (h11) 组合规则的数量，学习对组合规则的偏好；
- (h12) 词汇化规则的数量，学习对含有终结符规则的偏好；
- (h13) 低频规则的数量，学习对训练数据中出现频次低于 3 的规则偏好。低频规则大多不可靠，设计这个特征的目的也是为了区分不同质量的规则。

4.4.6 基于超图的推导空间表示

在完成建模后，剩下的问题是：如何组织这些翻译推导，完成高效的计算？本质上，基于句法的机器翻译与句法分析是一样的，因此关于翻译推导的组织可以借用句法分析中的一些概念。

在句法分析中，上下文无关文法（CFG）的分析过程可以被组织成一个叫**有向超图**（Directed Hyper-graph）的结构，或者简称为**超图** [141]：

定义 4.4.6 有向超图

一个有向超图 G 包含一个节点集合 N 和一个有向**超边**（Hyper-edge）集合 E 。每个有向超边包含一个头（Head）和一个尾（Tail），头指向 N 中的一个节点，尾是若干个 N 中的节点所构成的集合。

与传统的有向图不同，超图中的每一个边（超边）的尾可以包含多个节点。也就是说，每个超边从若干个节点出发最后指向同一个节点。这种定义完美契合了 CFG 的要求。比如，如果把节点看作是一个推导所对应树结构的根节点（含有句法标记），那么每个超边就可以表示一条 CFG 规则。

图4.64就展示了一个简单的超图。其中每个节点都有一个句法标记，句法标记下面记录了这个节点的跨度。超边 edge1 和 edge2 分别对应了两条 CFG 规则：

$$\begin{aligned} VP &\rightarrow VV NP \\ NP &\rightarrow NN NP \end{aligned}$$

对于规则“ $VP \rightarrow VV NP$ ”，超边的头指向 VP，超边的尾表示规则右部的两个变

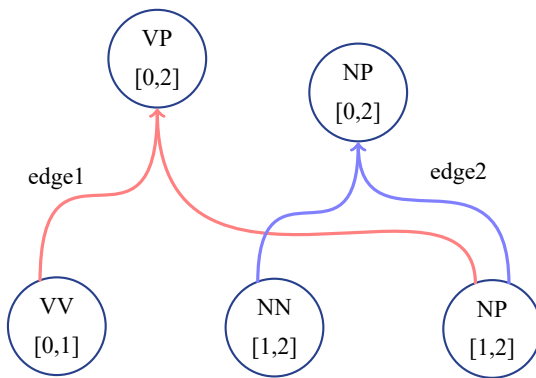


图 4.64: 超图实例

量 VV 和 NP。规则“ $NP \rightarrow NN NP$ ”也可以进行类似的解释。

不难发现，超图提供了一种非常紧凑的数据结构来表示多个推导，因为不同推导之间可以共享节点。如果把图4.64中的蓝色和红色部分看作是两个推导，那么它们就共享了同一个节点 $NN[1,2]$ 。能够想象，简单枚举一个句子所有的推导几乎是不可能的，但是用超图的方式却可以很有效地对指数级数量的推导进行表示。另一方面，超图上的运算常常被看作是一种基于半环的代数系统，而且人们发现许多句法分析和机器翻译问题本质上都是**半环分析**（Semi-ring Parsing）。不过，由于篇幅有限，这里不会对半环等结构展开讨论。感兴趣的读者可以查阅相关文献 [69, 91]。

从句法分析的角度看，超图最大程度地复用了局部的分析结果，使得分析可以“结构化”。比如，有两个推导：

$$d_1 = r_1 \circ r_2 \circ r_3 \circ r_4 \quad (4.30)$$

$$d_2 = r_1 \circ r_2 \circ r_3 \circ r_5 \quad (4.31)$$

其中， r_1 - r_5 分别表示不同的规则。 $r_1 \circ r_2 \circ r_3$ 是两个推导的公共部分。在超图表示中， $r_1 \circ r_2 \circ r_3$ 可以对应一个子图，显然这个子图也是一个推导，记为 $d' = r_1 \circ r_2 \circ r_3$ 。这样， d_1 和 d_2 不需要重复记录 $r_1 \circ r_2 \circ r_3$ ，重新写作：

$$d_1 = d' \circ r_4 \quad (4.32)$$

$$d_2 = d' \circ r_5 \quad (4.33)$$

引入 d' 的意义在于，整个分析过程具有了递归性。从超图上看， d' 可以对应以一个（或几个）节点为“根”的子图，因此只需要在这个（或这些）子图上增加新的超边就可以得到更大的推导。这个过程不断执行，最终完成对整个句子的分析。

在句法分析中，超图的结构往往被组织为一种 **Chart** 结构。所谓 **Chart**，就是一个表格，每个格代表了一个跨度，因此可以把所有覆盖这个跨度的推导都放入相应

的表格单元（Chart Cell）。对于上下文无关文法，表格里的每一项还会增加一个句法标记，用来区分不同句法功能的推导。

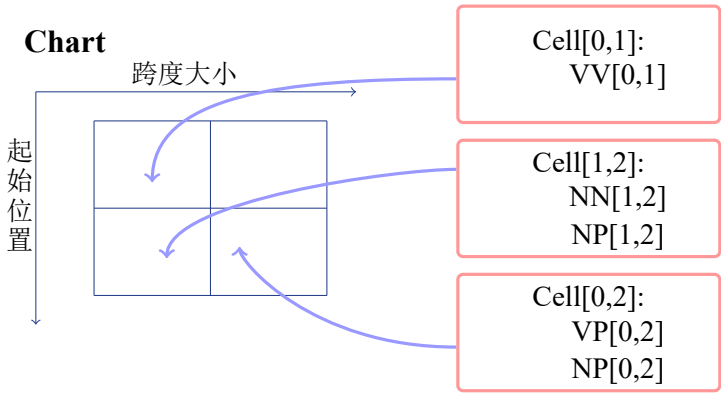


图 4.65: Chart 结构

如图4.65所示，覆盖相同跨度的节点会被放入同一个 Chart Cell，但是不同句法标记的节点会被看作是不同的项（Item）。这种组织方式建立了一个索引，通过索引可以很容易的访问同一个跨度下的所有推导。比如，如果采用自下而上的分析，可以从小跨度的 Chart Cell 开始，构建推导，并填写 Chart Cell。这个过程中，可以访问之前的 Chart Cell 来获得所需的局部推导（类似于前面提到的 d' ）。该过程重复执行，直到处理完最大跨度的 Chart Cell。而最后一个 Chart Cell 就保存了完整推导的根节点。通过回溯的方式，能够把所有推导都生成出来。

基于句法的机器翻译仍然可以使用超图进行翻译推导的表示。和句法分析一样，超图的每条边可以对应一个基于树结构的文法，超边的头代表文法的左部，超边的尾代表规则中变量所对应的超图中的节点¹⁰。图4.66 给出了一个使用超图来表示机器翻译推导的实例。可以看到，超图的结构是按源语言组织的，但是每个规则（超边）会包含目标语言的信息。由于同步翻译文法可以确保规则的源语言端和目标语言端都覆盖连续的词串，因此超图中的每个节点都对应一个源语言跨度，同时对应一个目标语的连续译文。这样，每个节点实际上代表了一个局部的翻译结果。

不过，机器翻译与句法分析也有不同之处。最主要的区别在于机器翻译使用了语言模型作为一个特征，比如 n -gram 语言模型。因为语言模型并不是上下文无关的，因此机器翻译中计算最优推导的方法和句法分析会有不同。常用的方法是，直接在每个 Chart Cell 中融合语言模型的分数，保留前 k 个结果；或者，在构建超图时不计算语言模型得分，等到构建完整个超图之后对最好的若干个推导用语言模型重新排序；又或者，将译文和语言模型都转化为加权有限状态自动机，之后直接对两个自动机做**组合**（Composition）得到新的自动机，最后得到融合语言模型得分的译文表示。

基于超图的推导表示方法有着很广泛的应用。比如，4.3节介绍的层次短语系统

¹⁰也可以把每个终结符看作是一个节点，这样一个超边的尾就对应规则的树片段中所有的叶子。

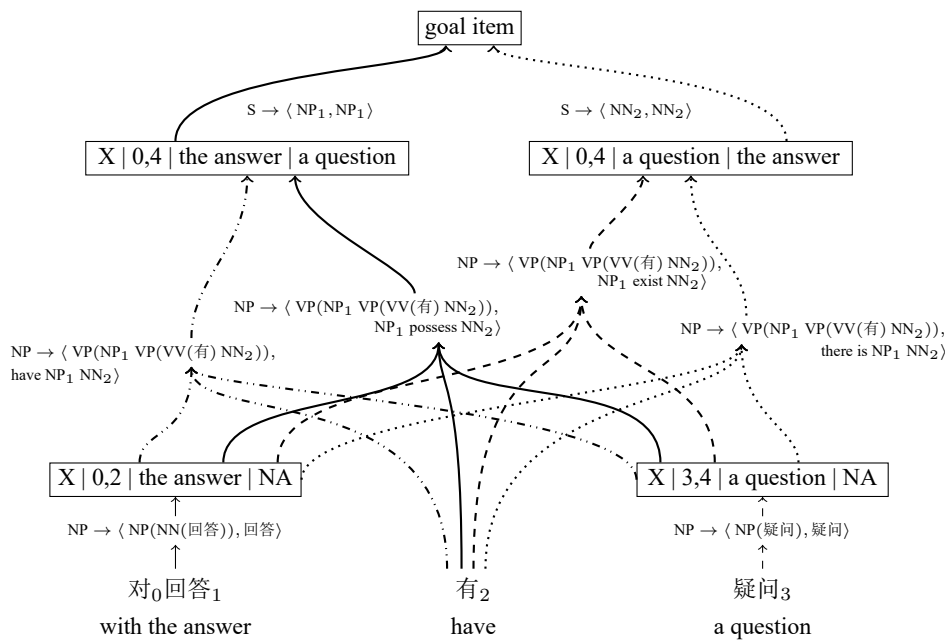


图 4.66: 机器翻译推导的超图表示

也可以使用超图进行建模，因为它也使用了同步文法。从这个角度说，基于层次短语的模型和基于语言学句法的模型本质上是一样的。它们的主要区别在于规则中的句法标记和抽取规则的方法不同。

4.4.7 基于树的解码 vs 基于串解码

解码的目标是找到得分 $\text{score}(d)$ 最大的推导 d 。这个过程通常被描述为：

$$\hat{d} = \arg \max_d \text{score}(d, \mathbf{s}, \mathbf{t}) \tag{4.34}$$

这也是一种标准的**基于串解码**（String-based Decoding），即通过句法模型对输入的源语言句子进行翻译得到译文串。不过，搜索所有的推导会导致巨大的解码空间。对于树到串和树到树翻译来说，源语言句法树是可见的，因此可以使用另一种解码方法——**基于树的解码**（Tree-based Decoding），即把输入的源语句法树翻译为目标语串。

表 4.4: 基于串解码 vs 基于树的解码

对比	基于树的解码	基于串解码
解码方法	$\hat{d} = \arg \max_{d \in D_{\text{tree}}} \text{score}(d)$	$\hat{d} = \arg \max_{d \in D} \text{score}(d)$
搜索空间	与输入的源语句法树兼容的推导 D_{tree}	所有的推导 D

对比	基于树的解码	基于串的解码
适用模型	树到串、树到树	所有的句法模型
解码算法	Chart 解码	CKY + 规则二义化
速度	快	一般较慢

表4.4对比了基于串和基于树的解码方法。可以看到，基于树的解码只考虑了与源语言句法树兼容的推导，因此搜索空间更小，解码速度会更快。

这里需要注意的是，不论是基于串的解释还是基于树的解码都是使用句法模型的方法，在翻译过程中都会生成翻译推导和树结构。二者的本质区别在于，基于树的解码把句法树作为显性的输入，而基于串的解释把句法树看作是翻译过程中的隐含变量。图4.67进一步解释了这个观点。

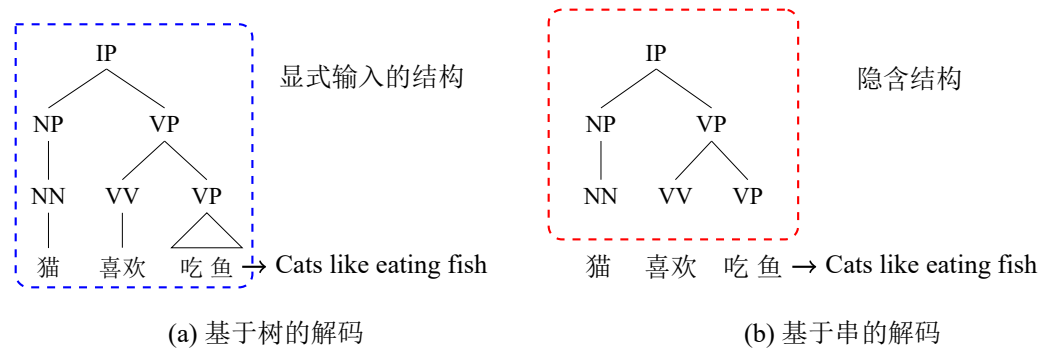


图 4.67: 句法树在不同解码方法中的角色

基于树的解码

基于树和基于串的解释都可以使用前面的超图结构进行推导的表示。基于树的解码方法相对简单。直接使用 Chart 结构组织解码空间即可。这里采用自底向上的策略，具体步骤如下：

- 从源语言句法树的叶子节点开始，自下而上访问输入句法树的节点；
- 对于每个树节点，匹配相应的规则；
- 从树的根节点可以得到翻译推导，最终生成最优推导所对应的译文。

这个过程如图4.68所示，可以看到，不同的 Chart Cell 对应不同跨度，每个 Chart Cell 会保存相应的句法标记（还有译文的信息）。

这里的问题在于规则匹配。对于每个树节点，需要知道以它为根可以匹配的规则有哪些。比较直接的解决方法是遍历这个节点下一定深度的句法树片段，用每个树片段在文法中找出相应的匹配规则，如图4.69所示。不过这种匹配是一种严格匹

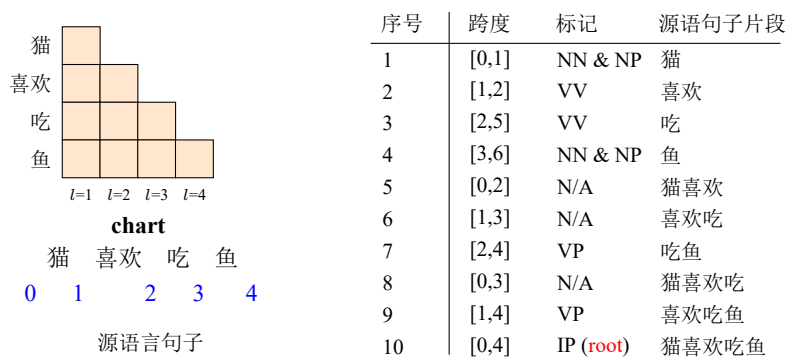


图 4.68: 基于树的解码中 Chart 的内容

配，因为它要求句法树片段内的所有内容都要与规则的源语言部分严格对应。有时，句法结构中的细微差别都会导致规则匹配不成功。因此，也可以考虑采用模糊匹配的方式提高规则的命中率，进而增加可以生成推导的数量 [351]。

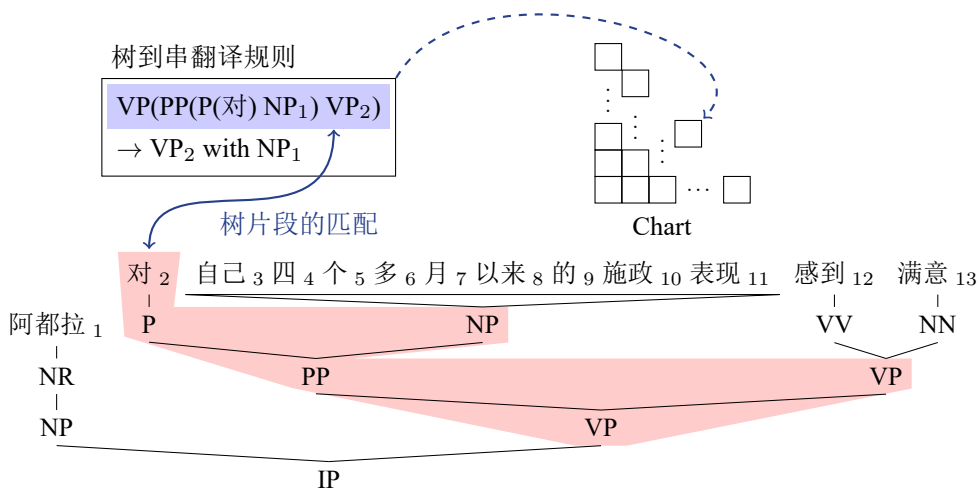


图 4.69: 基于树的规则匹配

基于串的解码

基于串的解码过程和句法分析几乎一样。对于输入的源语言句子，基于串的解码需要找到这个句子上的最优推导。唯一不同的地方在于，机器翻译需要考虑译文

的生成（语言模型的引入会使问题稍微复杂一些），但是源语言部分的处理和句法分析是一样的。因为不要求用户输入句法树，所以这种方法同时适用于树到串、串到树、树到树等多种模型。本质上，基于串的解码可以探索更多潜在的树结构，并增大

搜索空间（相比基于树的解码），因此该方法更有可能找到高质量翻译结果。

基于串的解码仍然可以用 Chart 来组织翻译推导。不过，一个比较有挑战的问题是如何找到每个规则能够匹配的源语言跨度。也就是，对于每个 Chart Cell，需要知道哪些规则可以被填入其中。因为，没有用户输入的句法树做指导，理论上输入句子的所有子串要与所有规则进行匹配。匹配时，需要考虑规则中源语言端的符号串（或者树结构的叶子序列）与输入词串匹配的全部可能性。

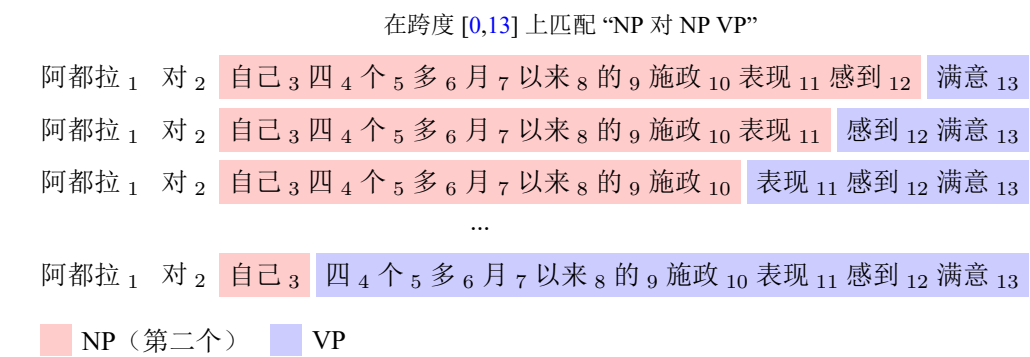


图 4.70: 在一个词串上匹配 “NP 对 NP VP”。连续变量的匹配对应了对词串不同位置的切割。

图4.70展示了规则匹配输入句子（包含 13 个词）的所有可能。可以看到，规则源语言端的连续变量会使得匹配情况变得复杂。对于长度为 n 的词串，匹配含有 m 个连续变量的规则的时间复杂度是 $O(n^{m-1})$ 。显然当变量个数增加时规则匹配是相当耗时的操作，甚至当变量个数过多时解码无法在可接受的时间内完成。

对于这个问题，有两种常用的解决办法：

- 对文法进行限制。比如，可以限制规则中变量的数量；或者不允许连续的变量，这样的规则也被称作满足 **Lexicalized Norm Form**（LNF）的规则。比如，层次短语规则就是 LNF 规则。由于 LNF 中单词（终结符）可以作为锚点，因此规则匹配时所有变量的匹配范围是固定的；
- 对规则进行二义化，使用 CKY 方法进行分析。这个方法也是句法分析中常用的策略。所谓规则二义化是把规则转化为最多只含两个变量或连续词串的规则（串到树规则）。比如，对于如下的规则：

$$\text{喜欢 } VP_1 NP_2 \rightarrow VP(VBZ(\text{likes}) VP_1 NP_2)$$

二义化的结果为：

$$\begin{aligned} \text{喜欢 } V103 &\rightarrow VP(VBZ(\text{likes}) V103) \\ VP_1 NP_2 &\rightarrow V103(VP_1 NP_2) \end{aligned}$$

可以看到,这两条新的规则源语言端只有两个部分,代表两个分叉。V103 是一个新的标签,它没有任何句法含义。不过,为了保证二义化后规则目标语部分的连续性,需要考虑源语言和目标语二义化的同步性 [315, 343]。这样的规则与 CKY 方法一起使用完成解码,具体内容可以参考4.3.4节的内容。

总的来说,基于句法的解码器较为复杂。无论是算法的设计还是工程技巧的运用,对开发者的能力都有一定要求。因此开发一个优秀的基于句法的机器翻译系统是一项有挑战的工作。

4.5 小结及深入阅读

统计机器翻译模型是近三十年内自然语言处理的重要里程碑之一。其统计建模的思想长期影响着自然语言处理的研究。无论是基于短语的模型,还是基于层次短语的模型,还是基于语言学句法的模型都在尝试回答:究竟应该用什么样的知识对机器翻译进行统计建模?不过,这个问题至今还没有确定的答案。但是,显而易见,统计机器翻译为机器翻译的研究提供了一种范式,即让计算机用概率化的“知识”描述翻译问题。这些“知识”就是统计模型的参数,模型可以从大量的双语和单语数据中自动学习参数。这种建模思想在今天的机器翻译研究中仍然随处可见。

本章对统计机器翻译的经典模型进行了介绍。从早期的基于短语的模型,再到层次短语模型,以及更为复杂的基于语言学句法的模型,本章尝试对不同的建模思想进行阐释。只是,统计机器翻译的内容非常丰富,很难通过一章的内容进行面面俱到的介绍。还有很多方向值得读者进一步了解:

- 统计机器翻译的成功很大程度上来自判别式模型引入任意特征的能力。因此,在统计机器翻译时代,很多工作都集中在新特征的设计上。比如,可以基于不同的统计特征和先验知识设计翻译特征 [41, 85, 217],也可以模仿分类任务设计大规模的稀疏特征 [43]。另一方面,模型训练和特征权重调优也是统计机器翻译中的重要问题,除了最小错误率训练,还有很多方法,比如,最大似然估计 [25, 153]、判别式方法 [22]、贝叶斯方法 [21, 46]、最小风险训练 [175, 260]、基于 Margin 的方法 [41, 306] 以及基于排序模型的方法 (PRO) [62, 113]。实际上,统计机器翻译的训练和解码也存在不一致的问题,比如,特征值由双语数据上的极大似然估计得到(没有剪枝),而解码时却使用束剪枝,而且模型的目标是最大化机器翻译评价指标。对于这个问题也可以通过调整训练的目标函数进行缓解 [195, 317]。
- 统计机器翻译的另一个基础问题是如何表示并获取翻译单元(如短语)。传统方法中,研究者大多使用词对齐或者句法树等结构化信息,通过启发性方法进行短语和翻译规则的获取。不过这类方法最大的问题是上游系统(比如,词对齐、句法分析等)中的错误会影响到下游系统。因此,很多研究者尝试使用更多样的对齐或者句法分析来指导翻译单元的获取。比如,可以绕过词对齐,直

接进行短语对齐 [55]; 也可以使用多个句法树或者句法森林来覆盖更多的句法现象, 进而增加规则抽取的召回率 [202, 323]。另一个有趣的方向是用更紧凑的方式表示更多样的翻译假设, 比如, 直接将翻译结果用有限状态自动机表示, 进行更大搜索空间上的解码 [27, 86]。

- 系统融合是具有统计机器翻译时代特色的研究方向。某种意义上说, 系统融合的兴起源于本世纪初各种机器翻译比赛。因为当时提升翻译性能的主要方法之一就是多个翻译引擎进行融合。系统融合的出发点是: 多样的翻译候选有助于生成更好的译文。系统融合有很多思路, 比较简单的方法是假设选择, 即从多个翻译系统的输出中直接选择一个译文 [8, 240, 319]; 另一种方法是用多个系统的输出构建解码格或者混淆网络, 这样可以生成新的翻译结果 [75, 105, 166]; 此外, 还可以在解码过程中动态融合不同模型 [168, 182]。另一方面, 也有研究者探讨如何在一个翻译系统中让不同的模型进行互补, 而不是简单的融合。比如, 可以控制句法在机器翻译中使用的程度, 让句法模型和层次短语模型处理各自擅长的问题 [321]。
- 语言模型是统计机器翻译系统所使用的重要特征。但是, 即使引入 n -gram 语言模型, 机器翻译系统仍然会产生语法上不正确的译文, 甚至会生成结构完全错误的译文。对于这个问题, 研究者尝试使用基于句法的语言模型。早期的探索有 Charniak 等人 [29] 和 Och 等人 [217] 的工作, 不过当时的结果并没有显示出基于句法的语言模型可以显著提升机器翻译的品质。后来, BBN 的研究团队提出了基于依存树的语言模型 [256], 这个模型可以显著提升层次短语模型的性能。正是凭借着这项技术, BBN 的系统也在多个机器翻译评测比赛中名列前茅, 引起了广泛关注。除此之外, 也有研究工作探索基于树替换文法等结构的语言模型 [324]。实际上, 树到树、串到树模型也可以被看作是一种对目标语言句法合理性的度量, 只不过目标语言的句法信息被隐含在翻译规则中。这时, 可以在翻译规则上设计相应的特征, 以达到引入目标语句法语言模型的目的。

神经机器翻译

5	人工神经网络和神经语言建模	207
5.1	深度学习与人工神经网络	
5.2	神经网络基础	
5.3	神经网络的张量实现	
5.4	神经网络的参数训练	
5.5	神经语言模型	
5.6	小结及深入阅读	
6	神经机器翻译模型	275
6.1	神经机器翻译的发展简史	
6.2	编码器-解码器框架	
6.3	基于循环神经网络的翻译模型及注意力机制	
6.4	Transformer	
6.5	序列到序列问题及应用	
6.6	小结及深入阅读	
7	神经机器翻译实战	341
7.1	神经机器翻译并不简单	
7.2	数据处理	
7.3	建模与训练	
7.4	推断	
7.5	进阶技术	
7.6	小结及深入阅读	



5. 人工神经网络和神经语言建模

人工神经网络（Artificial Neural Networks）或**神经网络**（Neural Networks）是描述客观世界的一种数学模型。这种模型和生物学上的神经系统在行为上有一些相似之处，但是人们更多的是把它作为一种计算工具，而非一个生物学模型。近些年，随着机器学习领域的快速发展，人工神经网络被更多的使用在对图像和自然语言处理问题的建模上。特别是，研究人员发现深层神经网络可以被成功训练后，学术界也逐渐形成了一种新的机器学习范式——深度学习。可以说，深度学习是近几年最受瞩目的研究领域之一，其应用也十分广泛。比如，图像识别的很多重要进展都来自深度学习模型的使用。包括机器翻译在内的很多自然语言处理任务中，深度学习也已经成为了一种标准模型。基于深度学习的表示学习方法也为自然语言处理开辟了新的思路。

本章将对深度学习的概念和技术进行介绍，目的是为第六章和第七章神经机器翻译的内容进行铺垫。此外，本章也会对深度学习在语言建模方面的应用进行介绍。这样，读者可以更容易地理解如何使用深度学习方法描述自然语言处理问题。同时，进一步了解一些相关的学术前沿，如预训练模型。

5.1 深度学习与人工神经网络

深度学习（Deep Learning）是机器学习研究中一个非常重要的分支，其概念来源于对人工神经网络的研究：通过人工神经元之间的连接建立一种数学模型，使计算机可以像人一样进行分析、学习和推理。

近几年来，随着深度学习技术的广泛传播与使用，“人工智能”这个名词在有些场合下甚至与“深度学习”划上了等号。这种理解非常片面，比较准确地说，“深度学习”是实现“人工智能”的一种技术手段。但从这种现象中，深度学习的火爆情况可见一斑。深度学习的技术浪潮以惊人的速度席卷世界，也改变了很多领域的现状，在数据挖掘、自然语言处理、语音识别、图像识别等各个领域随处可见深度学习的身影。自然语言处理领域中，深度学习在很多任务中已经处于“统治”地位。特别是，基于深度学习的表示学习方法已经成为自然语言处理的新范式，在机器翻译任务中更是衍生出了“神经机器翻译”这样全新的模型。

5.1.1 发展简史

神经网络最早出现在控制论中，随后更多地在连接主义中被提及。神经网络被提出的初衷并不是利用神经网络做一个简单的计算模型，而是希望将神经网络应用到一些自动控制相关的场景中。然而随着神经网络技术的持续发展，神经网络方法已经被广泛应用到各行各业的研究和实践工作中。

人工神经网络自 1943 年诞生至今，经历了多次高潮和低谷，这是任何一种技术都无法绕开的命运。然而，好的技术和方法终究不会被埋没，直到今天，神经网络和深度学习迎来了最好的 20 年。

早期的人工神经网络和第一次寒冬

最初，神经网络设计的初衷是用计算模型来模拟生物大脑中神经元的运行机理，这种想法哪怕是现在看来也是十分超前的。例如，目前很多机构关注的概念——“类脑计算”就是希望研究人脑的运行机制及相关的计算机实现方法。然而模拟大脑这件事并没有想象中的那么简单，众所周知，生物学中对人脑机制的研究是十分困难的，我们对人脑的运行机制尚不明确又何谈模拟呢？因而，神经网络技术一直在摸索着前行，发展到现在，其计算过程与人脑的运行机制已经大相径庭。

人工神经网络的第一个发展阶段是在二十世纪 40 年代到 70 年代，这个时期的人工神经网络还停留在利用线性模型模拟生物神经元的阶段，比如使用线性加权函数来描述输入 \mathbf{x} 和输出 y 之间的联系： $y = x_1 \cdot w_1 + \cdots + x_n \cdot w_n$ 。举一个简单例子，输入 \mathbf{x} 是某个地区的坐标和时间，输出 y 是该地区的温度，尽管真实的问题可能要复杂的多，但是线性模型确实有能力去拟合简单的函数关系。

这种线性模型在现在看来可能比较“简陋”，但是这类模型对后来的随机梯度下降等经典方法产生了深远影响。不过，显而易见的是，这种结构也存在着非常明显的缺陷，单层结构限制了它的学习能力，使它无法描述非线性问题，如著名的异或

函数（XOR）学习问题，然而非线性才是现实世界的普遍特征，第一代人工神经网络对很多事物的规律都无法准确描述。此后，神经网络的研究陷入了很长一段时间的低迷期。

神经网络的第二次高潮和第二次寒冬

虽然第一代神经网络受到了打击，但是 20 世纪 80 年代，第二代人工神经网络开始萌发新的生机。在这个发展阶段，生物属性已经不再是神经网络的唯一灵感来源，在**连接主义**（Connectionism）和**分布式表示**（Distributed representation）两种思潮的影响下，神经网络方法再次走入了人们的视线。

（1）符号主义与连接主义

人工智能领域始终存在着符号主义和连接主义之争。早期的人工智能研究在认知学中被称为**符号主义**（Symbolicism），符号主义认为人工智能源于数理逻辑，希望将世界万物的所有运转方式归纳成像文法一样符合逻辑规律的推导过程。符号主义的支持者们坚信基于物理符号系统（即符号操作系统）假设和有限合理性原理，就能通过逻辑推理来模拟智能。但被他们忽略的一点是，模拟智能的推理过程需要大量的先验知识支持，哪怕是在现代，生物学界也很难解释大脑中神经元的工作原理，因此也很难用符号系统刻画人脑逻辑。另一方面，连接主义则侧重于利用人工神经网络中神经元的连接去探索并模拟输入与输出之间存在的某种关系，这个过程不需要任何先验知识，其核心思想是“大量简单的计算单元连接到一起可以实现智能行为”，这种思想也推动了反向传播等多层神经网络方法的应用，并发展了包括长短时记忆模型在内的经典建模方法。2019 年 3 月 27 日，ACM 正式宣布将图灵奖授予 Yoshua Bengio, Geoffrey Hinton 和 Yann LeCun，以表彰他们提出的概念和工作使得深度学习神经网络有了重大突破，这三位获奖人均是人工智能连接主义学派的主要代表，从这件事中也可以看出连接主义对当代人工智能和深度学习的巨大影响。

（2）分布式表示

分布式表示的主要思想是“一个复杂系统的任何部分的输入都应该是多个特征共同表示的结果”，这种思想在自然语言处理领域的影响尤其深刻，它改变了刻画世界的角度，将世界万物从离散空间映射到多维连续空间。例如，在现实世界中，“张三”这个代号就代表着一个人。如果想要知道这个人亲属都有谁，因为有“A 和 B 如果姓氏相同，在一个家谱中，那么 A 和 B 是本家”这个先验知识在，在知道代号“张三”的情况下，可以得知“张三”的亲属是谁。但是如果不依靠这个先验知识，就无法得知“张三”的亲属是谁。但在分布式表示中，可以用一个实数向量，如 (0.1, 0.3, 0.4) 来表示“张三”这个人，这个人的所有特征信息都包含在这个实数向量中，通过在向量空间中的一些操作（如计算距离等），哪怕没有任何先验知识的存在，也完全可以找到这个人的所有亲属。在自然语言处理中，一个单词也用实数向量（词向量或词嵌入）表示，通过这种方式将语义空间重新刻画，将这个离散空间转化成了一个连续空间，这时单词就不再是一个简单的词条，而是由成百上千个特征共同描述

出来的，其中每个特征分别代表这个词的某个“方面”。

随着第二代人工神经网络的“脱胎换骨”，学者们又对神经网络方法燃起了希望之火，这也导致有些时候过分夸大了神经网络的能力。20 世纪 90 年代后期，由于在语音识别、自然语言处理等应用中，人们对神经网络方法期望过高，但是结果并没有达到预期，这也让很多人丧失了对神经网络方法的信任。相反，核方法、图模型等机器学习方法取得了很好的效果，这导致神经网络研究又一次进入低谷。

深度学习和神经网络方法的崛起

21 世纪初，随着深度学习浪潮席卷世界，人工神经网络又一次出现在人们的视野中。深度学习的流行源于 2006 年 Hinton 等人成功训练了一个深度信念网络（Deep Belief Network），在深度神经网络方法完全不受重视的情况下，大家突然发现深度神经网络完全是一个魔鬼般的存在，可以解决很多当时其他方法无法解决的问题。神经网络方法终于在一次又一次的被否定后，迎来了它的春天。随之针对神经网络和深度学习的一系列研究前赴后继地展开了，延续至今。

回过头来看，现代深度学习的成功主要有三方面的原因：

- 第一，模型和算法的不断完善和改进。这是现代深度学习能够获得成功的最主要原因；
- 第二，并行计算能力的提升使大规模的实践成为了可能。早期的计算机设备根本无法支撑深度神经网络训练所需要的计算量，导致实践变得十分困难。而设备的进步、计算能力的提升则彻底改变了这种窘境；
- 第三，以 Hinton 等人为代表的学者的坚持和持续投入。

另外，从应用的角度，数据量的快速提升和模型容量的增加也为深度学习的成功提供了条件，数据量的增加使得深度学习有了用武之地，例如，2000 年以来，双文本数据量无论在学术研究还是在工业实践中的使用数量都在逐年上升（如图5.1所示）。现在的深度学习模型参数量往往很大，因此需要大规模数据才能保证模型学习的充分性，而大数据时代的到来为训练这样的模型提供了数据基础。

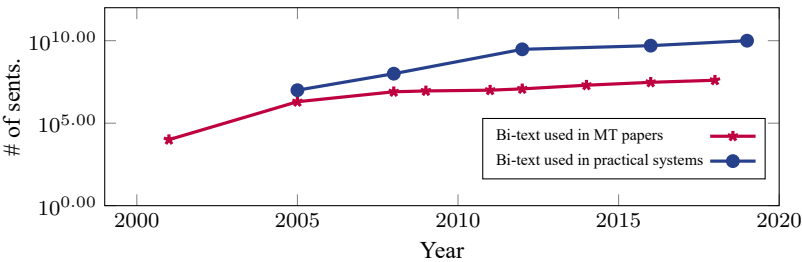


图 5.1: 2000 年以来各年的双语数据量

5.1.2 为什么需要深度学习

深度神经网络提供了一种简单的学习机制，即直接学习输入与输出的关系，通常把这种机制称为**端到端学习**（End-to-End Learning）。与传统方法不同，端到端学习并不需要人工定义特征或者进行过多的先验性假设，所有的学习过程都是由一个模型完成。从外面看这个模型只是建立了一种输入到输出的映射，而这种映射具体是如何形成的完全由模型的结构和参数决定。这样做的最大好处是，模型可以更加“自由”的进行学习。此外，端到端学习也引发了一个新的思考——如何表示问题？这也就是所谓的**表示学习**（Representation Learning）问题。在深度学习时代，问题的输入和输出的表示已经不再是人类通过简单的总结得到的规律，而是可以让计算机自己进行描述的一种可计算“量”，比如一个实数向量。由于这种表示可以被自动学习，因此也大大促进了计算机对语言文字等复杂现象的处理能力。

端到端学习和表示学习

端到端学习使机器学习不再像以往传统的特征工程方法一样需要经过繁琐的数据预处理、特征选择、降维等过程，而是直接利用人工神经网络自动从简单特征中提取、组合更复杂的特征，大大提升了模型能力和工程效率。以图5.2中的图像分类为例，在传统方法中，图像分类需要很多阶段的处理。首先，需要提取一些手工设计的图像特征，在将其降维之后，需要利用 SVM 等分类算法对其进行分类。与这种多阶段的流水线似的处理流程相比，端到端深度学习只训练一个神经网络，输入就是图片的像素表示，输出直接是分类类别。

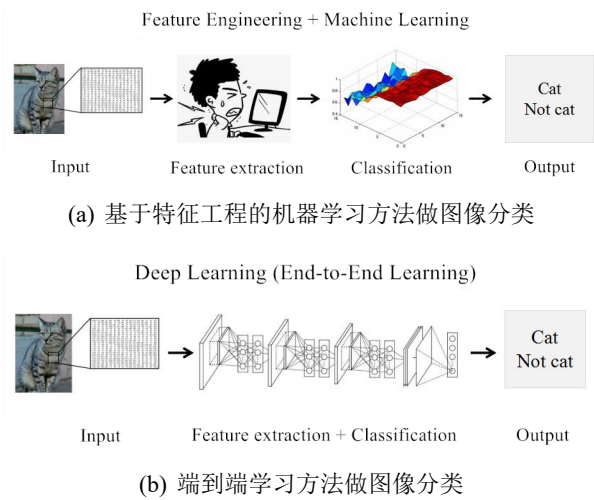


图 5.2: 特征工程 vs 端到端学习

传统的机器学习需要大量人工定义的特征，这些特征的构建往往会带来对问题的隐含假设。这种方法存在三方面的问题：

- 特征的构造需要耗费大量的时间和精力。在传统机器学习的特征工程方法中，

特征提取过程往往依赖于大量的先验假设，都基于人力完成的，这样导致相关系统的研发周期也大大增加：

- 最终的系统性能强弱非常依赖特征的选择。有一句话在业界广泛流传：“数据和特征决定了机器学习的上限”，但是人的智力和认知是有限的，因此人工设计的特征的准确性和覆盖度会受到限制；
- 通用性差。针对不同的任务，传统机器学习的特征工程方法需要选择出不同的特征，在这个任务上表现很好的特征在其他任务上可能没有效果。

端到端学习将人们从大量的特征提取工作之中解放出来，可以不需要太多人的先验知识。从某种意义上讲，对问题的特征提取全是自动完成的，这也意味着哪怕我们不是该任务的“专家”也可以完成相关系统的开发。此外，端到端学习实际上也隐含了一种新的对问题的表示形式——**分布式表示**（Distributed Representation）。在这种框架下，模型的输入可以被描述为分布式的实数向量，这样模型可以有更多的维度描述一个事物，同时避免传统符号系统对客观事物离散化的刻画。比如，在自然语言处理中，表示学习重新定义了什么是词，什么是句子。在本章后面的内容中也会看到，表示学习可以让计算机对语言文字的描述更加准确和充分。

深度学习的效果

相比于传统的基于特征工程的方法，基于深度学习的模型更加方便、通用，在系统性能上也普遍更优。这里以语言建模任务为例。语言建模的目的是开发一个模型来描述词串出现的可能性（见第二章）。这个任务已经有着很长的历史。表5.1给出了不同方法在标准的 PTB 上的困惑度结果¹。传统的 n -gram 语言模型由于面临维度灾难和数据稀疏问题，最终语言模型的性能并不是很好。而在深度学习模型中，通过引入循环神经网络等结构，所得到的语言模型可以更好地描述序列生成的问题。而最新的基于 Transformer 架构的语言模型将 PPL 从最初的 178.0 下降到了惊人的 35.7。可见深度学习为这个任务所带来的进步是巨大的。

5.2 神经网络基础

神经网络是一种由大量的节点（或称神经元）之间相互连接构成的计算模型。那么什么是神经元？神经元之间又是如何连接的？神经网络的数学描述又是什么样的？这一节将围绕这些问题对神经网络的基础知识进行系统的介绍。

5.2.1 线性代数基础

线性代数作为一个数学分支，广泛应用于科学和工程中，神经网络的数学描述中也大量使用了线性代数工具。因此，这里对线性代数的一些概念进行简要介绍，以

¹ 困惑度越低标明语言建模的效果越好。

表 5.1: 不同方法在 PTB 语言建模任务上的困惑度 (PPL)

模型	作者	年份	PPL
3-gram LM[23]	Brown et al.	1992	178.0
Feed-forward Neural LM[14]	Bengio et al.	2003	162.2
Recurrent NN-based LM[204]	Mikolov et al.	2010	124.7
Recurrent NN-LDA[206]	Mikolov et al.	2012	92.0
LSTM [336]	Zaremba et al.	2014	78.4
RHN[355]	Zilly et al.	2016	65.4
AWD-LSTM[201]	Merity et al.	2018	58.8
GPT-2 (Transformer)[237]	Radford et al.	2019	35.7

方便后续对神经网络的数学建模。

标量、向量和矩阵

标量 (Scalar)：标量亦称“无向量”，是一种只具有数值大小而没有方向的量，通俗地说，一个标量就是一个单独的数，这里特指实数²。一般用小写斜体表示标量。比如，对于 $a = 5$ ， a 就是一个标量。

向量 (Vector)：向量是由一组实数组成的有序数组。与标量不同，向量既有大小也有方向。可以把向量看作空间中的点，每个元素是不同坐标轴上的坐标。公式5.1和公式5.2展示了一个行向量和一个列向量。本章默认使用行向量，如 $\mathbf{a} = (a_1, a_2, a_3)$ ， \mathbf{a} 对应的列向量记为 \mathbf{a}^T 。

$$\mathbf{a} = \left(1 \quad 2 \quad 5 \quad 7 \right)$$

(5.1)

$$\mathbf{a}^T = \begin{pmatrix} 1 \\ 2 \\ 5 \\ 7 \end{pmatrix}$$

(5.2)

矩阵 (Matrix)：矩阵是一个按照长方阵列排列的实数集合，最早来自于方程组的系数及常数所构成的方阵。在计算机领域，通常将矩阵看作二维数组。我们用粗体的符号 \mathbf{a} 表示一个矩阵，如果该矩阵有 m 行 n 列，那么有 $\mathbf{a} \in R^{m \times n}$ 。这里，用不加粗的符号来表示矩阵中的元素，其中每个元素都被一个行索引和一个列索引所确定。例如， a_{ij} 表示第 i 行、第 j 列的矩阵元素。如下，公式5.3中 \mathbf{a} 定义了一个 2 行

²严格意义上，标量可以是复数等其他形式。这里为了方便讨论，仅以实数为对象。

2 列的矩阵。

$$\begin{aligned}\mathbf{a} &= \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}\end{aligned}\quad (5.3)$$

矩阵的转置

转置 (Transpose) 是矩阵的重要操作之一。矩阵的转置可以看作是将矩阵以对角线为镜像进行翻转：假设 \mathbf{a} 为 m 行 n 列的矩阵，第 i 行、第 j 列的元素是 a_{ij} ，即： $\mathbf{a} = (a_{ij})_{m \times n}$ ，把 $m \times n$ 矩阵 \mathbf{a} 的行换成同序数的列得到一个 $n \times m$ 矩阵，则得到 \mathbf{a} 的转置矩阵，记为 \mathbf{a}^T ，其中 $a_{ji}^T = a_{ij}$ 。例如：

$$\mathbf{a} = \begin{pmatrix} 1 & 3 & 2 & 6 \\ 5 & 4 & 8 & 2 \end{pmatrix}\quad (5.4)$$

$$\mathbf{a}^T = \begin{pmatrix} 1 & 5 \\ 3 & 4 \\ 2 & 8 \\ 6 & 2 \end{pmatrix}\quad (5.5)$$

向量可以看作只有一行（列）的矩阵。对应地，向量的转置可以看作是只有一列（行）的矩阵。标量可以看作是只有一个元素的矩阵。因此，标量的转置等于它本身，即 $a^T = a$ 。

矩阵加法和数乘

矩阵加法又被称作**按元素加法** (Element-wise Addition)。它是指两个矩阵把其对应元素加在一起的运算，通常的矩阵加法被定义在两个形状相同的矩阵上。两个 $m \times n$ 矩阵 \mathbf{a} 和 \mathbf{b} 的和，标记为 $\mathbf{a} + \mathbf{b}$ ，它也是个 $m \times n$ 矩阵，其内的各元素为其相对应元素相加后的值。如果矩阵 $\mathbf{c} = \mathbf{a} + \mathbf{b}$ ，则 $c_{ij} = a_{ij} + b_{ij}$ 。公式5.6展示了矩阵之间进行加法的计算过程。

$$\begin{pmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 1+0 & 3+0 \\ 1+7 & 0+5 \\ 1+2 & 2+1 \end{pmatrix} = \begin{pmatrix} 1 & 3 \\ 8 & 5 \\ 3 & 3 \end{pmatrix}\quad (5.6)$$

矩阵加法满足以下运算规律：

- 交换律： $\mathbf{a} + \mathbf{b} = \mathbf{b} + \mathbf{a}$ 。

- 结合律: $(\mathbf{a} + \mathbf{b}) + \mathbf{c} = \mathbf{a} + (\mathbf{b} + \mathbf{c})$ 。
- $\mathbf{a} + \mathbf{0} = \mathbf{a}$, 其中 $\mathbf{0}$ 指的是零矩阵, 即元素皆为 0 的矩阵。
- $\mathbf{a} + (-\mathbf{a}) = \mathbf{0}$, 其中 $-\mathbf{a}$ 是矩阵 \mathbf{a} 的负矩阵, 即将矩阵 \mathbf{a} 的每个元素取负得到的矩阵。

矩阵的**数乘** (Scalar Multiplication) 是指标量 (实数) 与矩阵的乘法运算, 计算过程是将标量与矩阵的每个元素相乘, 最终得到与原矩阵形状相同的矩阵。例如, 矩阵 $\mathbf{a} = (a_{ij})_{m \times n}$ 与标量 k 进行数乘运算, 其结果矩阵 $\mathbf{b} = (ka_{ij})_{m \times n}$, 即 $k(a_{ij})_{m \times n} = (ka_{ij})_{m \times n}$ 。下面的式子展示了矩阵数乘的计算过程:

$$\mathbf{a} = \begin{pmatrix} 3 & 2 & 7 \\ 5 & 8 & 1 \end{pmatrix} \quad (5.7)$$

$$2\mathbf{a} = \begin{pmatrix} 6 & 4 & 14 \\ 10 & 16 & 2 \end{pmatrix} \quad (5.8)$$

矩阵的数乘满足以下运算规律, 其中 k 和 l 是实数, \mathbf{a} 和 \mathbf{b} 是形状相同的矩阵:

- 右分配律: $k(\mathbf{a} + \mathbf{b}) = k\mathbf{a} + k\mathbf{b}$ 。
- 左分配律: $(k + l)\mathbf{a} = k\mathbf{a} + l\mathbf{a}$ 。
- 结合律: $(kl)\mathbf{a} = k(l\mathbf{a})$ 。

矩阵乘法和矩阵点乘

矩阵乘法是矩阵运算中最重要的操作之一, 为了与矩阵点乘区分, 通常也把矩阵乘法叫做矩阵叉乘。假设 \mathbf{a} 为 $m \times p$ 的矩阵, \mathbf{b} 为 $p \times n$ 的矩阵, 对 \mathbf{a} 和 \mathbf{b} 作矩阵乘法的结果是一个 $m \times n$ 的矩阵 \mathbf{c} , 其中矩阵 \mathbf{c} 中第 i 行、第 j 列的元素可以表示为:

$$(\mathbf{ab})_{ij} = \sum_{k=1}^p a_{ik}b_{kj} \quad (5.9)$$

只有当第一个矩阵的列数与第二个矩阵的行数相等时, 两个矩阵才可以作矩阵乘法。公式5.10展示了矩阵乘法的运算过程, 若 $\mathbf{a} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$, 则有:

$$\begin{aligned} \mathbf{c} &= \mathbf{ab} \\ &= \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{pmatrix} \end{aligned} \quad (5.10)$$

矩阵乘法满足以下运算规律：

- 结合律：若 $\mathbf{a} \in R^{m \times n}$, $\mathbf{b} \in R^{n \times p}$, $\mathbf{c} \in R^{p \times q}$, 则 $(\mathbf{ab})\mathbf{c} = \mathbf{a}(\mathbf{bc})$ 。
- 左分配律：若 $\mathbf{a} \in R^{m \times n}$, $\mathbf{b} \in R^{m \times n}$, $\mathbf{c} \in R^{n \times p}$, 则 $(\mathbf{a} + \mathbf{b})\mathbf{c} = \mathbf{ac} + \mathbf{bc}$ 。
- 右分配律：若 $\mathbf{a} \in R^{m \times n}$, $\mathbf{b} \in R^{n \times p}$, $\mathbf{c} \in R^{n \times p}$, 则 $\mathbf{a}(\mathbf{b} + \mathbf{c}) = \mathbf{ab} + \mathbf{ac}$ 。

可以将线性方程组用矩阵乘法表示，如对于线性方程组
$$\begin{cases} 5x_1 + 2x_2 = y_1 \\ 3x_1 + x_2 = y_2 \end{cases}, \text{ 可}$$

以表示为 $\mathbf{ax}^T = \mathbf{y}^T$, 其中 $\mathbf{a} = \begin{pmatrix} 5 & 2 \\ 3 & 1 \end{pmatrix}$, $\mathbf{x}^T = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, $\mathbf{y}^T = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$ 。

矩阵的点乘就是两个形状相同的矩阵各个对应元素相乘，矩阵点乘也被称为**按元素乘积**（Element-wise Product）或 Hadamard 乘积，记为 $\mathbf{a} \odot \mathbf{b}$ 。例如，对于

$$\mathbf{a} = \begin{pmatrix} 1 & 0 \\ -1 & 3 \end{pmatrix} \quad (5.11)$$

$$\mathbf{b} = \begin{pmatrix} 3 & 1 \\ 2 & 1 \end{pmatrix} \quad (5.12)$$

矩阵点乘的计算如下：

$$\begin{aligned} \mathbf{c} &= \mathbf{a} \odot \mathbf{b} \\ &= \begin{pmatrix} 1 \times 3 & 0 \times 1 \\ -1 \times 2 & 3 \times 1 \end{pmatrix} \end{aligned} \quad (5.13)$$

线性映射

线性映射（Linear Mapping）或**线性变换**（Linear Transformation）是从一个向量空间 V 到另一个向量空间 W 的映射函数 $f: v \rightarrow w$, 且该映射函数保持加法运算和数量乘法运算，即对于空间 V 中任何两个向量 \mathbf{u} 和 \mathbf{v} 以及任何标量 c , 有：

$$f(\mathbf{u} + \mathbf{v}) = f(\mathbf{u}) + f(\mathbf{v}) \quad (5.14)$$

$$f(c\mathbf{v}) = cf(\mathbf{v}) \quad (5.15)$$

利用矩阵 $\mathbf{a} \in R^{m \times n}$, 可以实现两个有限维欧氏空间的映射函数 $f: R^n \rightarrow R^m$ 。例如 n 维列向量 \mathbf{x}^T 与 $m \times n$ 的矩阵 \mathbf{a} , 向量 \mathbf{x}^T 左乘矩阵 \mathbf{a} , 可将向量 \mathbf{x}^T 映射为 m 列向量，对于

$$\mathbf{x}^T = (x_1, x_2, \dots, x_n)^T \quad (5.16)$$

$$\mathbf{a} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & \cdots & \cdots & a_{mn} \end{pmatrix} \quad (5.17)$$

可以得到：

$$\begin{aligned} \mathbf{y}^T &= \mathbf{a}\mathbf{x}^T \\ &= \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{pmatrix} \end{aligned} \quad (5.18)$$

上例中矩阵 \mathbf{a} 定义了一个从 R^n 到 R^m 的线性映射：向量 $\mathbf{x}^T \in R^n$ 和 $\mathbf{y}^T \in R^m$ 别为两个空间中的列向量，即大小为 $n \times 1$ 和 $m \times 1$ 的矩阵。

范数

工程领域，经常会使用被称为**范数**（Norm）的函数衡量向量大小，范数为向量空间内的所有向量赋予非零的正长度或大小。对于一个 n 维向量 \mathbf{x} ，一个常见的范数函数为 l_p 范数，通常表示为 $\|\mathbf{x}\|_p$ ，其中 $p \geq 0$ ，是一个标量形式的参数。常用的 p 的取值有 1、2、 ∞ 等。范数的计算公式为：

$$\begin{aligned} l_p(\mathbf{x}) &= \|\mathbf{x}\|_p \\ &= \left(\sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}} \end{aligned} \quad (5.19)$$

l_1 范数为向量的各个元素的绝对值之和：

$$\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i| \quad (5.20)$$

l_2 范数为向量的各个元素平方和的二分之一次方：

$$\begin{aligned} \|\mathbf{x}\|_2 &= \sqrt{\sum_{i=1}^n x_i^2} \\ &= \sqrt{\mathbf{x}^T \mathbf{x}} \end{aligned} \quad (5.21)$$

l_2 范数被称为**欧几里得范数**（Euclidean Norm）。从几何角度，向量也可以表示为从原点出发的一个带箭头的有向线段，其 l_2 范数为线段的长度，也常被称为向量的模。 l_2 范数在机器学习中非常常用，向量 \mathbf{x} 的 l_2 范数经常简化为 $\|\mathbf{x}\|$ ，可以简单地通过点积 $\mathbf{x}^T \mathbf{x}$ 计算。

l_∞ 范数为向量的各个元素的最大绝对值：

$$\|\mathbf{x}\|_\infty = \max\{x_1, x_2, \dots, x_n\} \tag{5.22}$$

广义上讲，范数是将向量映射到非负值的函数，其作用是衡量向量 \mathbf{x} 到坐标原点的距离。更严格的说，范数并不拘于 l_p 范数，任何一个同时满足下列性质的函数都可以作为范数：

- 若 $f(x) = 0$ ，则 $x = 0$ 。
- 三角不等式： $f(x + y) \leq f(x) + f(y)$ 。
- 任意实数 α ， $f(\alpha x) = |\alpha| f(x)$ 。

在深度学习中，有时候希望衡量矩阵的大小，这时可以考虑使用 **Frobenius 范数**（Frobenius Norm）。计算方式为：

$$\|\mathbf{a}\|_F = \sqrt{\sum_{i,j} a_{i,j}^2} \tag{5.23}$$

5.2.2 人工神经元和感知机

生物学中，神经元是神经系统的基本组成单元。图5.3展示了一个生物神经元实例。

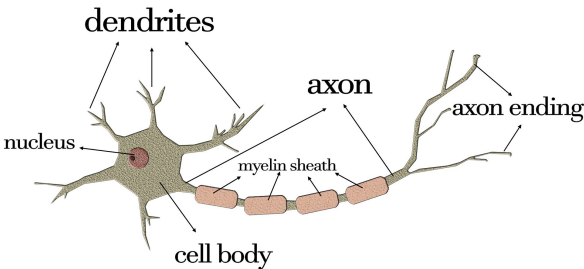


图 5.3: 生物神经元

同样，人工神经元是人工神经网络的基本单元。在人们的想象中，人工神经元应该与生物神经元类似。但事实上，二者在形态上是有明显差别的。如图5.4 是一个典型的人工神经元，其本质是一个形似 $y = f(\mathbf{x} \cdot \mathbf{w} + b)$ 的函数。显而易见，一个神经元主要由 \mathbf{x} , \mathbf{w} , b , f 四个部分构成。其中 \mathbf{x} 是一个形如 (x_0, x_1, \dots, x_n) 的实数向

量，在一个神经元中担任“输入”的角色。 \mathbf{w} 是一个权重矩阵，其中的每一个元素都对应着一个输入和一个输出，代表着“某输入对某输出的贡献程度”，通常也被理解为神经元连接的**权重**（weight）。 b 被称作偏置，是一个实数。 f 被称作激活函数，其本质是一个非线性函数。可见，一个人工神经元的功能是将输入向量与权重矩阵右乘（做内积）后，加上偏置量，经过一个非线性激活函数得到一个标量结果。

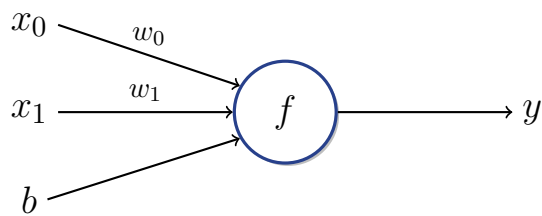


图 5.4: 人工神经元

感知机 —— 最简单的人工神经元模型

感知机是人工神经元的一种实例，在上世纪 50-60 年代被提出后，对神经网络研究产生了深远的影响。感知机模型如图5.5所示，其输入是一个 n 维二值向量 $\mathbf{x} = (x_0, x_1, \dots, x_n)$ ，其中 $x_i = 0$ 或 1 。权重 $\mathbf{w} = (w_0, w_1, \dots, w_n)$ ，每个输入变量对应一个权重 w_i （实数）。偏置 b 是一个实数变量 $(-\sigma)$ 。输出也是一个二值结果，即 $y = 0$ 或 1 。 y 值的判定由输入的加权和是否大于（或小于）一个阈值 σ 决定（公式5.24）：

$$y = \begin{cases} 0 & \sum_i x_i \cdot w_i - \sigma < 0 \\ 1 & \sum_i x_i \cdot w_i - \sigma \geq 0 \end{cases} \tag{5.24}$$

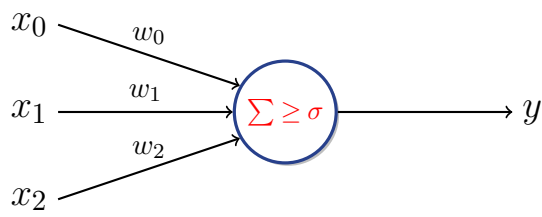


图 5.5: 感知机模型

感知机可以做一些简单的决策。举一个非常简单的例子，有一场音乐会，你正在纠结是否去参加，有三个因素会影响你的决定：

- x_0 ：剧场是否离你足够近（是，则 $x_0 = 1$ ；否则 $x_0 = 0$ ）；
- x_1 ：票价是否低于 300 元（是，则 $x_1 = 1$ ；否则 $x_1 = 0$ ）；
- x_2 ：女朋友是否喜欢音乐会（是，则 $x_2 = 1$ ；否则 $x_2 = 0$ ）。

在这种情况下应该如何做出决定呢？比如，女朋友很希望和你一起去看音乐会，但是剧场很远而且票价 500 元，如果这些因素对你都是同等重要的（即 $w_0 = w_1 = w_2$ ，假设这三个权重都设置为 1）那么会得到一个综合得分：

$$\begin{aligned} x_0 \cdot w_0 + x_1 \cdot w_1 + x_2 \cdot w_2 &= 0 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 \\ &= 1 \end{aligned} \tag{5.25}$$

如果你不是十分纠结的人，能够接受不完美的事情，你可能会把 σ 设置为 1，于是 $\sum w_i \cdot x_i - \sigma \geq 0$ ，那么你会去看音乐会。可以看出，上面的例子的本质就是一个如图 5.6 的感知机：

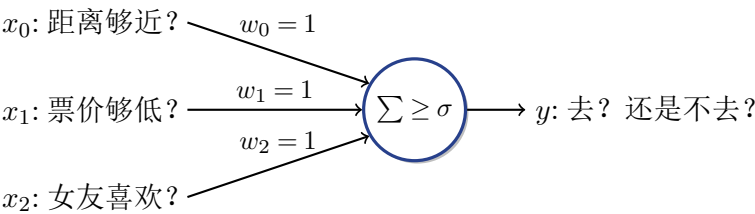


图 5.6: 预测是否去剧场的感知机（权重相同）

神经元内部权重

在上面的例子中，连接权重代表着每个输入因素对最终输出结果的重要程度，为了得到令人满意的决策，需要不断调整权重。如果你是守财奴，则会对票价看得更重一些，这样你会用不均匀的权重计算每个因素的影响，比如： $w_0 = 0.5$ ， $w_1 = 2$ ， $w_2 = 0.5$ ，此时感知机模型如图 5.7 所示。在这种情况下，女友很希望和你一起去看音乐会，但是剧场很远而且票价 500 元，会导致你不去看音乐会，因为

$$\begin{aligned} \sum_i x_i \cdot w_i &= 0 \cdot 0.5 + 0 \cdot 2 + 1 \cdot 0.5 \\ &= 0.5 \\ &< \sigma = 1 \end{aligned} \tag{5.26}$$

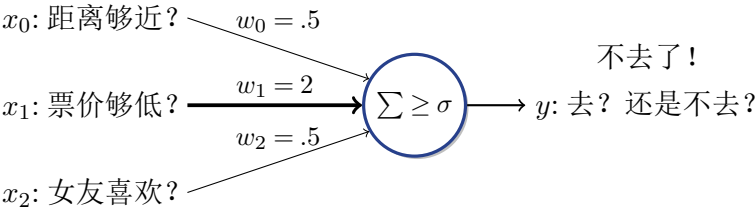


图 5.7: 预测是否去剧场的感知机（改变权重）

当然，结果是女友对这个决定非常不满意，让你跪键盘上反思一下自己。

神经元的输入 —— 离散 vs 连续

在遭受了女友一万点伤害之后，你意识到决策考虑的因素（即输入）不应该只是非 0 即 1，而应该把“程度”考虑进来，于是你改变了三个输入的形式：

- x_0 : 10/距离
- x_1 : 150/票价
- x_2 : 女朋友是否喜欢

在新修改的模型中， x_0 和 x_1 变成了连续变量， x_2 仍然是离散变量，如图5.8。

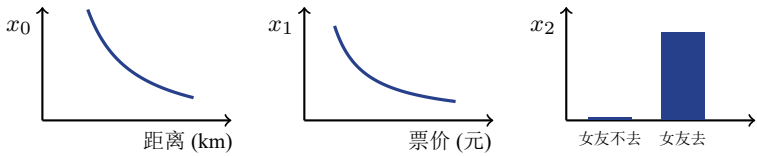


图 5.8: 神经元输入的不同形式

使用修改后的模型做决策：女朋友很希望和你一起，但是剧场有 20km 远而且票价有 500 元。于是有 $x_0 = 10/20$ ， $x_1 = 150/500$ ， $x_2 = 1$ 。

$$\begin{aligned} \sum_i x_i \cdot w_i &= 0.5 \cdot 0.5 + 0.3 \cdot 2 + 1 \cdot 0.5 \\ &= 1.35 \\ &> \sigma = 1 \end{aligned} \tag{5.27}$$

虽然剧场很远，价格有点贵，但是女友很满意，你还是很高兴。

神经元内部的参数学习

一次成功的音乐会之后，你似乎掌握了一个真理：其他什么都不重要，女友的喜好最重要，所以你又将决策模型的权重做出了调整：最简单的方式就是 $w_0 = w_1 = 0$ ，同时令 $w_2 > 0$ ，相当于只考虑 x_2 的影响而忽略其他因素，于是你得到了如图5.9所示的决策模型：

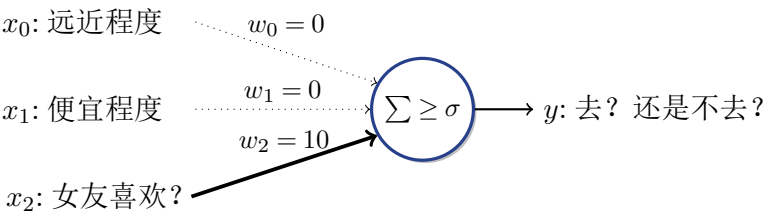


图 5.9: 预测是否去剧场的决策模型（只考虑女友喜好）

很快又有了一场音乐会，距你 1000 公里，票价 3000 元，当然女友是一直喜欢音乐会的。根据新的决策模型，你义无反顾地选择去看音乐会。然而，女友又不高

兴了，喜欢浪漫的女友觉得去看这场音乐会太奢侈了。在这几次看音乐会的经历中，你发现每个因素的权重需要准确地设置才能达到最好的决策效果。

那么如何确定最好的权重的？方法其实很简单，不断地尝试，根据结果不断地调整权重。在经过成百上千次的尝试之后，终于找到了一组合适的权重，使每次决策的正确率都很高。上面这个过程就类似于参数训练的过程，利用大量的数据来模拟成百上千次的尝试，根据输出的结果来不断地调整权重。

可以看到，在“是否参加音乐会”这个实际问题中，主要涉及到三方面的问题：

- 对问题建模，即定义输入 $\{x_i\}$ 的形式；
- 设计有效的决策模型，即定义 y ；
- 决定模型所涉及的参数（如权重 $\{w_i\}$ ）的最优值。

上面的例子对这三个问题都简要地做出了回答。下面的内容将继续对它们进行详细阐述。

5.2.3 多层神经网络

感知机是一种最简单的单层神经网络。一个非常自然的问题是：能否把多个这样的网络叠加在一起，获得建模更复杂问题的能力？如果可以，那么在多层神经网络的每一层，神经元之间是怎么组织、工作的呢？单层网络又是通过什么方式构造成多层的呢？

线性变换和激活函数

为了建立多层神经网络，首先需要把前面提到的简单的神经元进行扩展，把多个神经元组成一“层”神经元。比如，很多实际问题需要同时有多个输出，这时可以把多个相同的神经元并列起来，每个神经元都会有一个单独的输出，这就构成一“层”，形成了单层神经网络。单层神经网络中的每一个神经元都对应着一组权重和一个输出，可以把单层神经网络中的不同输出看作一个事物不同角度的描述。

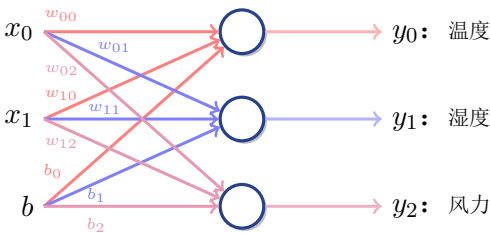


图 5.10: 权重矩阵中的元素与输出的对应关系

举个简单的例子，预报天气时，往往需要预测温度、湿度和风力，这就意味着如

果使用单层神经网络进行预测，需要设置 3 个神经元。如图5.10，权重矩阵为：

$$\mathbf{w} = \begin{pmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \end{pmatrix}$$

(5.28)

它的第一列元素 $\begin{pmatrix} w_{00} \\ w_{10} \end{pmatrix}$ 是输入相对第一个输出 y_0 的权重，参数向量 $\mathbf{b} = (b_0, b_1, b_2)$ 的第一个元素 b_0 是对应于第一个输出 y_0 的偏置量；类似的，可以得到 y_1 和 y_2 。预测天气的单层模型如图5.11所示（在本例中，假设输入 $\mathbf{x} = (x_0, x_1)$ ）。

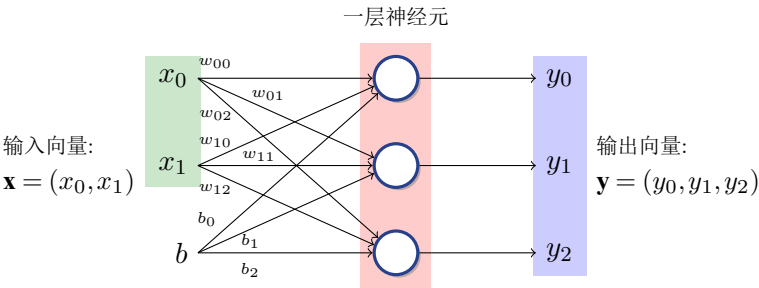


图 5.11: 预测天气的单层神经网络

在神经网络中，对于输入向量 $\mathbf{x} \in R^m$ ，一层神经网络首先将其经过线性变换映射到 R^n ，再经过激活函数变成 $\mathbf{y} \in R^n$ 。还是上面天气预测的例子，每个神经元获得相同的输入，权重矩阵 \mathbf{w} 是一个 2×3 矩阵，矩阵中每个元素 w_{ij} 代表第 j 个神经元中 x_i 对应的权重值，假设编号为 0 的神经元负责预测温度，则 w_{i0} 含义为预测温度时，输入 x_i 对其影响程度。此外所有神经元的偏置 b_0, b_1, b_2 组成了最终的偏置向量 \mathbf{b} 。在该例中则有，权重矩阵 $\mathbf{w} = \begin{pmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \end{pmatrix}$ ，偏置向量 $\mathbf{b} = (b_0, b_1, b_2)$ 。

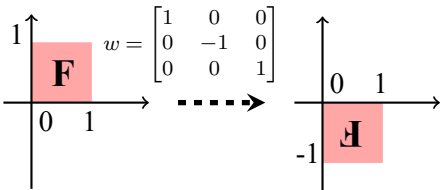


图 5.12: \mathbf{w} 对 \mathbf{x} 的旋转作用

那么，线性变换的本质是什么？

- 从代数角度看，对于线性空间 V ，任意 $a, b \in V$ 和数域中的任意 α ，线性变换 $T(\cdot)$ 需满足： $T(a + b) = T(a) + T(b)$ ，且 $T(\alpha a) = \alpha T(a)$ ；
- 从几何角度上看，公式中的 $\mathbf{x} \cdot \mathbf{w} + \mathbf{b}$ 将 \mathbf{x} 右乘 \mathbf{w} 相当于对 \mathbf{x} 进行旋转变换，如

图5.12所示，对三个点 (0,0)，(0,1)，(1,0) 及其围成的矩形区域右乘如下矩阵：

$$\mathbf{w} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.29}$$

这样，矩形区域由第一象限旋转 90 度到了第四象限。

公式 $\mathbf{x} \cdot \mathbf{w} + \mathbf{b}$ 中的公式中的 \mathbf{b} 相当于对其进行平移变换。其过程如图5.13所示，偏置矩阵 $\mathbf{b} = \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ 将矩形区域沿 x 轴向右平移了一段距离。

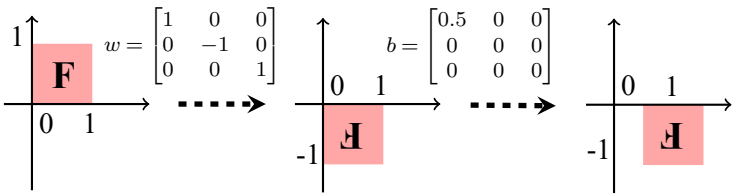


图 5.13: 线性变换示意图

也就是说，线性变换提供了对输入数据进行空间中旋转、平移的能力。当然，线性变换也适用于更加复杂的情况，这也为神经网络提供了拟合不同函数的能力。比如，可以利用线性变换将三维图形投影到二维平面上，或者将二维平面上的图形映射到三维空间。如图5.14，通过一个简单的线性变换，可以将三维图形投影到二维平面上。

$$\underbrace{\left\{ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \dots \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right\}}_5 \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \underbrace{\left\{ \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \dots \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\}}_5$$

图 5.14: 线性变换 3 维 → 2 维数学示意

那激活函数又是什么？神经元在接收到经过线性变换的结果后，通过激活函数的处理，得到最终的输出 \mathbf{y} 。激活函数的目的是实际问题中的非线性变换，线性变换只能拟合直线，而激活函数的加入，使神经网络具有了拟合曲线的能力。特别是在实际问题中，很多现象都无法用简单的线性关系描述，这时激活函数的非线性就为描述更加复杂的问题提供了工具。常见的非线性函数有 Sigmoid、ReLU、Tanh 等。如图5.15列举了几种激活函数的形式。

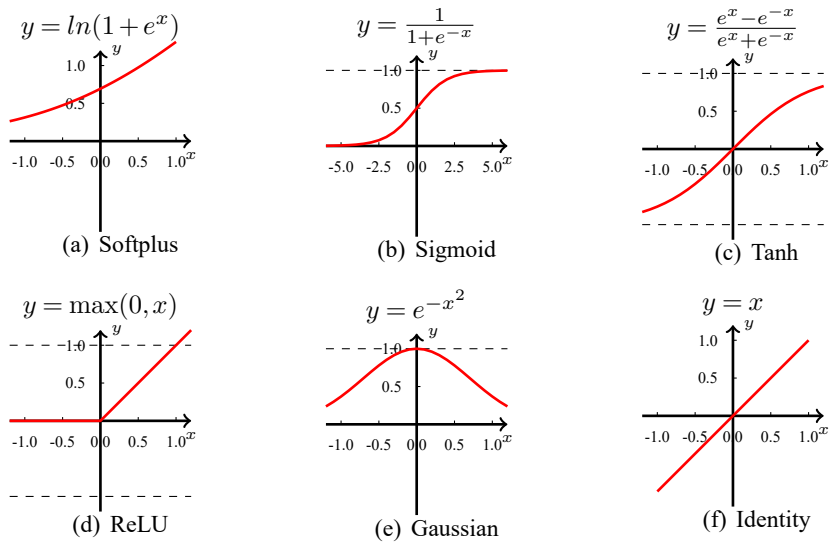


图 5.15: 几种常见的激活函数

单层神经网络 → 多层神经网络

单层神经网络由线性变换和激活函数两部分构成，但在实际问题中，单层网络并不能很好地拟合复杂函数。因此很自然地想到将单层网络扩展到多层神经网络，即深层神经网络。将一层神经网络的最终输出向量作为另一层神经网络的输入向量，通过这种方式可以将多个单层神经网络连接在一起。

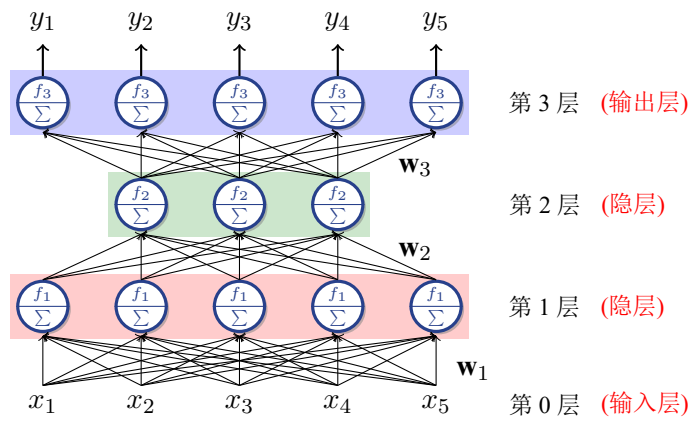


图 5.16: 具有四层神经元的三层神经网络

在多层神经网络中，通常包括输入层、输出层和至少一个隐藏层。如图5.16是一个由四层神经网络构成的模型，包括输入层、输出层和两个隐藏层。

5.2.4 函数拟合能力

神经网络方法之所以受到青睐一方面是由于它提供了端到端学习的模式，另一方面是由于它强大的函数拟合能力。理论上说，神经网络可以拟合任何形状的函数。下面就来看一下为什么神经网络会有这样的能力。

众所周知，单层神经网络无法解决线性不可分问题，比如经典的异或问题。但是具有一个隐藏层的两层神经网络在理论上就可以拟合所有的函数了。接下来我们分析一下为什么仅仅是多了一层，神经网络就能变得如此强大。在此之前，需要明确的一点是，“拟合”是把平面上一系列的点，用一条光滑的曲线连接起来，并用函数来表示这条拟合的曲线。在用神经网络解决问题时，可以通过拟合训练数据中的“数据点”来获得输入与输出之间的函数关系，并利用其对未知数据做出判断。可以假设输入与输出之间存在一种函数关系，而神经网络的“拟合”能力是要尽可能地逼近原函数输出值，与原函数输出值越逼近，则意味着拟合得越优秀。

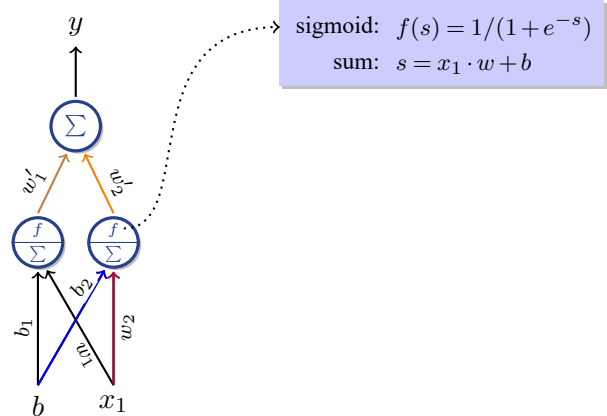


图 5.17: 以 Sigmoid 作为隐藏层激活函数的两层神经网络

如图5.17是一个以 Sigmoid 作为隐藏层激活函数的两层神经网络。通过调整参数 $\mathbf{w} = (w_1, w_2)$, $\mathbf{b} = (b_1, b_2)$ 和 $\mathbf{w}' = (w_1', w_2')$ 的值，可以不断地改变目标函数的形状。

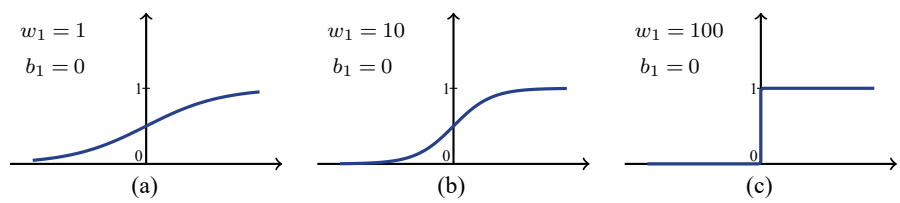


图 5.18: 通过改变权重 w_1 改变目标函数平滑程度

设置 $w_1' = 1$, $w_1 = 1$, $b_1 = 0$, 其他参数设置为 0。可以得到如图5.18(a) 所示的目标函数，此时目标函数还是比较平缓的。通过调大 w_1 ，可以将图5.18(a) 中函数的坡度调得更陡：当 $w_1 = 10$ 时，如图5.18(b) 所示，目标函数的坡度与图5.18(a) 相比变得更陡了；当 $w_1 = 100$ 时，如图5.18(c) 所示，目标函数的坡度变得更陡、更尖锐，

已经逼近一个阶梯函数。

设置 $w'_1 = 1$, $w_1 = 100$, $b_1 = 0$, 其他参数设置为 0。可以得到如图5.19(a) 所示的目标函数, 此时目标函数是一个阶梯函数, 其“阶梯”恰好与 y 轴重合。通过改变 b_1 , 可以将整个函数沿 x 轴向左右平移: 当 $b_1 = -2$ 时, 如图5.19(b) 所示, 与图5.19(a) 相比目标函数的形状没有发生改变, 但其位置沿 x 轴向右平移; 当 $b_1 = -4$ 时, 如图5.19(c) 所示, 目标函数的位置继续沿 x 轴向右平移。

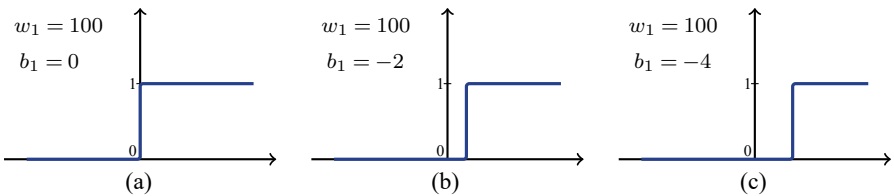


图 5.19: 通过改变偏置量 b_1 改变目标函数位置

设置 $w'_1 = 1$, $w_1 = 100$, $b_1 = -4$, 其他参数设置为 0。可以得到如图5.20(a) 所示的目标函数, 此时目标函数是一个阶梯函数, 该阶梯函数取得最大值的分段处为 $y = 1$ 。通过改变 w'_1 , 可以将目标函数“拉高”或是“压扁”。如图5.20(b) 和 (c) 所示, 目标函数变得“扁”了。最终, 该阶梯函数取得最大值的分段处约为 $y = 0.7$ 。

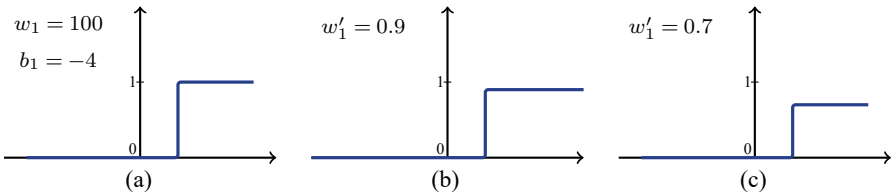


图 5.20: 通过改变权重 w'_1 将目标函数“拉高”或“压扁”

设置 $w'_1 = 0.7$, $w_1 = 100$, $b_1 = -4$, 其他参数设置为 0。可以得到如图5.21(a) 所示的目标函数, 此时目标函数是一个阶梯函数。若是将其他参数设置为 $w'_2 = 0.7$, $w_2 = 100$, $b_2 = 16$, 由图5.21(b) 可以看出, 原来目标函数的“阶梯”由一级变成了两级, 由此可以推测, 将第二组参数进行设置, 可以使目标函数分段数增多; 若将第二组参数中的 w'_2 由原来的 0.7 设置为 -0.7 , 可得到如图5.21(c) 所示的目标函数, 与图5.21(b) 相比, 原目标函数的“第二级阶梯”向下翻转, 由此可见 w' 的符号决定了目标函数的翻转方向。

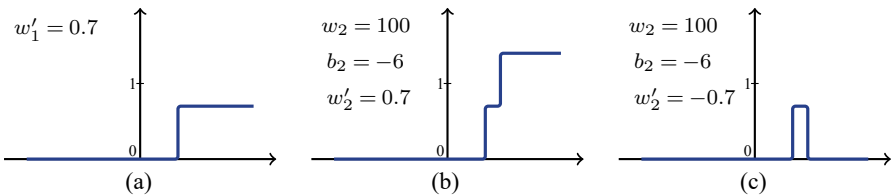


图 5.21: 通过设置第二组参数将目标函数分段数增加

由上面的内容，已经看到通过设置神经元中的参数将目标函数的形状做各种变换，但是看起来目标函数的类型还是比较单一的。而在实际问题中，输入与输出之间的函数关系甚至复杂到无法人为构造或是书写，神经网络又是如何拟合这种复杂的函数关系的呢？

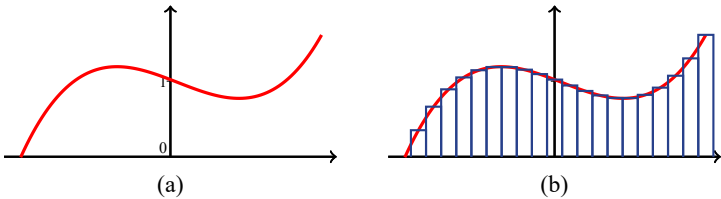


图 5.22: 将目标函数作分段处理

以如图5.22(a)所示的目标函数为例，为了拟合该函数，可以将其看成分成无数小段的分段函数，如图5.22(b)所示。

如图5.23(a)所示，上例中两层神经网络的函数便可以拟合出目标函数的一小段。为了使两层神经网络可以拟合出目标函数更多的一小段，需要增加隐层神经元的个数。如图5.23(b)，将原本的两层神经网络神经元个数增多一倍，由2个神经元扩展到4个神经元，其函数的分段数也增加一倍，而此时的函数恰好可以拟合目标函数中的两个小段。以此类推，理论上，该两层神经网络便可以通过不断地增加隐层神经元数量去拟合任意函数。

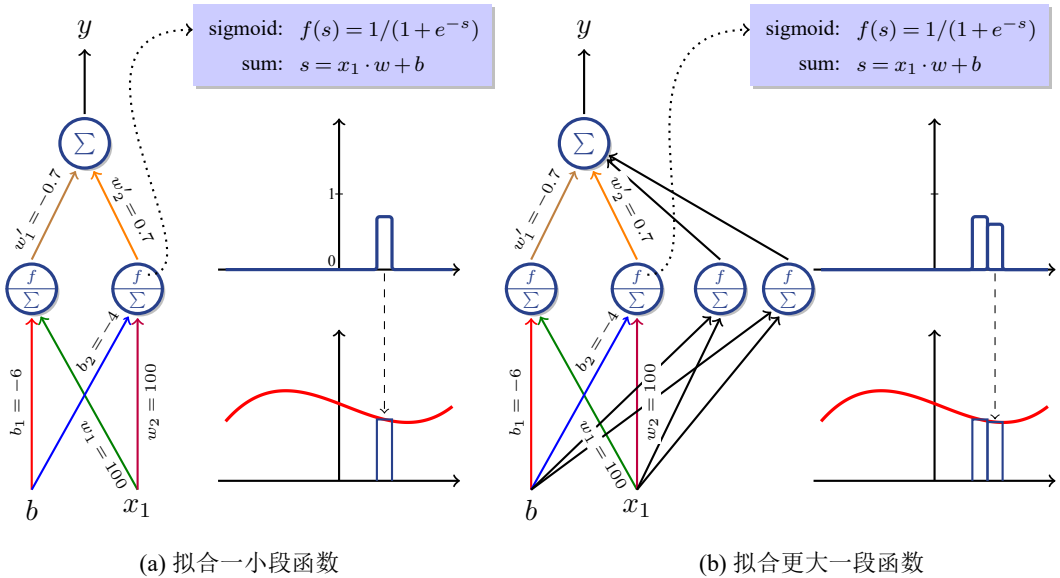


图 5.23: 扩展隐层神经元个数去拟合目标函数更多的“一小段”

两层神经元的神经网络在理论上可以拟合所有函数了，但是在实际问题中所使用的神经网络都远远超过了两层，这也是对深度学习这个概念中“深度”的一种体现。

主要是有以下几方面的原因：

- 使用较浅的神经网络去拟合一个比较复杂的函数关系，需要数量极其庞大的神经元和参数，训练难度大。在上面的例子中可以看出，两层神经元仅仅拟合目标函数的两小段，其隐层就需要 4 个神经元。从另一个角度说，加深网络也可能达到与宽网络（更多神经元）类似的效果。
- 更多层的网络可以提供更多的线性变换和激活函数，对输入的抽象程度更好，因而可以更好的表示数据的特征。

在本书后面的内容中还会看到，深层网络在机器翻译中可以带来明显的性能提升。

5.3 神经网络的张量实现

在神经网络内部，输入经过若干次变换，最终得到输出的结果。这个过程类似于一种逐层的数据“流动”。不禁会产生这样的疑问：在神经网络中，数据是以哪种形式“流动”的？如何去编程实现这种数据“流动”呢？

为了解决上面的问题，本节将介绍人工神经网络更加通用的描述形式——张量计算。随后也会看到，使用基于张量的数学工具，可以方便的搭建神经网络。

5.3.1 张量及其计算

张量

对于神经网络中的某层神经元 $y = f(x \cdot w + b)$ ，其中 w 是权重矩阵，例如 $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ ， b 是偏置向量，例如 $(1,3)$ 。在这里，输入 x 和输出 y ，可以不是简单的向量或是矩阵形式，而是深度学习中更加通用的数学量——张量（Tensor），比如下式中的几种情况都可以看作是深度学习中定义数据的张量：

$$x = (-1 \ 3) \quad x = \begin{pmatrix} -1 & 3 \\ 0.2 & 2 \end{pmatrix} \quad x = \begin{pmatrix} \begin{pmatrix} -1 & 3 \\ 0.2 & 2 \end{pmatrix} \\ -1 & 3 \\ 0.2 & 2 \end{pmatrix}$$

简单来说，张量是一种通用的工具，用于描述由多个数据构成的量。比如，输入的量有三个维度在变化，用矩阵不容易描述，但是用张量却很容易。

从计算机实现的角度来看，现在所有深度学习框架都把张量定义为“多维数组”。张量有一个非常重要的属性——阶（Rank）。可以将多维数组中“维”的属性与张量的“阶”的属性作类比，这两个属性都表示多维数组（张量）有多少个独立的方向。例如，3 是一个标量（Scalar），相当于一个 0 维数组或 0 阶张量； $(2 \ -3 \ 0.8 \ 0.2)^T$

是一个向量 (Vector)，相当于一个 1 维数组或 1 阶张量； $\begin{pmatrix} -1 & 3 & 7 \\ 0.2 & 2 & 9 \end{pmatrix}$ 是一个矩阵 (Matrix)，相当于一个 2 维数组或 2 阶张量；如图5.24，这是一个 3 维数组或 3 阶张量，其中，每个 4×4 的方形代表一个 2 阶张量，这样的方形有 4 个，最终形成 3 阶张量。

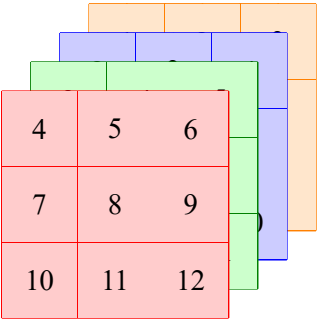


图 5.24: 3 阶张量示例 ($4 \times 4 \times 4$)

虽然这里所使用的张量是出于编程实现的视角，但是数学中张量有严格的定义。从数学上看“张量并不是向量和矩阵的简单扩展，多维数组也并不是张量所必须的表达形式”。从某种意义上说，矩阵才是张量的扩展。当然，这个逻辑可能和大家在深度学习中的认知是不一致的。现在就一起看一下，张量究竟为何物。

张量的严格定义是利用线性映射来描述的。与矢量相类似，定义由若干坐标系改变时满足一定坐标转化关系的有序数组成的集合为张量。从几何角度讲，它是一个真正的几何量，也就是说，它是不随参照系的坐标变换而变化的，是若干向量和协向量通过张量乘法定义的量。

不过，更广泛接受的定义是：张量是多重线性函数，是定义在一些向量空间和笛卡尔积上的多重线性映射。张量的多重线性表现在，对于每一个输入变量都是线性的。比如，张量 $\mathbf{T}(v_0, v_1, \dots, v_r)$ ，其输入是 r 个向量 $\{v_0, v_1, \dots, v_r\}$ ，对于张量 \mathbf{T} 的任意一个 v_i ，都有：

$$\mathbf{T}(v_0, \dots, v_i + u, \dots, v_r) = \mathbf{T}(v_0, \dots, v_i, \dots, v_r) + \mathbf{T}(v_0, \dots, u, \dots, v_r) \tag{5.30}$$

$$\mathbf{T}(v_0, \dots, c \cdot v_i, \dots, v_r) = c \cdot \mathbf{T}(v_0, \dots, v_i, \dots, v_r) \tag{5.31}$$

其中， u 为与 v_i 形状相同的向量， c 为任意实数。这个性质非常重要，根据这个性质可以推导出张量的其他定义。

从我们的物理世界看，如果一个物理量在物体的某个位置上只是一个单值，那么它是一个标量，例如密度；如果一个物理量在同一个位置、从多个方向上看，有不同的值，那么这个物理量就是一个张量，比如物理学中常用的应力的描述就是一个典型的张量。举一个简单的例子： $\mathbf{T}(\mathbf{v}, \mathbf{u})$ 是一个三维空间 (x, y, z) 上的 2 阶张量，其中 \mathbf{v} 和 \mathbf{u} 是两个向量，如图5.25所示，向量 \mathbf{v} 在某个两两垂直的三维坐标

系中可以表示为 $(a \ b \ c)^T$ ，同理向量 \mathbf{u} 在某个两两垂直的三维坐标系中可以表示为 $(a' \ b' \ c')^T$ 。但在三维空间 (x,y,z) 中，向量 \mathbf{v} 和向量 \mathbf{u} 分别被表示为 $(v_x \ v_y \ v_z)^T$ 和 $(u_x \ u_y \ u_z)^T$ 。

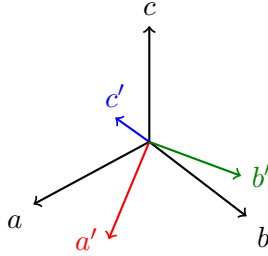


图 5.25: 向量 \mathbf{v} 和向量 \mathbf{u} 在不同坐标系中的值

2 阶张量 $\mathbf{T}(\mathbf{v}, \mathbf{u})$ 可表示为:

$$\mathbf{T}(\mathbf{v}, \mathbf{u}) = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}^T \begin{pmatrix} T_{xx} & T_{xy} & T_{xz} \\ T_{yx} & T_{yy} & T_{yz} \\ T_{zx} & T_{zy} & T_{zz} \end{pmatrix} \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} \quad (5.32)$$

其中， $\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$ 是向量 \mathbf{v} 在基向量 (x,y,z) 上的投影， $\begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}$ 是向量 \mathbf{u} 在基向量 (x,y,z) 上的投影， $\begin{pmatrix} T_{xx} & T_{xy} & T_{xz} \\ T_{yx} & T_{yy} & T_{yz} \\ T_{zx} & T_{zy} & T_{zz} \end{pmatrix}$ 是张量 \mathbf{T} 在 3×3 个方向上的分量，恰巧用“矩阵”表示，记为 $[\mathbf{T}]$ 。

以上的内容是帮助大家明确张量的原始定义，以避免对这个概念的误解。但是，本书仍然遵循深度学习中常用的概念，把张量理解为多维数组。在保证数学表达的简洁性的同时，使程序实现接口更加统一。

张量的矩阵乘法

对于一个单层神经网络， $\mathbf{y} = f(\mathbf{x} \cdot \mathbf{w} + \mathbf{b})$ 中的 $\mathbf{x} \cdot \mathbf{w}$ 表示对输入 \mathbf{x} 进行线性变换，其中 \mathbf{x} 是输入张量， \mathbf{w} 是权重矩阵。 $\mathbf{x} \cdot \mathbf{w}$ 表示的是矩阵乘法，需要注意的是这里是矩阵乘法而不是张量乘法。

张量乘以矩阵是怎样计算呢？可以先回忆一下5.2.1节的线性代数的知识。假设 \mathbf{a} 为 $m \times p$ 的矩阵， \mathbf{b} 为 $p \times n$ 的矩阵，对 \mathbf{a} 和 \mathbf{b} 作矩阵乘积的结果是一个 $m \times n$ 的

矩阵 \mathbf{c} ，其中矩阵 \mathbf{c} 中第 i 行、第 j 列的元素可以表示为：

$$(\mathbf{ab})_{ij} = \sum_{k=1}^p a_{ik}b_{kj} \tag{5.33}$$

例如 $\mathbf{a} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$ ， $\mathbf{b} = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$ ，则

$$\begin{aligned} \mathbf{c} &= \mathbf{ab} \\ &= \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{pmatrix} \end{aligned} \tag{5.34}$$

将矩阵乘法扩展到高阶张量中：一个张量 \mathbf{x} 若要与矩阵 \mathbf{w} 做矩阵乘法，则 \mathbf{x} 的最后一维度需要与 \mathbf{w} 的行数大小相等，即：若张量 \mathbf{x} 的形状为 $\cdot \times n$ ， \mathbf{w} 须为 $n \times \cdot$ 的矩阵。如下是一个例子：

$$\mathbf{x}(1:4, 1:4, \mathbf{1:4}) \times \mathbf{w}(\mathbf{1:4}, 1:2) = \mathbf{s}(1:4, 1:4, 1:2) \tag{5.35}$$

其中，张量 \mathbf{x} 沿第 1 阶所在的方向与矩阵 \mathbf{w} 进行矩阵运算（张量 \mathbf{x} 第 1 阶的每个维度都可以看做一个 4×4 的矩阵）。图5.26演示了这个计算过程。张量 \mathbf{x} 中编号为①的子张量（可看作矩阵）与矩阵 \mathbf{w} 进行矩阵乘法，其结果对应张量 \mathbf{s} 中编号为①的子张量。这个过程会循环四次，因为有四个这样的矩阵（子张量）。最终，图5.26给出了结果张量的形式（ $4 \times 4 \times 2$ ）。

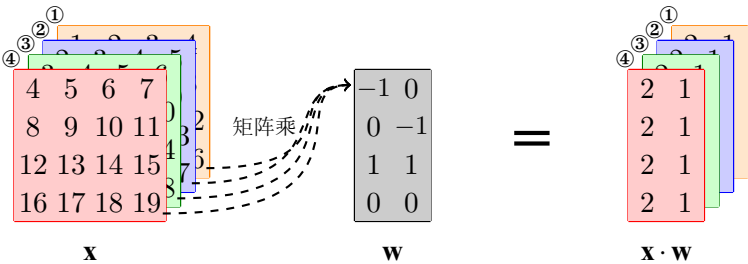


图 5.26: 张量与矩阵的矩阵乘法

张量的单元操作

对于神经网络中的某层神经元 $\mathbf{y} = f(\mathbf{x} \cdot \mathbf{w} + \mathbf{b})$ ，也包含有其他张量单元操作：1) 加法： $\mathbf{s} + \mathbf{b}$ ，其中张量 $\mathbf{s} = \mathbf{x} \cdot \mathbf{w}$ ；2) 激活函数： $f(\cdot)$ 。具体来说：

- $\mathbf{s} + \mathbf{b}$ 中的单元加就是对张量中的每个位置都进行加法。在上例中 \mathbf{s} 是形状为

(1:4,1:4,1:2) 的 3 阶张量，而 **b** 是含有 4 个元素的向量，在形状不同的情况下是怎样进行单元加的呢？在这里需要引入**广播机制**：如果两个数组的后缘维度（即从末尾开始算起的维度）的轴长度相符或其中一方的长度为 1，则认为它们是广播兼容的。广播会在缺失或长度为 1 的维度上进行，它是深度学习框架中常用的计算方式。来看一个具体的例子，如图5.27所示，**s** 是一个 2×4 的矩阵而 **b** 是一个长度为 4 的向量，这两者进行单元加运算时，广播机制会将 **b** 沿第一个维度复制后，再与 **s** 做加法运算。

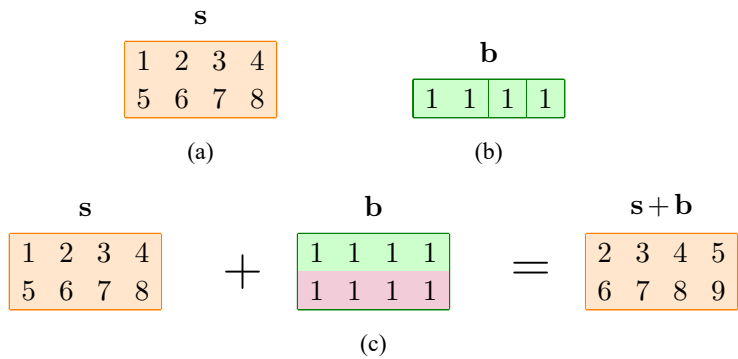


图 5.27: 广播机制

- 除了单位加之外，张量之间也可以使用减法操作、乘法操作。此外也可以对张量作激活操作，这里将其称作为函数的**向量化**（Vectorization）。例如，对向量（1 阶张量）作 ReLU 激活，ReLU 激活函数的公式为：

$$f(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases} \tag{5.36}$$

例如 $\text{ReLU} \left(\begin{pmatrix} 2 \\ -3 \end{pmatrix} \right) = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$ 。

5.3.2 张量的物理存储形式

在深度学习世界中，张量就是多维数组，无论是向量还是矩阵都可以看作是数学上“张量”的扩展。因此，张量的物理存储方式也与多维数组相同：

- 张量 **T**(1:3) 表示一个含有三个元素的向量（1 阶张量），其物理存储如图5.28(a)所示。
- 张量 **T**(1:2,1:3) 表示一个 2×3 的矩阵（2 阶张量），其物理存储如图5.28(b)所示。
- 张量 **T**(1:2,1:2,1:3) 表示一个大小 2×2×3 的 3 阶张量，其物理存储如

图5.28(c) 所示。

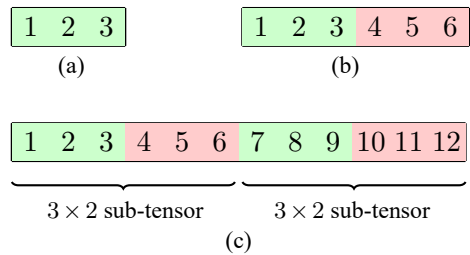


图 5.28: 1 阶 (a)、2 阶 (b)、3 阶张量 (c) 的物理存储

高阶张量的物理存储方式与多维数组在 C++、Python 中的物理存储方式相同。

5.3.3 使用开源框架实现张量计算

实现神经网络的开源系统有很多，比如，一个简单好用的 Python 工具包 —— Numpy (<https://numpy.org/>)。Numpy 提供了张量表示和使用的范式，可以很方便地定义、使用多维数组。

此外，如今深度学习框架已经非常成熟。比如，Tensorflow 和 Pytorch 就是非常受欢迎的深度学习工具包，除此之外还有很多其他优秀的框架：CNTK、MXNet、PaddlePaddle、Keras、Chainer、dl4j、NiuTensor 等。开发者可以根据自身的喜好和开发项目的要求选择所采用的框架。

本节将使用 NiuTensor 来描述张量计算。NiuTensor 是一个面向自然语言处理任务的张量库，它支持丰富的张量计算接口。此外，该 NiuTensor 内核基于 C++ 语言编写，代码高度优化。该工具包获取网址为<https://developer.niutrans.com/ArticleContent/technicaldoc/doc1/1.NiuTensor>。

NiuTensor 的使用非常简单，如图5.29是一个使用 NiuTensor 声明、定义张量的 C++ 代码：

```
#include "source/tensor/XTensor.h"           // 引用 XTensor 定义的头文件
using namespace nts;                          // 引用 nts 命名空间

int main(int argc, const char ** argv){
    XTensor tensor;                           // 声明张量 tensor
    InitTensor2D(&tensor, 2, 2, X_FLOAT);      // 定义张量为 2*2 的矩阵
    tensor.SetDataRand();                     // [0,1] 均匀分布初始化张量
    tensor.Dump(stdout);                      // 输出张量内容
    return 0;
}
```

图 5.29: 使用 NiuTensor 声明定义张量

这段程序定义了一个形状为 2×2 (`dimsizes = 2, 2`)，数据类型是单精度浮点 (`dtype = X_FLOAT`)，非稀疏 (`dense = 1.00`) 的 2 阶 (`order = 2`) 张量。运行这个

程序会显示张量每个元素的值，如图5.30所示。

```
order=2 dimsize=2,2 dtype=X_FLOAT dense=1.000000
3.605762e-001 2.992340e-001 1.393780e-001 7.301248e-001
```

图 5.30: 程序运行结果

在 NiuTensor 中，张量由类 XTensor 表示，利用 InitTensor 定义，主要有四个参数：

- 指向 XTensor 类型变量的指针，如 &tensor。
- 张量的阶，如 6。
- 各个方向维度的大小，约定该参数形式与传统的多维数组形式相同，如 {2,3,4, 2,3,4}。
- 张量的数据类型，该参数有缺省值。

定义 XTensor 的程序示例如图5.31(a) 所示：

```
XTensor tensor; // 声明张量 tensor
int sizes[6] = {2,3,4,2,3,4}; // 张量的形状为 2*3*4*2*3*4
InitTensor(&tensor, 6, sizes, X_FLOAT); // 定义形状为 sizes 的 6 阶张量
```

(a) NiuTensor 定义张量程序

```
XTensor a, b, c; // 声明张量 tensor
InitTensor1D(&a, 10, X_INT); // 10 维的整数型向量
InitTensor1D(&b, 10); // 10 维的向量，缺省类型 (浮点)
InitTensor4D(&c, 10, 20, 30, 40); // 10*20*30*40 的 4 阶张量 (浮点)
```

(b) 定义张量的简便方式程序

```
XTensor tensorGPU; // 声明张量 tensor
InitTensor2D(&tensorGPU, 10, 20, \ // 在编号为 0 的 GPU 上定义张量
            X_FLOAT, 0);
```

(c) 在 GPU 上定义张量程序

图 5.31: NiuTensor 定义张量程序示例

除此之外，NiuTensor 还提供更简便的张量定义方式，如图5.31(b) 所示。也可以在 GPU 上定义张量，如图5.31(c) 所示。NiuTensor 支持张量的各种代数运算，各种单元算子，如 +、-、*、/、Log（取对数）、Exp（指数运算）、Power（幂方运算）、Absolute（绝对值）等，还有 Sigmoid、Softmax 等激活函数。图5.32(a) 是一段对张量进行 1 阶运算的程序示例。

除了上述单元算子外，NiuTensor 还支持张量之间的高阶运算，其中最常用的是矩阵乘法，图5.32(b) 是张量之间进行矩阵乘法的程序示例。表5.2展示了一些 NiuTensor 支持的其他函数操作，除此还有很多其他操作无法在此一一列举，有兴趣可以参考网站上的详细说明。

```
XTensor a, b, c, d, e;           // 声明张量 tensor
InitTensor3D(&a, 2, 3, 4);       // a 为 2*3*4 的 3 阶张量
InitTensor3D(&b, 2, 3, 4);       // b 为 2*3*4 的 3 阶张量
InitTensor3D(&c, 2, 3, 4);       // c 为 2*3*4 的 3 阶张量
a.SetDataRand();                 // 随机初始化 a
b.SetDataRand();                 // 随机初始化 b
c.SetDataRand();                 // 随机初始化 c
d = a + b * c;                   // d 被赋值为 a + b * c
d = ((a + b) * d - b / c) * d;   // d 可以被嵌套使用
e = Sigmoid(d);                  // d 经过激活函数 Sigmoid 赋值给 e
```

(a) 张量进行 1 阶运算

```
XTensor a, b, c;                 // 声明张量 tensor
InitTensor4D(&a, 2, 2, 3, 4);    // a 为 2*2*3*4 的 4 阶张量
InitTensor2D(&b, 4, 5);          // b 为 4*5 的矩阵
a.SetDataRand();                 // 随机初始化 a
b.SetDataRand();                 // 随机初始化 b
c = MMul(a, b);                  // 矩阵乘的结果为 2*2*3*5 的 4 阶张量
```

(b) 张量之间的矩阵乘法

图 5.32: 张量代数运算程序示例

表 5.2: NiuTensor 支持的部分函数

函数	描述
a.Reshape(o,s)	把 a 变换成阶为 o、形状为 s 的张量
a.Get(pos)	取张量中位置为 pos 的元素
a.Set(v,pos)	把张量中位置为 pos 的元素值设为 v
a.Dump(file)	把张量存到 file 中，file 为文件句柄
a.Read(file)	从 file 中读取张量，file 为文件句柄
Power(a,p)	计算指数 a^p
Linear(a,s,b)	计算 $a*s + b$ ，s 和 b 都是一个实数
CopyValue(a)	构建 a 的一个拷贝
ReduceMax(a,d)	对 a 沿着方向 d 进行规约，得到最大值
ReduceSum(a,d)	对 a 沿着方向 d 进行规约，得到和
Concatenate(a,b,d)	把两个张量 a 和 b 沿 d 方向级联
Merge(a,d)	对张量 a 沿 d 方向合并
Split(a,d,n)	对张量 a 沿 d 方向分裂成 n 份
Sigmoid(a)	对 a 进行 Sigmoid 变换
Softmax(a)	对 a 进行 Softmax 变换，沿最后一个方向
HardTanh(a)	对 a 进行 hard Tanh 变换（双曲正切的近似）
Relu(a)	对 a 进行 ReLU 变换

随后的内容会使用 NiuTensor 作为一种张量“语言”来完成对神经网络的描述，以便于读者理解一个抽象的神经网络是如何和具体的程序对应起来的。当然，神经网络也可以使用 TensorFlow 和 PyTorch 等框架进行定义，方法都是非常相似的。

5.3.4 前向传播与计算图

有了张量这个工具，可以很容易地实现任意的神经网络。反过来，神经网络都可以被看作是张量的函数。一种经典的神经网络计算模型是：给定输入张量，通过各个神经网络层所对应的张量计算之后，最后得到输出张量。这个过程也被称作**前向传播**，它常常被应用在使用神经网络对新的样本进行推断中。

来看一个具体的例子，如图5.33(a) 是一个根据天气情况判断穿衣指数（穿衣指数是人们穿衣薄厚的依据）的过程，将当天的天空状况、低空气温、水平气压作为输入，通过一层神经元在输入数据中提取温度、风速两方面的特征，并根据这两方面的特征判断穿衣指数。需要注意的是，在实际的神经网络中，并不能准确地知道神经元究竟可以提取到哪方面的特征，以上表述是为了让读者更好地理解神经网络的建模过程和前向传播过程。这里将上述过程建模为如图5.33(b) 所示的两层神经网络。

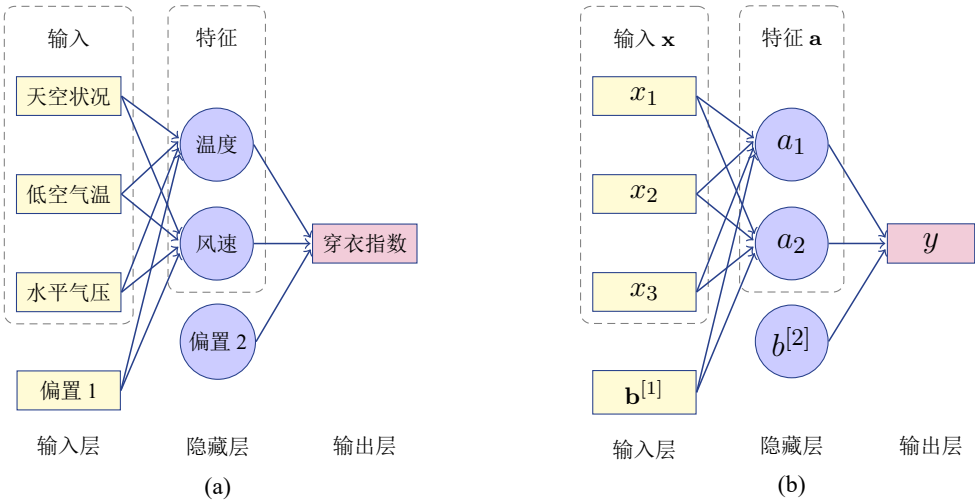


图 5.33: 判断穿衣指数问题的神经网络过程

它可以被描述为公式5.37，其中隐藏层的激活函数是 Tanh 函数，输出层的激活函数是 Sigmoid 函数， $\mathbf{w}^{[1]}$ 和 $\mathbf{b}^{[1]}$ 分别表示第一层的权重矩阵和偏置， $\mathbf{w}^{[2]}$ 和 $b^{[2]}$ 分别表示第二层的权重矩阵和偏置且偏置 $b^{[2]}$ 是标量：

$$y = \text{Sigmoid}(\text{Tanh}(\mathbf{x} \cdot \mathbf{w}^{[1]} + \mathbf{b}^{[1]}) \cdot \mathbf{w}^{[2]} + b^{[2]}) \tag{5.37}$$

前向计算实现如图5.34所示，图中对各张量和其他参数的形状做了详细说明。输入 $\mathbf{x} = (x_1, x_2, x_3)$ 是一个 1×3 的张量，其三个维度分别对应天空状况、低空气温、水平气压三个方面的数据。输入数据经过隐藏层的线性变换 $\mathbf{x} \cdot \mathbf{w}^{[1]} + \mathbf{b}^{[1]}$ 和 Tanh 函

数的激活，得到新的张量 $\mathbf{a} = (a_1, a_2)$ ，其中 a_1, a_2 分别对应着从输入数据中提取出的温度和风速两方面特征；神经网络在获取到天气情况的特征 \mathbf{a} 后，继续对其进行线性变换 $\mathbf{a} \cdot \mathbf{w}^{[2]} + b^{[2]}$ 和 Sigmoid 函数的激活操作，得到神经网络的最终输出 y ，即神经网络此时预测的穿衣指数。

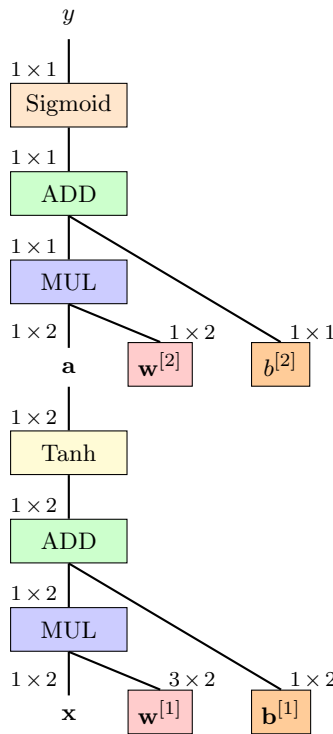


图 5.34: 前向计算示例（计算图）

图5.34实际上是神经网络的一种**计算图**（Computation Graph）表示。现在很多深度学习框架都是把神经网络转化为计算图，这样可以把复杂的运算分解为简单的运算。通过对计算图中节点的遍历，可以方便地完成神经网络的计算。比如，可以对图中节点进行拓扑排序（由输入到输出），之后依次访问每个节点，同时完成相应的计算，这也就实现了一个前向计算的过程。构建计算图的方式有很多，比如，动态图、静态图等。在5.4.2节会进一步对计算图在模型参数训练中的应用进行介绍。

5.3.5 神经网络实例

下面用几个实例来说明搭建神经网络的过程。搭建神经网络的过程本质上就是定义前向计算的过程。首先构造一个单层神经网络。如图5.35(a) 所示，简单的定义输入、权重和偏置后，定义激活函数为 Sigmoid 函数，输入 \mathbf{x} 经过线性变换和激活函数，得到输出 \mathbf{y} 。

图5.35(b) 是一个构造三层神经网络的程序示例。在第一层中， \mathbf{x} 作为输入， $\mathbf{h1}$ 作为输出，其中 $\mathbf{h1} = \text{Sigmoid}(\mathbf{x} \cdot \mathbf{w1} + \mathbf{b1})$ 。在第二层中， $\mathbf{h1}$ 作为输入， $\mathbf{h2}$ 作为输出，其中

$h2 = \text{Tanh}(h1 \cdot w2)$ 。在第三层中， $h2$ 作为输入， y 作为输出，其中 $y = \text{ReLU}(h2 \cdot w3)$ 。 y 也会作为整个神经网络的输出。

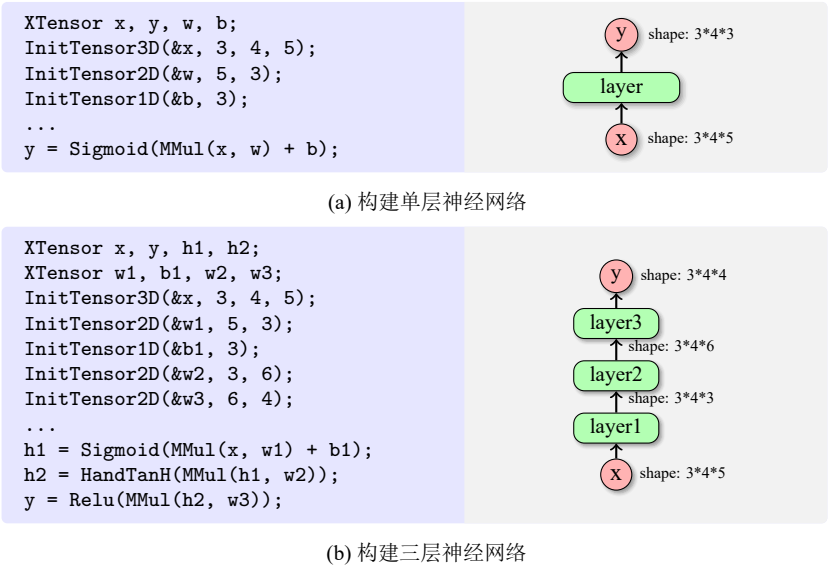


图 5.35: 构建神经网络的程序示例

除了简单对若干全连接网络进行堆叠，也可以用张量定义更加复杂的神经网络结构，如循环神经网络和 Transformer 等结构（见5.5节）。如图5.36所示，输入 x_1 、 x_2 、 x_3 经过一个循环神经网络送给更上层的网络进行处理。关于循环神经网络及其它神经网络结构在本章的后续内容还会有进一步介绍。

5.4 神经网络的参数训练

简单来说，神经网络可以被看作是由变量和函数组成的表达式，例如： $y = x + b$ 、 $y = \text{ReLU}(x \cdot w + b)$ 、 $y = \text{Sigmoid}(\text{ReLU}(x \cdot w^{[1]} + b^{[1]}) \cdot w^{[2]} + b^{[2]})$ 等等，其中的 x 和 y 作为输入和输出变量， w 、 b 等其他变量作为**模型参数**（Model Parameters）。确定了函数表达式和模型参数，也就确定了神经网络模型。通常，表达式的形式需要系统开发者设计，而模型参数的数量有时会非常巨大，因此需要自动学习，这个过程也被称为**模型学习**或**训练**（Training）。为了实现这个目标，通常会准备一定量的带有标准答案的数据，称之为**有标注数据**（Annotated Data/Labeled Data）。这些数据会用于对模型参数的学习，这也对应了统计模型中的参数估计过程。在机器学习中，一般把这种使用有标注数据进行统计模型参数训练的过程称为**有指导的训练**或**有监督的训练**（Supervised Training）。在本章中，如果没有特殊说明，模型训练都是指有监督的训练。那么神经网络内部是怎样利用有标注数据对参数进行训练的呢？

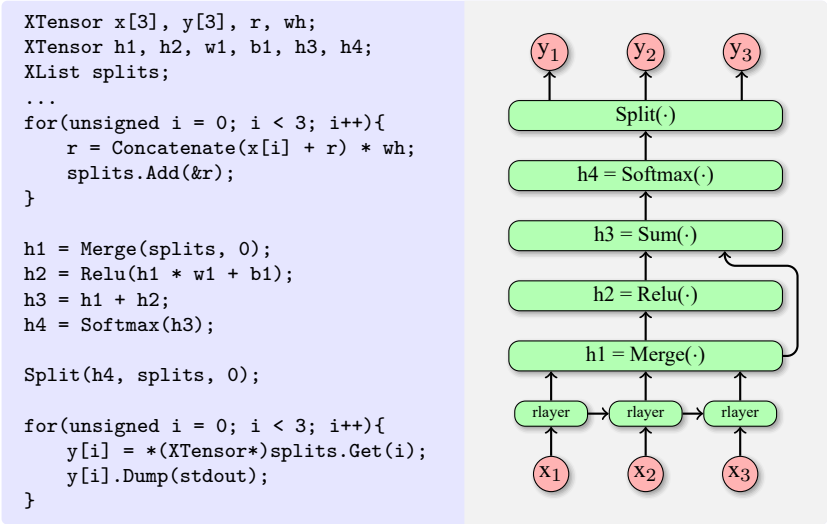


图 5.36: 构建含有循环结构的神经网络的示例

为了回答这个问题，可以把模型参数的学习过程看作是一个优化问题，即找到一组参数，使得模型达到某种最优的状态。这个问题又可以被转化为两个新的问题：

- 优化的目标是什么？
- 如何调整参数以达到优化目标？

下面会围绕这两个问题对神经网络的参数学习方法展开介绍。

5.4.1 损失函数

在神经网络的有监督学习中，训练模型的数据是由输入和正确答案所组成的样本构成的。假设有多个输入样本 $\{x_1, x_2, \dots, x_n\}$ ，每一个 x_i 都对应一个正确答案 \tilde{y}_i ， $\{x_i, \tilde{y}_i\}$ 就构成一个优化神经网络的**训练数据集**（Training Data Set）。对于一个神经网络模型 $y = f(x)$ ，每个 x_i 也会有一个输出 y_i 。如果可以度量正确答案 \tilde{y}_i 和神经网络输出 y_i 之间的偏差，进而通过调整网络参数减小这种偏差，就可以得到更好的模型。

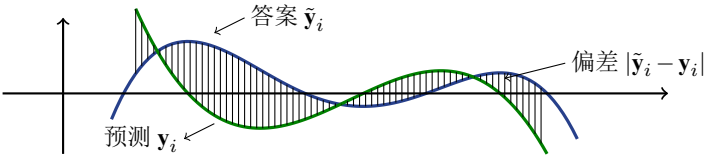


图 5.37: 正确答案与神经网络输出之间的偏差

通常，可以通过设计**损失函数**（Loss Function）来度量正确答案 \tilde{y}_i 和神经网络输出 y_i 之间的偏差。而这个损失函数往往充当训练的**目标函数**（Objective Function），神

神经网络训练就是通过不断调整神经网络内部的参数而使损失函数最小化。图5.37展示了一个绝对值损失函数的实例。

这里用 $Loss(\tilde{\mathbf{y}}_i, \mathbf{y}_i)$ 表示网络输出 \mathbf{y}_i 相对于答案 $\tilde{\mathbf{y}}_i$ 的损失，简记为 L 。表5.3是几种常见损失函数的定义。需要注意的是，没有一种损失函数可以适用于所有的问题。损失函数的选择取决于许多因素，包括：数据中是否有离群点、模型结构的选择、是否易于找到函数的导数以及预测结果的置信度等。对于相同的神经网络，不同的损失函数会对训练得到的模型产生不同的影响。对于新的问题，如果无法找到已有的、适合于该问题的损失函数，研究人员也可以自定义损失函数。因此设计新的损失函数也是神经网络中有趣的研究方向。

表 5.3: 常见的损失函数

名称	定义	应用
0-1 损失	$L = \begin{cases} 0 & \tilde{\mathbf{y}}_i = \mathbf{y}_i \\ 1 & \tilde{\mathbf{y}}_i \neq \mathbf{y}_i \end{cases}$	感知机
Hinge 损失	$L = \max(0, 1 - \tilde{\mathbf{y}}_i \cdot \mathbf{y}_i)$	SVM
绝对值损失	$L = \tilde{\mathbf{y}}_i - \mathbf{y}_i $	回归
Logistic 损失	$L = \log(1 + \tilde{\mathbf{y}}_i \cdot \mathbf{y}_i)$	回归
平方损失	$L = (\tilde{\mathbf{y}}_i - \mathbf{y}_i)^2$	回归
指数损失	$L = \exp(-\tilde{\mathbf{y}}_i \cdot \mathbf{y}_i)$	AdaBoost
交叉熵损失	$L = -\sum_k \mathbf{y}_i^{[k]} \log \tilde{\mathbf{y}}_i^{[k]}$	多分类

在实际系统开发中，损失函数中除了损失项（即用来度量正确答案 $\tilde{\mathbf{y}}_i$ 和神经网络输出 \mathbf{y}_i 之间的偏差的部分）之外，还可以包括正则项，比如 L1 正则和 L2 正则。设置正则项本质上是加入一些偏置，使模型在优化的过程中偏向某个方向多一些。关于正则项的内容将在5.4.5节详细阐述。此外，在第七章的内容中还会看到，使用恰当的正则项可以大大提升基于神经网络的机器翻译系统性能。

5.4.2 基于梯度的参数优化

对于第 i 个样本 $(\mathbf{x}_i, \tilde{\mathbf{y}}_i)$ ，把损失函数 $Loss(\tilde{\mathbf{y}}_i, \mathbf{y}_i)$ 看作是参数 \mathbf{w} 的函数³。因为输出 \mathbf{y}_i 是由输入 \mathbf{x}_i 和模型参数 \mathbf{w} 决定，因此也把损失函数写为 $L(\mathbf{x}_i, \tilde{\mathbf{y}}_i; \mathbf{w})$ 。参数学习过程可以被描述为

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^n L(\mathbf{x}_i, \tilde{\mathbf{y}}_i; \mathbf{w})$$

(5.38)

其中， $\hat{\mathbf{w}}$ 表示在训练数据上使损失的平均值达到最小的参数。 $\frac{1}{n} \sum_{i=1}^n L(\mathbf{x}_i, \tilde{\mathbf{y}}_i; \mathbf{w})$ 也

³为了简化描述，可以用 \mathbf{w} 表示神经网络中的所有参数。

被称作**代价函数**（Cost Function），它是损失函数均值期望的估计，记为 $J(\mathbf{w})$ 。

参数优化的核心问题是：找到使代价函数 $J(\mathbf{w})$ 达到最小的 \mathbf{w} 。然而 $J(\mathbf{w})$ 可能会包含大量的参数，比如，基于神经网络的机器翻译模型的参数量可能会超过一亿个。这时不可能用手动方法进行调参。为了实现高效的参数优化，比较常用的手段是使用**梯度下降方法**（Gradient Descent Method）。

梯度下降

梯度下降法是一种常用的优化方法，非常适用于目标函数可微分的问题。它的基本思想是：给定函数上的第一个点，找到使函数值变化最大的方向，然后前进一“步”，这样模型就可以朝着更大（或更小）的函数值以最快的速度移动⁴。具体来说，梯度下降通过迭代更新参数 \mathbf{w} ，不断沿着梯度的反方向让参数 \mathbf{w} 朝着损失函数更小的方向移动：如果 $J(\mathbf{w})$ 对 \mathbf{w} 可微分，则 $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$ 将指向 $J(\mathbf{w})$ 在 \mathbf{w} 处变化最大的方向，这里将其称之为梯度方向。 \mathbf{w} 沿着梯度方向更新，新的 \mathbf{w} 可以使函数更接近极值，其过程如图5.38所示。

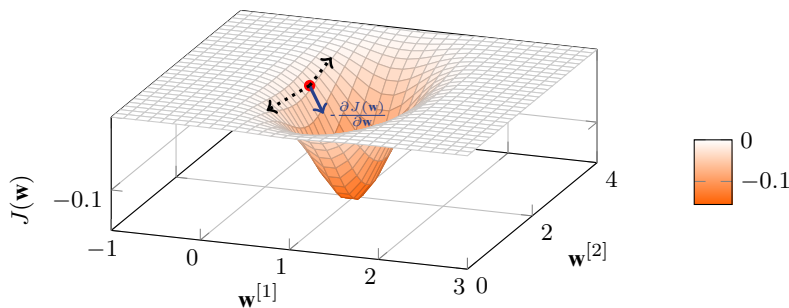


图 5.38: 函数上一个点沿着不同方向移动的示例

应用梯度下降算法时，首先需要初始化参数 \mathbf{w} 。一般情况下深度学习中的参数应该初始化为一个不太大的随机数。一旦初始化 \mathbf{w} 后，就开始对模型进行不断的更新，**参数更新的规则**（Update Rule）为：

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \tag{5.39}$$

其中 t 表示更新的步数， α 是一个超参数，被称作**学习率**（Learning Rate），表示更新步幅的大小。 α 的设置需要根据任务进行调整。

从优化的角度看，梯度下降是一种典型的**基于梯度的方法**（Gradient-based Method），属于基于一阶导数的方法。其他类似的方法还有牛顿法、共轭方向法、拟牛顿法等。在具体实现时，公式5.39可以有以下不同的形式。

⁴梯度下降的一种实现是**最速下降**（Steepest Descent）。该方法的每一步移动都选取合适的步长，进而使目标函数能得到最大程度的增长（或下降）。

批量梯度下降 (Batch Gradient Descent)

批量梯度下降是梯度下降方法中最原始的形式，这种梯度下降方法在每一次迭代时使用所有的样本进行参数更新。参数优化的目标函数是

$$J(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n L(\mathbf{x}_i, \tilde{\mathbf{y}}_i; \mathbf{w}) \quad (5.40)$$

式5.40是式5.39的严格实现，也就是将全部训练样本的平均损失作为目标函数。由全数据集确定的方向能够更好地代表样本总体，从而朝着模型在数据上整体优化所在的方向更新参数。

不过，这种方法的缺点也十分明显，因为要在全部训练数据上最小化损失，每一次参数更新都需要计算在所有样本上的损失。在使用海量数据进行训练的情况下，这种计算是非常消耗时间的。当训练数据规模很大时，很少使用这种方法。

随机梯度下降 (Stochastic Gradient Descent)

随机梯度下降（简称 SGD）不同于批量梯度下降，每次迭代只使用一个样本对参数进行更新。SGD 的目标函数是

$$J(\mathbf{w}) = L(\mathbf{x}_i, \tilde{\mathbf{y}}_i; \mathbf{w}) \quad (5.41)$$

由于每次只随机选取一个样本 $(\mathbf{x}_i, \tilde{\mathbf{y}}_i)$ 进行优化，这样更新的计算代价低，参数更新的速度大大加快，而且也适用于利用少量样本进行在线学习的情况⁵。

因为随机梯度下降算法每次优化的只是某一个样本上的损失，所以它的问题也非常明显：单个样本上的损失无法代表在全部样本上的损失，因此参数更新的效率低，方法收敛速度极慢。即使在目标函数为强凸函数的情况下，SGD 仍旧无法做到线性收敛。

小批量梯度下降 (Mini-Batch Gradient Descent)

为了综合批量梯度下降和随机梯度下降的优缺点，在实际应用中一般采用这两个算法的折中——小批量梯度下降。其思想是：每次迭代计算一小部分训练数据的损失函数，并对参数进行更新。这一小部分数据被称为一个批次（mini-batch 或者 batch）。小批量梯度下降的参数优化的目标函数如下：

$$J(\mathbf{w}) = \frac{1}{m} \sum_{i=j}^{j+m-1} L(\mathbf{x}_i, \tilde{\mathbf{y}}_i; \mathbf{w}) \quad (5.42)$$

其中 m 表示一个批次中的样本的数量， j 表示这个批次在全体训练数据的起始位置。

⁵ 比如，训练数据不是一次给定的，而是随着模型的使用不断追加的。这时，需要不断地用新的训练样本更新模型，这种模式也被称作**在线学习**（Online Learning）。

这种方法可以更充分的利用 GPU 设备，因为批次中的样本可以一起计算。而且每次使用多个样本可以大大减小使模型收敛所需要的参数更新次数。但是需要注意的是批次大小的选择对模型的最终性能是存在一定影响的。

梯度获取

梯度下降算法的一个核心是要得到目标函数相对于参数的梯度。下面将介绍三种常见的求梯度方法：数值微分、符号微分和自动微分，深度学习实现过程中多是采用自动微分方法计算梯度 [12]。

数值微分 (Numerical Differentiation)

数学上，梯度的求解其实就是求函数偏导的问题。导数是用极限来定义的，如下：

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = \lim_{\Delta \mathbf{w} \rightarrow 0} \frac{L(\mathbf{w} + \Delta \mathbf{w}) - L(\mathbf{w} - \Delta \mathbf{w})}{2\Delta \mathbf{w}} \quad (5.43)$$

其中， $\Delta \mathbf{w}$ 表示参数的一个很小的变化值。式5.43也被称作导数的双边定义。如果一个函数是初等函数，可以用求导法则来求得其导函数。如果不知道函数导数的解析式，则必须利用数值方法来求解该函数在某个点上的导数，这种方法就是数值微分。

数值微分根据导数的原始定义完成，根据公式可知，要得到损失函数在某个参数状态 \mathbf{w} 下的梯度，可以将 \mathbf{w} 增大或减小一点 ($\Delta \mathbf{w}$)，例如，取 $\Delta \mathbf{w} = 0.0001$ ，之后观测损失函数的变化与 $\Delta \mathbf{w}$ 的比值。 $\Delta \mathbf{w}$ 的取值越小计算的结果越接近导数的真实值，但是对计算的精度要求越高。

这种求梯度的方法很简单，但是计算量很大，求解速度非常慢，而且这种方法会造成**截断误差** (Truncation Error) 和**舍入误差** (Round-off Error)。在网络比较复杂、参数量稍微有点大的模型上一般不会使用这种方法。

截断误差和舍入误差是如何造成的呢？数值微分方法求梯度时，需用极限或无穷过程来求得。然而计算机需要将求解过程化为一有限系列的算术运算和逻辑运算。这样就要对某种无穷过程进行“截断”，即仅保留无穷过程的前段有限序列而舍弃它的后段。这就带来截断误差；舍入误差，是指运算得到的近似值和精确值之间的差异。由于数值微分方法计算复杂函数的梯度问题时，经过无数次的近似，每一次近似都产生了舍入误差，在这样的情况下，误差会随着运算次数增加而积累得很大，最终得出没有意义的运算结果。实际上，截断误差和舍入误差在训练复杂神经网络中，特别是使用低精度计算时，也会出现，因此是实际系统研发中需要注意的问题。

尽管数值微分不适用于大模型中的梯度求解，但是由于其非常简单，因此经常被用于检验其他梯度计算方法的正确性。比如在实现反向传播的时候（详见5.4.6节），可以检验求导是否正确 (Gradient Check)，这个过程就是利用数值微分实现的。

符号微分 (Symbolic Differentiation)

顾名思义，符号微分就是通过建立符号表达式求解微分的方法：借助符号表达式和求导公式，推导出目标函数关于自变量的微分表达式，最后再带入具体数值得到微分结果。例如，对于表达式 $L(\mathbf{w}) = \mathbf{x} \cdot \mathbf{w} + 2\mathbf{w}^2$ ，可以手动推导出微分表达式 $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = \mathbf{x} + 4\mathbf{w}$ ，最后将具体数值 $\mathbf{x} = (2 \ -3)$ 和 $\mathbf{w} = (-1 \ 1)$ 带入后，得到微分结果 $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = (2 \ -3) + 4(-1 \ 1) = (-2 \ 1)$ 。

使用这种求梯度的方法，要求必须将目标函数转化成一种完整的数学表达式，这个过程中存在**表达式膨胀** (Expression Swell) 的问题，很容易导致符号微分求解的表达式急速“膨胀”，大大增加系统存储和处理表达式的负担。关于这个问题的一个实例请看表5.4。在深层的神经网络中，神经元数量和参数量极大，损失函数的表达式会非常冗长，不易存储和管理，而且，仅仅写出损失函数的微分表达式就是一个很庞大的工作量。从另一方面来说，这里真正需要的是微分的结果值，而不是微分表达式，推导微分表达式仅仅是求解过程中的中间产物。

表 5.4: 符号微分的表达式随函数的规模增加而膨胀

函数	微分表达式	化简的微分表达式
x	1	1
$x \cdot (x + 1)$	$(x + 1) + x$	$2x + 1$
$x \cdot (x + 1) \cdot (x^2 + x + 1)$	$(x + 1) \cdot (x^2 + x + 1)$ $+ x \cdot (x^2 + x + 1)$ $+ x \cdot (x + 1) \cdot (2x + 1)$	$4x^3 + 6x^2$ $+ 4x + 1$
$(x^2 + x) \cdot (x^2 + x + 1) \cdot (x^4 + 2x^3 + 2x^2 + x + 1)$	$(2x + 1) \cdot (x^2 + x + 1) \cdot (x^4 + 2x^3 + 2x^2 + x + 1)$ $+ (x^2 + x) \cdot (2x + 1) \cdot (x^4 + 2x^3 + 2x^2 + x + 1)$ $+ (x^2 + x) \cdot (x^2 + x + 1) \cdot (4x^3 + 6x^2 + 4x + 1)$	$8x^7 + 28x^6$ $+ 48x^5 + 50x^4$ $+ 36x^3 + 18x^2$ $+ 6x + 1$

自动微分 (Automatic Differentiation)

自动微分是一种介于数值微分和符号微分的方法：将符号微分应用于最基本的算子，如常数、幂函数、指数函数、对数函数、三角函数等，然后代入数值，保留中间结果，最后再应用于整个函数。通过这种方式，将复杂的微分变成了简单的步骤，这些步骤完全自动化，而且容易进行存储和计算。

由于它只对基本函数或常数运用符号微分法则，所以它非常适合嵌入编程语言的循环条件等结构中，形成一种程序化的微分过程。在具体实现时，自动微分往往被当做是一种基于图的计算，相关的理论和技术方法相对成熟，因此是深度学习中
使用最广泛的一种方法。不同于一般的编程模式，图计算先生成计算图，然后按照

计算图执行计算过程。

自动微分可以用一种**反向模式**（Backward Mode）即反向传播思想进行描述。首先，从神经网络的输入，逐层计算每层网络的输出值。如图5.39，第 i 层的输出 \mathbf{h}_i 作为第 $i+1$ 层的输入，数据流在神经网络内部逐层传递。

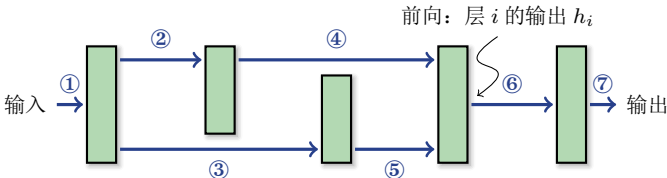


图 5.39: 前向计算示意图

前向计算实际上就是网络构建的过程，所有的计算都会被转化为计算图上的节点，前向计算和反向计算都依赖计算图来完成。构建计算图有以下两种实现方式：

- 动态图（如 Pytorch、NiuTensor 等）：前向计算与计算图的搭建同时进行，函数表达式写完即能得到前向计算的结果，有着灵活、易于调试的优点。
- 静态图（如 Tensorflow）：先搭建计算图，后执行运算，函数表达式完成后，并不能得到前向计算结果，需要显性调用一个 Forward 函数。但是计算图可以进行深度优化，执行效率较高。

对于反向计算的实现，一般从神经网络的输出开始，逆向逐层计算每层网络输入所对应的微分结果。如图5.40，在第 i 层计算此处的梯度 $\frac{\partial L}{\partial \mathbf{h}_i}$ ，并将微分值向前一层传递，根据链式法则继续计算梯度。

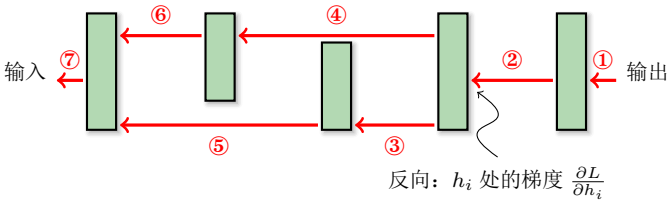


图 5.40: 反向计算示意图

反向计算也是深度学习中反向传播方法的基础。其实现的内部细节将在5.4.6节详细阐述，所以在这里不再赘述。

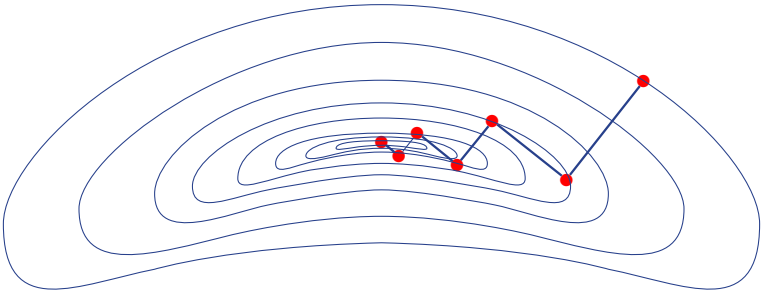
基于梯度的方法的变种和改进

参数优化通常基于梯度下降算法，即在每个更新步骤 t ，沿梯度方向更新参数：

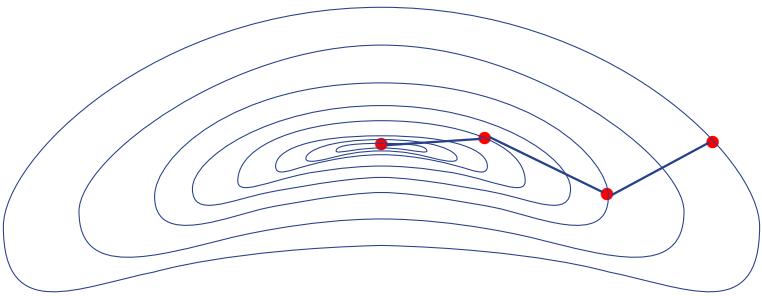
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial J(\mathbf{w}_t)}{\partial \mathbf{w}_t} \tag{5.44}$$

其中 α 是一个超参数，表示更新步幅的大小，称作**学习率**（Learning Rate）。当然，这是一种最基本的梯度下降方法。如果函数的形状非均向，比如呈延伸状，搜索最优点的路径就会非常低效，因为这时梯度的方向并没有指向最小值的方向，并且随着参数的更新，梯度方向往往呈锯齿状，这将是一条相当低效的路径；此外这种梯度下降算法并不是总能到达最优点，而是在其附近徘徊；还有一个最令人苦恼的问题——设置学习率，如果学习率设置的比较小，会导致训练收敛速度慢，如果学习率设置的比较大，会导致训练过程中因为优化幅度过大而频频跳过最优点。我们希望网络在优化的时候损失函数有一个很好的收敛速度同时又不至于摆动幅度太大。

针对以上问题，很多学者尝试对梯度下降方法做出改进，如 Momentum, Adagrad, Adadelata, RMSprop, Adam, AdaMax, Nadam, AMSGrad 等等，在这里将介绍 Momentum、AdaGrad、RMSprop、Adam 这 4 种方法。



(a) 梯度下降算法中的“锯齿”现象



(b) Momentum 梯度下降算法更加“平滑”地更新

图 5.41: Momentum 梯度下降 vs 普通梯度下降

Momentum

Momentum 梯度下降算法的参数更新公式如下⁶:

$$v_t = \beta v_{t-1} + (1 - \beta) \frac{\partial L}{\partial w_t} \tag{5.45}$$

$$w_{t+1} = w_t - \alpha v_t \tag{5.46}$$

⁶在梯度下降算法的几种改进方法的公式中，其更新对象是某个具体参数而非参数矩阵，因此不再使用加粗样式

该算法引入了一个“动量”的理念 [235]，它是基于梯度的移动指数加权平均。公式中的 v_t 是损失函数在前 $t-1$ 次更新中累积的梯度动量， β 是梯度累积的一个指数，这里一般设置值为 0.9。所以 Momentum 梯度下降算法的主要思想就是对网络的参数进行平滑处理，让梯度的摆动幅度变得更小。

这里的“梯度”不再只是现在的损失函数的梯度，而是之前的梯度的加权和。在原始的梯度下降算法中，如果在某个参数状态下，梯度方向变化特别大，甚至与上一次参数更新中梯度方向成 90 度夹角，下一次参数更新中梯度方向可能又是一次 90 度的改变，这时参数优化路径将会成“锯齿”状（如图 5.41 所示），优化效率极慢。而 Momentum 梯度下降算法不会让梯度发生 90 度的变化，而是让梯度慢慢发生改变：如果当前的梯度方向与之前的梯度方向相同，在原梯度方向上加速更新参数；如果当前的梯度方向与之前的梯度方向相反，并不会产生一个急转弯，而是尽量把优化路径平滑地进行改变。这样做的优点也非常明显，一方面杜绝了“锯齿”状优化路径的出现，另一方面将优化幅度变得更加平滑，不会导致频频跳过最优解。

AdaGrad

在神经网络的学习中，学习率的设置很重要。学习率过小，会导致学习花费过多时间；反过来，学习率过大，则会导致学习发散，甚至造成模型的“跑偏”。在深度学习实现过程中，有一种被称为学习率**衰减**（Decay）的方法，即最初设置较大的学习率，随着学习的进行，使学习率逐渐减小，这种方法相当于将“全体”参数的学习率值一起降低。AdaGrad 梯度下降算法进一步发展了这个思想 [64]。

AdaGrad 会为参数的每个元素适当地调整学习率，与此同时进行学习。其参数更新公式为：

$$z_t = z_{t-1} + \frac{\partial L}{\partial w_t} \cdot \frac{\partial L}{\partial w_t} \quad (5.47)$$

$$w_{t+1} = w_t - \eta \frac{1}{\sqrt{z_t}} \cdot \frac{\partial L}{\partial w_t} \quad (5.48)$$

这里新出现了变量 z ，它保存了以前的所有梯度值的平方和，在更新参数时，通过乘以 $\frac{1}{\sqrt{z_t}}$ ，就可以调整学习的尺度。这意味着，变动较大（被大幅度更新）的参数的学习率将变小。也就是说，可以按参数的元素进行学习率衰减，使变动大的参数的学习率逐渐减小。

RMSprop

RMSprop 算法是一种自适应学习率的方法 [283]，它是对 AdaGrad 算法的一种改进，可以避免 AdaGrad 算法中学习率不断单调下降以至于过早衰减的缺点。

RMSProp 算法沿袭了 Momentum 梯度下降算法中指数加权平均的思路，不过 Momentum 算法中加权平均的对象是梯度（即 $\frac{\partial L}{\partial w}$ ），而 RMSProp 算法加权平均的对

象是梯度的平方（即 $\frac{\partial L}{\partial w} \cdot \frac{\partial L}{\partial w}$ ）。RMSProp 算法的参数更新公式为：

$$z_t = \gamma z_{t-1} + (1 - \gamma) \frac{\partial L}{\partial w_t} \cdot \frac{\partial L}{\partial w_t} \quad (5.49)$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{z_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t} \quad (5.50)$$

公式中的 ϵ 是为了维持数值稳定性而添加的常数，一版可设为 10^{-8} 。和 AdaGrad 的想法类似，模型参数中每个元素都拥有各自的学习率。

RMSProp 与 AdaGrad 相比，学习率的分母部分（即两种梯度下降算法迭代公式中的 z ）的计算由累积方式变成了指数衰减移动平均。于是，每个参数的学习率并不是呈衰减趋势，而是既可以变小也可以变大，从而避免 AdaGrad 算法中学习率不断单调下降以至于过早衰减的缺点。

Adam

Adam 梯度下降算法是在 RMSProp 算法的基础上进行改进的，可以将其看成是带有动量项的 RMSProp 算法 [140]。该算法在自然语言处理领域非常流行。Adam 算法的参数更新公式如下，

$$v_t = \beta v_{t-1} + (1 - \beta) \frac{\partial L}{\partial w_t} \quad (5.51)$$

$$z_t = \gamma z_{t-1} + (1 - \gamma) \frac{\partial L}{\partial w_t} \cdot \frac{\partial L}{\partial w_t} \quad (5.52)$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{z_t + \epsilon}} v_t \quad (5.53)$$

可以看到 Adam 算法相当于在 RMSProp 算法中引入了 Momentum 算法中的动量项，这样做使得 Adam 算法兼具了 Momentum 算法和 RMSProp 算法的优点：既能使梯度更为“平滑”地更新，同时可以为神经网络中的每个参数设置不同的学习率。

需要注意的是包括 Adam 在内的很多参数更新算法中的学习率都需要人为设置。而且模型学习的效果与学习率的设置关系极大，甚至在研发实际系统时工程师需要进行大量的实验，才能得到最佳的模型。第六章还会具体介绍在机器翻译中参数更新学习率设置的策略。

5.4.3 参数更新的并行化策略

虽然通过 GPU 可以加速神经网络的运算，但当神经网络较为复杂时，模型训练还是需要几天甚至几周的时间。如果希望尽可能缩短一次学习所需的时间，最直接的想法就是把不同的训练样本分配给多个 GPU 或 CPU，然后在这些设备上同时进行训练，即实现并行化训练。这种方法也被称作**数据并行**。具体实现时，有两种常用的并行化策略：（参数）同步更新和（参数）异步更新。

- **同步更新**（Synchronous Update）是指所有计算设备完成计算后，统一汇总并更新参数。当所有设备的反向传播算法完成之后同步更新参数，不会出现单个设备单独对参数进行更新的情况。这种方法效果稳定，但是效率比较低，在同步更新时，每一次参数更新都需要所有设备统一开始、统一结束，如果设备的运行速度不一致，那么每一次参数更新都需要等待最慢的设备结束才能开始。
- **异步更新**（Asynchronous Update）是指每个计算设备可以随时更新参数。不同设备可以随时读取参数的最新值，然后根据当前参数值和分配的训练样本，各自执行反向传播过程并独立更新参数。由于设备间不需要相互等待，这种方法并行度高。但是不同设备读取参数的时间可能不同，会造成不同设备上的参数不同步，导致这种方法不十分稳定，有可能无法达到较好的训练结果。

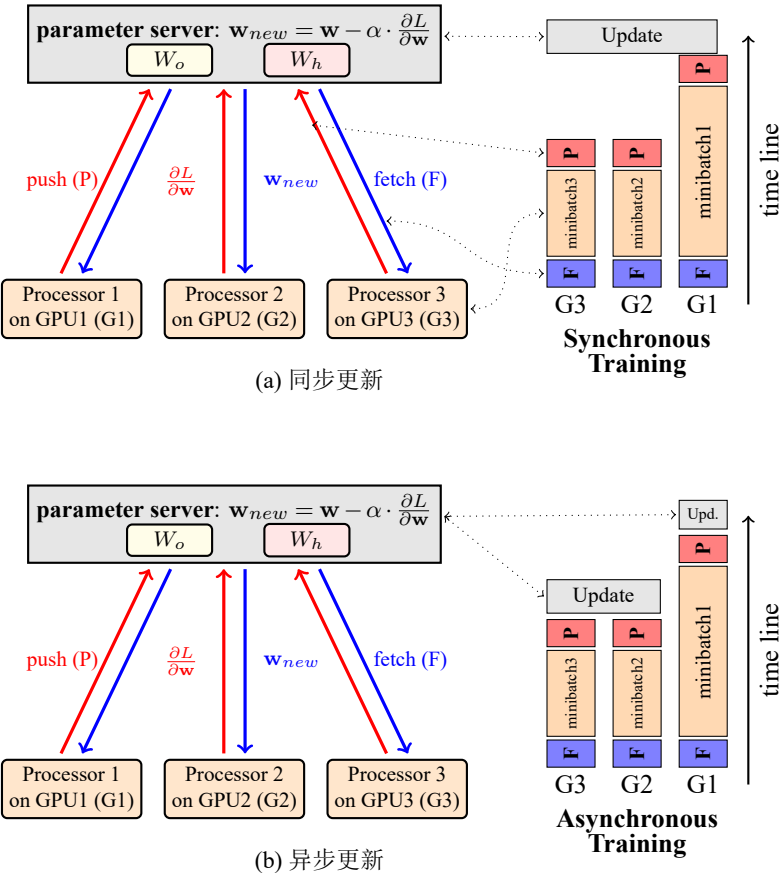


图 5.42: 同步更新与异步更新对比

图5.42对比了同步更新和异步更新的区别，在这个例子中，使用 4 台设备对一个两层神经网络中的参数进行更新，其中使用了一个**参数服务器**（Parameter Server）来保存最新的参数，不同设备（Worker，图中的 G1、G2、G3）可以通过同步或者异步的方式访问参数服务器。图中的 w_o 和 w_h 分别代表输出层和隐藏层的全部参数，操

作 `push(P)` 表示设备向参数服务器传送梯度，操作 `fetch(F)` 表示参数服务器向设备传送更新后的参数。

此外，在使用多个设备进行并行训练的时候，由于设备间带宽的限制，大量的数据传输会有较高的延时。对于复杂神经网络来说，设备间参数和梯度传递的时间消耗也会成为一个不得不考虑的因素。有时候，设备间数据传输的时间甚至比模型计算的时间都长，大大降低了并行度 [320]。对于这种问题，可以考虑对数据进行压缩或者减少传输的次数来缓解问题。

5.4.4 梯度消失、梯度爆炸和稳定性训练

深度学习中随着神经网络层数的增加，导数可能会出现指数级的下降或者指数级的增加，这种现象分别称为**梯度消失**（Gradient Vanishing）和**梯度爆炸**（Gradient Explosion）。出现这两种现象的本质原因是反向传播过程中链式法则导致梯度矩阵的多次相乘。这类问题很容易导致训练的不稳定。

易于优化的激活函数

网络训练过程中，如果每层网络的梯度都小于 1，各层梯度的偏导数会与后面层传递而来的梯度相乘得到本层的梯度，并向前一层传递。该过程循环进行，最后导致梯度指数级地减小，这就产生了梯度消失现象。这种情况会导致神经网络层数较浅的部分梯度接近 0。一般来说，产生很小梯度的原因是使用了类似于 Sigmoid 这样的激活函数，当输入的值过大或者过小的时候这类函数曲线会趋于直线，梯度近似为零。针对这个问题，主要的解决办法是使用更加易于优化的激活函数，比如，使用 ReLU 代替 Sigmoid 和 Tanh 作为激活函数。

缓解梯度消失问题最直接的想法就是希望各层的偏导数大于或等于 1。图5.43展示了 Sigmoid 激活函数 $y = \frac{1}{1+e^{-x}}$ 的函数曲线和导函数曲线，如果使用 Sigmoid 作为损失函数，其梯度不可能超过 0.25，这样经过链式求导之后，很容易发生梯度消失。

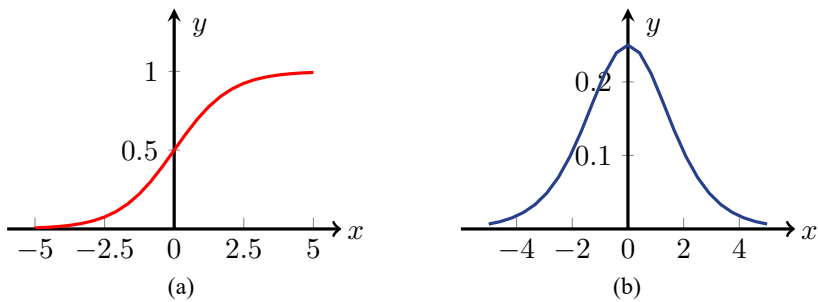


图 5.43: Sigmoid 激活函数的函数曲线和导函数曲线

同理，Tanh 作为激活函数也容易出现梯度消失现象，图5.44展示了 Tanh 激活函数 $y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ 的函数曲线和导函数曲线，可以看出，Tanh 激活函数比 Sigmoid 激活函数要好一些，但是 Tanh 激活函数的导数也小于 1，因此无法避免梯度消失现象。

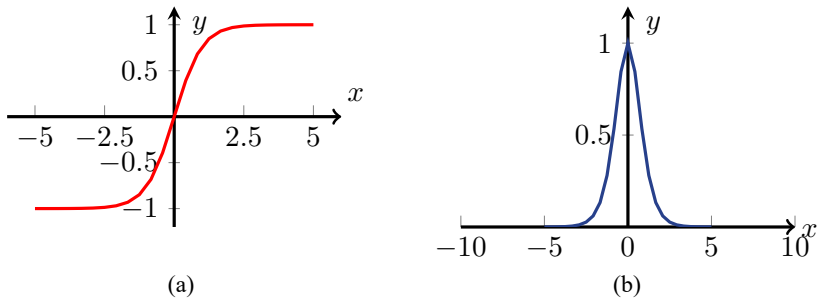


图 5.44: Tanh 激活函数的函数曲线和导函数曲线

ReLU 激活函数的思想也很简单，如果激活函数的导数为 1，那么就不存在梯度消失爆炸的问题了。图5.45展示了 ReLU 激活函数 $y = \max(0, x)$ 的函数曲线和导函数曲线。可以很容易看出，ReLU 函数的导数在正数部分是恒等于 1 的，因此在深层网络中使用 ReLU 激活函数就不会产生很小的梯度。

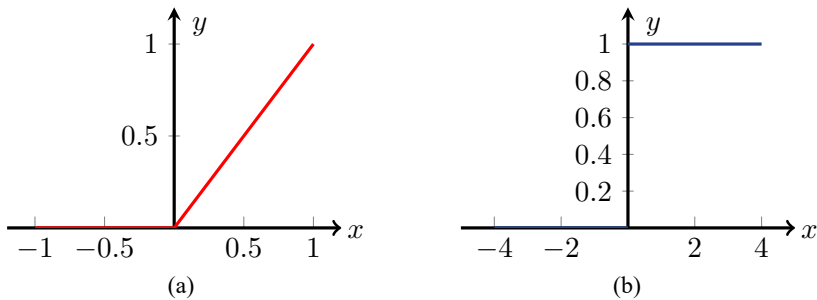


图 5.45: ReLU 激活函数的函数曲线和导函数曲线

当然，梯度消失并不是仅仅可以通过改变激活函数就可以完全消除掉。随着网络层数的增加，很多因素都可能会造成梯度消失。后面也会进一步介绍其他手段，可以综合运用这些方法达到很好的缓解梯度消失问题的目的。

梯度裁剪

网络训练过程中，如果参数的初始值过大，而且每层网络的梯度都大于 1，反向传播过程中，各层梯度的偏导数都会比较大，会导致梯度指数级地增长直至超出浮点数表示的范围，这就产生了梯度爆炸现象。如果发生这种情况，模型中离输入近的部分比离输入远的部分参数更新得更快，使网络变得非常不稳定。在极端情况下，模型的参数值变得非常大，甚至于溢出。针对梯度爆炸的问题，常用的解决办法为**梯度裁剪**（Gradient Clipping）。

梯度裁剪的思想是设置一个梯度剪切阈值。在更新梯度的时候，如果梯度超过这个阈值，就将其强制限制在这个范围之内。假设梯度为 g ，梯度剪切阈值为 θ ，梯

度裁剪的公式为

$$\mathbf{g} = \min(\frac{\theta}{\|\mathbf{g}\|}, 1)\mathbf{g} \tag{5.54}$$

其中， $\|\cdot\|$ 表示 l_2 范数。梯度裁剪经常被使用在层数较多的模型中，如循环神经网络。

稳定性训练

为了使神经网络模型训练更加稳定，通常还会考虑其他策略。

(1) 批量归一化 (Batch Normalization)

批量归一化，顾名思义，是以进行学习时的小批量样本为单位进行归一化 [123]。具体而言，就是对神经网络隐层输出的每一个维度，沿着批次的方向进行均值为 0、方差为 1 的归一化。在深层神经网络中，每一层网络都可以使用批量归一化操作。这样使神经网络任意一层的输入不至于过大或过小，从而防止隐层中异常值导致模型状态的巨大改变。

(2) 层归一化 (Layer Normalization)

类似的，层归一化更多是针对自然语言处理这种序列处理任务 [5]，它和批量归一化的原理是一样的，只是归一化操作是在序列上同一层网络的输出结果上进行的，也就是归一化操作沿着序列方向进行。这种方法可以很好的避免序列上不同位置神经网络输出结果的不可比性。同时由于归一化后所有的结果都转化到一个可比的范围，使得隐层状态可以在不同层之间进行自由组合。

(3) 残差网络 (Residual Networks)

最初，残差网络是为了解决神经网络持续加深时的模型退化问题 [101]，但是残差结构对解决梯度消失和梯度爆炸问题也有所帮助。有了残差结构，可以很轻松的构建几十甚至上百层的神经网络，而不用担心层数过深造成的梯度消失问题。残差网络的结构如图5.46所示：

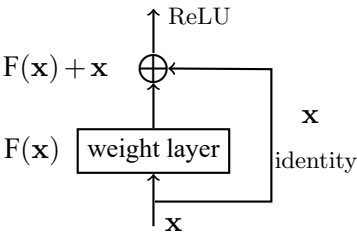


图 5.46: 残差网络的结构

图5.46中右侧的曲线叫做**跳接** (Shortcut Connection)，通过跳接在激活函数前，将上一层（或几层）之前的输出与本层计算的输出相加，将求和的结果输入到激活

函数中作为本层的输出。假设残差结构的输入为 \mathbf{x}_l ，输出为 \mathbf{x}_{l+1} ，则有

$$\mathbf{x}_{l+1} = F(\mathbf{x}_l) + \mathbf{x}_l \quad (5.55)$$

相比较于简单的多层堆叠的结构，残差网络提供了跨层连接结构。这种结构在反向传播中有很大的好处，比如，对于 \mathbf{x}_l 处的梯度可以进行如下计算：

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{x}_l} &= \frac{\partial L}{\partial \mathbf{x}_{l+1}} \cdot \frac{\partial \mathbf{x}_{l+1}}{\partial \mathbf{x}_l} \\ &= \frac{\partial L}{\partial \mathbf{x}_{l+1}} \cdot \left(1 + \frac{\partial F(\mathbf{x}_l)}{\partial \mathbf{x}_l} \right) \\ &= \frac{\partial L}{\partial \mathbf{x}_{l+1}} + \frac{\partial L}{\partial \mathbf{x}_{l+1}} \cdot \frac{\partial F(\mathbf{x}_l)}{\partial \mathbf{x}_l} \end{aligned} \quad (5.56)$$

由上式可知，残差网络可以将后一层的梯度 $\frac{\partial L}{\partial \mathbf{x}_{l+1}}$ 不经过任何乘法项直接传递到 $\frac{\partial L}{\partial \mathbf{x}_l}$ ，从而缓解了梯度经过每一层后多次累乘造成的梯度消失问题。在第六章中还会看到，在机器翻译中残差结构可以和层归一化一起使用，而且这种组合可以取得很好的效果。

5.4.5 过拟合

理想中，我们总是希望尽可能地拟合输入和输出之间的函数关系，即让模型尽量模拟训练数据的中由输入预测答案的行为。然而，在实际应用中，模型在训练数据上的表现不一定代表了其在未见数据上的表现。如果模型训练过程中过度拟合训练数据，最终可能无法对未见数据做出准确的判断，这种现象叫做**过拟合**（Overfitting）。随着模型复杂度增加，特别在神经网络变得更深、更宽时，过拟合问题会表现得更为突出。如果训练数据量较小，而模型又很复杂，可以“完美”地拟合这些数据，这时过拟合也很容易发生。所以在模型训练时，往往不希望去完美拟合训练数据中的每一个样本。

正则化（Regularization）是常见的缓解过拟合问题的手段，通过在损失函数中加上用来刻画模型复杂程度的正则项来惩罚过度复杂的模型，从而避免神经网络过度学习造成过拟合。引入正则化处理之后目标函数变为 $J(\mathbf{w}) + \lambda R(\mathbf{w})$ ，其中 $J(\mathbf{w})$ 是原来的代价函数， $R(\mathbf{w})$ 即为正则项， λ 用来调节正则项对结果影响的程度。

过拟合的模型通常会表现为部分非零参数过多或者参数的值过大。这种参数产生的原因在于模型需要复杂的参数才能匹配样本中的个别现象甚至噪声。基于此，常见的正则化方法有 L1 正则化和 L2 正则化，其命名方式是由 $R(\mathbf{w})$ 的计算形式来决定的。在 L1 正则化中， $R(\mathbf{w})$ 即为参数 w 的 l_1 范数，即 $R(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{i=1}^n |w_i|$ ；在 L2 正则化中， $R(\mathbf{w})$ 即为参数 w 的 l_2 范数的平方，即 $R(\mathbf{w}) = (\|\mathbf{w}\|_2)^2 = \sum_{i=1}^n w_i^2$ 。L1 正则化中的正则项衡量了模型权数中的绝对值大小，倾向于生成值为 0 的参数，从而让参数变得更加稀疏；而 L2 正则化由于平方的加入，当参数中的某一项小到一

定程度，比如 0.001 的时候，参数的平方结果已经可以忽略不计了，因此 L2 正则化会倾向生成很小的参数，在这种情况下，即便训练数据中含有少量随机噪音，模型也不大容易通过增加个别参数的值来对噪声进行过度拟合，即提高了模型的抗扰动能力。

此外，在第六章即将介绍的 Dropout 和 Label Smoothing 方法也可以被看作是一种正则化操作。它们都可以提高模型在未见数据上的泛化能力。

5.4.6 反向传播

为了获取梯度，最常用的做法是使用自动微分技术，通常通过**反向传播**（back propagation）来实现。该方法分为两个计算过程：前向计算和反向计算。前向计算的目的是从输入开始，逐层计算，得到网络的输出，并记录计算图中每个节点的局部输出。反向计算过程从输出端反向计算梯度，这个过程可以被看作是一种梯度的“传播”，最终计算图中所有节点都会得到相应的梯度结果。

这里，首先对反向传播算法中涉及到的符号进行统一说明。图5.47是一个多层神经网络，其中层 $k-1$ 、层 k 、层 $k+1$ 均为神经网络中的隐藏层，层 K 为神经网络中的输出层。为了化简问题，这里每层网络没有使用偏置项。

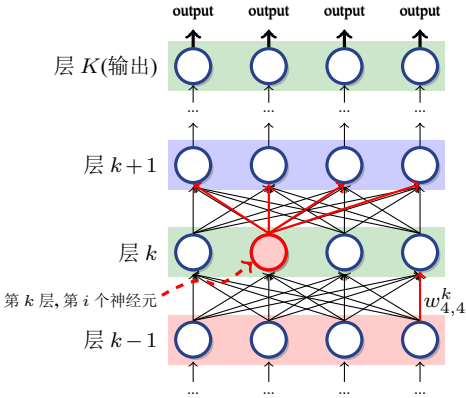


图 5.47: 多层神经网络实例

下面是一些符号的定义：

- h_i^k : 第 k 层第 i 个神经元的输出；
- \mathbf{h}^k : 第 k 层的输出。若第 k 层有 n 个神经元，则：

$$\mathbf{h}^k = (h_1^k, h_2^k, \dots, h_n^k) \tag{5.57}$$

- $w_{j,i}^k$: 第 $k-1$ 层神经元 j 与第 k 层神经元 i 的连接权重；

- \mathbf{w}^k : 第 $k-1$ 层与第 k 层的连接权重。若第 $k-1$ 层有 m 个神经元, 第 k 层有 n 个神经元, 则:

$$\mathbf{w}^k = \begin{pmatrix} w_{1,1}^k & w_{1,2}^k & \cdots & w_{1,n}^k \\ w_{2,1}^k & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ w_{m,1}^k & \cdots & \cdots & w_{m,n}^k \end{pmatrix} \quad (5.58)$$

- \mathbf{h}^K : 整个网络的输出;
- \mathbf{s}^k : 第 k 层的线性变换结果, 有:

$$\begin{aligned} \mathbf{s}^k &= \mathbf{h}^{k-1} \mathbf{w}^k \\ &= \sum h_j^{k-1} w_{j,i}^k \end{aligned} \quad (5.59)$$

- f^k : 第 k 层的激活函数, $\mathbf{h}^k = f^k(\mathbf{s}^k)$ 。

于是, 在神经网络的第 k 层, 前向计算过程为:

$$\begin{aligned} \mathbf{h}^k &= f^k(\mathbf{s}^k) \\ &= f^k(\mathbf{h}^{k-1} \mathbf{w}^k) \end{aligned} \quad (5.60)$$

输出层的反向传播

反向传播是由输出层开始计算梯度, 之后逆向传播到每一层网络, 直至到达输入层。这里首先讨论输出层的反向传播机制。输出层 (即第 K 层) 可以被描述为:

$$\mathbf{h}^K = f^K(\mathbf{s}^K) \quad (5.61)$$

$$\mathbf{s}^K = \mathbf{h}^{K-1} \mathbf{w}^K \quad (5.62)$$

也就是, 输出层 (第 K 层) 的输入 \mathbf{h}^{K-1} 先经过线性变换右乘 \mathbf{w}^K 转换为中间状态 \mathbf{s}^K , 之后 \mathbf{s}^K 再经过激活函数 $f^K(\cdot)$ 变为 \mathbf{h}^K , \mathbf{h}^K 即为第 K 层 (输出层) 的输出。最后, \mathbf{h}^K 和标准答案一起计算得到损失函数的值, 记为 L 。以上过程如图5.48所示。这里将输出层的前向计算过程细化为两个阶段: 线性变换阶段和激活函数 + 损失函数阶段。

在前向过程中, 计算次序为 $\mathbf{h}^{K-1} \rightarrow \mathbf{s}^K \rightarrow \mathbf{h}^K \rightarrow L$ 。而反向计算中节点访问的次序与之相反:

- 第一步, 获取 $\frac{\partial L}{\partial \mathbf{h}^K}$, 即计算损失函数 L 关于网络输出结果 \mathbf{h}^K 的梯度, 并将梯度向前传递;

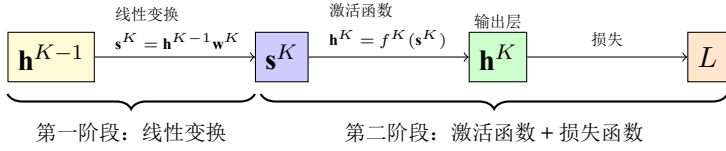


图 5.48: 输出层的前向计算过程

- 第二步，获取 $\frac{\partial L}{\partial \mathbf{s}^K}$ ，即计算损失函数 L 关于中间状态 \mathbf{s}^K 的梯度，并将梯度向前传递；
- 第三步，获取 $\frac{\partial L}{\partial \mathbf{h}^{K-1}}$ 和 $\frac{\partial L}{\partial \mathbf{w}^K}$ ，即计算损失函数 L 关于第 $K-1$ 层输出结果 \mathbf{h}^{K-1} 的梯度，并将梯度向前传递；同时计算损失函数 L 关于第 K 层参数 \mathbf{w}^K 的梯度，并用于参数更新。

对于前两个步骤，如图5.49所示：

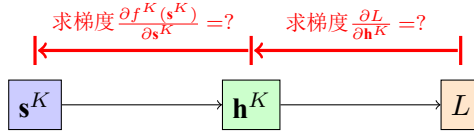


图 5.49: 从损失到中间状态的反向传播（输出层）

在第一阶段，计算的目标是得到损失函数 L 关于第 K 层中间状态 \mathbf{s}^K 的梯度，这里令 $\pi^K = \frac{\partial L}{\partial \mathbf{s}^K}$ ，利用链式法则有：

$$\begin{aligned}
 \pi^K &= \frac{\partial L}{\partial \mathbf{s}^K} \\
 &= \frac{\partial L}{\partial \mathbf{h}^K} \cdot \frac{\partial \mathbf{h}^K}{\partial \mathbf{s}^K} \\
 &= \frac{\partial L}{\partial \mathbf{h}^K} \cdot \frac{\partial f^K(\mathbf{s}^K)}{\partial \mathbf{s}^K}
 \end{aligned} \tag{5.63}$$

其中：

- $\frac{\partial L}{\partial \mathbf{h}^K}$ 表示损失函数 L 相对网络输出 \mathbf{h}^K 的梯度。比如，对于平方损失 $L = \frac{1}{2} \|\tilde{\mathbf{y}} - \mathbf{h}^K\|^2$ ，有 $\frac{\partial L}{\partial \mathbf{h}^K} = \tilde{\mathbf{y}} - \mathbf{h}^K$ 。计算结束后，将 $\frac{\partial L}{\partial \mathbf{h}^K}$ 向前传递。
- $\frac{\partial f^K(\mathbf{s}^K)}{\partial \mathbf{s}^K}$ 表示激活函数相对于其输入 \mathbf{s}^K 的梯度。比如，对于 Sigmoid 函数 $f(\mathbf{s}) = \frac{1}{1+e^{-\mathbf{s}}}$ ，有 $\frac{\partial f(\mathbf{s})}{\partial \mathbf{s}} = f(\mathbf{s})(1 - f(\mathbf{s}))$

这个过程可以得到 \mathbf{s}^K 节点处的梯度 $\pi^K = \frac{\partial L}{\partial \mathbf{s}^K}$ ，在后续的过程中可以直接使用其作为前一层提供的梯度计算结果，而不需要从 \mathbf{h}^K 节点处重新计算。这也体现了自动微分与符号微分的差别，对于计算图的每一个阶段，并不需要得到完成的微分

表达式，而是通过前一层提供的梯度，直接计算当前的梯度即可，这样避免了大量的重复计算。

在得到 $\pi^K = \frac{\partial L}{\partial \mathbf{s}^K}$ 之后，下一步的目标是：1) 计算损失函数 L 相对于第 $K-1$ 层与输出层之间连接权重 \mathbf{w}^K 的梯度；2) 计算损失函数 L 相对于神经网络第 $K-1$ 层输出结果 \mathbf{h}^{K-1} 的梯度。这部分内容如图5.50所示。

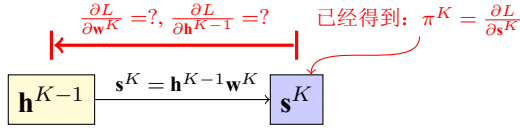


图 5.50: 从中间状态到输入的反向传播（输出层）

具体来说：

- 计算 $\frac{\partial L}{\partial \mathbf{w}^K}$ ：由于 $\mathbf{s}^K = \mathbf{h}^{K-1}\mathbf{w}^K$ ，且损失函数 L 关于 \mathbf{s}^K 的梯度 $\pi^K = \frac{\partial L}{\partial \mathbf{s}^K}$ 已经得到，于是有

$$\frac{\partial L}{\partial \mathbf{w}^K} = [\mathbf{h}^{K-1}]^T \pi^K \quad (5.64)$$

其中 $[\cdot]^T$ 表示转置操作⁷。

- 计算 $\frac{\partial L}{\partial \mathbf{h}^{K-1}}$ ：与求解 $\frac{\partial L}{\partial \mathbf{w}^K}$ 类似，可以得到

$$\frac{\partial L}{\partial \mathbf{h}^{K-1}} = \pi^K [\mathbf{w}^K]^T \quad (5.65)$$

梯度 $\frac{\partial L}{\partial \mathbf{h}^{K-1}}$ 需要继续向前一层传递，用于计算网络中间层的梯度。 $\frac{\partial L}{\partial \mathbf{w}^K}$ 会作为参数 \mathbf{w}^K 的梯度计算结果，用于模型参数的更新⁸。

隐藏层的反向传播

对于第 k 个隐藏层，有：

$$\mathbf{h}^k = f^k(\mathbf{s}^k) \quad (5.66)$$

$$\mathbf{s}^k = \mathbf{h}^{k-1}\mathbf{w}^k \quad (5.67)$$

其中， \mathbf{h}^k 、 \mathbf{s}^k 、 \mathbf{h}^{k-1} 、 \mathbf{w}^k 和分别表示隐藏层的输出、中间状态、隐藏层的输入和参数矩阵。隐藏层的前向计算过程如图5.51所示，第 $k-1$ 层神经元的输出 \mathbf{h}^{k-1} 经过

⁷如果 \mathbf{h}^{K-1} 是一个向量， $[\mathbf{h}^{K-1}]^T$ 表示向量的转置，比如，行向量变成列向量；如果 \mathbf{h}^{K-1} 是一个高阶张量， $[\mathbf{h}^{K-1}]^T$ 表示沿着张量最后两个方向的转置。

⁸ \mathbf{w}^K 可能会在同一个网络中被多次使用（类似于网络不同部分共享同一个参数），这时需要累加相关计算节点处得到的 $\frac{\partial L}{\partial \mathbf{w}^K}$ 。

线性变换和激活函数后，将计算结果 \mathbf{h}^k 向后一层传递。

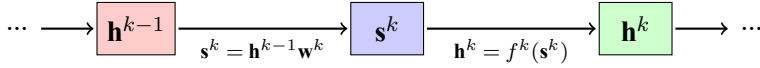


图 5.51: 隐藏层的前向计算过程

与输出层类似，隐藏层的反向传播也是逐层逆向计算。

- 第一步，获取 $\frac{\partial L}{\partial \mathbf{s}^k}$ ，即计算损失函数 L 关于第 k 层中间状态 \mathbf{s}^k 的梯度，并将梯度向前传递；
- 第二步，获取 $\frac{\partial L}{\partial \mathbf{h}^{k-1}}$ 和 $\frac{\partial L}{\partial \mathbf{w}^k}$ ，即计算损失函数 L 关于第 $k-1$ 层输出结果 \mathbf{h}^{k-1} 的梯度，并将梯度向前传递。同时计算损失函数 L 关于参数 \mathbf{w}^k 的梯度，并用于参数更新。

这两步和输出层的反向传播十分类似。可以利用链式法则得到：

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{s}^k} &= \frac{\partial L}{\partial \mathbf{h}^k} \cdot \frac{\partial \mathbf{h}^k}{\partial \mathbf{s}^k} \\ &= \frac{\partial L}{\partial \mathbf{h}^k} \cdot \frac{\partial f^k(\mathbf{s}^k)}{\partial \mathbf{s}^k}\end{aligned}\quad (5.68)$$

其中 $\frac{\partial L}{\partial \mathbf{h}^k}$ 表示损失函数 L 相对该隐藏层输出 \mathbf{h}^k 的梯度。进一步，由于 $\mathbf{s}^k = \mathbf{h}^{k-1} \mathbf{w}^k$ ，可以得到

$$\frac{\partial L}{\partial \mathbf{w}^k} = [\mathbf{h}^{k-1}]^T \cdot \frac{\partial L}{\partial \mathbf{s}^k} \quad (5.69)$$

$$\frac{\partial L}{\partial \mathbf{h}^{k-1}} = \frac{\partial L}{\partial \mathbf{s}^k} \cdot [\mathbf{w}^k]^T \quad (5.70)$$

$\frac{\partial L}{\partial \mathbf{h}^{k-1}}$ 需要继续向第 $k-1$ 隐藏层传递。 $\frac{\partial L}{\partial \mathbf{w}^k}$ 会作为参数的梯度用于参数更新。图5.52展示了隐藏层反向传播的计算过程。

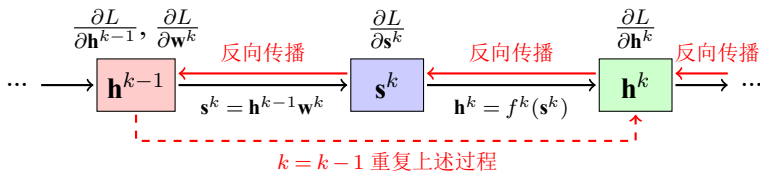


图 5.52: 隐藏层的反向传播

综合输出层和隐藏层的反向传播方法，可以得到神经网络中任意位置和任意参数的梯度信息。只需要根据网络的拓扑结构，逆向访问每一个节点，并执行上述反向计算过程。

程序实现

在了解了反向传播的原理之后，实现反向传播就变得非常容易了。实际上，现在主流的深度学习框架都支持自动微分。为了进一步说明反向传播的过程，这里使用 NiuTensor 工具构建两个简单的实例，并分别尝试手动编写反向传播代码和使用 NiuTensor 自带的自动微分模块。

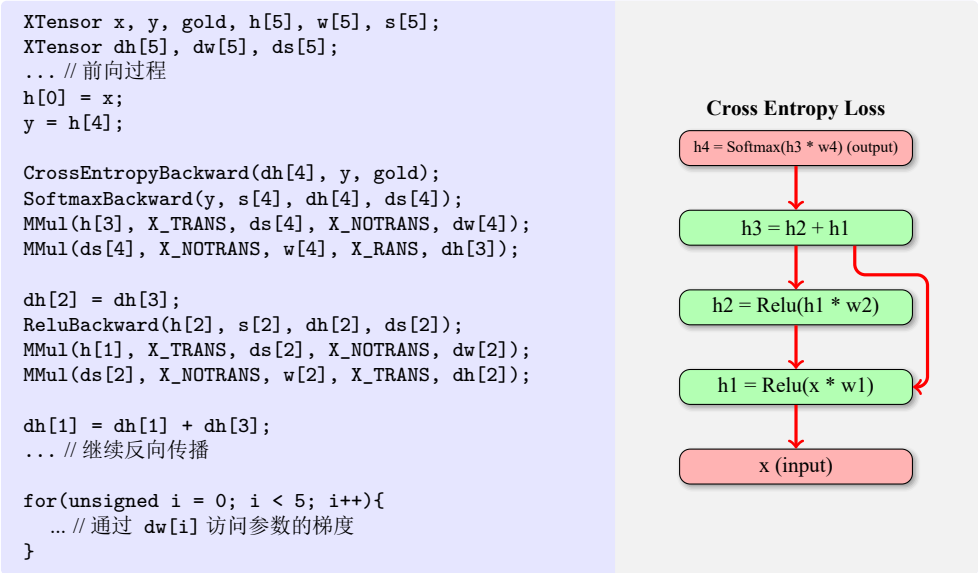


图 5.53: 手动编写反向传播代码（NiuTensor）

图5.53展示了一个简单的神经网络的反向传播程序示例。这种反向传播的实现方式正是上一节内容的代码实现：按层实现自动微分并将梯度向上一层传播。

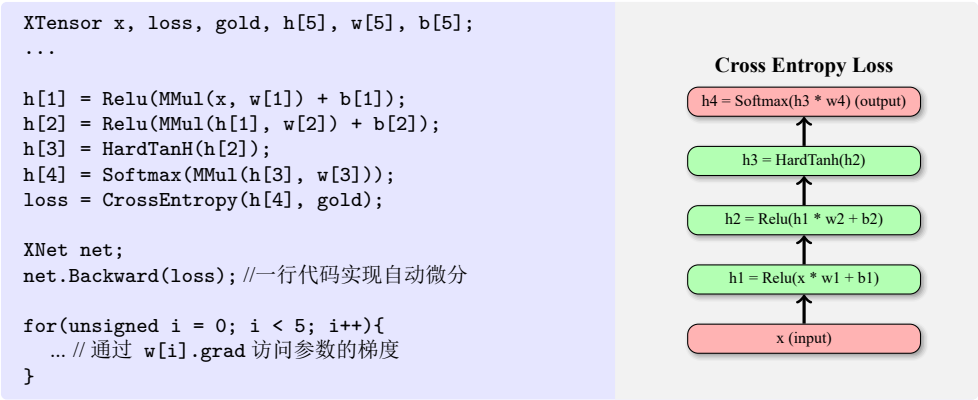


图 5.54: 反向传播的自动微分实现（NiuTensor）

此外，NiuTensor 还提供了一种更简单的一步完成神经网络反向传播的实现方式。如图5.54所示，在完成神经网络的搭建后，无论前向计算过程是怎样的，直接利

用 Backward 函数就可以实现整个神经网络的反向传播，系统开发人员可以完全不用关心其求解过程。

5.5 神经语言模型

神经网络给我们提供了一种工具，只要将问题的输入和输出定义好，就可以学习输入和输出之间的对应关系。显然，很多自然语言处理任务都可以用神经网络进行实现。比如，在机器翻译中，可以把输入的源语言句子和输出的目标语言句子用神经网络建模；在文本分类中，可以把输入的文本内容和输出的类别标签进行神经网络建模，等等。

为了更好地理解神经网络和深度学习在自然语言处理中的应用。这里介绍一种基于神经网络的语言建模方法——**神经语言模型**（Neural Language Model）。可以说，神经语言模型是深度学习时代下自然语言处理的标志性成果，它所涉及的许多概念至今仍是研究的热点，比如：词嵌入、表示学习、预训练等。此外，神经语言模型也为机器翻译的建模提供了很好的思路。从某种意义上说，机器翻译的深度学习建模的很多灵感均来自神经语言模型，二者在一定程度上是统一的。

5.5.1 基于神经网络的语言建模

回顾一下第二章的内容，语言建模的问题被定义为：对于一个词序列 $w_1 w_2 \dots w_m$ ，如何计算该词序列的可能性？词序列出现的概率可以通过链式法则得到：

$$P(w_1 w_2 \dots w_m) = P(w_1)P(w_2|w_1)P(w_3|w_1 w_2) \dots P(w_m|w_1 \dots w_{m-1}) \quad (5.71)$$

由于 $P(w_m|w_1 \dots w_{m-1})$ 需要建模 $m-1$ 个词构成的历史信息，这个模型仍然很复杂。于是就有了基于局部历史的 n -gram 语言模型，即：

$$P(w_m|w_1 \dots w_{m-1}) = P(w_m|w_{m-n+1} \dots w_{m-1}) \quad (5.72)$$

其中， $P(w_m|w_{m-n+1} \dots w_{m-1})$ 可以通过相对频次估计进行计算， $\text{count}(\cdot)$ 表示在训练数据上的频次：

$$P(w_m|w_{m-n+1} \dots w_{m-1}) = \frac{\text{count}(w_{m-n+1} \dots w_m)}{\text{count}(w_{m-n+1} \dots w_{m-1})} \quad (5.73)$$

这里， $w_{m-n+1} \dots w_m$ 也被称作 n -gram，即 n 元语法单元。 n -gram 语言模型是一种典型的基于离散表示的模型。在这个模型中，所有的词都被看作是离散的符号。因此，不同单词之间是“完全”不同的。另一方面，语言现象是十分多样的，即使在很大的语料库上也无法得到所有 n -gram 的准确统计。甚至很多 n -gram 在训练数据中从未出现过。由于不同 n -gram 间没有建立直接的联系， n -gram 语言模型往往面临数据稀

疏的问题。比如，虽然在训练数据中见过“景色”这个词，但是测试数据中却出现了“风景”这个词，恰巧“风景”在训练数据中没有出现过。即使“风景”和“景色”表达的是相同的意思，*n*-gram 语言模型仍然会把“风景”看作未登录词，赋予一个很低的概率值。

上面这个问题的本质是 *n*-gram 语言模型对词使用了离散化表示，即每个单词都孤立的对应词表中的一个索引，词与词之间在语义上没有任何“重叠”。神经语言模型重新定义了这个问题。这里并不需要显性地通过统计离散的 *n*-gram 的频度，而是直接设计一个神经网络模型 $g(\cdot)$ 来估计单词生成的概率，

$$P(w_m|w_1 \dots w_{m-1}) = g(w_1 \dots w_m) \tag{5.74}$$

$g(w_1 \dots w_m)$ 实际上是一个多层神经网络。与 *n*-gram 语言模型不同的是 $g(w_1 \dots w_m)$ 并不包含对 $w_1 \dots w_m$ 的任何假设，比如，在神经网络模型中，单词不再是离散的符号，而是连续空间上的点。这样两个单词之间也不再是简单的非 0 即 1 的关系，而是具有可计算的距离。此外，由于没有对 $w_1 \dots w_m$ 进行任何结构性的假设，神经语言模型对问题进行端到端学习。通过设计不同的神经网络 $g(\cdot)$ ，可以从不同的角度“定义”序列的表示问题。当然，这么说可能还有一些抽象，下面就一起看看神经语言模型究竟是什么样子的。

基于前馈神经网络的语言模型

最具代表性的神经语言模型是 Bengio 等人提出的**前馈神经网络语言模型**（Feed-forward Neural Network Language Model，简称 FNNLM）。这种语言模型的目标是用神经网络计算 $P(w_m|w_{m-n+1} \dots w_{m-1})$ ，之后将多个 *n*-gram 的概率相乘得到整个序列的概率 [14]。

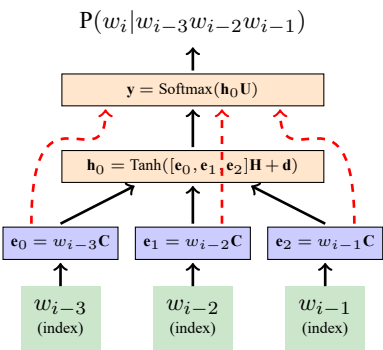


图 5.55: 4-gram 前馈神经网络语言架构

为了有一个直观的认识，这里以 4-gram 的 FNNLM 语言模型为例，即根据前三个单词 w_{i-3} 、 w_{i-2} 、 w_{i-1} 预测当前单词 w_i 的概率。如图5.55所示， w_{i-3} 、 w_{i-2} 、 w_{i-1} 为该语言模型的输入（绿色方框），输入为每个词的 One-hot 向量表示（维度大

小与词表大小一致)，每个 One-hot 向量仅一维为 1，其余为 0，比如：(0,0,1,...,0) 表示词表中第三个单词。之后把 One-hot 向量乘以一个矩阵 **C** 得到单词的分布式表示（紫色方框）。令 w_i 为第 i 个词的 One-hot 表示， e_i 为第 i 个词的分布式表示，有：

$$e_i = w_i C \tag{5.75}$$

这里的 **C** 可以被理解为一个查询表，根据 w_i 中为 1 的那一维，在 **C** 中索引到相应的行进行输出（结果是一个行向量）。随后，把得到的 e_0 、 e_1 、 e_2 三个向量级联在一起，经过两层网络，最后通过 Softmax 函数（橙色方框）得到输出：

$$y = \text{Softmax}(h_0 U) \tag{5.76}$$

$$h_0 = \text{Tanh}([e_{i-3}, e_{i-2}, e_{i-1}]H + d) \tag{5.77}$$

输出 y 是词表上的一个分布，通过单词 w_i 可以索引到对应概率 $P(w_i|w_{i-1}, w_{i-2}, w_{i-3})$ 。**U**、**H** 和 **d** 是模型的参数。从结构上看，FNNLM 主要有三层：1) 词的分布式表示层，即把输入的离散的单词变为分布式表示对应的实数向量；2) 隐藏层，即将得到的词的分布式表示进行线性和非线性变换；3) 输出层，根据隐藏层的输出预测单词的概率分布。这三层堆叠在一起构成了整个网络，而且也可以加入从词的分布式表示直接到输出层的连接（红色虚线箭头）。

值得注意的是，在 FNNLM 中，单词已经不再是一个孤立的符号串，而是被表示为一个实数向量。这样，两个单词之间可以通过向量计算某种相似度或距离。这导致相似的单词会具有相似的分布，进而缓解 n -gram 语言模型的问题——明明意思很相近的两个词但是概率估计的结果差异性却很大。

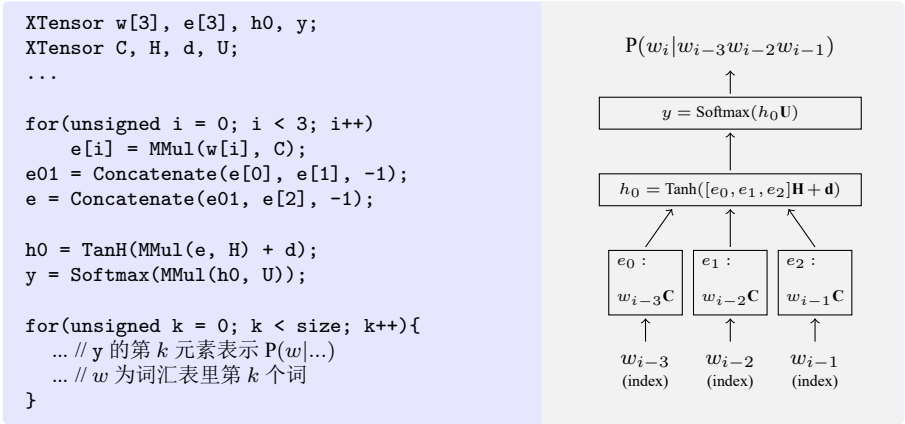


图 5.56: FNNLM 模型代码示例（NiuTensor）

在 FNNLM 中，所有的参数、输入、输出都是连续变量，因此 FNNLM 也是典型的一个连续空间模型。通过使用交叉熵等损失函数，FNNLM 很容易进行优化。比

如，可以使用梯度下降方法对 FNNLM 的模型参数进行训练。

FNNLM 的实现也非常简单，图5.56展示了基于 NiuTensor 的 FNNLM 的部分代码。需要注意的是，在程序实现时，Tanh 函数一般会用 HardTanh 函数代替。因为 Tanh 函数中的指数运算容易导致溢出：

$$\text{Tanh}(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \quad (5.78)$$

而 HardTanh 函数不存在这个问题，因此具有数值计算的稳定性。HardTanh 函数表达式如下：

$$\text{HardTanh}(x) = \begin{cases} -1 & x < -1 \\ x & -1 \leq x \leq 1 \\ 1 & x > 1 \end{cases} \quad (5.79)$$

虽然 FNNLM 模型形式简单，却为处理自然语言提供了一个全新的视角。首先，该模型重新定义了“词是什么”——它并非词典的一项，而是可以用一个连续实数向量进行表示的可计算的“量”。此外，由于 n -gram 不再是离散的符号序列，模型不需要记录 n -gram，所以很好的缓解了上面所提到的数据稀疏问题，模型体积也大大减小。

当然，FNNLM 模型也引发后人的许多思考，比如：神经网络每一层都学到了什么？是词法、句法，还是一些其他知识？如何理解词的分布式表示？等等。在随后的内容中也会看到，随着近几年深度学习和自然语言处理的发展，部分问题已经得到了很好的解答，但是仍有许多问题需要进一步探索。

基于循环神经网络的语言模型

FNNLM 模型固然有效，但是和传统的 n -gram 语言模型一样需要依赖有限上下文假设，也就是 w_i 的生成概率只依赖于之前的 $n-1$ 个单词。很自然的一个想法是引入更大范围的历史信息，这样可以捕捉单词间的长距离依赖。

对于这个问题，可以通过**循环神经网络**（Recurrent Neural Network，或 RNN）进行求解。通过引入循环单元这种特殊的结构，循环神经网络可以对任意长度的历史进行建模，因此在一定程度上解决了传统 n -gram 语言模型有限历史的问题。正是基于这个优点，**循环神经网络语言模型**（RNNLM）应运而生 [204]。

在循环神经网络中，输入和输出都是一个序列，分别记为 $(\mathbf{x}_1, \dots, \mathbf{x}_m)$ 和 $(\mathbf{y}_1, \dots, \mathbf{y}_m)$ 。它们都可以被看作是时序序列，其中每个时刻 t 都对应一个输入 \mathbf{x}_t 和输出 \mathbf{y}_t 。循环神经网络的核心是**循环单元**（RNN Cell），它读入前一个时刻循环单元的输出和当前时刻的输入，生成当前时刻循环单元的输出。图5.57展示了一个简单的循环单元

结构，对于时刻 t ，循环单元的输出被定义为：

$$\mathbf{h}_t = \text{Tanh}(\mathbf{x}_t \mathbf{U} + \mathbf{h}_{t-1} \mathbf{W}) \tag{5.80}$$

其中， \mathbf{h}_t 表示 t 时刻循环单元的输出， \mathbf{h}_{t-1} 表示 $t-1$ 时刻循环单元的输出， \mathbf{U} 和 \mathbf{W} 是模型的参数。可以看出，循环单元的结构其实很简单，只是一个对 \mathbf{h}_{t-1} 和 \mathbf{x}_t 的线性变换再加上一个 Tanh 函数。通过读入上一时刻的输出，当前时刻可以访问以前的历史信息。这个过程可以循环执行，这样就完成了对所有历史信息的建模。 \mathbf{h}_t 可以被看作是序列在 t 时刻的一种表示，也可以被看作是网络的一个隐藏层。进一步， \mathbf{h}_t 可以被送入输出层，得到 t 时刻的输出：

$$\mathbf{y}_t = \text{Softmax}(\mathbf{h}_t \mathbf{V}) \tag{5.81}$$

其中， \mathbf{V} 是输出层的模型参数。

图5.57展示了一个基于循环神经网络的语言模型结构。首先，所有输入的单词会被转换成分布式表示（红色部分），这个过程和 FNNLM 是一样的。之后，该模型堆叠了两层循环神经网络（绿色部分）。最后通过 Softmax 层（紫色部分）得到每个时刻的预测结果 $\mathbf{y}_t = P(w_t|w_1 \dots w_{t-1})$ 。

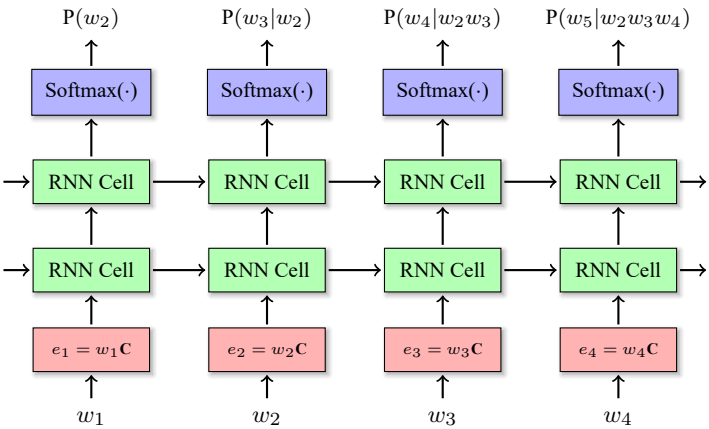


图 5.57: 基于循环神经网络的语言模型结构

RNNLM 体现了一种“记忆”的能力。对于每一个时刻，循环单元都会保留一部分“以前”的信息，并加入“现在”的信息。从这个角度说，RNNLM 本质上是一种记忆模型。在简单的循环单元结构的基础上，也有很多改进工作，如 LSTM、GRU 等模型，这部分内容将会在第六章进行介绍。

基于自注意力机制的语言模型

通过引入记忆历史的能力，RNNLM 缓解了 n -gram 模型中有限上下文的局限性，但依旧存在一些问题。随着序列变长，不同单词之间信息传递路径变长，信息传递的

效率变低。对于长序列，很难通过很多次的循环单元操作保留很长的历史信息。过长的序列还容易引起梯度消失和梯度爆炸问题（详见5.4.4节），增加模型训练的难度。

对于这个问题，研究者又提出了一种新的结构——**自注意力机制**（Self-Attention Mechanism）。自注意力是一种特殊的神经网络结构，它可以对序列上任意两个词的相互作用直接进行建模，这样也就避免了循环神经网络中随着距离变长信息传递步骤增多的缺陷。在自然语言处理领域，自注意力机制被成功地应用在机器翻译任务上，形成了著名的 Transformer 模型 [288]。第六章会系统地介绍自注意力机制和 Transformer 模型。

这里，先简单了解一下基于 Transformer 的语言模型结构（图5.58）。与 FNNLM 和 RNNLM 一样，Transformer 首先对输入单词进行分布式表示，同时加上每个单词的位置编码构成了整个模型的输入（蓝色方框）。之后，利用自注意力机制对输入的向量进行处理（绿色方框）。自注意力的结果会被送入一个前馈神经网络，之后再送给 Softmax 输出层（橙色方框）。

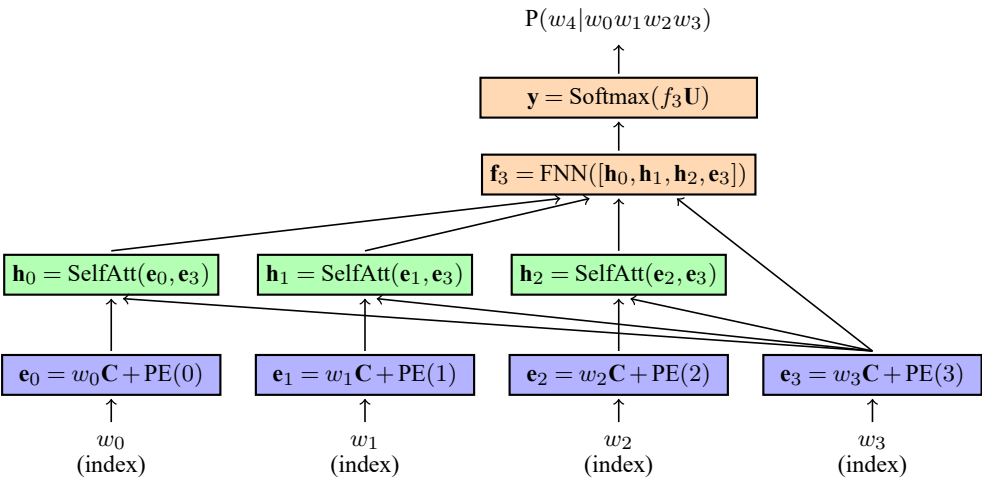


图 5.58: Transformer 语言模型结构

在传统的语言模型中，给定一个单词 w_i ，其他单词对它的影响并没有被显性地建模。而在基于注意力机制的语言模型中，当前需要预测的单词会更加关注与该位置联系较大的单词。具体来说，注意力机制会计算位置 i 与其他任意位置之间的相关度，称为**注意力权重**（Attention Weight），通过这个权重可以更多地使用与 w_i 关联紧密的位置的信息。举个简单的例子，在“我 喜欢 学习 数学”这个句子中，需要预测“数学”这个词，通过注意力机制很可能知道“数学”与“学习”的联系更紧密，所以在预测过程中“学习”所占的权重会更大，预测结果会更加精确。

语言模型的评价

在使用语言模型时，往往需要知道模型的质量。**困惑度**（Perplexity, PPL）是一种衡量语言模型的好坏的指标。对于一个真实的词序列 $w_1 \dots w_m$ ，困惑度被定义为

$$\text{PPL} = P(w_1 \dots w_m)^{-\frac{1}{m}} \tag{5.82}$$

本质上，PPL 反映了语言模型对序列可能性预测能力的一种评估。如果 $w_1 \dots w_m$ 是真实的自然语言，“完美”的模型会得到 $P(w_1 \dots w_m) = 1$ ，它对应了最低的困惑度 $\text{PPL} = 1$ ，这说明模型可以完美地对词序列出现的可能性进行预测。当然，真实的语言模型是无法达到 $\text{PPL} = 1$ 的，比如，在著名的 Penn Treebank（PTB）数据上最好的语言模型的 PPL 值也只能到达 35 左右。可见自然语言处理任务的困难程度。

5.5.2 单词表示模型

在神经语言建模中，每个单词都会被表示为一个实数向量。这对应了一种单词的表示模型。下面就来看看传统的单词表示模型和这种基于实数向量的单词表示模型有何不同。

One-hot 编码

One-hot 编码（也称**独热编码**）是传统的单词表示方法。One-hot 编码把单词表示为词汇表大小的 0-1 向量，其中只有该词所对应的那一项是 1，而其余所有项都是零。举个简单的例子，假如有一个词典，里面包含 10k 个单词，并进行编号。那么每个单词都可以表示为一个 10k 维的 One-hot 向量，它仅在对应编号那个维度为 1，其他维度都为 0，如图5.59所示。

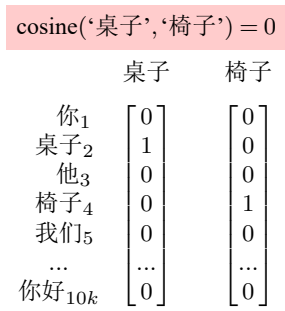


图 5.59: 单词的 One-hot 表示

One-hot 编码的优点是形式简单、易于计算，而且这种表示与词典具有很好的对应关系，因此每个编码都可以进行解释。但是，One-hot 编码把单词都看作是相互正交的向量。这导致所有单词之间没有任何的相关性。只要是不同的单词，在 One-hot 编码下都是完全不同的东西。比如，大家可能会期望诸如“桌子”和“椅子”之类的词具有一些相似性，但是 One-hot 编码把它们看作相似度为 0 的两个单词。

分布式表示

神经语言模型中使用的是一种**分布式表示**（Distributed Representation）。在神经语言模型里，每个单词不再是完全正交的 0-1 向量，而是在多维实数空间中的一个点，具体表现为一个实数向量。很多时候，也会把单词的这种分布式表示叫做**词嵌入**（Word Embedding）。

单词的分布式表示可以被看作是欧式空间中的一个点，因此单词之间的关系也可以通过空间的几何性质进行刻画。如图5.60所示，可以在一个 512 维空间上表示不同的单词。在这种表示下，“桌子”与“椅子”之间是具有一定的联系的。

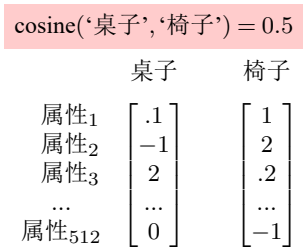


图 5.60: 单词的分布式表示 (词嵌入)

那么，分布式表示中每个维度的含义是什么？可以把每一维度都理解为一种属性，比如一个人的身高、体重等。但是，神经网络模型更多的是把每个维度看作是单词的一种抽象“刻画”，是一种统计意义上的“语义”，而非简单的人工归纳的事物的一个个属性。使用这种连续空间的表示的好处在于，表示的内容（实数向量）可以进行计算和学习，因此可以通过模型训练得到更适用于自然语言处理的单词表示结果。

为了方便理解，看一个简单的例子。假如现在有个“预测下一个单词”的任务：有这样一个句子“屋里要摆放一个 _____”，其中下划线的部分表示需要预测的下一个单词。如果模型在训练数据中看到过类似于“摆放一个桌子”这样的片段，那么就可以很自信的预测出“桌子”。另一方面，很容易知道，实际上与“桌子”相近的单词，如“椅子”，也是可以预测的单词的。但是，“椅子”恰巧没有出现在训练数据中，这时如果用 One-hot 编码来表示单词，显然无法把“椅子”填到下划线处；而如果使用单词的分布式表示，很容易就知道“桌子”与“椅子”是相似的，因此预测“椅子”在一定程度上也是合理的。

■ 实例 5.1 屋里要摆放一个 _____ 预测下个词

屋里要摆放一个 桌子 见过

屋里要摆放一个 椅子 没见过，但是仍然是合理预测

■

关于单词的分布式表示还有一个经典的例子：通过词嵌入可以得到如下关系：“国王”=“女王”-“女人”+“男人”。从这个例子可以看出，词嵌入也具有一些代数性质，比如，词的分布式表示可以通过加、减等代数运算相互转换。图5.61展示了词嵌入在一个二维平面上的投影，不难发现，含义相近的单词分布比较临近。



图 5.61: 分布式表示的可视化

语言模型的词嵌入是通过词嵌入矩阵进行存储的，矩阵中的每一行对应了一个词的分布式表示结果。图5.62展示了一个词嵌入矩阵的实例。

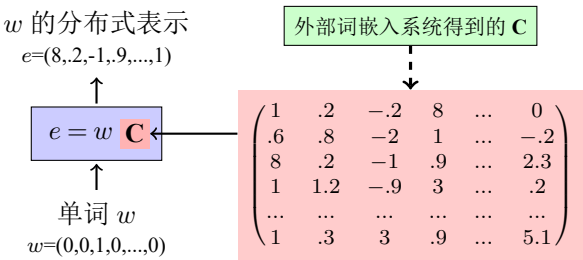


图 5.62: 词嵌入矩阵 \mathbf{C}

通常，有两种方法得到词嵌入矩阵。一种方法是把词嵌入作为语言模型的一部分进行训练，不过由于语言模型往往较复杂，这种方法非常耗时；另一种方法使用更加轻便的外部训练方法，如 word2vec[205]、Glove[228] 等。由于这些方法的效率较高，因此可以使用更大规模的数据得到更好的词嵌入结果。

5.5.3 句子表示模型及预训练

目前，词嵌入已经成为诸多自然语言处理系统的标配，也衍生出很多有趣的研究法方向，甚至有人开玩笑的喊出“embed everything”的口号。但是，冷静地看，词嵌入依旧存在一些问题：每个词都对应唯一的向量表示，那么对于一词多义现象，词义需要通过上下文进行区分，这时使用简单的词嵌入式是无法处理的。有一个著名的例子：

- **实例 5.2** Jobs was the CEO of apple.
He finally ate the apple.

这两句中“apple”的语义显然是不同的，第一句中的上下文“Jobs”和“CEO”可以帮助我们判断“apple”是一个公司名字，而不是水果。但是词嵌入只有一个结果，因此无法区分这两种情况。这个例子给我们一个启发：在一个句子中，不能孤立的看待单词，应同时考虑其上下文的信息。也就是需要一个能包含句子中上下文信息的表示模型。

简单的上下文表示模型

回忆一下神经语言模型的结构，它需要在每个位置预测单词生成的概率。这个概率是由若干层神经网络进行计算后，通过输出层得到的。实际上，在送入输出层之前，系统已经得到了这个位置的一个向量（隐藏层的输出），因此可以把它看作是含有一部分上下文信息的表示结果。以 RNN 为例，图5.63展示了一个由四个词组成的句子，这里使用了一个两层循环神经网络对其进行建模。可以看到，对于第三个位置，RNN 已经积累了从第 1 个单词到第 3 个单词的信息，因此可以看作是单词 1-3（“乔布斯 就职 于”）的一种表示；另一方面，第 4 个单词的词嵌入可以看作是“苹果”自身的表示。这样，可以把第 3 个位置 RNN 的输出和第 4 个位置的词嵌入进行合并，就得到了第 4 个位置上含有上下文信息的表示结果。从另一个角度说，这里得到了“苹果”的一种新的表示，它不仅包含苹果这个词自身的信息，也包含它前文的信息。

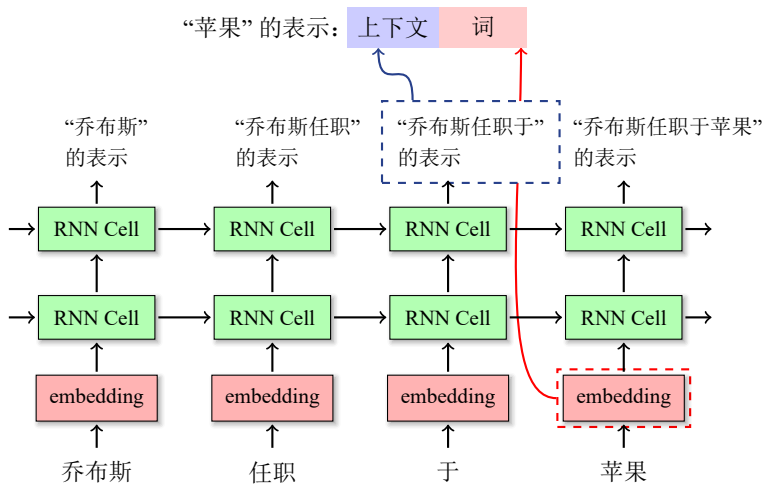


图 5.63: 基于 RNN 的表示模型（词 + 上下文）

在自然语言处理中，**句子表示模型**是指把输入的句子进行分布式表示。不过表示的形式不一定是一个单独的向量。现在广泛使用的句子表示模型可以被描述为：给定一个输入的句子 $\{w_1, \dots, w_m\}$ ，得到一个表示序列 $\{\mathbf{h}_1, \dots, \mathbf{h}_m\}$ ，其中 h_i 是句子在第 i 个位置的表示结果。 $\{\mathbf{h}_1, \dots, \mathbf{h}_m\}$ 就被看作是**句子的表示**，它可以被送入下游模块。比如，在机器翻译任务中，可以用这种模型表示源语言句子，然后通过这种表示结果进行目标语译文的生成；在序列标注（如词性标注）任务中，可以对输入的句子进行表示，然后在这个表示之上构建标签预测模块。很多自然语言处理任务都可以用句子表示模型进行建模，因此句子的表示模型也是应用最广泛的深度学习模型之一。而学习这种表示的过程也被称作**表示学习**（Representation Learning）。

句子表示模型有两种训练方法。最简单的方法是把它作为目标系统中的一个模块进行训练，比如把句子表示模型作为机器翻译系统的一部分。也就是，并不单独训练句子表示模型，而是把它作为一个内部模块放到其他系统中。另一种方法是把

句子表示作为独立的模块，用外部系统进行训练，之后把训练好的表示模型放入目标系统中，再进行微调。这种方法构成了一种新的范式：预训练 + 微调（pre-training + fine-tuning）。图5.64对比了这两种不同的方法。

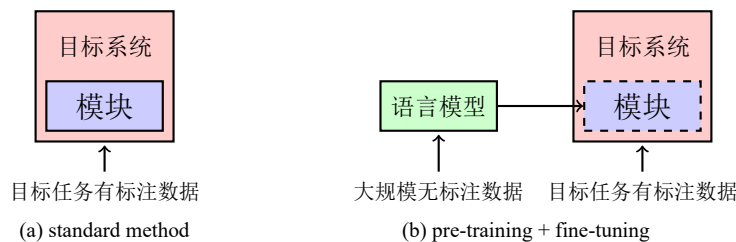


图 5.64: 表示模型的训练方法（与目标任务联合训练 vs 用外部任务预训练）

目前，句子表示模型的预训练方法在多项自然语言处理任务上取得了很好的效果。预训练模型也成为了当今自然语言处理中的热点方向，相关系统也在很多评测任务上刷榜。除了上面提到的简单的神经语言模型的方法，还有各式各样基于预训练的表示模型被提出。接下来，会简单介绍其中比较有代表性的三种模型——ELMO、GPT 和 BERT。

ELMO 模型

ELMO（Embedding from Language Models）掀起了基于语言模型的预训练的热潮 [231]。ELMO 的论文也获得了自然语言处理领域顶级会议 NAACL2018 的最佳论文。

在 ELMO 中，作者认为词的表示应该能够包含丰富的句子结构信息，并且能够对多义词进行建模。而传统的词嵌入（例如 word2vec）是上下文无关的，所以他们利用语言模型来获得一个上下文相关的预训练表示。EMLO 基于双向 LSTM 语言模型⁹，由一个正向语言模型和一个反向语言模型构成，目标函数是最大化这两个方向语言模型的似然（图5.65）。简单来说，ELMO 就是一个预训练好的双向语言模型，对于每个句子都可以生成相应的句子表示结果，这个结果会作为输入的特征被送入下游任务中。比如，ELMO 在问答、文本蕴含、情感分析等多个任务中都表现出非常好的效果。

GPT 模型

GPT（Generative Pre-Training）也是一种基于语言建模的句子表示模型 [236]。该工作的贡献在于利用 Transformer 结构代替了 LSTM。而且该模型基于 Pre-training + Fine-tuning 的框架，预训练的结果将作为下游系统的句子表示模块的参数初始值，因此可以更好的适应目标任务。

⁹LSTM（Long Short-Term Memory），即长短时记忆模型，是一种循环神经网络结构。

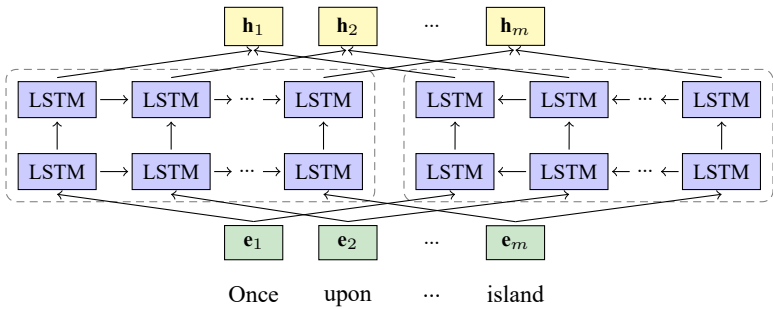


图 5.65: ELMO 模型结构

GPT 模型仍然使用标准的语言建模的思路，即通过前 $n - 1$ 个词预测第 n 个词。但是在网络结构上，GPT 模型使用了 Transformer（图5.66），而且模型参数会在目标任务上进行有监督的微调。与 ELMO 模型的做法不同，GPT 不需要对目标任务构建新的模型结构，而是直接在 Transformer 语言表示模型的最后一层加上 Softmax 层作为任务的输出层。实验结果证明，GPT 模型的性能较 ELMO 模型更为优越，在 12 个 NLP 任务中取得了 9 个任务当时最好的结果。

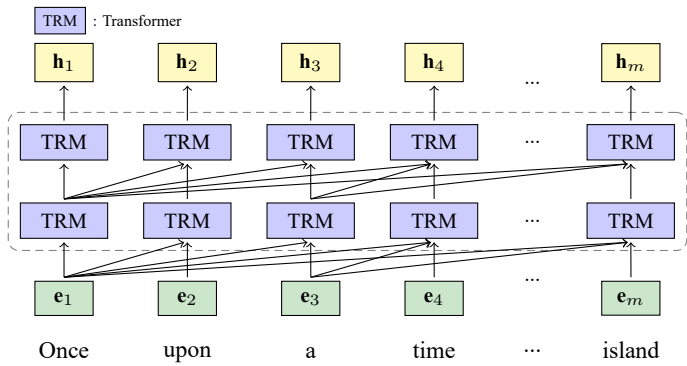


图 5.66: GPT 模型结构

BERT 模型

BERT（Bidirectional Encoder Representations from Transformers）是另一个非常有代表性的基于预训练的句子表示模型 [56]。某种意义上，BERT 把基于预训练的句子表示模型推向了新的高潮。BERT 的论文也获得了 NAACL2019 最佳论文奖。

与传统语言模型的训练目标不同，BERT 不使用预测下一个词作为训练目标，而是提出了两个新的任务。

- 第一个任务被称为 Masked LM。在输入的词序列中随机地挡住一定量的词（比如 15% 的单词被挡住），然后让模型去预测挡住的这些词。这个过程有些类似于英语考试中的完形填空。这么做的好处是，模型能够从多个方向去预测这些被遮罩住的词（图5.67），而不是像传统语言模型一样从单个方向预测（自左

向右或者自右向左)。Masked LM 的思想也影响了很多预训练模型的设计。

- 第二个任务是预测下一个句子。当选择句子 a 与 b 作为预训练样本时，b 有一半几率可能是 a 的下一句，也有一半几率来自语料库的随机句子，从而可以更好地学习句子之间的相关性。

BERT 的训练目标就是最小化这两个任务上的损失函数。实验结果证明，BERT 模型的性能十分强劲，在 11 个 NLP 任务上取得了当时最好的性能。

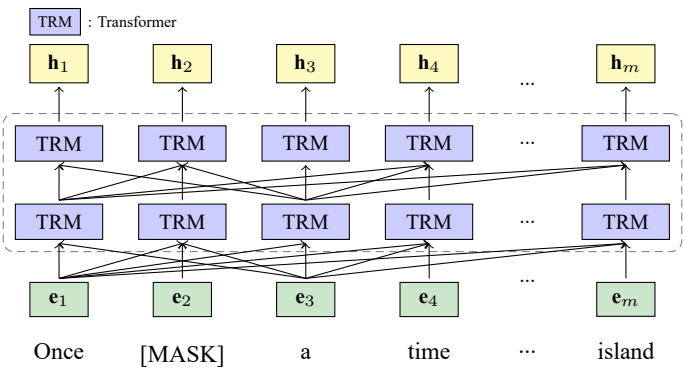


图 5.67: BERT 模型结构

为什么要预训练?

基于预训练的句子表示模型确实给自然语言处理带来了新的思路。相比传统的基于目标任务的建模和训练方法，预训练有如下优势：

- 目标任务依赖有指导训练，但是带标注的数据往往十分有限。预训练模型并不依赖带标注数据，因此可以从更大规模的无标注数据（比如训练语言模型中使用的纯文本数据）学习知识，这使得预训练模型可以捕捉更加通用的语言规律，甚至有时候会具有一些“常识”¹⁰；
- 由于预训练并不依赖具体的目标任务，因此不会对目标任务产生过拟合，模型具有较好的泛化能力；
- 从优化的角度，预训练相当于将模型“初始化”到参数空间中优质解密度更高的区域（图5.68），下游任务更容易通过微调找到更好的参数（面向具体任务）。此外，也有实验表明预训练可以让模型参数落到损失函数曲面比较光滑的区域，这样能够避免下游任务在凸凹不平的损失函数曲面上陷入局部最优。

¹⁰关于自然语言处理系统是否具有常识也存在不同的观点。从一些任务上的实验结果上看，很多模型确实能够进行一些常识性推理。但是，当问题的难度增加时，绝大多数模型还无法做到人类的水平。

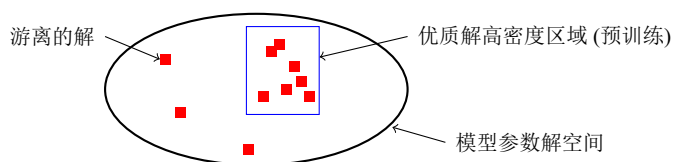



图 5.68: 模型的参数空间

5.6 小结及深入阅读

神经网络为解决自然语言处理问题提供了全新的思路。而所谓深度学习也是建立在多层神经网络结构之上的一系列模型和方法。本章从神经网络的基本概念到其在语言建模中的应用进行了概述。由于篇幅所限, 这里无法覆盖所有神经网络和深度学习的相关内容, 感兴趣的读者可以进一步阅读《Neural Network Methods in Natural Language Processing》[88] 和《Deep Learning》[90]。此外, 也有很多研究方向值得关注:

- 端到端学习是神经网络方法的特点之一。这样, 系统开发者不需要设计输入和输出的隐含结构, 甚至连特征工程都不再需要。但是, 另一方面, 由于这种端到端学习完全由神经网络自行完成, 整个学习过程没有人的先验知识做指导, 导致学习的结构和参数很难进行解释。针对这个问题也有很多研究者进行**可解释机器学习** (Explainable Machine Learning) 的研究 [97][154]。对于自然语言处理, 方法的可解释性是十分必要的。从另一个角度说, 如何使用先验知识改善端到端学习也是很多人关注的方向 [4][345], 比如, 如何使用句法知识改善自然语言处理模型 [356][31][269]。
- 词嵌入是自然语言处理近些年的重要进展。所谓“嵌入”是一类方法, 理论上, 把一个事物进行分布式表示的过程都可以被看作是广义上的“嵌入”。基于这种思想的表示学习也成为了自然语言处理中的前沿方法。比如, 如何对树结构, 甚至图结构进行分布式表示 [232][230] 成为了分析自然语言的重要方法。此外, 除了语言建模, 还有很多方式可以进行词嵌入的学习, 比如, SENNA[48]、word2vec[203][205]、Glove[228]、CoVe[200] 等。
- 预训练是表示学习的重要产物。预训练已经在图像处理等领域得到应用。在自然语言处理中, 以 BERT 为代表的预训练模型席卷了很多自然语言处理任务, 在阅读理解等比赛 (如 Stanford Question Answering) 中已经成为了所有参赛系统的标配。除了 ELMO、GPT、BERT, 还有很多优秀的预训练模型, 包括 GPT-2[237]、XLM[49]、MASS[262]、XLNet[331], 等等。但是, 预训练往往依赖大规模的数据和并行运算设备, 这使得很多普通研究者对训练这样的模型望而却步。不过, 也有一些研究轻量的预训练方法, 也受到了很多关注, 例如 ALBERT[161]。



6. 神经机器翻译模型

神经机器翻译（Neural Machine Translation）是机器翻译的前沿方法。近几年，随着深度学习技术的发展和在各领域中的深入应用，基于端到端表示学习的方法正在改变着我们处理自然语言的方式，神经机器翻译在这种趋势下应运而生。一方面，神经机器翻译仍然延续着统计建模和基于数据驱动的思想，因此在基本问题的定义上与前人的研究是一致的；另一方面，神经机器翻译脱离了统计机器翻译中对隐含翻译结构的假设，同时使用分布式表示来对文字序列进行建模，这使得它可以从一个全新的视角看待翻译问题。现在，神经机器翻译已经成为了机器翻译研究及应用的热点，译文质量得到了巨大的提升。本章将对神经机器翻译的基础模型和方法进行介绍。

6.1 神经机器翻译的发展简史

纵观机器翻译的发展历程，神经机器翻译诞生很晚。无论是早期的基于规则的方法，还是逐渐发展起来的基于实例的方法，再到上世纪末的统计方法，每次机器翻译框架级的创新都需要很长时间的酝酿，而技术走向成熟甚至需要更长的时间。但是，神经机器翻译的出现和后来的发展速度多少有些令人“出人意料”。神经机器翻译的概念出现在 2013-2014 年间，当时机器翻译领域的主流方法仍然是统计机器翻译。虽然那个时期深度学习已经在图像、语音等领域取得令人瞩目的效果，但是对于自然语言处理来说深度学习仍然不是主流。这也导致当时的研究者对神经机器翻译这种方法还有一些排斥。

不过，有人也意识到了神经机器翻译在表示学习等方面的优势。特别是，以

Yoshua Bengio 团队为代表的研究力量对包括机器翻译在内的序列到序列问题进行了广泛而深入的研究，注意力机制等新的模型不断被推出。这使得神经机器翻译系统在翻译品质上逐渐体现出优势，甚至超越了当时的统计机器翻译系统。正当大家在讨论神经机器翻译是否能取代统计机器翻译成为下一代机器翻译范式的时候，谷歌、百度等企业推出以神经机器翻译技术为内核的在线机器翻译服务，在很多场景下的翻译品质显著超越了当时最好的统计机器翻译系统。这也引发了学术界和产业界对神经机器翻译的讨论。随着关注度的不断升高，神经机器翻译的研究吸引了更多的科研机构和企业投入，神经机器翻译系统的翻译品质得到进一步提升。

在短短 5-6 年间，神经机器翻译从一个新生的概念已经成长为机器翻译领域的最前沿技术之一，在各种机器翻译评测和应用中呈全面替代统计机器翻译之势。比如，从近几年 WMT、CCMT 等评测的结果来看，神经机器翻译已经处于绝对的统治地位，在不同语种和领域的翻译任务中，成为各参赛系统的标配。此外，从 ACL 等自然语言处理顶级会议的发表论文看，神经机器翻译是毫无疑问的焦点，在论文数量上呈明显的增长趋势，这也体现了学术界对该方法的热情。至今，无论是国外的著名企业，如谷歌、微软、脸书，还是国内的团队，如百度、腾讯、阿里巴巴、有道、搜狗、小牛翻译，都推出了自己研发的神经机器翻译系统，整个研究和产业生态欣欣向荣。图6.1展示了包含神经机器翻译在内的机器翻译发展简史。

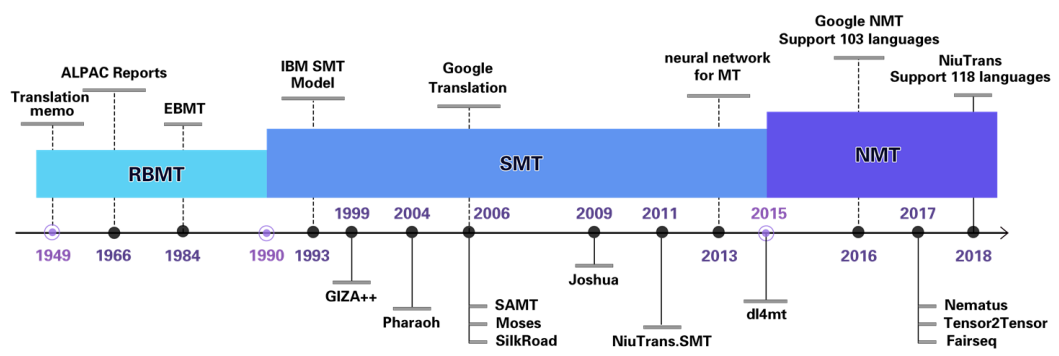


图 6.1: 机器翻译发展简史

神经机器翻译的迅速崛起确实让所有人都有些措手不及，甚至有一种一觉醒来天翻地覆的感觉。也有人评价，神经机器翻译的出现给整个机器翻译领域带来了前所未有的发展机遇。不过，客观地看，机器翻译达到今天这样的状态也是一种历史必然，其中有几方面原因：

- 自上世纪末所发展起来的基于数据驱动的方法为神经机器翻译提供了很好的基础。本质上，神经机器翻译仍然是一种基于统计建模的数据驱动的方法，因此无论是对问题的基本建模方式，还是训练统计模型所使用到的带标注数据，都可以复用机器翻译领域以前的研究成果。特别是机器翻译长期的发展已经积累了大量的双语、单语数据，这些数据在统计机器翻译时代就发挥了很大作用。随着时间的推移，数据规模和质量又得到进一步提升，包括一些评测基准、任

务设置都已经非常完备，研究者可以直接在数据条件全部具备的情况下开展神经机器翻译的研究工作，这些都省去了大量的时间成本。从这个角度说，神经机器翻译是站在巨人的肩膀上才发展起来的。

- 深度学习经过长时间的酝酿终于爆发，为机器翻译等自然语言处理任务提供了新的思路和技术手段。神经机器翻译的不断壮大伴随着深度学习技术的发展。在深度学习的视角下，语言文字可以被表示成抽象的实数向量。这种文字的表达方法可以被自动学习，为机器翻译建模提供了更大的灵活性。相对于神经机器翻译，深度学习的发展更加曲折。虽然深度学习经过了漫长的起伏过程，但是神经机器翻译恰好出现在深度学习逐渐走向成熟的阶段。反过来说，受到深度学习及相关技术空前发展的影响，自然语言处理的范式也发生了变化，神经机器翻译的出现只是这种趋势下的一种必然。
- 此外，计算机算力的提升也为神经机器翻译提供了很好的支撑。与很多神经网络方法一样，神经机器翻译也依赖大量的基于浮点数的矩阵运算。在 2000 年前，大规模的矩阵运算仍然依赖非常昂贵的 CPU 集群系统，但是随着 GPU 等相关技术的发展，在相对低成本的设备上已经可以完成非常复杂的浮点并行运算。这使得包括神经机器翻译在内的很多基于深度学习的系统可以进行大规模实验，随着实验周期的缩短，相关研究和系统的迭代周期大大缩短。实际上，计算机硬件运算能力一直是稳定提升的，神经机器翻译只是受益于运算能力的阶段性突破。
- 还有，翻译需求的不断增加也为机器翻译技术提供了新的机会。在近几年，无论是翻译品质，还是翻译语种数量，甚至不同的翻译场景，都对机器翻译有了更高的要求。人们迫切需要一种品质更高、翻译效果稳定的机器翻译方法，神经机器翻译恰好满足了这些要求。当然，应用端需求的增加也会反推机器翻译技术的发展，二者相互促进。

至今，神经机器翻译已经成为带有时代特征的标志性方法。当然，机器翻译的发展也远没有达到终点。下面将介绍神经机器翻译的起源和优势，以便读者在正式了解神经机器翻译的技术方法前对其现状有一个充分的认识。

6.1.1 神经机器翻译的起源

从广义上讲，神经机器翻译是一种基于人工神经网络的方法，它把翻译过程描述为可以用人工神经网络表示的函数。所有的训练和推断都在这些函数上进行。由于神经机器翻译中的神经网络可以用连续可微函数表示，因此这类方法也可以用基于梯度的方法进行优化，相关技术非常成熟。更为重要的是，在神经网络的设计中，研究者引入了**分布式表示**（Distributed Representation）的概念，这也是近些年自然语言处理领域的重要成果之一。传统统计机器翻译仍然把词序列看作离散空间里的由多个特征函数描述的点，类似于 n -gram 语言模型，这类模型对数据稀疏问题非常敏

感。此外，人工设计特征也在一定程度上限制了模型对问题的表示能力。神经机器翻译把文字序列表示为实数向量，一方面避免了特征工程繁重的工作，另一方面使得系统可以对文字序列的“表示”进行学习。可以说，神经机器翻译的成功很大程度上源自“表示学习”这种自然语言处理的新范式的出现。在表示学习的基础上，注意力机制、深度神经网络等技术都被应用于神经机器翻译，使其得以进一步发展。

虽然神经机器翻译中大量的使用了人工神经网络方法，但是它并不是最早在机器翻译中使用人工神经网络的框架。实际上，人工神经网络在机器翻译中应用的历史要远早于现在的神经机器翻译。在统计机器翻译时代，也有很多研究者利用人工神经网络进行机器翻译系统模块的构建 [57, 247]，比如，Jacob Devlin 等人就成功地在统计机器翻译系统中使用了基于神经网络的联合表示模型，取得了令人振奋的结果，这项工作也获得了 ACL2014 的最佳论文奖（Best Paper Award）。

不过，以上这些工作大多都是在系统的局部模块中使用人工神经网络和深度学习方法。与之不同的是，神经机器翻译是用人工神经网络完成整个翻译过程的建模，这样做的一个好处是，整个系统可以进行端到端学习，无需引入对任何翻译的隐含结构假设。这种利用端到端学习对机器翻译进行神经网络建模的方式也就成为了现在大家所熟知的神经机器翻译。这里简单列出部分代表性的工作：

- 早在 2013 年，牛津大学的 Nal Kalchbrenner 和 Phil Blunsom 提出了一个基于编码器-解码器结构的新模型 [135]。该模型用卷积神经网络（CNN）将源语言编码成实数向量，之后用循环神经网络（RNN）将连续向量转换成目标语言。这使得模型不需要进行词对齐、特征提取等工作，就能够自动学习源语言的信息。这也是一种端到端学习的方法。不过，这项工作的实现较复杂，而且方法存在梯度消失/爆炸等问题 [13, 111]，因此并没有成为后来神经机器翻译的基础框架。
- 2014 年，谷歌的 Ilya Sutskever 等人提出了序列到序列（seq2seq）学习的方法，同时将长短记忆结构（LSTM）引入到神经机器翻译中，这个方法解决了梯度爆炸/消失的问题，并且通过遗忘门的设计让网络选择性地记忆信息，缓解了序列中长距离依赖的问题 [276]。但是该模型在进行编码的过程中，将不同长度的源语言句子压缩成了一个固定长度的向量，句子越长，损失的信息越多，同时该模型无法对输入和输出序列之间的对齐进行建模，因此并不能有效的保证翻译质量。
- 同年 Dzmitry Bahdanau 等人首次将注意力机制（Attention Mechanism）应用到机器翻译领域，在机器翻译任务上对翻译和局部翻译单元之间的对应关系同时建模 [7]。Bahdanau 等人工作的意义在于，使用了更加有效的模型来表示源语言的信息，同时使用注意力机制对两种语言不同部分之间的相互联系进行建模。这种方法可以有效地处理长句子的翻译，而且注意力的中间结果具有一定的可解释性¹。然而相比于前人的神经机器翻译模型，注意力模型也引入了额外的

¹比如，目标语言和源语言句子不同单词之间的注意力强度能够在一定程度上反应单词之间的互译

成本，计算量较大。

- 2016 年谷歌发布了基于多层循环神经网络方法的 GNMT 系统。该系统集成了当时的神经机器翻译技术，并进行了诸多的改进。它的性能显著优于基于短语的机器翻译系统 [312]，引起了研究者的广泛关注。在之后不到一年的时间里，Facebook 采用卷积神经网络（CNN）研发了新的神经机器翻译系统 [84]，实现了比基于循环神经网络（RNN）系统更高的翻译水平，并大幅提升翻译速度。
- 2017 年，谷歌的 Ashish Vaswani 等人提出了新的翻译模型 Transformer。其完全抛弃了 CNN、RNN 等结构，仅仅通过自注意力机制（Self-Attention）和前向神经网络，不需要使用序列对齐的循环框架就展示出强大的性能，并且巧妙的解决了翻译中长距离依赖问题 [288]。Transformer 是第一个完全基于注意力机制搭建的模型，不仅训练速度更快，在翻译任务上也获得了更好的结果，一跃成为目前最主流的神经机器翻译框架。

神经机器翻译的工作远不止以上这些内容，实际上全面介绍所有神经机器翻译的方法也是非常具有挑战的工作。感兴趣的读者可以参考一篇关于神经机器翻译的综述文章——Neural Machine Translation: A Review[265]。本章会对神经机器翻译的典型方法进行细致的介绍。

6.1.2 神经机器翻译的品质

图6.2是一个真实的机器翻译的例子。其中译文 1 是统计机器翻译系统的结果，译文 2 是神经机器翻译系统的结果。为了保证公平性，两个系统使用完全相同的数据进行训练。

原文：During Soviet times, if a city’s population topped one million, it would become eligible for its own metro. Planners wanted to brighten the lives of everyday Soviet citizens, and saw the metros, with their tens of thousands of daily passengers, as a singular opportunity to do so. In 1977, Tashkent, the capital of Uzbekistan, became the seventh Soviet city to have a metro built. Grand themes celebrating the history of Uzbekistan and the Soviet Union were brought to life, as art was commissioned and designers set to work. The stations reflected different themes, some with domed ceilings and painted tiles reminiscent of Uzbekistan’s Silk Road mosques, while others ...

译文 1: 在苏联时代，如果一个城市的人口突破一百万，这将成为合资格为自己的地铁。规划者想去照亮每天的苏联公民的生命，看到地铁，与他们的数十每天数千乘客，作为一个独特的机会来这样做。1977 年，塔什干，乌兹别克斯坦的首都，成了苏联第七城市建有地铁。宏大主题，庆祝乌兹别克斯坦和苏联的历史被带到生活，因为艺术是委托和设计师开始工作。车站反映了不同的主题，有的圆顶天花板和绘瓷砖让人想起乌兹别克斯坦是丝绸之路的清真寺，而另一些则装饰着...

译文 2: 在苏联时期, 如果一个城市的人口超过一百万, 它就有资格拥有自己的地铁。规划者想要照亮日常苏联公民的生活, 并把拥有数万名每日乘客的地铁看作是这样做的一个绝佳机会。1977 年, 乌兹别克斯坦首都塔什干成为苏联第七个修建地铁的城市。随着艺术的委托和设计师们的工作, 乌兹别克斯坦和苏联历史的宏伟主题被赋予了生命力。这些电台反映了不同的主题, 有的有穹顶和彩砖, 让人想起乌兹别克斯坦的丝绸之路清真寺, 有的则用...

图 6.2: 机器翻译实例对比

可以明显地看到译文 2 更加通顺, 意思的表达更加准确, 翻译质量明显高于译文 1。这个例子基本反应出统计机器翻译和神经机器翻译的差异性。当然, 这里并不是要讨论统计机器翻译和神经机器翻译孰优孰劣。但是, 很多场景中都不难发现神经机器翻译可以生成非常流畅的译文, 易于人工阅读和修改。

在很多量化的评价中也可以看到神经机器翻译的优势。回忆一下第一章提到的机器翻译质量的自动评估指标中, 使用最广泛的一种指标是 BLEU。在统计机器翻译时代, 在由美国国家标准和科技机构 (NIST) 举办的汉英机器翻译评测中 (比如汉英 MT08 数据集), 基于统计方法的翻译系统能够得到 30% 以上的 BLEU 值已经是当时最顶尖的结果了。而现在的神经机器翻译系统, 则可以轻松的将 BLEU 提高至 45% 以上。

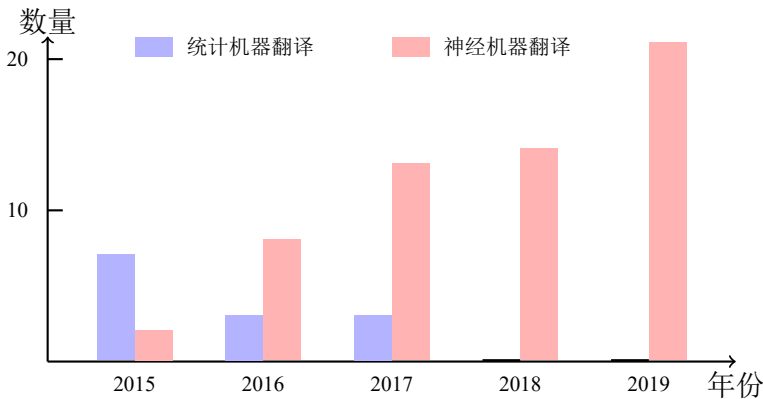


图 6.3: WMT 冠军系统的数量

同样, 在机器翻译领域中最具影响力的评测比赛 WMT (Workshop of Machine Translation) 中, 使用统计机器翻译方法的参赛系统也在逐年减少。而现在获得比赛冠军的系统中几乎没有只使用纯统计机器翻译模型的系统²。图6.3展示了近年来 WMT 比赛冠军系统中神经机器翻译系统的占比, 可见神经机器翻译系统的占比在逐年提高。

除了上面例子中展示的流畅度和准确度外, 神经机器翻译在其他评价指标上的

²但是, 仍然有大量的统计机器翻译和神经机器翻译融合的方法。比如, 在无指导机器翻译中, 统计机器翻译仍然被作为初始模型。

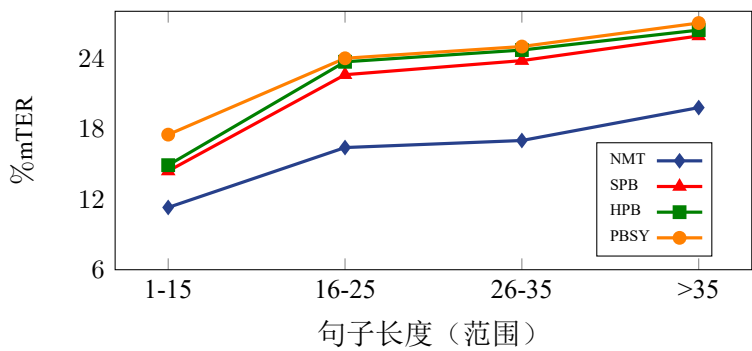


图 6.4: 不同系统在不同长度句子上的 mTER 分值 (得分越低越好)

表现也全面超越统计机器翻译 [16]。比如，在 IWSLT 2015 英语-德语任务中，与三个最先进的统计机器翻译系统（PBSY、HPB、SPB）相比，神经机器翻译系统的 mTER 得分在不同长度句子上都有明显的下降，如图6.4³。其次，神经机器翻译的单词形态错误率和单词词义错误率都远低于统计机器翻译系统（表6.1）。

表 6.1: NMT 与 SMT 系统的译文错误率 [16]

system	word	lemma	%Δ
PBSY	27.1	22.5	-16.9
HPB	28.7	23.5	-18.4
SPB	28.3	23.2	-18.0
NMT	21.7*	18.7*	-13.7

更振奋人心的是，神经机器翻译在某些任务上的结果已经相当惊艳，比如在汉英新闻翻译任务中，神经机器翻译就取得了至少和专业翻译人员相媲美的效果 [99]。在该任务中，神经机器系统（Combo-4、Combo-5 和 Combo-6）的人工评价得分与 Reference-HT（专业翻译人员翻译）的得分无显著差别，且远超 Reference-WMT（WMT 的参考译文，也是由人类翻译）得分（表6.2）。

表 6.2: 不同机器翻译系统人类评价结果 [99]

#	Ave% (平均原始分数)	System
1	69.0	Combo-6
	68.5	Reference-HT
	68.9	Combo-5
	68.6	Combo-4
2	62.1	Reference-WMT

在最近两年，神经机器翻译的发展更加迅速，新的模型、方法层出不穷。表6.3给

³mTER 是一种错误率度量，值越低表明译文越好。

出了 2019 年一些主流的神经机器翻译模型的对比 [299]。可以看到，相比 2017 年，2018-2019 年中机器翻译仍然有明显的进步。

表 6.3: WMT14 英德数据集上不同神经机器翻译系统的表现 [299]

模型	作者	年份	BLEU
ConvS2S	Gehring 等	2017	25.2
Transformer-Base	Vaswani 等	2017	27.3
Transformer-Big	Vaswani 等	2017	28.4
RNMT+	Chen 等	2018	28.5
Layer-Wise Coordination	Xu 等	2018	29
Transformer-RPR	Shaw 等	2018	29.2
Transformer-DLCL	Wang 等	2019	29.3

6.1.3 神经机器翻译的优势

表 6.4: 统计机器翻译 vs 神经机器翻译

统计机器翻译	神经机器翻译
基于离散空间的表示模型	基于连续空间的表示模型
NLP 问题的隐含结构假设	无隐含结构假设，端到端学习
特征工程为主	无显性特征，但需要设计网络
特征、规则的存储耗资源	模型存储相对小，但计算慢

既然神经机器翻译如此强大，它的优势在哪里呢？为了回答这个问题，表6.4给出了神经机器翻译与统计机器翻译的简单对比。具体来说，神经机器翻译有如下特点：

（一）分布式连续空间表示模型，能捕获更多隐藏信息

神经机器翻译与统计机器翻译最大的区别在于对语言文字串的表达方法上。在统计机器翻译中，所有词串本质上都是由更小的词串（短语、规则）组合而成，也就是统计机器翻译模型利用了词串之间的组合性来表示更大的词串。统计机器翻译使用多个特征描述翻译结果，但是其仍然对应着离散的字符串的组合，因此可以把模型对问题的表示空间看做是由一个离散结构组成的集合。在神经机器翻译中，词串的表达已经被神经网络转化为多维实数向量，而且也不依赖任何的可组合性假设等其他假设来刻画离散的语言结构，从这个角度说，所有的词串分别对应了一个连续空间上的点（比如，对应 n 维实数空间中一个点）。这样，模型可以更好地进行优化，而且对未见样本有更好的泛化能力。此外，基于连续可微函数的机器学习算法已经相对完备，可以很容易的对问题进行建模和优化。

（二）不需要特征工程，特征学习更加全面

经典的统计机器翻译可以通过判别式模型引入任意特征，不过这些特征需要人工设计，因此这个过程也被称为**特征工程**（Feature Engineering）。特征工程依赖大量的人工，特别是对不同语种、不同场景的翻译任务，所采用的特征可能不尽相同，这也使得设计有效的特征成为了统计机器翻译时代最主要的工作之一。但是，由于人类自身的思维和认知水平的限制，人工设计的特征可能不全面，甚至会遗漏一些重要的翻译现象。神经机器翻译并不依赖任何人工特征的设计，或者说它的特征都隐含在分布式表示中。这些“特征”都是自动学习得到的，因此神经机器翻译并不会受到人工思维的限制，学习到的特征对问题描述更加全面。

（三）不含隐含结构假设，端到端学习对问题建模更加直接

传统的自然语言处理任务会对问题进行隐含结构假设。比如，进行翻译时，统计机器翻译会假设翻译过程由短语的拼装完成。这些假设可以大大化简问题的复杂度，但是另一方面也带来了各种各样的约束条件。错误的隐含假设往往会导致建模错误。神经机器翻译是一种端到端模型，它并不依赖任何隐含结构假设。这样，模型并不会受到错误的隐含结构的引导。从某种意义上说，端到端学习可以让模型更加“自由”地进行学习，因此往往可以学到很多传统认知上不容易理解或者不容易观测到的现象。

（四）模型结构统一，存储消耗更小

统计机器翻译系统依赖于很多模块，比如词对齐、短语（规则）表、目标语言模型等等，因为所有的信息（如 n -gram）都是离散化表示的，因此模型需要消耗大量的存储资源。同时，由于系统模块较多，开发的难度也较大。神经机器翻译的模型都是用神经网络进行表示，模型参数大多是实数矩阵，因此存储资源的消耗很小。而且神经网络可以作为一个整体进行开发和调试，系统搭建的代价相对较低。实际上，由于模型体积小，神经机器翻译也非常合适于离线小设备上的翻译任务。

当然，神经机器翻译也并不完美，很多问题有待解决。首先，神经机器翻译需要大规模浮点运算的支持，模型的推断速度较低。为了获得优质的翻译结果，往往需要大量 GPU 设备的支持，计算资源成本很高；其次，由于缺乏人类的先验知识对翻译过程的指导，神经机器翻译的运行过程缺乏可解释性，系统的可干预性也较差；此外，虽然脱离了繁重的特征工程，神经机器翻译仍然需要人工设计网络结构，包括在模型的各种超参的设置、训练策略的选择等方面，仍然需要大量人工参与。这也导致很多实验结果不容易重现。显然，完全不依赖人工进行机器翻译还很遥远。不过，随着研究者的不断攻关，很多问题也得到了解决。

6.2 编码器-解码器框架

说到神经机器翻译就不得不提**编码器-解码器模型**（Encoder-Decoder Paradigm），或**编码器-解码器框架**。本质上，编码器-解码器模型是描述输入-输出之间关系的一种方式。编码器-解码器这个概念在日常生活中并不少见。例如在电视系统上为了便于

视频的传播，会使用各种编码器将视频编码成数字信号，在客户端，相应的解码器组件会把收到的数字信号解码为视频。另外一个更贴近生活的例子是电话，它通过对声波和电信号进行相互转换，达到传递声音的目的。这种“先编码，再解码”的思想被应用到密码学、信息论等多个领域。

不难看出，机器翻译问题也完美的贴合编码器-解码器结构的特点。可以将源语言编码为类似信息传输中的数字信号，然后利用解码器对其进行转换，生成目标语言。下面就来看一下神经机器翻译是如何在编码器-解码器框架下进行工作的。

6.2.1 框架结构

编码器-解码器框架是一种典型的基于“表示”的模型。编码器的作用是将输入的文字序列通过某种转换变为一种新的“表示”形式，这种“表示”包含了输入序列的所有信息。之后，解码器把这种“表示”重新转换为输出的文字序列。这其中的一个核心问题是表示学习，即：如何定义对输入文字序列的表示形式，并自动学习这种表示，同时应用它生成输出序列。一般来说，不同的表示学习方法可以对应不同的机器翻译模型，比如，在最初的神经机器翻译模型中，源语言句子都被表示为一个独立的向量，这时表示结果是静态的；而在注意力机制中，源语言句子的表示是动态的，也就是翻译目标语的每个单词时都会使用不同的表示结果。

图6.5是一个应用编码器-解码器结构来解决机器翻译问题的简单实例。给定一个中文句子“我 对 你 感 到 满 意”，编码器会将这句话编码成一个实数向量 (0.2, -1, 6, 5, .7, -2)，这个向量就是源语言句子的“表示”结果。虽然有些不可思议，但是神经机器翻译模型把这个向量等同于输入序列。向量中的数字并没有实际的意义，然而解码器却能从中提取到源语句子中所包含的信息。也有研究者把向量的每一个维度看作是一个“特征”，这样源语言句子就被表示成多个“特征”的联合，而且这些特征可以被自动学习。有了这样的源语言句子的“表示”，解码器可以把这个实数向量作为输入，然后逐词生成目标语句子“I am satisfied with you”。

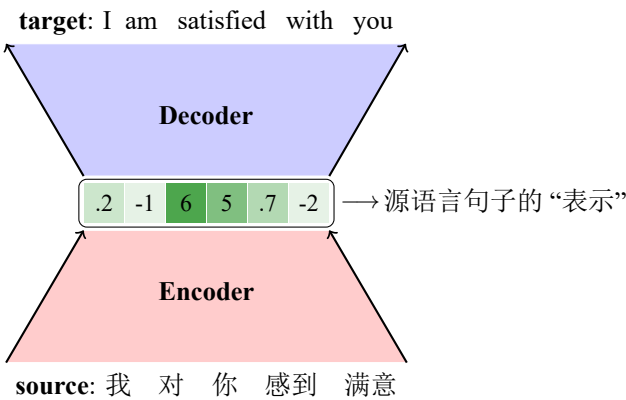


图 6.5: Encoder-Decoder 过程

在源语言句子的表示形式确定之后，需要设计相应的编码器和解码器结构。在

大多数情况下，神经机器翻译系统中的编码器由词嵌入层和中间网络层组成。当输入一串单词序列时，词嵌入层会将以一维空间表示的离散的单词映射到连续的多维表示空间，这个过程也被称为词嵌入。之后中间层会对词嵌入向量进行更深层的抽象，得到输入单词序列的中间表示。中间层的实现方式有很多，比如：循环神经网络、卷积神经网络、Transformer 等模型都是常用的结构。解码器的结构基本上和编码器是一致的，只不过多了输出层，用于输出每个目标语位置的单词生成概率。

现在，编码器-解码器框架已经成为了神经机器翻译系统的标准架构。当然，也有一些研究工作在探索编码器-解码器框架之外的结构 [171]，但是还没有太多颠覆性的进展。因此，本章仍然以编码器-解码器框架为基础对相关模型和方法进行介绍。

6.2.2 表示学习

编码器-解码器框架的创新之处在于，将传统基于符号的离散型知识转化为分布式的连续型知识。比如，对于一个句子，它可以由离散的符号所构成的文法规则来生成，也可以直接被表示为一个实数向量记录句子的各个“属性”。这种分布式的实数向量可以不依赖任何离散化的符号系统，简单来说，它就是一个函数，把输入的词串转化为实数向量。更为重要的是，这种分布式表示可以被自动学习。或者从某种意义上说，编码器-解码器框架的作用之一就是学习输入序列的表示。表示结果学习的好与坏很大程度上会影响神经机器翻译系统的性能。

图6.6形象地对比了统计机器翻译和神经机器翻译的表示模型的区别。传统的统计机器翻译（a）通过短语或者规则组合来获得更大的翻译片段，直至覆盖整个句子。这本质上是在一个离散的结构空间中不断组合的过程。神经机器翻译（b）与之不同，它并没有所谓的“组合”的过程，整个句子的处理是直接在连续空间上进行计算得到的。这二者的区别也体现了符号系统与神经网络系统的区别。前者更适合处理离散化的结构表示，后者更适合处理连续化的表示。

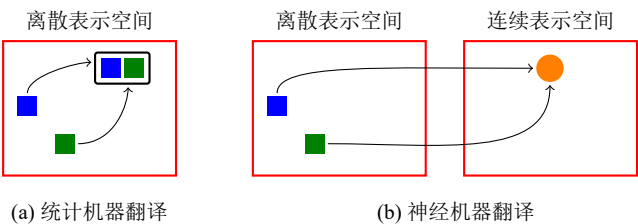


图 6.6: 统计机器翻译和神经机器翻译的表示空间

实际上，编码器-解码器模型也并不是表示学习实现的唯一途径。比如，在第五章提到的神经语言模型实际上也是一种有效的学习句子表示的方法，它所衍生出的预训练模型可以从大规模单语数据上学习句子的表示形式。这种学习会比使用少量的双语数据进行编码端和解码端的学习更加充分。相比机器翻译任务，语言模型相

当于一个编码器的学习⁴，可以无缝嵌入到神经机器翻译模型中。不过，值得注意的是，机器翻译的目的是解决双语字符串之间的映射问题，因此它所使用的句子表示是为了更好地进行翻译。从这个角度说，机器翻译中的表示学习又和语言模型中的表示学习有不同。不过，这里不会深入讨论神经语言模型和预训练与神经机器翻译之间的异同，感兴趣的读者可以参看第五章的相关内容。

还有一点，在神经机器翻译中，句子的表示形式可以有很多选择。使用单个向量表示一个句子是一种最简单的方法。当然，也可以用矩阵、高阶张量完成表示。甚至，在解码时动态地生成源语言的表示结果。这部分技术也会在随后的内容中进行介绍。

6.2.3 简单的运行实例

为了对编码器-解码器框架和神经机器翻译的运行过程有一个直观的认识，这里演示一个简单的翻译实例。这里采用标准的循环神经网络作为编码器和解码器的结构。假设系统的输入和输出为：

源语（中文）输入：{“我”，“很”，“好”，“<eos>”}
目标语（英文）输出：{“I”，“am”，“fine”，“<eos>”}

其中，<eos>（End of Sequence）表示序列的终止，<sos>（Start of Sequence）表示序列的开始。

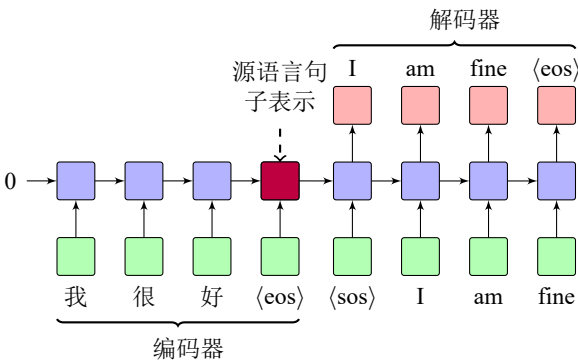


图 6.7: 神经机器翻译的运行实例

翻译过程的神经网络结构如图6.7所示，其中左边是编码器，右边是解码器。编码器会顺序处理源语言单词，将每个单词都表示成一个实数向量，也就是每个单词的词嵌入结果（绿色方框）。在词嵌入的基础上运行循环神经网络（蓝色方框）。在编码下一个时间步状态的时候，上一个时间步的隐藏状态会作为历史信息传入给循环神经网络。这样，句子中每个位置的信息都被向后传递，最后一个时间步的隐藏

⁴相比神经机器翻译的编码器，神经语言模型会多出一个输出层，这时可以直接把神经语言模型的中间层的输出作为编码器的输出。

状态（红色方框）就包含了整个源语言句子的信息，也就得到了编码器的编码结果——源语言句子的分布式表示。

解码器直接把源语言句子的分布式表示作为输入的隐层状态，之后像编码器一样依次读入目标语言单词，这是一个标准的循环神经网络的执行过程。与编码器不同的是，解码器会有一个输出层，用于根据当前时间步的隐层状态生成目标语单词及其概率分布。可以看到，解码端当前时刻的输出单词与下一个时刻的输入单词是一样的。从这个角度说，解码器也是一种神经语言模型，只不过它会从另外一种语言（源语言）获得一些信息，而不是仅仅做单语句子的生成。具体来说，当生成第一个单词“I”时，解码器利用了源语言句子表示（红色方框）和目标语的起始词“<sos>”。在生成第二个单词“am”时，解码器利用了上一个时间步的隐藏状态（隐藏层变量）和已经生成的“I”的信息。这个过程会循环执行，直到生成完整的目标语句子。

从这个例子可以看出，神经机器翻译的流程其实并不复杂：首先通过编码器神经网络将源语言句子编码成实数向量，然后解码器神经网络利用源语言句子的表示结果逐词生成译文。几乎所有的神经机器翻译系统都是类似架构。

6.2.4 机器翻译范式的对比

对于不同类型的机器翻译方法，人类所扮演的作用是不同的。在统计机器翻译时代，往往需要人工来定义翻译时所需要的特征和翻译单元，翻译中的每一个步骤对于人来说都是透明的，翻译过程具有一定的可解释性。而在神经机器翻译时代，神经机器翻译将所有的工作都交给神经网络，翻译的过程完全由神经网络计算得到。在整个神经网络的运行过程中并不需要人工先验知识，其中所生成的中间表示也只有神经网络自身才可以理解。有时候也会把神经机器翻译系统看作“黑盒”。所谓“黑盒”并不是指神经网络计算的过程不可见，而是这种复杂的计算过程无法控制也很难解释。那么是神经机器翻译会魔法吗，不需要任何人为的干预就可以进行翻译吗？其实不然，相对于统计机器翻译，真正变化的是人类使用知识的形式。

表 6.5: 不同机器翻译范式中人类的作用

机器翻译方法	人类参与方式
基于规则的方法	设计翻译规则
传统统计方法	设计翻译特征
神经网络方法	设计网络架构

在机器翻译的不同时期，人类参与到机器翻译中的形式并不相同。如表6.5所述，在早期基于规则的方法中，规则的编写、维护均需要人来完成，也就是人类直接提供了计算机可读的知识形式；在统计机器翻译方法中，则需要人为的设计翻译特征，并且定义基本翻译单元的形式，然后剩下的事情（比如翻译过程）交由统计机器翻译算法完成，也就是人类间接的提供了翻译所需要的知识；在神经机器翻译方法中，特征的设计完全不需要人的参与，但是完成特征提取的网络结构仍然需要人为地设

计，训练网络所需要的参数也需要工程师的不断调整，才能发挥神经机器翻译的强大性能。

可见，不管是基于规则的机器翻译方法，还是统计机器翻译方法，甚至今天的神经机器翻译方法，人类的作用是不可替代的。虽然神经机器翻译很强大，但是它的成功仍然依赖人工设计网络结构、调参。纵然，也有一些研究工作通过结构搜索的方法自动获得神经网络结构，但是搜索的算法和模型仍然需要人工设计。当然，这里不是要讨论一个新的悖论，因为结论还是很简单的：机器翻译是人类设计的，脱离了人的工作，机器翻译是不可能成功的。

6.3 基于循环神经网络的翻译模型及注意力机制

早期神经机器翻译的进展主要来自两个方面：1) 使用循环神经网络对单词序列进行建模；2) 注意力机制的使用。表6.6列出了 2013-2015 年间有代表性的部分研究工作。从这些工作的内容上看，当时的研究重点还是如何有效地使用循环神经网络进行翻译建模以及使用注意力机制捕捉双语单词序列间的对应关系。

表 6.6: 2013-2015 期间神经机器翻译方面的部分论文

时间	作者	论文
2013	Kalchbrenner 和 Blunsom	Recurrent Continuous Translation Models
2014	Sutskever 等	Sequence to Sequence Learning with neural networks
2014	Bahdanau 等	Neural Machine Translation by Jointly Learning to Align and Translate
2014	Cho 等	On the Properties of Neural Machine Translation
2015	Jean 等	On Using Very Large Target Vocabulary for Neural Machine Translation
2015	Luong 等	Effective Approches to Attention-based Neural Machine Translation

可以说循环神经网络和注意力机制构成了当时神经机器翻译的标准框架。比较有代表性的工作是谷歌公司于 2016 年上线的谷歌神经机器翻译系统（GNMT），它是由多层循环神经网络（长短时记忆模型）以及注意力机制搭建，且在当时来看性能很强劲的翻译模型 [312]。这项工作也引起了广泛的关注（图6.8），甚至可以被看作是神经机器翻译进入飞速发展时期的一个重要的标志。在 GNMT 推出后，很多企业也推出了基于循环神经网络的神经机器翻译系统，出现了百花齐放的局面。

本章将会从基于循环神经网络的翻译模型和注意力机制入手，介绍神经机器翻

译的基本方法。同时也会以 GNMT 系统为例，对神经机器翻译的其他相关技术进行讨论。



图 6.8: 对 GNMT 的报道

6.3.1 建模

同大多数自然语言处理任务一样，神经机器翻译要解决的一个基本问题是如何描述文字序列，称为序列表示问题。例如，处理语音数据、文本数据都可以被看作是典型的序列表示问题。如果把一个序列看作一个时序上的一系列变量，不同时刻的变量之间往往是存在相关性的。也就是说，一个时序中某个时刻变量的状态会依赖其他时刻变量的状态，即上下文的语境信息。下面是一个简单的例子，假设有一个句子，但是最后两个单词被擦掉了，如何猜测被擦掉的单词是什么？

中午没吃饭，又刚打了一下午篮球，我现在很饿，我想_____。

显然，根据上下文中提到的“没吃饭”、“很饿”，最佳的答案是“吃饭”或者“吃东西”。也就是，对序列中某个位置的答案进行预测时需要记忆当前时刻之前的序列信息，因此，**循环神经网络**（Recurrent Neural Network, RNN）应运而生。实际上循环神经网络有着极为广泛的应用，例如语音识别、语言建模以及即将要介绍的神经机器翻译。

第五章已经对循环神经网络的基本知识进行过介绍。这里再回顾一下。简单来说，循环神经网络由循环单元组成。对于序列中的任意时刻，都有一个循环单元与之对应，它会融合当前时刻的输入和上一时刻循环单元的输出，生成当前时刻的输出。这样每个时刻的信息都会被传递到下一时刻，这也间接达到了记录历史信息的目的。比如，对于序列 $\mathbf{x} = \{x_1, x_2, \dots, x_m\}$ ，循环神经网络会按顺序输出一个序列 $\mathbf{h} = \{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_m\}$ ，其中 \mathbf{h}_i 表示 i 时刻循环神经网络的输出（通常为一个向量）。

图6.9展示了一个循环神经网络处理序列问题的实例。当前时刻循环单元的输入由上一个时刻的输入和当前时刻的输入组成，因此也可以理解为，网络当前时刻计算得到的输出是由之前的序列共同决定的，即网络在不断地传递信息的过程中记忆了历史信息。以最后一个时刻的循环单元为例，它在对“开始”这个单词的信息进行处理时，参考了之前所有词（“<sos> 让我们”）的信息。

在神经机器翻译里使用循环神经网络也很简单。我们只需要把源语言句子和目

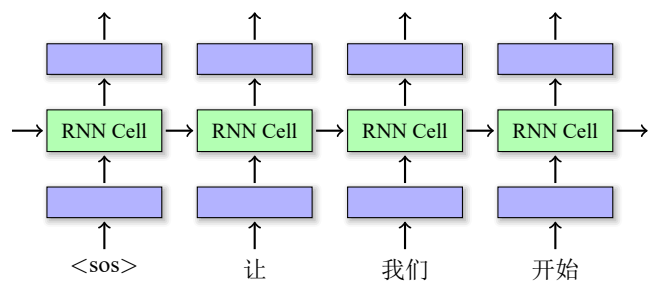


图 6.9: 循环神经网络处理序列的实例

标语言句子分别看作两个序列，之后使用两个循环神经网络分别对其进行建模。这个过程如图6.10所示。图中，下半部分是编码器，上半部分是解码器。编码器利用循环神经网络对源语言序列逐词进行编码处理，同时利用循环单元的记能力，不断累积序列信息，遇到终止符 <eos> 后便得到了包含源语言句子全部信息的表示结果。解码器利用编码器的输出和起始符 <sos> 开始逐词的进行解码，即逐词翻译，每得到一个译文单词，便将其作为当前时刻解码端循环单元的输入，这也是一个典型的神经语言模型的序列生成过程。解码器通过循环神经网络不断地累积已经得到的译文的信息，并继续生成下一个单词，直到遇到结束符 <eos>，便得到了最终完整的译文。

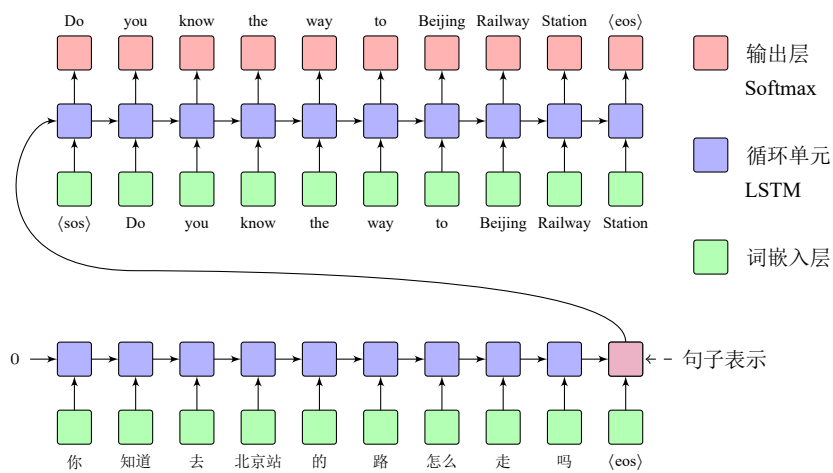


图 6.10: 基于循环神经网络翻译的模型结构

从数学模型上看，神经机器翻译模型与统计机器翻译的目标是一样的：在给定源语言句子 \mathbf{x} 的情况下，找出翻译概率最大的目标语译文 $\hat{\mathbf{y}}$:

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y}} P(\mathbf{y} | \mathbf{x}) \tag{6.1}$$

这里, 用 $\mathbf{x} = \{x_1, x_2, \dots, x_m\}$ 表示输入的源语言单词序列, $\mathbf{y} = \{y_1, y_2, \dots, y_n\}$ 表示生成的目标语单词序列。由于神经机器翻译在生成译文时采用的是自左向右逐词生成的方式, 并在翻译每个单词时考虑已经生成的翻译结果, 因此对 $P(\mathbf{y}|\mathbf{x})$ 的求解可以转换为:

$$P(\mathbf{y}|\mathbf{x}) = \prod_{j=1}^n P(y_j|\mathbf{y}_{<j}, \mathbf{x}) \quad (6.2)$$

其中, $\mathbf{y}_{<j}$ 表示目标语第 j 个位置之前已经生成的译文单词序列。 $P(y_j|\mathbf{y}_{<j}, \mathbf{x})$ 可以被解释为: 根据源语句子 \mathbf{x} 和已生成的目标语言译文片段 $\mathbf{y}_{<j} = \{y_1, y_2, \dots, y_{j-1}\}$, 生成第 j 个目标语言单词 y_j 的概率。举个简单的例子, 已知源文为 $\mathbf{x} = \{\text{“我”, “很好”}\}$, 则译文 $\mathbf{y} = \{\text{“I’m”, “fine”}\}$ 的概率为:

$$P(\{\text{“I’m”, “fine”}\}|\{\text{“我”, “很好”}\}) = P(\text{“I’m”}|\{\text{“我”, “很好”}\}) \cdot P(\text{“fine”}|\text{“I’m”, }\{\text{“我”, “很好”}\}) \quad (6.3)$$

求解 $P(y_j|\mathbf{y}_{<j}, \mathbf{x})$ 有三个关键问题 (图6.11):

- 如何对 \mathbf{x} 和 $\mathbf{y}_{<j}$ 进行分布式表示, 即**词嵌入** (Word Embedding)。首先, 将由 one-hot 向量表示的源语言单词, 即由 0 和 1 构成的离散化向量表示, 转化为实数向量。可以把这个过程记为 $\mathbf{e}_x(\cdot)$ 。类似的, 可以把目标语序列 $\mathbf{y}_{<j}$ 中的每个单词用同样的方式进行表示, 记为 $\mathbf{e}_y(\cdot)$ 。
- 如何在词嵌入的基础上获取整个序列的表示, 即句子的**表示学习** (Representation Learning)。可以把词嵌入的序列作为循环神经网络的输入, 循环神经网络最后一个时刻的输出向量便是整个句子的表示结果。如图6.11中, 编码器最后一个循环单元的输出 \mathbf{h}_m 被看作是一种包含了源语句子信息的表示结果, 记为 \mathbf{C} 。
- 如何得到每个目标语单词的概率, 即译文单词的**生成** (Generation)。与神经语言模型一样, 可以用一个 Softmax 输出层来获取当前时刻所有单词的分布, 即利用 Softmax 函数计算目标语词表中每个单词的概率。令目标语序列 j 时刻的循环神经网络的输出向量 (或状态) 为 \mathbf{s}_j 。根据循环神经网络的性质, y_j 的生成只依赖前一个状态 \mathbf{s}_{j-1} 和当前时刻的输入 (即词嵌入 $\mathbf{e}_y(y_{j-1})$)。同时考虑源语言信息 \mathbf{C} , $P(y_j|\mathbf{y}_{<j}, \mathbf{x})$ 可以被重新定义为:

$$P(y_j|\mathbf{y}_{<j}, \mathbf{x}) \equiv P(y_j|\mathbf{s}_{j-1}, y_{j-1}, \mathbf{C}) \quad (6.4)$$

$P(y_j|\mathbf{s}_{j-1}, y_{j-1}, \mathbf{C})$ 由 Softmax 实现, Softmax 的输入是循环神经网络 j 时刻的输出。在具体实现时, \mathbf{C} 可以被简单的作为第一个时刻循环单元的输入, 即, 当 $j = 1$ 时, 解码器的循环神经网络会读入编码器最后一个隐层状态 \mathbf{h}_m (也就

个实数矩阵 \mathbf{E} ，得到的结果（行向量）就是这个单词所对应的词嵌入结果。

$$\mathbf{e}_y(y_j) = y_j \mathbf{E} \tag{6.6}$$

这里， \mathbf{E} 也被称作词嵌入矩阵，它可以作为模型的一部分参数共同参与机器翻译系统的训练，也可以由外部其他模块训练得到（如预训练模型）。 \mathbf{E} 的大小为 $|V| \times d$ ，这里 $|V|$ 表示词表 V 的大小， d 表示循环神经网络输入和输出向量的维度。

图6.12以单词“you”为例，展示了词嵌入的生成过程。词嵌入层首先将输入的单词“you”转化成 One-hot 表示，对应虚线框中的 0-1 向量，即除了 you 在词表中的索引位置为 1，其余位置均为 0。然后词嵌入层将这个 0-1 向量乘以 \mathbf{E} 就得到了词嵌入的结果（绿色圆角框框起来部分），这里用 $\mathbf{e}_y(\cdot)$ 表示这个过程，即 you 的词嵌入表示 $\mathbf{e}_y(\text{“you”})$ 。最后，将单词的词嵌入表示作为当前时间步循环单元（蓝色方框）的输入。

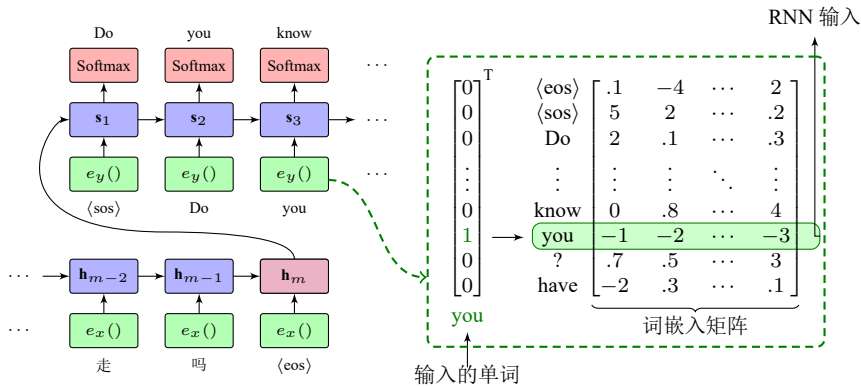


图 6.12: 词嵌入的生成过程

需要注意的是，在上面这个过程中 One-hot 表示和词嵌入矩阵并不必须调用矩阵乘法才得到词嵌入结果。只需要获得 One-hot 向量中 1 对应的索引，从词嵌入矩阵中取出对应的行即可。这种利用索引“取”结果的方式避免了计算代价较高的矩阵乘法，因此在实际系统中很常用。

在解码端，需要在每个位置预测输出的单词。在循环神经网络中，每一时刻循环单元的输出向量为 \mathbf{s}_j ，它可以被看作这个时刻的目标语单词的一种表示，但是我们无法根据这个向量得出要生成的目标语单词的概率。而输出层的目的便是通过向量 \mathbf{s}_j 计算词表中每个单词的生成概率，进而选取概率最高的单词作为当前时刻的输出。图6.13展示了一个输出层进行单词预测的实例。

输出层的构造很简单，对于输入的向量 \mathbf{s}_j 经过一个线性变换之后再经过 Softmax 函数，即可得到一个 V 上的分布，具体描述如下：

$$\mathbf{o}_j = \text{Softmax}(\mathbf{s}_j \mathbf{W}_o) \tag{6.7}$$

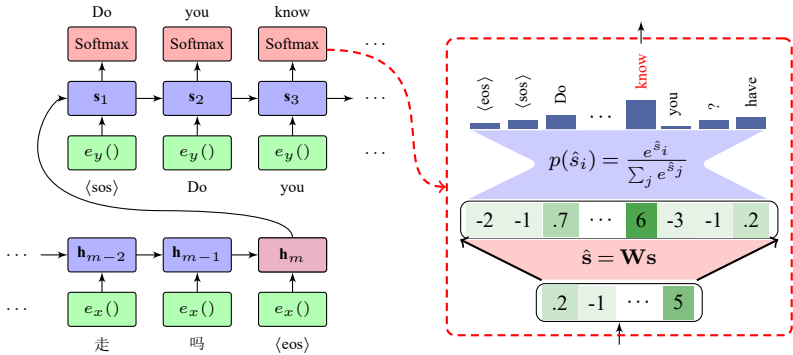


图 6.13: 输出层的预测过程

其中， \mathbf{W}_o 是线性变换的参数矩阵，矩阵的大小为 $d \times |V|$ ，也就是它会把 d 维的向量变为 $|V|$ 维的向量； \mathbf{o}_j 表示输出的结果向量， \mathbf{o}_j 的每一维 \mathbf{o}_{jk} 表示，在时刻 j 词表 V 中一个第 k 个单词出现的概率。这里把 $\mathbf{o}_j(y_j)$ 记作目标语单词 y_j 的生成概率，显然有

$$P(y_j | \mathbf{y}_{<j}, \mathbf{x}) = \mathbf{o}_j(y_j) \tag{6.8}$$

$\text{Softmax}(\cdot)$ 的作用是根据输入的 $|V|$ 维向量（即 $\mathbf{s}_j \mathbf{W}_o$ ），得到一个 $|V|$ 维的分布。令 τ 表示 $\text{Softmax}(\cdot)$ 的输入向量， τ_k 表示向量的第 k 维。 Softmax 函数可以被定义为

$$\text{Softmax}(\tau_k) = \frac{\exp(\tau_k)}{\sum_{k'=1}^{|V|} \exp(\tau_{k'})} \tag{6.9}$$

这里， $\exp(\cdot)$ 表示指数函数。 Softmax 函数是一个典型的归一化函数，它可以将输入的向量的每一维都转化为 0-1 之间的数，同时保证所有维的和等于 1。 Softmax 的另一个优点是，它本身（对于输出的每一维）都是可微的（如图6.14所示），因此可以直接使用基于梯度的方法进行优化。实际上， Softmax 经常被用于分类任务。也可以把机器翻译中目标语单词的生成看作一个分类问题，它的类别数是 $|V|$ 。

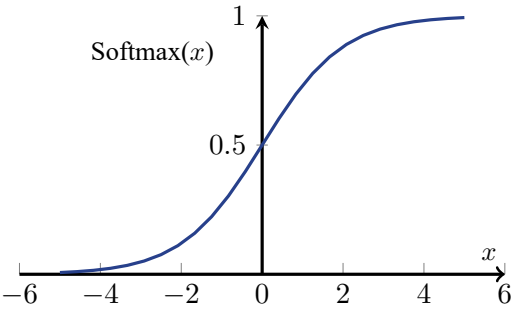


图 6.14: Softmax 函数（一维）所对应的曲线

为了进一步理解 Softmax 的计算过程, 来看一个简单的例子。假设词表为 (“吃饭”, “睡觉”, “学习”), 当预测下一个译文单词时, 可以将循环神经网络的输出通过矩阵 \mathbf{W}_0 映射到词表大小的向量, 得到 $\tau = (-3, 1.5, 2.7)$, 此时再使用 Softmax 激活函数将其进行归一化:

$$\text{Softmax}(\tau) = \begin{pmatrix} \frac{0.05}{0.05+4.48+14.88} \\ \frac{4.48}{0.05+4.48+14.88} \\ \frac{14.88}{0.05+4.48+14.88} \end{pmatrix} = \begin{pmatrix} 0.0026 \\ 0.2308 \\ 0.7666 \end{pmatrix} \quad (6.10)$$

最终得到在整个词表上的概率分布 (0.0026, 0.2308, 0.7666), 其中概率最大的单词 “学习”, 便是最终的译文单词。

6.3.3 循环神经网络结构

循环神经网络的核心是设计循环单元的结构。至今, 研究人员已经提出了很多优秀的循环单元结构, 这里将介绍其中三种基本结构: RNN, LSTM 和 GRU。LSTM 和 GRU 是 RNN 的变体, 在自然语言处理任务中得到了广泛的应用。

循环神经单元 (RNN)

RNN (Recurrent Neural Network) 是最原始的循环神经网络结构。在 RNN 中, 对于序列 $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$, 每个时刻 t 都对应一个循环单元, 它的输出是一个向量 \mathbf{h}_t , 可以被描述为:

$$\mathbf{h}_t = f(\mathbf{x}_t \mathbf{U} + \mathbf{h}_{t-1} \mathbf{W} + \mathbf{b}) \quad (6.11)$$

其中 \mathbf{x}_t 是当前时刻的输入, \mathbf{h}_{t-1} 是上一时刻循环单元的输出, $f(\cdot)$ 是激活函数, \mathbf{U} 和 \mathbf{W} 是参数矩阵, \mathbf{b} 是偏置。

虽然 RNN 的结构很简单, 但是已经具有了对序列信息进行记忆的能力。实际上, 基于 RNN 结构的神经语言模型已经能够取得比传统 n -gram 语言模型更优异的性能。在机器翻译中, RNN 也可以做为入门或者快速原型所使用的神经网络结构。

长短期记忆网络 (LSTM)

RNN 结构使得当前时刻循环单元的状态包含了之前时间步的状态信息。但是这种对历史信息的记忆并不是无损的, 随着序列变长, RNN 的记忆信息的损失越来越严重。在很多长序列处理任务中 (如长文本生成) 都观测到了类似现象。对于这个问题, Hochreiter 和 Schmidhuber 提出了**长短期记忆** (Long Short-term Memory) 模型, 也就是常说的 LSTM 模型 [112]。

LSTM 模型是 RNN 模型的一种改进。相比 RNN 仅传递前一时刻的状态 \mathbf{h}_{t-1} , LSTM 会同时传递两部分信息: 状态信息 \mathbf{h}_{t-1} 和记忆信息 \mathbf{c}_{t-1} 。这里, \mathbf{c}_{t-1} 是新引入的变量, 它也是循环单元的一部分, 用于显性的记录需要记录的历史内容, \mathbf{h}_{t-1} 和

c_{t-1} 在循环单元中会相互作用。LSTM 通过“门”单元来动态地选择遗忘多少以前的信息和记忆多少当前的信息。LSTM 中所使用的门结构如图6.15所示，包括遗忘门，输入门和输出门。图中 σ 代表 Sigmoid 函数，它将函数输入映射为 0-1 范围内的实数，用来充当门控信号。

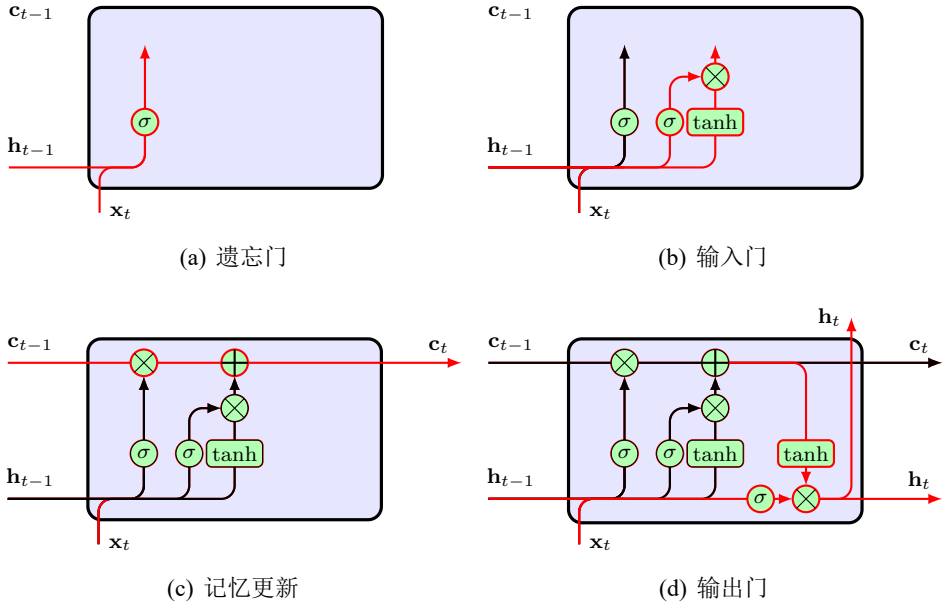


图 6.15: LSTM 中的门控结构

LSTM 的结构主要分为三个部分：

- **遗忘**。顾名思义，遗忘的目的是忘记一些历史，在 LSTM 中通过遗忘门实现，其结构如图6.15(a)所示。 \mathbf{x}_t 表示时刻 t 的输入向量， \mathbf{h}_{t-1} 是时刻 $t-1$ 的循环单元的输出， \mathbf{x}_t 和 \mathbf{h}_{t-1} 都作为 t 时刻循环单元的输入。 σ 将对 \mathbf{x}_t 和 \mathbf{h}_{t-1} 进行筛选，以决定遗忘的信息，其计算公式如下：

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \quad (6.12)$$

这里， \mathbf{W}_f 是权值， \mathbf{b}_f 是偏置， $[\mathbf{h}_{t-1}, \mathbf{x}_t]$ 表示两个向量的拼接。该公式可以解释为，对 $[\mathbf{h}_{t-1}, \mathbf{x}_t]$ 进行变换，并得到一个实数向量 \mathbf{f}_t 。 \mathbf{f}_t 的每一维都可以被理解为一个“门”，它决定可以有多少信息被留下（或遗忘）。

- **记忆更新**。首先，要生成当前时刻需要新增加的信息，该部分由输入门完成，其结构如图6.15(b)红色线部分，图中“ \otimes ”表示进行点乘操作。输入门的计算分为两部分，首先利用 σ 决定门控参数 \mathbf{i}_t ，然后通过 Tanh 函数得到新的信息 $\hat{\mathbf{c}}_t$ ，

具体公式如下：

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \quad (6.13)$$

$$\hat{\mathbf{c}}_t = \text{Tanh}(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c) \quad (6.14)$$

之后，用 \mathbf{i}_t 点乘 $\hat{\mathbf{c}}_t$ ，得到当前需要记忆的信息，记为 $\mathbf{i}_t \cdot \hat{\mathbf{c}}_t$ 。接下来需要更新旧的信息 \mathbf{c}_{t-1} ，得到新的记忆信息 \mathbf{c}_t ，更新的操作如图6.15(c) 红色线部分所示，“ \oplus ”表示相加。具体规则是通过遗忘门选择忘记一部分上文信息 \mathbf{f}_t ，通过输入门计算新增的信息 $\mathbf{i}_t \cdot \hat{\mathbf{c}}_t$ ，然后根据“ \otimes ”门与“ \oplus ”门进行相应的乘法和加法计算：

$$\mathbf{c}_t = \mathbf{f}_t \cdot \mathbf{c}_{t-1} + \mathbf{i}_t \cdot \hat{\mathbf{c}}_t \quad (6.15)$$

- **输出。**该部分使用输出门计算最终的输出信息 \mathbf{h}_t ，其结构如图6.15(d) 红色线部分所示。在输出门中，首先将 \mathbf{x}_t 和 \mathbf{h}_{t-1} 通过 σ 函数变换得到 \mathbf{o}_t 。其次，将上一步得到的新记忆信息 \mathbf{c}_t 通过 Tanh 函数进行变换，得到值在 $[-1, 1]$ 范围的向量。最后将这两部分进行点乘，具体公式如下：

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \quad (6.16)$$

$$\mathbf{h}_t = \mathbf{o}_t \cdot \text{Tanh}(\mathbf{c}_t) \quad (6.17)$$

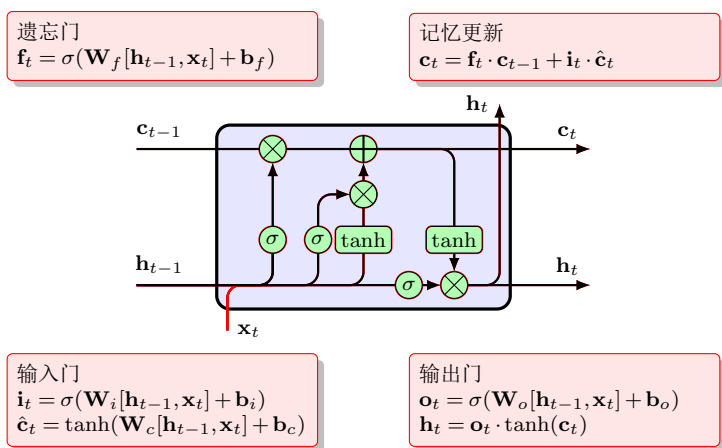


图 6.16: LSTM 的整体结构

LSTM 的完整结构如图6.16所示，模型的参数包括：参数矩阵 \mathbf{W}_f 、 \mathbf{W}_i 、 \mathbf{W}_c 、 \mathbf{W}_o 和偏置 \mathbf{b}_f 、 \mathbf{b}_i 、 \mathbf{b}_c 、 \mathbf{b}_o 。可以看出， \mathbf{h}_t 是由 \mathbf{c}_{t-1} 、 \mathbf{h}_{t-1} 与 \mathbf{x}_t 共同决定的。此外，上述公式中激活函数的选择是根据函数各自的特点决定的。

门控循环单元 (GRU)

LSTM 通过门控单元控制传递状态，忘记不重要的信息，记住必要的历史信息，在长序列上取得了很好的效果，但是其进行了许多门信号的计算，较为繁琐。**门循环单元** (Gated Recurrent Unit, GRU) 作为一个 LSTM 的变种，它继承了 LSTM 中利用门控单元控制信息传递的思想，并对 LSTM 进行了简化 [44]。它把循环单元状态 \mathbf{h}_t 和记忆 \mathbf{c}_t 合并成一个状态 \mathbf{h}_t ，同时使用了更少的门控单元，大大提升了计算效率。

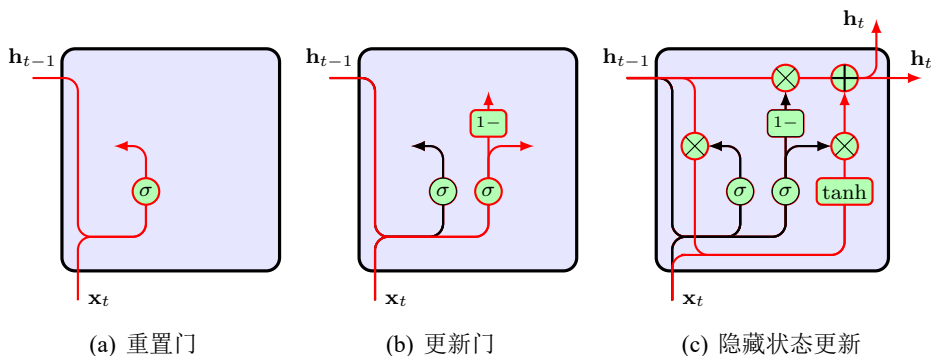


图 6.17: GRU 中的门控结构

GRU 的输入和 RNN 是一样的，由输入 \mathbf{x}_t 和 $t-1$ 时刻的状态 \mathbf{h}_{t-1} 组成。GRU 只有两个门信号，分别是重置门和更新门。重置门 \mathbf{r}_t 用来控制前一时刻隐藏状态的记忆程度，其结构如图6.17(a)。更新门用来更新记忆，使用一个门同时完成遗忘和记忆两种操作，其结构如图6.17(b)。重置门和更新门的计算公式如下：

$$\mathbf{r}_t = \sigma(\mathbf{W}_r[\mathbf{h}_{t-1}, \mathbf{x}_t]) \quad (6.18)$$

$$\mathbf{u}_t = \sigma(\mathbf{W}_u[\mathbf{h}_{t-1}, \mathbf{x}_t]) \quad (6.19)$$

当完成了重置门和更新门计算后，就需要更新当前隐藏状态，如图6.17(c) 所示。在计算得到了重置门的权重 \mathbf{r}_t 后，使用其对前一时刻的状态 \mathbf{h}_{t-1} 进行重置 ($\mathbf{r}_t \cdot \mathbf{h}_{t-1}$)，将重置后的结果与 \mathbf{x}_t 拼接，通过 Tanh 激活函数将数据变换到 $[-1,1]$ 范围内：

$$\hat{\mathbf{h}}_t = \text{Tanh}(\mathbf{W}_h[\mathbf{r}_t \cdot \mathbf{h}_{t-1}, \mathbf{x}_t]) \quad (6.20)$$

$\hat{\mathbf{h}}_t$ 在包含了输入信息 \mathbf{x}_t 的同时，引入了 \mathbf{h}_{t-1} 的信息，可以理解为，记忆了当前时刻的状态。下一步是计算更新后的隐藏状态也就是更新记忆，如下所示：

$$\mathbf{h}_t = (1 - \mathbf{u}_t) \cdot \mathbf{h}_{t-1} + \mathbf{u}_t \cdot \hat{\mathbf{h}}_t \quad (6.21)$$

这里， \mathbf{u}_t 是更新门中得到的权重，将 \mathbf{u}_t 作用于 $\hat{\mathbf{h}}_t$ 表示对当前时刻的状态进行“遗忘”，舍弃一些不重要的信息，将 $(1 - \mathbf{u}_t)$ 作用于 \mathbf{h}_{t-1} ，用于对上一时刻隐藏状态进

行选择性记忆。

GRU 的输入输出和 RNN 类似，其采用与 LSTM 类似的门控思想，达到捕获长距离依赖信息的目的。此外，GRU 比 LSTM 少了一个门结构，而且参数只有 \mathbf{W}_r 、 \mathbf{W}_u 和 \mathbf{W}_h 。因此，GRU 具有比 LSTM 高的运算效率，在系统研发中也经常被使用。

双向模型

前面提到的循环神经网络都是自左向右运行的，也就是说在处理一个单词的时候只能访问它前面的序列信息。但是，只根据句子的前文来生成一个序列的表示是不全面的，因为从最后一个词来看，第一个词的信息可能已经很微弱了。为了同时考虑前文和后文的信息，一种解决办法是使用双向循环网络，其结构如图6.18所示。这里，编码器可以看作有两个循环神经网络，第一个网络，即红色虚线框里的网络，从句子的右边进行处理，第二个网络从句子左边开始处理，最终将正向和反向得到的结果都融合后传递给解码器。

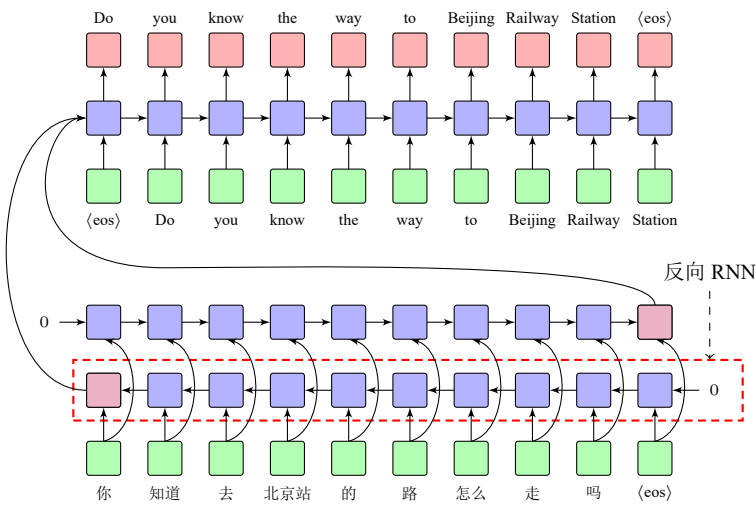


图 6.18: 基于双向循环神经网络的机器翻译模型结构

双向模型是自然语言处理领域的常用模型，包括前面提到的词对齐对称化、语言模型等中都大量地使用了类似的思路。实际上，这里也体现了建模时的非对称思想。也就是，建模时如果设计一个对称模型可能会导致问题复杂度增加，因此往往先对问题进行化简，从某一个角度解决问题。之后再融合多个模型，从不同角度得到相对合理的最终方案。

多层循环神经网络

实际上，对于单词序列所使用的循环神经网络是一种很“深”的网络，因为从第一个单词到最后一个单词需要经过至少句子长度相当层数的神经元。比如，一个包含几十个词的句子也会对应几十个神经元层。但是，在很多深度学习应用中，更习惯

把对输入序列的同一种处理作为“一层”。比如，对于输入序列，构建一个 RNN，那么这些循环单元就构成了网络的“一层”。当然，这里并不是要混淆概念。只是要明确，在随后的讨论中，“层”并不是指一组神经元的全连接，它一般指的是网络的拓扑结构。

单层循环神经网络对输入序列进行了抽象，为了得到更深入的抽象能力，可以把多个循环神经网络叠在一起，构成多层循环神经网络。比如，图6.19就展示基于两层循环神经网络的解码器和编码器结构。通常来说，层数越多模型的表示能力越强，因此在很多基于循环神经网络的机器翻译系统中一般会使用 4~8 层的网络。但是，过多的层也会增加模型训练的难度，甚至导致模型无法进行训练。第七章还会对这个问题进行深入讨论。

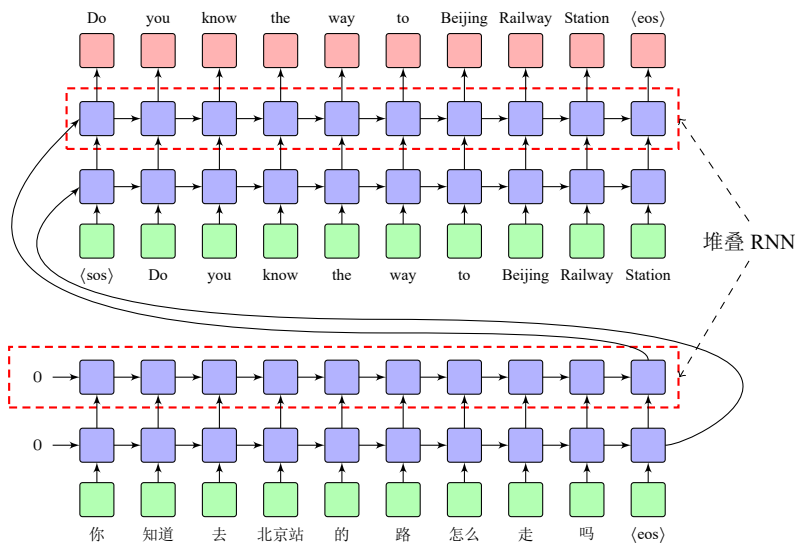


图 6.19: 基于双层循环神经网络的机器翻译模型结构

6.3.4 注意力机制

第二章提到过“上帝是不公平的”，这个观点主要是表达了：世界上事物之间的联系不是均匀的，有些事物之间的联系会很强，而其他的联系可能很弱。自然语言也完美地契合了这个观点。比如，再重新看一下前面提到的根据上下文补全缺失单词的例子，

中午没吃饭，又刚打了一下午篮球，我现在很饿，我想_____。

之所以能想到在横线处填“吃饭”、“吃东西”很有可能是因为看到了“没吃饭”、“很饿”等关键信息。也就是这些关键的片段对预测缺失的单词起着关键性作用。而预测“吃饭”与前文中的“中午”、“又”之间的联系似乎不那么紧密。也就是说，在形成

“吃饭”的逻辑时，在潜意识里会更注意“没吃饭”、“很饿”等关键信息。也就是我们的关注度并不是均匀地分布在整个句子上的。

这个现象可以用注意力机制进行解释。注意力机制的概念来源于生物学的一些现象：当待接收的信息过多时，人类会选择性地关注部分信息而忽略其他信息。它在人类的视觉、听觉、嗅觉等方面均有体现，当我们在感受事物时，大脑会自动过滤或衰减部分信息，仅关注其中少数几个部分。例如，当我们看到图6.20时，往往不是“均匀地”看图像中的所有区域，可能最先注意到的是大狗头上戴的帽子，然后才会关注图片中其他的部分。

那么注意力机制和神经机器翻译又有什么关系呢？它如何解决神经机器翻译的问题呢？下面就一起来看一看。



图 6.20: 戴帽子的狗

翻译中的注意力机制

如前文所述，早期的神经机器翻译只使用循环神经网络最后一个单元的输出作为整个序列的表示。这种方式有两个明显的缺陷：

- 首先，虽然编码器把一个源语言句子的表示传递给解码器，但是一个维度固定的向量所能包含的信息是有限的，随着源语言序列的增长，将整个句子的信息编码到一个固定维度的向量中可能会造成源语言句子信息的丢失。显然，在翻译较长的句子时，解码端可能无法获取完整的源语言信息，降低翻译性能；
- 此外，当生成某一个目标语单词时，并不是均匀的使用源语言句子中的单词信息。更普遍的情况是，系统会参考与这个目标语单词相对应的源语言单词进行翻译。这有些类似于词对齐的作用，即翻译是基于单词之间的某种对应关系。但是，使用单一的源语言表示根本无法区分源语言句子的不同部分，更不用说对源语言单词和目标语言单词之间的联系进行建模了。

看一个翻译实例，如图6.21，目标语中的“very long”仅依赖于源文中的“很长”。这时如果将所有源语编码成一个固定的实数向量，“很长”的信息就很可能被其他词的信息淹没掉，而翻译“very long”时也无法区分不同源语单词的贡献。

显然，以上问题的根本原因在于所使用的表示模型还比较“弱”。因此需要一个更强大的表示模型，在生成目标语单词时能够有选择地获取源语言句子中更有用的部分。更准确的说，对于要生成的目标语单词，相关性更高的源语言片段应该在源语言句子的表示中体现出来，而不是将所有的源语言单词一视同仁。在神经机器翻

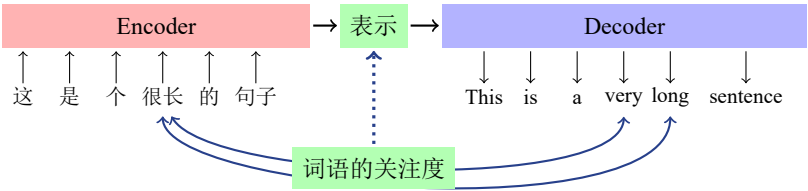


图 6.21: 源语词和目标语词的关注度

译中引入注意力机制正是为了达到这个目的 [7, 188]。实际上，除了机器翻译，注意力机制也被成功地应用于图像处理、语音识别、自然语言处理等其他任务。而正是注意力机制的引入，使得包括机器翻译在内很多自然语言处理系统得到了飞跃发展。

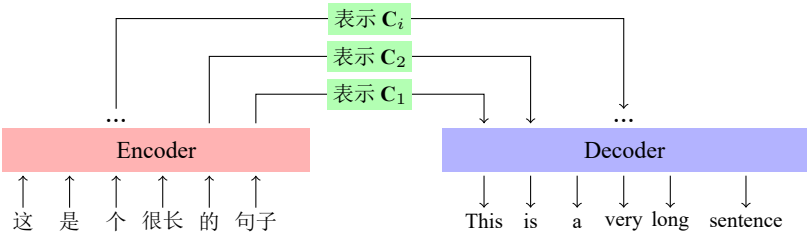
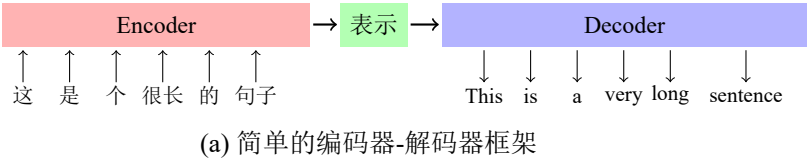


图 6.22: (a) 不使用和 (b) 使用注意力机制的翻译模型对比

神经机器翻译中的注意力机制并不复杂。对于每个目标语单词 y_j ，系统生成一个源语言表示向量 C_j 与之对应， C_j 会包含生成 y_j 所需的源语言的信息，或者说 C_j 是一种包含目标语言单词与源语言单词对应关系的源语言表示。相比用一个静态的表示 C ，注意机制使用的是动态的表示 C_j 。 C_j 也被称作对于目标语位置 j 的上下文向量。图6.22对比了未引入注意力机制和引入了注意力机制的编码器-解码器结构。可以看出，在注意力模型中，对于每一个目标单词的生成，都会额外引入一个单独的上下文向量参与运算。

上下文向量的计算

那么注意力机制是如何针对不同单词生成不同的上下文向量呢？这里，可以将注意力机制看做是一种对接收到的信息的加权处理。对于更重要的信息赋予更高的权重即更高的关注度，对于贡献度较低的信息分配较低的权重，弱化其对结果的影响。这样， C_j 可以包含更多对当前目标语言位置有贡献的源语言片段的信息。

根据这种思想, 上下文向量 \mathbf{C}_j 被定义为对不同时间步编码器输出的状态序列 $\{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_m\}$ 进行加权求和, 如下:

$$\mathbf{C}_j = \sum_i \alpha_{i,j} \mathbf{h}_i \quad (6.22)$$

其中, $\alpha_{i,j}$ 是**注意力权重** (Attention Weight), 它表示目标语第 j 个位置与源语第 i 个位置之间的相关性大小。这里, 将每个时间步编码器的输出 \mathbf{h}_i 看作源语言位置 i 的表示结果。进行翻译时, 解码端可以根据当前的位置 j , 通过控制不同 \mathbf{h}_i 的权重得到 \mathbf{C}_j , 使得对目标语位置 j 贡献大的 \mathbf{h}_i 对 \mathbf{C}_j 的影响增大。也就是说, \mathbf{C}_j 实际上就是 $\{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_m\}$ 的一种组合, 只不过不同的 \mathbf{h}_i 会根据对目标端的贡献给予不同的权重。图6.23展示了上下文向量 \mathbf{C}_j 的计算过程。

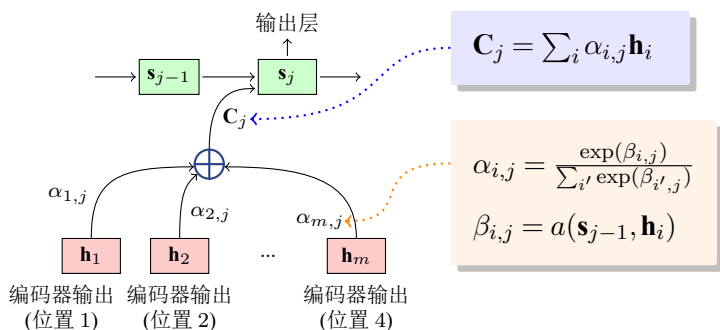


图 6.23: 上下文向量 \mathbf{C}_j 的计算过程

如图6.23所示, 注意力权重 $\alpha_{i,j}$ 的计算分为两步:

- 使用目标语言上一时刻循环单元的输出 \mathbf{s}_{j-1} 与源语言第 i 个位置的表示 \mathbf{h}_i 之间的相关性, 其用来表示目标语言位置 j 对源语言位置 i 的关注程度, 记为 $\beta_{i,j}$, 由函数 $a(\cdot)$ 实现

$$\beta_{i,j} = a(\mathbf{s}_{j-1}, \mathbf{h}_i) \quad (6.23)$$

$a(\cdot)$ 可以被看作是目标语言表示和源语言表示的一种“统一化”, 即把源语言和目标语言表示映射在同一个语义空间, 进而语义相近的内容有更大的相似性。该函数有多种计算方式, 比如, 向量乘、向量夹角、线性模型、单词神经网络等, 数学表达如下:

$$a(\mathbf{s}, \mathbf{h}) = \begin{cases} \mathbf{s}\mathbf{h}^T & \text{向量乘} \\ \cos(\mathbf{s}, \mathbf{h}) & \text{向量夹角} \\ \mathbf{s}\mathbf{W}\mathbf{h}^T & \text{线性模型} \\ \text{TanH}(\mathbf{W}[\mathbf{s}, \mathbf{h}])\mathbf{v}^T & \text{拼接}[\mathbf{s}, \mathbf{h}] + \text{单层网络} \end{cases} \quad (6.24)$$

其中 \mathbf{W} 和 \mathbf{v} 是可学习的参数。

- 进一步，利用 Softmax 函数，将相关性系数 $\beta_{i,j}$ 进行指数归一化处理，得到注意力权重 $\alpha_{i,j}$ ：

$$\alpha_{i,j} = \frac{\exp(\beta_{i,j})}{\sum_{i'} \exp(\beta_{i',j})} \tag{6.25}$$

最终， $\{\alpha_{i,j}\}$ 可以被看作是一个矩阵，它的长为目标语言句子长度，宽为源语言句子长度，矩阵中的每一项对应一个 $\alpha_{i,j}$ 。图6.24给出了 $\{\alpha_{i,j}\}$ 的一个矩阵表示。图中蓝色方框的大小表示不同的注意力权重 $\alpha_{i,j}$ 的大小，方框越大，源语言位置 i 和目标语言位置 j 的相关性越高。能够看到，对于互译的中英文句子， $\{\alpha_{i,j}\}$ 可以较好的反应两种语言之间不同位置的对应关系。

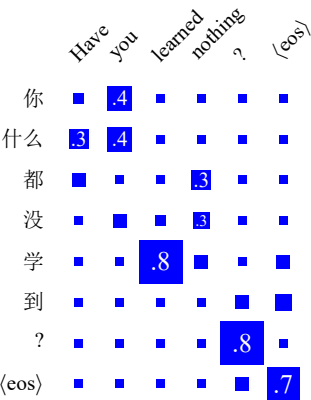


图 6.24: 一个汉英句对之间的注意力权重 $\alpha_{i,j}$ 的矩阵表示

图6.25展示了一个上下文向量的计算过程实例。首先，计算目标语第一个单词“Have”与源语中的所有单词的相关性，即注意力权重，对应图中第一列 $\alpha_{i,1}$ ，则当前时刻所使用的上下文向量 $\mathbf{C}_1 = \sum_{i=1}^8 \alpha_{i,1} \mathbf{h}_i$ ；然后，计算第二个单词“you”的注意力权重对应第二列 $\alpha_{i,2}$ ，其上下文向量 $\mathbf{C}_2 = \sum_{i=1}^8 \alpha_{i,2} \mathbf{h}_i$ ，以此类推，可以得到任意目标语位置 j 的上下文向量 \mathbf{C}_j 。很容易看出，不同目标语单词的上下文向量对应的源语言词的权重 $\alpha_{i,j}$ 是不同的，不同的注意力权重为不同位置赋予了不同重要性，对应了注意力机制的思想。

在6.3.1节中，我们使用公式6.5描述了目标语单词生成概率 $P(y_j | \mathbf{y}_{<j}, \mathbf{x})$ 。在引入注意力机制后，不同时刻的上下文向量 \mathbf{C}_j 替换了传统模型中固定的句子表示 \mathbf{C} 。描述如下：

$$P(y_j | \mathbf{y}_{<j}, \mathbf{x}) \equiv P(y_j | \mathbf{s}_{j-1}, y_{j-1}, \mathbf{C}_j) \tag{6.26}$$

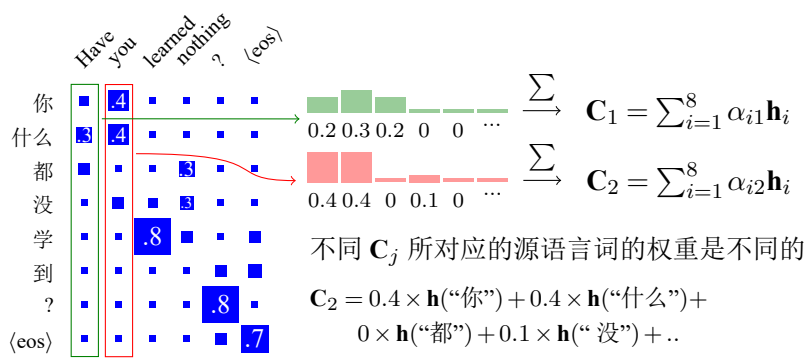


图 6.25: 上下文向量计算过程实例

这样，可以在生成每个 y_j 时动态的使用不同的源语言表示 C_j ，并更准确地捕捉源语言和目标语言不同位置之间的相关性。表6.7展示了引入注意力机制前后译文单词生成公式的对比。

表 6.7: 引入注意力机制前后译文单词生成公式

引入注意力之前	引入注意力之后
“have” = $\arg \max_{y_1} P(y_1 \mathbf{C}, y_0)$	“have” = $\arg \max_{y_1} P(y_1 \mathbf{C}_1, y_0)$
“you” = $\arg \max_{y_2} P(y_2 \mathbf{s}_1, y_1)$	“you” = $\arg \max_{y_2} P(y_2 \mathbf{s}_1, \mathbf{C}_2, y_1)$

注意力机制的解读

从前面的描述可以看出，注意力机制在机器翻译中就是要回答一个问题：给定一个目标语位置 j 和一系列源语言的不同位置上的表示 $\{\mathbf{h}_i\}$ ，如何得到一个新的表示 $\hat{\mathbf{h}}$ ，使得它与目标语位置 j 对应得最好？

那么，如何理解这个过程？注意力机制的本质又是什么呢？换一个角度来看，实际上，目标语位置 j 本质上是一个查询，我们希望从源语言端找到与之最匹配的源语言位置，并返回相应的表示结果。为了描述这个问题，可以建立一个查询系统。假设有一个库，里面包含若干个 key-value 单元，其中 key 代表这个单元的索引关键字，value 代表这个单元的值。比如，对于学生信息系统，key 可以是学号，value 可以是学生的身高。当输入一个查询 query，我们希望这个系统返回与之最匹配的结果。也就是，希望找到匹配的 key，并输出其对应的 value。比如，当查询某个学生的身高信息时，可以输入学生的学号，之后在库中查询与这个学号相匹配的记录，并把这个记录中的 value（即身高）作为结果返回。

图6.26展示了一个这样的查询系统。里面包含四个 key-value 单元，当输入查询 query，就把 query 与这四个 key 逐个进行匹配，如果完全匹配就返回相应的 value。在图中的例子中，query 和 key_3 是完全匹配的（因为都是横纹），因此系统返回第三

个单元的值，即 $value_3$ 。当然，如果库中没有与 query 匹配的 key，则返回一个空结果。

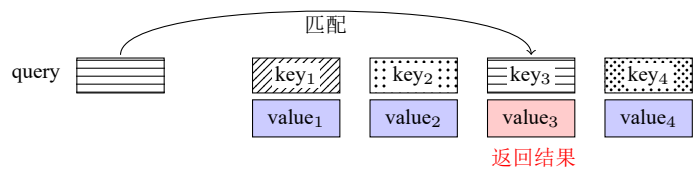


图 6.26: 传统查询模型

也可以用这个系统描述翻译中的注意力问题，其中，query 即目标语言位置 j 的某种表示，key 和 value 即源语言每个位置 i 上的 h_i （这里 key 和 value 是相同的）。但是，这样的系统在机器翻译问题上并不好用，因为目标语言的表示和源语言的表示都在多维实数空间上，所以无法要求两个实数向量像字符串一样进行严格匹配，或者说这种严格匹配的模型可能会导致 query 几乎不会命中任何的 key。既然无法严格精确匹配，注意力机制就采用了一个“模糊”匹配的方法。这里定义每个 key_i 和 query 都有一个 $0 \sim 1$ 之间的匹配度，这个匹配度描述了 key_i 和 query 之间的相关程度，记为 α_i 。而查询的结果（记为 \overline{value} ）也不再是某一个单元的 value，而是所有单元 value 用 α_i 的加权和：

$$\overline{value} = \sum_i \alpha_i \cdot value_i \tag{6.27}$$

也就是说所有的 $value_i$ 都会对查询结果有贡献，只是贡献度不同罢了。可以通过设计 α_i 来捕捉 key 和 query 之间的相关性，以达到相关度越大的 key 所对应的 value 对结果的贡献越大。

重新回到神经机器翻译问题上来。这种基于模糊匹配的查询模型可以很好的满足对注意力建模的要求。实际上，公式6.27中的 α_i 就是前面提到的注意力权重，它可以由注意力函数 $a(\cdot)$ 计算得到。这样， \overline{value} 就是得到的上下文向量，它包含了所有 $\{h_i\}$ 的信息，只是不同 h_i 的贡献度不同罢了。图6.27展示了将基于模糊匹配的查询模型应用于注意力机制的实例。

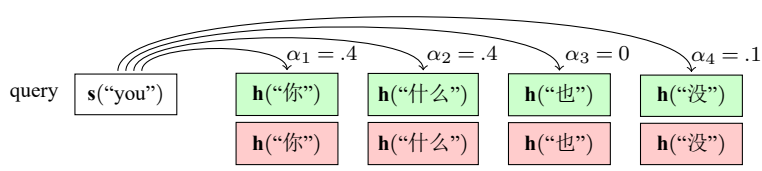


图 6.27: 注意力机制所对应的查询模型

最后，从统计学的角度，如果把 α_i 作为每个 $value_i$ 出现的概率的某种估计，即：

$P(\text{value}_i) = \alpha_i$ ，于是可以把公式6.27重写为：

$$\overline{\text{value}} = \sum_i P(\text{value}_i) \cdot \text{value}_i \quad (6.28)$$

显然， $\overline{\text{value}}$ 就是 value_i 在分布 $P(\text{value}_i)$ 下的期望，即

$$\mathbb{E}_{\sim P(\text{value}_i)}(\text{value}_i) = \sum_i P(\text{value}_i) \cdot \text{value}_i \quad (6.29)$$

从这个观点看，注意力机制实际上是得到了一个变量（value）的期望。当然，严格意义上说， α_i 并不是从概率角度定义的，这里也并不是要追求严格的统计学意义。不过这确实说明了，往往看似简单的模型背后的数学原理可能会很深刻。

6.3.5 训练

第五章已经介绍了神经网络的训练方法。其中最常用的是基于梯度的方法，即：使用一定量样本进行神经网络的前向计算，之后进行反向计算，并得到所有参数的梯度信息，再使用下面的规则进行参数更新：

$$\mathbf{w}_{step+1} = \mathbf{w}_{step} - \alpha \cdot \frac{\partial L(\mathbf{w}_{step})}{\partial \mathbf{w}_{step}} \quad (6.30)$$

其中， \mathbf{w}_{step} 表示更新前的模型参数， \mathbf{w}_{step+1} 表示更新后的模型参数， $L(\mathbf{w}_{step})$ 表示模型相对于 \mathbf{w}_{step} 的损失， $\frac{\partial L(\mathbf{w}_{step})}{\partial \mathbf{w}_{step}}$ 表示损失函数的梯度， α 是更新的步进值。也就是说，给定一定量的训练数据，不断执行公式6.30的过程。反复使用训练数据，直至模型参数达到收敛或者损失函数不再变化。通常，把公式的一次执行称为“一步”更新/训练，把访问完所有样本的训练称为“一轮”训练。

将公式6.30应用于神经机器翻译有几个基本问题需要考虑：1）损失函数的选择；2）参数初始化的策略，也就是如何设置 \mathbf{w}_0 ；3）优化策略和学习率调整策略；4）训练加速。下面对这些问题进行讨论。

损失函数

因为神经机器翻译在每个目标语位置都会输出一个概率分布，表示这个位置上不同单词出现的可能性，因此需要知道当前位置输出的分布相比于标准答案的“损失”。对于这个问题，常用的是交叉熵损失函数。令 \mathbf{y} 表示机器翻译模型输出的分布， $\hat{\mathbf{y}}$ 表示标准答案，则交叉熵损失可以被定义为 $L_{cc}(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{k=1}^{|V|} \mathbf{y}[k] \log(\hat{\mathbf{y}}[k])$ ，其中 $\mathbf{y}[k]$ 和 $\hat{\mathbf{y}}[k]$ 分别表示向量 \mathbf{y} 和 $\hat{\mathbf{y}}$ 的第 k 维， $|V|$ 表示输出向量的维度（等于词表大小）。对于一个模型输出的概率分布 $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n\}$ 和标准答案分布

$\hat{\mathbf{Y}} = \{\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \dots, \hat{\mathbf{y}}_n\}$, 损失函数可以被定义为

$$L(\mathbf{Y}, \hat{\mathbf{Y}}) = \sum_{j=1}^n L_{ce}(\mathbf{y}_j, \hat{\mathbf{y}}_j) \quad (6.31)$$

公式6.31是一种非常通用的损失函数形式, 除了交叉熵, 也可以使用其他的损失函数, 这时只需要替换 $L_{ce}(\cdot)$ 即可。这里使用交叉熵损失函数的好处在于, 它非常容易优化, 特别是与 **Softmax** 组合, 其反向传播的实现非常高效。此外, 交叉熵损失(在一定条件下)也对应了极大似然的思想, 这种方法在自然语言处理中已经被证明是非常有效的。

除了交叉熵, 很多系统也使用了面向评价的损失函数, 比如, 直接利用评价指标 BLEU 定义损失函数。不过这类损失函数往往不可微分, 因此无法直接获取梯度。这时可以引入强化学习技术, 通过策略梯度等方法进行优化。不过这类方法需要采样等手段, 这里不做重点讨论, 相关内容会在后面技术部分进行介绍。

参数初始化

神经网络的参数主要是各层中的线性变换矩阵和偏置。在训练开始时, 需要对参数进行初始化。但是, 由于神经机器翻译的网络结构复杂, 因此损失函数往往不是凸函数, 不同初始化会导致不同的优化结果。而且在大量实践中已经发现, 神经机器翻译模型对初始化方式非常敏感, 性能优异的系统往往需要特定的初始化方式。

下面以 LSTM 循环神经网络为例(见6.3.3节), 介绍机器翻译模型的初始化方法。这些方法也可以推广到 GRU 等结构。具体内容如下:

- LSTM 遗忘门偏置初始化为 1, 也就是始终选择遗忘记忆 \mathbf{c} , 这样可以有效防止初始化时 \mathbf{c} 里包含的错误信号传播到后面的所有时刻。
- 网络中的其他偏置一般都初始化为 0, 可以有效防止加入过大或过小的偏置后使得激活函数的输出跑到“饱和区”, 也就是梯度接近 0 的区域, 防止训练一开始就无法跳出局部极小的区域。
- 网络的权重矩阵 \mathbf{w} 一般使用 Xavier 参数初始化方法 [87], 可以有效稳定训练过程, 特别是对于比较“深”的网络。令 d_{in} 和 d_{out} 分别表示 \mathbf{w} 的输入和输出的维度大小, 则该方法的具体实现如下:

$$\mathbf{w} \sim U\left(-\sqrt{\frac{6}{d_{in} + d_{out}}}, \sqrt{\frac{6}{d_{in} + d_{out}}}\right) \quad (6.32)$$

其中 $U(a, b)$ 表示以 $[a, b]$ 为范围的均匀分布, 6 是固定值。

优化策略

公式6.30展示了最基本的优化策略，也被称为标准的 SGD 优化器。实际上，训练神经机器翻译模型时，还有非常多的优化器可以选择，在第五章也有详细介绍，这里考虑 Adam 优化器。Adam 通过对梯度的一阶矩估计（First Moment Estimation）和二阶矩估计（Second Moment Estimation）进行综合考虑，计算出更新步长。

表6.8从效果上对比了 Adam 和 SGD 的区别。通常，Adam 收敛的比較快，不同任务基本上可以使用一套配置进行优化，虽性能不算差，但很难达到最优效果。相反，SGD 虽能通过在不同的数据集上进行调整，来达到最优的结果，但是收敛速度慢。因此需要根据不同的需求来选择合适的优化器。若需要快得到模型的初步结果，选择 Adam 较为合适，若是需要在一个任务上得到最优的结果，选择 SGD 更为合适。

表 6.8: Adam / SGD 对比

	使用	性能
Adam	一套配置包打天下	不算差，但没到极限
SGD	换一个任务就得调	效果好

梯度裁剪

需要注意的是，训练循环神经网络时，反向传播使得网络层之间的梯度重复相乘，在网络层数过深时，如果连乘因子小于 1 可能造成梯度指数级的减少，甚至趋近于 0，导致网络无法优化，也就是梯度消失问题。当连乘因子大于 1 时，可能会导致梯度的乘积变得异常大，造成梯度爆炸的问题。在这种情况下需要使用“梯度裁剪”来防止梯度超过阈值。梯度裁剪在第五章已经介绍过，这里简单回顾一下。梯度裁剪的具体公式如下：

$$\mathbf{w}' = \mathbf{w} \cdot \frac{\gamma}{\max(\gamma, \|\mathbf{w}\|_2)}$$

(6.33)

其中 γ 是手工设定的梯度大小阈值， $\|\cdot\|_2$ 是 L2 范数， \mathbf{w}' 表示梯度裁剪后的参数。这个公式的含义在于只要梯度大小超过阈值，就按照阈值与当前梯度大小的比例进行放缩。

学习率策略

在公式6.30中， α 决定了每次参数更新时更新的步幅大小，称之为**学习率**（Learning Rate）。学习率作为基于梯度方法中的重要超参数，它决定目标函数能否收敛到较好的局部最优点以及收敛的速度。合理的学习率能够使模型快速、稳定地达到较好的状态。但是，如果学习率太小，收敛过程会很慢；而学习率太大，则模型的状态可能

会出现震荡，很难达到稳定，甚至使模型无法收敛。图6.28 对比了不同学习率对优化过程的影响。

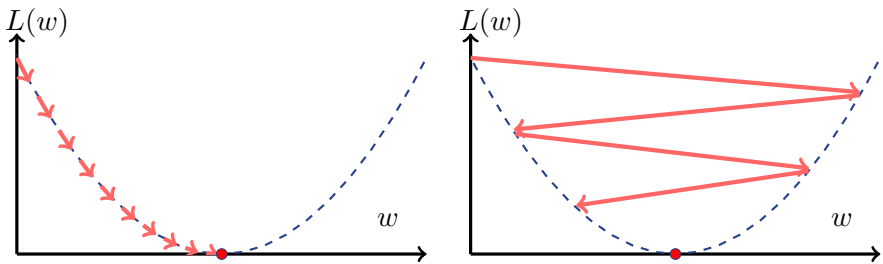


图 6.28: 学习率过小（左）vs 学习率过大（右）

不同优化器需要的学习率不同，比如 Adam 一般使用 0.001 或 0.0001，而 SGD 则在 0.1~1 之间进行挑选。在梯度下降法中，都是给定的统一的学习率，整个优化过程中都以确定的步长进行更新因此无论使用哪个优化器，为了保证训练又快又好，通常都需要根据当前的更新次数来动态调整学习率的大小。

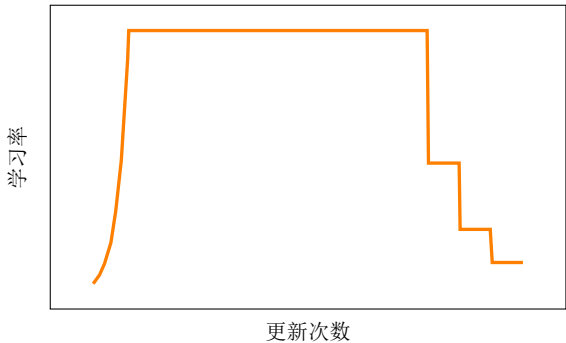


图 6.29: 学习率与更新次数的变化关系

图6.29展示了一种常用的学习率调整策略。它分为两个阶段：预热阶段和衰减阶段。模型训练初期梯度通常很大，如果直接使用较大的学习率很容易让模型陷入局部最优。学习率的预热阶段便是通过在训练初期使学习率从小到大逐渐增加来减缓在初始阶段模型“跑偏”的现象。一般来说，初始学习率太高会使得模型进入一种损失函数曲面非常不平滑的区域，进而使得模型进入一种混乱状态，后续的优化过程很难取得很好的效果。一个常用的学习率预热方法是**逐渐预热**（Gradual Warmup）。假设预热的更新次数为 T' ，初始学习率为 α_0 ，则预热阶段第 t 次更新的学习率为：

$$\alpha_t = \frac{t}{T'} \alpha_0 \quad , \quad 1 \leq t \leq T' \tag{6.34}$$

另一方面，当模型训练逐渐接近收敛的时候，使用太大学习率会很容易让模型在局部

最优解附近震荡，从而错过局部极小，因此需要通过减小学习率来调整更新的步长，以此来不断的逼近局部最优，这一阶段也称为学习率的衰减阶段。学习率衰减的方法有很多，比如指数衰减，余弦衰减等，图6.29右侧展示的是**分段常数衰减**（Piecewise Constant Decay），即每经过 m 次更新，学习率衰减为原来的 β_m ($\beta_m < 1$) 倍，其中 m 和 β_m 为经验设置的超参。

并行训练

机器翻译是自然语言处理中很“重”的任务。因为数据量巨大而且模型较为复杂，模型训练的时间往往很长。比如，使用一千万句的训练数据，性能优异的系统往往需要几天甚至一周的时间。更大规模的数据会导致训练时间更长。特别是使用多层网络同时增加模型容量时（比如增加隐层宽度时），神经机器翻译的训练会更加缓慢。对于这个问题，一个思路是从模型训练算法上进行改进。比如前面提到的 Adam 就是一种高效的训练策略。另一种思路是利用多设备进行加速，也称作分布式训练。

表 6.9: 数据并行与模型并行优缺点对比

	优点	缺点
数据并行	并行度高，理论上有多大的 batch（批次）就可以有多少个设备并行计算	模型不能大于单个设备的极限
模型并行	可以对很大的模型进行运算	只能有限并行，比如多少层就多少个设备

常用的多设备并行化加速方法有数据并行和模型并行，其优缺点的简单对比如表6.9所示。数据并行是指把同一个批次的不同样本分到不同设备上并行计算。其优点是并行度高，理论上有多大的批次就可以有多少个设备并行计算，但模型体积不能大于单个设备容量的极限。而模型并行是指把“模型”切分成若干模块后分配到不同设备上并行计算。其优点是可以对很大的模型进行运算，但只能有限并行，比如，如果按层对模型进行分割，那么有多少层就需要多少个设备，同时这两种方法可以一起使用进一步提高神经网络的训练速度。具体来说：

- **数据并行**。如果一台设备能完整放下一个神经机器翻译模型，那么数据并行可以把一个大批次均匀切分成 n 个小批次，然后分发到 n 个设备上并行计算，最后把结果汇总，相当于把运算时间变为原来的 $1/n$ ，数据并行的过程如图6.30所示。不过，需要注意的是，多设备并行需要对数据在不同设备间传输，特别是多个 GPU 的情况，设备间传输的带宽十分有限，设备间传输数据往往会造成额外的时间消耗 [320]。通常，数据并行的训练速度无法随着设备数量增加呈线性增长。不过这个问题也有很多优秀的解决方案，比如采用多个设备的异步训练，但是这些内容已经超出本章的内容，因此这里不做过多讨论。

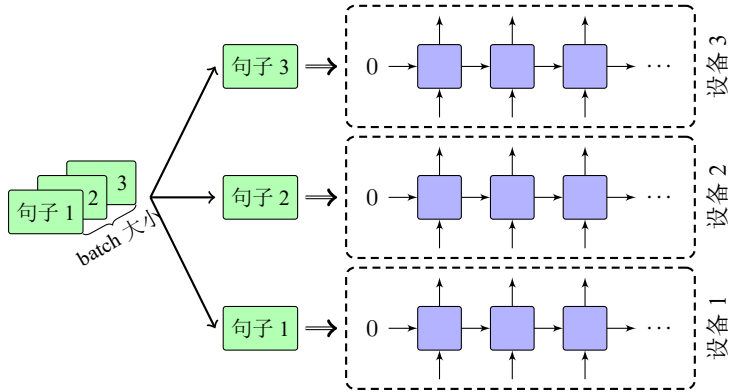


图 6.30: 数据并行过程

• **模型并行。**另一种思路是，把较大的模型分成若干小模型，之后在不同设备上训练小模型。对于循环神经网络，不同层的网络天然就是一个相对独立的模型，因此非常适合使用这种方法。比如，对于 l 层的循环神经网络，把每层都看做一个小模型，然后分发到 l 个设备上并行计算。在序列较长的时候，该方法使其运算时间变为原来的 $1/l$ 。图6.31以三层循环网络为例展示了对句子“你 很 不错。”进行模型并行的过程。其中，每一层网络都被放到了一个设备上。当模型根据已经生成的第一个词“你”预测下一个词时（图6.31(a)），同层的下一个时刻的计算和对“你”的第二层的计算就可以同时开展（图6.31(b)）。以此类推，就完成了模型的并行计算。

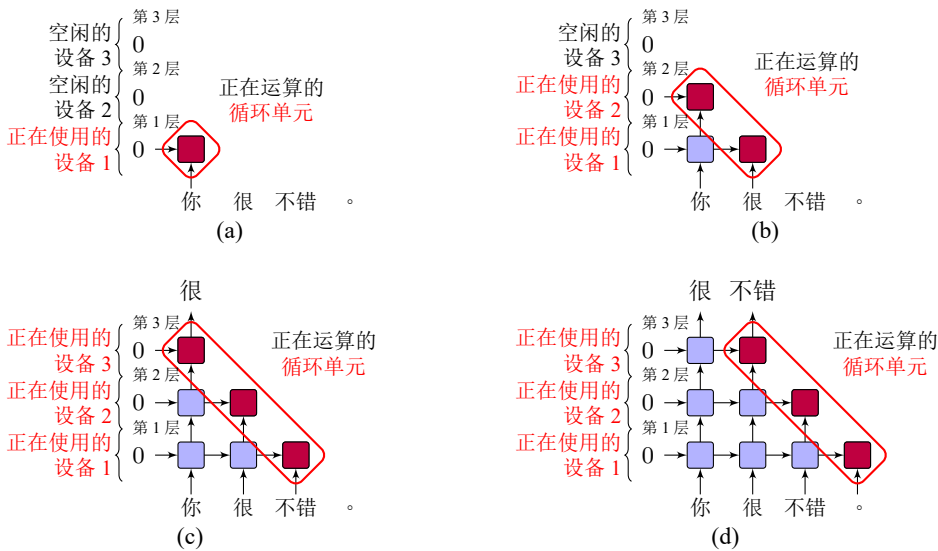




图 6.31: 一个三层循环神经网络的模型并行过程

6.3.6 推断

神经机器翻译的推断是指：利用已经训练好的模型对新的源语言句子进行翻译的过程。具体来说，首先利用编码器生成源语言句子的表示，之后利用解码器预测目标语译文。也就是，对于源语言句子 \mathbf{x} ，生成一个使翻译概率 $P(\mathbf{y}|\mathbf{x})$ 最大的目标语译文 $\hat{\mathbf{y}}$ ，如下（详细过程见6.3.1节）：

$$\begin{aligned}\hat{\mathbf{y}} &= \arg \max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) \\ &= \arg \max_{\mathbf{y}} \prod_{j=1}^n P(y_j | \mathbf{y}_{<j}, \mathbf{x})\end{aligned}\quad (6.35)$$

在具体实现时，由于当前目标语单词的生成需要依赖前面单词的生成，因此无法同时生成所有的目标语单词。理论上，可以枚举所有的 \mathbf{y} ，之后利用 $P(\mathbf{y}|\mathbf{x})$ 的定义对每个 \mathbf{y} 进行评价，然后找出最好的 \mathbf{y} 。这也被称作**全搜索**（Full Search）。但是，枚举所有的译文单词序列显然是不现实的。因此，在具体实现时，并不会访问所有可能的译文单词序列，而是用某种策略进行有效的搜索。常用的做法是自左向右逐词生成。比如，对于每一个目标语位置 j ，可以执行

$$\hat{y}_j = \arg \max_{y_j} P(y_j | \hat{\mathbf{y}}_{<j}, \mathbf{x}) \quad (6.36)$$

其中， \hat{y}_j 表示位置 j 概率最高的单词， $\hat{\mathbf{y}}_{<j} = \{\hat{y}_1, \dots, \hat{y}_{j-1}\}$ 表示已经生成的最优译文单词序列。也就是，把最优的译文看作是所有位置上最优单词的组合。显然，这是一种**贪婪搜索**（Greedy Search），因为无法保证 $\{\hat{y}_1, \dots, \hat{y}_n\}$ 是全局最优解。一种缓解这个问题的方法是，在每步中引入更多的候选。这里定义 \hat{y}_{jk} 表示在目标语第 j 个位置排名在第 k 位的单词。在每一个位置 j ，可以生成 K 个最可能的单词，而不是 1 个，这个过程可以被描述为

$$\{\hat{y}_{j1}, \dots, \hat{y}_{jk}\} = \arg \max_{\{\hat{y}_{j1}, \dots, \hat{y}_{jk}\}} P(y_j | \{\hat{\mathbf{y}}_{<j}^*, \mathbf{x}\}) \quad (6.37)$$

这里， $\{\hat{y}_{j1}, \dots, \hat{y}_{jk}\}$ 表示对于位置 j 翻译概率最大的前 K 个单词， $\{\hat{y}_{<j^*}\}$ 表示前 $j-1$ 步 top-K 单词组成的所有历史。 $\hat{y}_{<j^*}$ 可以被看作是一个集合，里面每一个元素都是一个目标语单词序列，这个序列是前面生成的一系列 top-K 单词的某种组成。 $P(y_j|\{\hat{y}_{<j^*}\}, \mathbf{x})$ 表示基于 $\{\hat{y}_{<j^*}\}$ 的某一条路径生成 y_j 的概率⁵。这种方法也被称为**束搜索**（Beam Search），意思是搜索时始终考虑一个集束内的候选。

不论是贪婪搜索还是束搜索都是一个自左向右的过程，也就是每个位置的处理需要等前面位置处理完才能执行。这是一种典型的**自回归模型**（Autoregressive Model），它通常用来描述时序上的随机过程，其中每一个时刻的结果对时序上其他部分的结果有依赖 [288]。相对应的，也有**非自回归模型**（Non-autoregressive Model），它消除了不同时刻结果之间的直接依赖 [96]。由于自回归模型是当今神经机器翻译主流的推断方法，这里仍以自回归的贪婪搜索和束搜索为基础进行讨论。

贪婪搜索

图6.32展示了一个基于贪婪方法的神经机器翻译解码过程。每一个时间步的单词预测都依赖于其前一步单词的生成。在解码第一个单词时，由于没有之前的单词信息，会用 <sos> 进行填充，作为起始的单词，且会用一个零向量（可以理解为没有之前时间步的信息）表示第 0 步的中间层状态。

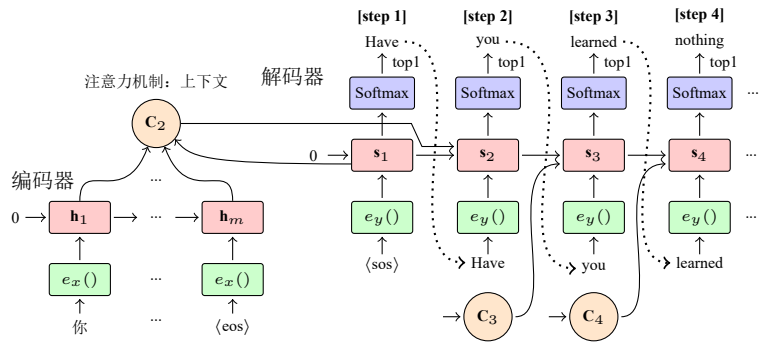


图 6.32: 基于贪婪方法的解码过程

解码端的每一步 Softmax 层会输出所有单词的概率，由于是基于贪心的方法，这里会选择概率最大（top-1）的单词作为输出。这个过程可以参考图6.33的内容。我们选择分布中概率最大的单词“Have”作为得到的第一个单词，并再次送入解码器，作为第二步的输入同时预测下一个单词。以此类推，直到生成句子的终止符为止，就得到了完整的译文。

贪婪搜索的优点在于速度快。在对翻译速度有较高要求的场景中，贪婪搜索是

⁵严格来说， $P(y_j|\hat{y}_{<j^*})$ 不是一个准确的数学表达，这里通过这种写法强调 y_j 是由 $\{\hat{y}_{<j^*}\}$ 中的某个译文单词序列作为条件生成的。

一种十分有效的对系统加速的方法。而且贪婪搜索的原理非常简单，易于快速原型。不过，由于每一步只保留一个最好的局部结果，贪婪搜索往往会带来翻译品质上的损失。

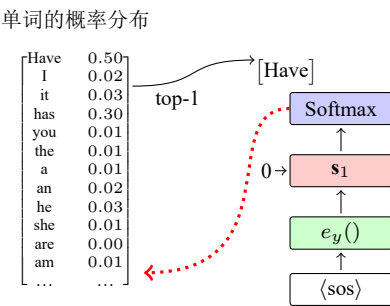


图 6.33: 解码第一个位置输出的单词概率分布（“Have” 的概率最高）

束搜索

束搜索是一种启发式图搜索算法。相比于全搜索，它可以减少搜索所占用的空间和时间，在每一步扩展的时候，剪掉一些质量比较差的结点，保留下一些质量较高的结点。具体到机器翻译任务，对于每一个目标语位置，束搜索选择了概率最大的前 K 个单词进行扩展（其中 K 叫做束宽度，或简称为束宽）。如图6.34所示，假设 $\{y_1, y_2, \dots, y_n\}$ 表示生成的目标语序列，且 $K = 3$ ，则束搜索的具体过程为：在预测第一个位置时，可以通过模型得到 y_1 的概率分布，选取概率最大的前 3 个单词作为候选结果（假设分别为 “have”，“has”，“it”）。在预测第二个位置的单词时，模型针对已经得到的三个候选结果（“have”，“has”，“it”）计算第二个单词的概率分布。例如，可以在将 “have” 作为第二步的输入，计算 y_2 的概率分布。此时，译文序列的概率为

$$\begin{aligned} P(y_2, y_1 | \mathbf{x}) &= P(y_2, \text{“have”} | \mathbf{x}) \\ &= P(y_2 | \text{“have”}, \mathbf{x}) \cdot P(\text{“have”} | \mathbf{x}) \end{aligned} \tag{6.38}$$

类似的,对 “has” 和 “it” 进行同样的操作, 分别计算得到 $P(y_2, \text{“have”} | \mathbf{x})$, $P(y_2, \text{“has”} | \mathbf{x})$, $P(y_2, \text{“it”} | \mathbf{x})$, 因为 y_2 对应 $|V|$ 种可能, 总共可以得到 $3 \times |V|$ 种结果。然后从中选取使序列概率 $P(y_2, y_1 | \mathbf{x})$ 最大的前三个 y_2 作为新的输出结果, 这样便得到了前两个位置的 top-3 译文。在预测其他位置时也是如此, 不断重复此过程直到推断结束。可以看到, 束搜索的搜索空间大小与束宽度有关, 也就是: 束宽度越大, 搜索空间越大, 更有可能搜索到质量更高的译文, 但同时搜索会更慢。束宽度等于 3, 意味着每次只考虑三个最有可能的结果, 贪婪搜索实际上便是集束宽度为 1 的情况。在神经机器翻译系统实现中, 一般束宽度设置在 4~8 之间。

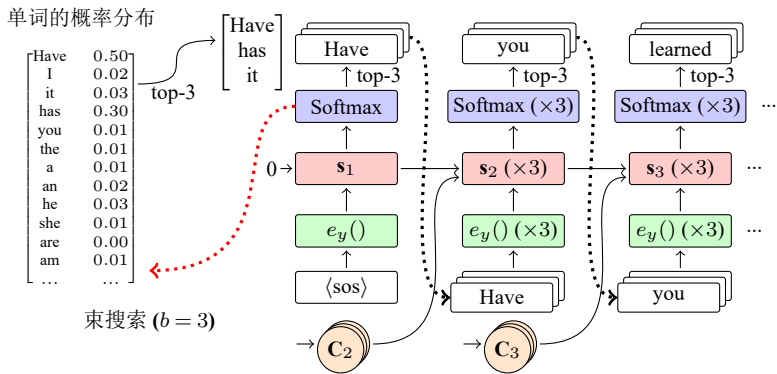


图 6.34: 束搜索过程

长度惩罚

这里用 $P(\mathbf{y}|\mathbf{x}) = \prod_{j=1}^n P(y_j|\mathbf{y}_{<j}, \mathbf{x})$ 作为翻译模型。直接实现这个公式有一个明显的缺点：当句子过长时乘法运算容易产生溢出，也就是多个数相乘可能会产生浮点数无法表示的运算结果。为了解决这个问题，可以利用对数操作将乘法转换为加法，得到新的概率公式： $\log P(\mathbf{y}|\mathbf{x}) = \sum_{j=1}^n \log P(y_j|\mathbf{y}_{<j}, \mathbf{x})$ ，对数函数不会改变函数的单调性，因此在具体实现时，通常用 $\log P(\mathbf{y}|\mathbf{x})$ 表示句子的得分，而不用 $P(\mathbf{y}|\mathbf{x})$ 。

不管是使用 $P(\mathbf{y}|\mathbf{x})$ 还是 $\log P(\mathbf{y}|\mathbf{x})$ 计算句子得分，还面临两个问题：

- $P(\mathbf{y}|\mathbf{x})$ 的范围是 $[0,1]$ ，如果句子过长，那么句子的得分就是很多个小于 1 的数相乘，或者说取 \log 之后很多个小于 0 的数相加。这也就是说，句子的得分会随着长度的增加而变小，即模型倾向于生成短句子。
- 模型本身并没有考虑每个源语言单词被使用的程度，比如一个单词可能会被翻译很多“次”。这个问题在统计机器翻译中并不存在，因为所有词在翻译中必须被“覆盖”到。但是早期的神经机器翻译模型没有所谓覆盖度的概念，因此也无法保证每个单词被翻译的“程度”是合理的 [172, 286]。

为了解决上面提到的问题，可以使用其他特征与 $\log P(\mathbf{y}|\mathbf{x})$ 一起组成新的模型得分 $\text{score}(\mathbf{y}, \mathbf{x})$ 。针对模型倾向于生成短句子的问题，常用的做法是引入惩罚机制。比如，可以定义一个惩罚因子，形式如下：

$$\text{lp}(\mathbf{y}) = \frac{(5 + |\mathbf{y}|)^\alpha}{(5 + 1)^\alpha} \tag{6.39}$$

其中， $|\mathbf{y}|$ 代表已经得到的译文长度， α 是一个固定的常数，用于控制惩罚的强度。同时在计算句子得分时，额外引入表示覆盖度的因子，如下：

$$\text{cp}(\mathbf{y}, \mathbf{x}) = \beta \cdot \sum_{i=1}^{|\mathbf{x}|} \log \left(\min \left(\sum_j \alpha_{ij}, 1 \right) \right) \tag{6.40}$$

$\text{cp}(\cdot)$ 会惩罚把某些源语单词对应到很多目标语单词的情况（覆盖度），被覆盖的程度用 $\sum_j^{|y|} \alpha_{ij}$ 度量。 β 也是需要经验性设置的超参数，用于对覆盖度惩罚的强度进行控制。

最终，模型得分定义如下：

$$\text{score}(\mathbf{y}, \mathbf{x}) = \frac{\log P(\mathbf{y}|\mathbf{x})}{\text{lp}(\mathbf{y})} + \text{cp}(\mathbf{y}, \mathbf{x}) \tag{6.41}$$

显然，当目标语 y 过短时， $\text{lp}(\mathbf{y})$ 的值越小，因为 $\log P(\mathbf{y}|\mathbf{x})$ 是负数，所以句子得分 $\text{score}(\mathbf{y}, \mathbf{x})$ 越小。也就是说，模型会惩罚译文过短的结果。当覆盖度较高时，同样会使得分变低。通过这样的惩罚机制，使模型得分更为合理，从而帮助模型选择出质量更高的译文。

6.3.7 实例-GNMT

循环神经网络在机器翻译中有很多成功的应用，比如、RNNSearch[7]、Nematus[249] 等系统就被很多研究者作为实验系统。在众多基于循环神经网络的系统中，Google’s Neural Machine Translation System（GNMT）系统是非常成功的一个 [312]。GNMT 是谷歌 2016 年发布的神经机器翻译系统。当时，神经机器翻译有三个弱点：训练和推理速度较慢、在翻译稀有单词上缺乏鲁棒性和有时无法完整翻译源语言句子中的所有单词。GNMT 的提出有效的缓解了上述问题。

GNMT 使用了编码器-解码器结构，构建了一个 8 层的深度网络，每层网络均由 LSTM 组成，且在编码器-解码器之间使用了多层注意力连接。其结构如图6.35，编码器只有最下面 2 层为双向 LSTM。GNMT 在束搜索中也加入了长度惩罚和覆盖度因子来确保输出高质量的翻译结果（公式6.41）。

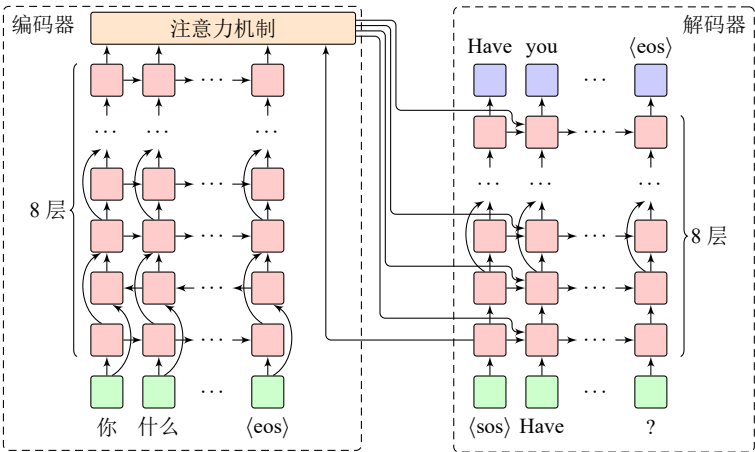


图 6.35: GNMT 结构

实际上，GNMT 的主要贡献在于集成了多种优秀的技术，而且在大规模数据上

证明了神经机器翻译的有效性。在引入注意力机制之前，神经机器翻译在较大规模的任务上的性能弱于统计机器翻译。加入注意力机制和深层网络后，神经机器翻译性能有了很大的提升。在英德和英法的任务中，GNMT 的 BLEU 值不仅超过了当时优秀的神经机器翻译系统 RNNSearch 和 LSTM（6 层），还超过了当时处于领导地位的基于短语的统计机器翻译系统（PBMT）（表6.10）。相比谷歌的基于短语的系统，在人工评价中，该模型能将翻译错误平均减少了 60%。这一结果也充分表明了神经机器翻译带来的巨大性能提升。

表 6.10: GNMT 与其他翻译模型对比 [312]

#	BLEU		CPU decoding time
	EN-DE	EN-FR	
PBMT	20.7	37.0	-
RNNSearch	16.5	-	-
LSTM(6 layers)	-	31.5	-
Deep-Att	20.6	37.7	-
GNMT	24.6	39.0	0.2s per sentence

6.4 Transformer

编码器-解码器框架提供了一个非常灵活的机制，因为开发者只需要设计编码器和解码器的结构就能完成机器翻译。但是，架构的设计是深度学习中最具挑战的工作，优秀的架构往往需要长时间的探索和大量的实验验证，而且还需要一点点“灵感”。

前面介绍的基于循环神经网络的翻译模型和注意力机制就是研究人员通过长期的实践发现的神经网络架构。除了神经机器翻译，它们也被广泛地应用于语音处理、图像处理等领域。虽然循环神经网络很强大，但是人们也发现了一些弊端。一个突出的问题是，循环神经网络每个循环单元都有向前依赖性，也就是当前时间步的处理依赖前一时间步处理的结果。这个性质可以使序列的“历史”信息不断被传递，但是也造成模型运行效率的下降。特别是对于自然语言处理任务，序列往往较长，无论是传统的 RNN 结构，还是更为复杂的 LSTM 结构，都需要很多次循环单元的处理才能够捕捉到单词之间的长距离依赖。由于需要多个循环单元的处理，距离较远的两个单词之间的信息传递变得很复杂。

针对这些问题，谷歌的研究人员提出了一种全新的模型 —— Transformer[288]。与循环神经网络等传统模型不同，Transformer 模型仅仅使用一种被称作自注意力机制的方法和标准的前馈神经网络，完全不依赖任何循环单元或者卷积操作。自注意力机制的优点在于可以直接对序列中任意两个单元之间的关系进行建模，这使得长距离依赖等问题可以更好地被求解。此外，自注意力机制非常适合在 GPU 上进行并行化，因此模型训练的速度更快。表6.11对比了 RNN、CNN、Transformer 三种模型

的时间复杂度。

表 6.11: RNN、CNN、Transformer 的对比 [288] (n 表示序列长度, d 表示隐层大小, k 表示卷积核大小)

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$

Transformer 在被提出之后,很快就席卷了整个自然语言处理领域。实际上,Transformer 也可以当作一种表示模型,因此也被大量地使用在自然语言处理的其他领域,甚至图像处理和语音处理中也能看到它的影子。比如,目前非常流行的 BERT 等预训练模型就是基于 Transformer。表6.12展示了 Transformer 在 WMT 英德和英法机器翻译任务上的性能。它能用更少的计算量(FLOPS)达到比其他模型更好的翻译品质⁶。

表 6.12: 不同翻译模型性能对比 [288]

#	BLEU		Training Cost (FLOPs)
	EN-DE	EN-FR	
GNMT+RL	24.6	39.92	1.4×10^{20}
ConvS2S	25.16	40.46	1.5×10^{20}
MoE	26.03	40.56	1.2×10^{20}
Transformer (Big)	28.4	41.8	2.3×10^{19}

注意,Transformer 并不简单等同于自注意力机制。Transformer 模型还包含了很多优秀的技术,比如:多头注意力、新的训练学习率调整策略等等。这些因素一起组成了真正的 Transformer。下面就一起看一看自注意力机制和 Transformer 是如何工作的。

6.4.1 自注意力模型

首先,再回顾一下循环神经网络处理文字序列的过程。如图6.36所示,对于单词序列 $\{w_1, \dots, w_m\}$, 处理第 m 个单词 w_m 时(绿色方框部分), 需要输入前一时刻的信息(即处理单词 w_{m-1}), 而 w_{m-1} 又依赖于 w_{m-2} , 以此类推。也就是说,如果想建立 w_m 和 w_1 之间的关系, 需要 $m-1$ 次信息传递。对于长序列来说, 词汇之间信息传递距离过长会导致信息在传递过程中丢失, 同时这种按顺序建模的方式也使得系统对序列的处理十分缓慢。

⁶FLOPS = floating-point operations per second, 即每秒浮点运算次数。它是度量计算机运算规模的常用单位

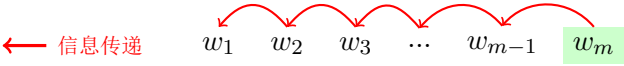


图 6.36: 循环神经网络中单词之间的依赖关系

那么能否摆脱这种顺序传递信息的方式，直接对不同位置单词之间的关系进行建模，即将信息传递的距离拉近为 1？**自注意力机制**（Self-Attention）的提出便有效解决了这个问题 [178]。图6.37给出了自注意力机制对序列进行建模的示例。对于单词 w_m ，自注意力机制直接建立它与前 $m - 1$ 个单词之间的关系。也就是说， w_m 与序列中所有其他单词的距离都是 1。这种方式很好地解决了长距离依赖问题，同时由于单词之间的联系都是相互独立的，因此也大大提高了模型的并行度。

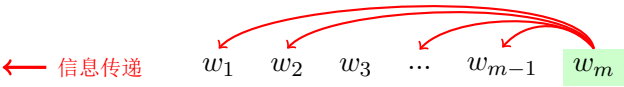


图 6.37: 自注意力机制中单词之间的依赖关系

自注意力机制也可以被看做是一个序列表示模型。比如，对于每个目标位置 j ，都生成一个与之对应的源语言句子表示，它的形式为： $\mathbf{C}_j = \sum_i \alpha_{i,j} \mathbf{h}_i$ ，其中 \mathbf{h}_i 为源语言句子每个位置的表示结果， $\alpha_{i,j}$ 是目标位置 j 对 \mathbf{h}_i 的注意力权重。而自注意力机制不仅可以处理两种语言句子之间的对应，它也可以对单语句子进行表示。以源语言句子为例，同时参考6.3.4节的内容，自注意力机制将序列中每个位置的表示 \mathbf{h}_i 看作 query（查询），并且将所有位置的表示看作 key（键）和 value（值）。自注意力模型通过计算当前位置与所有位置的匹配程度，也就是在注意力机制中提到的注意力权重，来对各个位置的 value 进行加权求和。得到的结果可以被看作是在这个句子中当前位置的抽象表示。这个过程，可以叠加多次，形成多层注意力模型，对输入序列中各个位置进行更深层的表示。

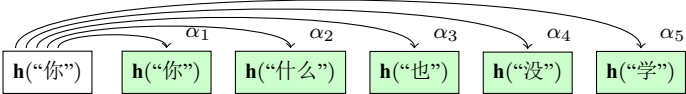


图 6.38: 自注意力计算实例

举个例子，如图6.38所示，一个汉语句子里包含 5 个词。这里，用 \mathbf{h} (“你”) 表示“你”当前的表示结果。如果把“你”看作目标，这时 query 就是 \mathbf{h} (“你”)，key 和 value 是图中所有位置的表示，即： \mathbf{h} (“你”)、 \mathbf{h} (“什么”)、 \mathbf{h} (“也”)、 \mathbf{h} (“没”)、 \mathbf{h} (“学”)。在自注意力模型中，首先计算 query 和 key 的相关度，这里用 α_i 表示 \mathbf{h} (“你”) 和位置 i 的表示之间的相关性。然后，把 α_i 作为权重，对不同位置上的 value 进行加权求和。最

终，得到新的表示结果 $\tilde{\mathbf{h}}$ (“你”)：

$$\tilde{\mathbf{h}}(\text{“你”}) = \alpha_1 \mathbf{h}(\text{“你”}) + \alpha_2 \mathbf{h}(\text{“什么”}) + \alpha_3 \mathbf{h}(\text{“也”}) + \alpha_4 \mathbf{h}(\text{“没”}) + \alpha_5 \mathbf{h}(\text{“学”}) \tag{6.42}$$

同理，也可以用同样的方法处理这个句子中的其他单词。可以看出，在注意力机制中，并不是使用类似于循环神经网络的记忆能力去访问历史信息。序列中所有单词之间的信息都是通过同一种操作（query 和 key 的相关度）进行处理。这样，表示结果 $\tilde{\mathbf{h}}$ (“你”) 在包含 “你” 这个单词的信息的同时，也包含了序列中其他词的信息。也就是，序列中每一个位置的表示结果中，都包含了其他位置的信息。从这个角度说， $\tilde{\mathbf{h}}$ (“你”) 已经不再是单词 “你” 自身的表示结果，而是一种在单词 “你” 的位置上的全局信息的表示。

通常，也把生成 $\{\tilde{\mathbf{h}}(\mathbf{w}_i)\}$ 的过程称为**特征提取**，而实现这个过程模型被称为特征提取器。循环神经网络、自注意力模型都是典型的特征提取器。特征提取是神经机器翻译系统的关键步骤，在随后的内容中可以看到自注意力模型是一个非常适合机器翻译任务的特征提取器。

6.4.2 Transformer 架构

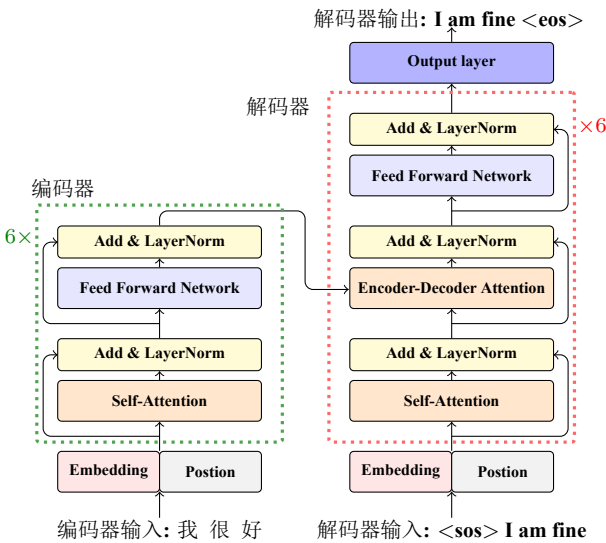


图 6.39: Transformer 结构

图6.39展示了经典的 Transformer 结构。解码器由若干层组成（绿色虚线框就代表一层）。每一层（layer）的输入都是一个向量序列，输出是同样大小的向量序列，而 Transformer 层的作用是对输入进行进一步的抽象，得到新的表示结果。不过这里的层并不是指单一的神经网络结构，它里面由若干不同的模块组成，包括：

- **自注意力子层**（Self-attention Sub-layer）：使用自注意力机制对输入的序列进行

新的表示；

- **前馈神经网络子层**（Feed-forward Sub-layer）：使用全连接的前馈神经网络对输入向量序列进行进一步变换；
- **残差连接**（Residual Connection，标记为“Add”）：对于自注意力子层和前馈神经网络子层，都有一个从输入直接到输出的额外连接，也就是一个跨子层的直连。残差连接可以使深层网络的信息传递更为有效；
- **层正则化**（Layer Normalization）：自注意力子层和前馈神经网络子层进行最终输出之前，会对输出的向量进行层正则化，规范结果向量取值范围，这样易于后面进一步的处理。

以上操作就构成了 Transformer 的一层，各个模块执行的顺序可以简单描述为：Self-Attention → Residual Connection → Layer Normalization → Feed Forward Network → Residual Connection → Layer Normalization。编码器可以包含多个这样的层，比如，可以构建一个六层编码器，每层都执行上面的操作。最上层的结果作为整个编码的结果，会被传入解码器。

解码器的结构与编码器十分类似。它也是由若干层组成，每一层包含编码器中的所有结构，即：自注意力子层、前馈神经网络子层、残差连接和层正则化模块。此外，为了捕捉源语言的信息，解码器又引入了一个额外的**编码-解码注意力子层**（Encoder-decoder Attention Sub-layer）。这个新的子层，可以帮助模型使用源语言句子的表示信息生成目标语不同位置的表示。编码-解码注意力子层仍然基于自注意力机制，因此它和自注意力子层的结构是相同的，只是 query、key、value 的定义不同。比如，在解码端，自注意力子层的 query、key、value 是相同的，它们都等于解码端每个位置的表示。而在编码-解码注意力子层中，query 是解码端每个位置的表示，此时 key 和 value 是相同的，等于编码端每个位置的表示。图6.40给出了这两种不同注意力子层输入的区别。

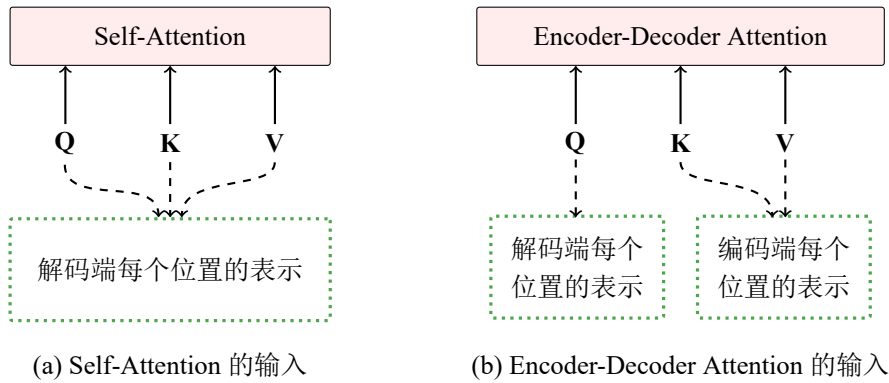


图 6.40: 注意力模型的输入（自注意力子层 vs 编码-解码注意力子层）

此外，编码端和解码端都有输入的词序列。编码端的词序列输入是为了对其进

行表示，进而解码端能从编码端访问到源语言句子的全部信息。解码端的词序列输入是为了进行目标语的生成，本质上它和语言模型是一样的，在得到前 $n-1$ 个单词的情况下输出第 n 个单词。除了输入的词序列的词嵌入，Transformer 中也引入了位置嵌入，以表示每个位置信息。原因是，自注意力机制没有显性地对位置进行表示，因此也无法考虑词序。在输入中引入位置信息可以让自注意力机制间接地感受到每个词的位置，进而保证对序列表示的合理性。最终，整个模型的输出由一个 Softmax 层完成，它和循环神经网络中的输出层是完全一样的（6.3.2节）。

在进行更详细的介绍前，先利用图6.39简单了解一下 Transformer 模型是如何进行翻译的。首先，Transformer 将源语“我 很 好”的**词嵌入**（Word Embedding）融合**位置编码**（Position Embedding）后作为输入。然后，编码器对输入的源语言句子进行逐层抽象，得到包含丰富的上下文信息的源语表示并传递给解码器。解码器的每一层，使用自注意力子层对输入解码端的表示进行加工，之后再使用编码-解码注意力子层融合源语言句子的表示信息。就这样逐词生成目标语译文单词序列。解码器的每个位置的输入是当前单词（比如，“I”），而这个位置输出是下一个单词（比如，“am”），这个设计和标准的神经语言模型是完全一样的。

了解到这里，可能大家还有很多疑惑，比如，什么是位置编码？Transformer 的自注意力机制具体是怎么进行计算的，其结构是怎样的？Add& LayerNorm 又是什么？等等。下面就一一展开介绍。

6.4.3 位置编码

在使用循环神经网络进行序列的信息提取时，每个时刻的运算都要依赖前一个时刻的输出，具有一定的时序性，这也与语言具有顺序的特点相契合。而采用自注意力机制对源语言和目标语言序列进行处理时，直接对当前位置和序列中的任意位置进行建模，忽略了词之间的顺序关系，例如图6.41中两个语义不同的句子，通过自注意力得到的表示 \tilde{h} (“机票”) 却是相同的。

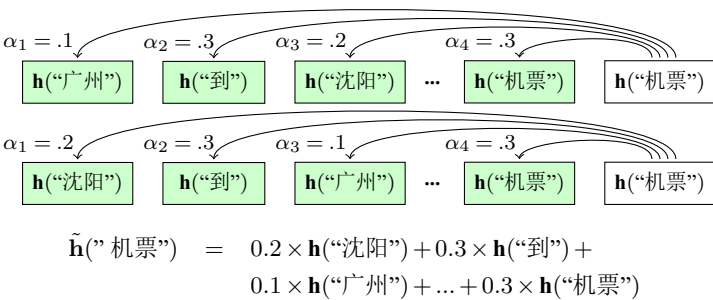


图 6.41: “机票”的更进一步抽象表示 \tilde{h} 的计算

为了解决这个问题，Transformer 在原有的词向量输入基础上引入了位置编码，来表示单词之间的顺序关系。位置编码在 Transformer 结构中的位置如图6.42，它是

Transformer 成功的一个重要因素。

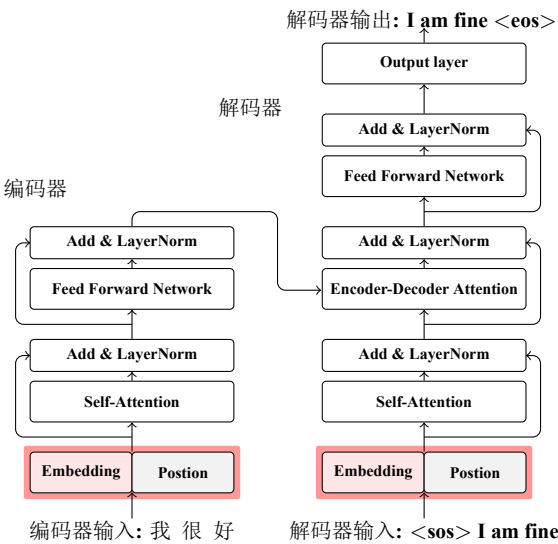


图 6.42: Transformer 输入与位置编码

位置编码的计算方式有很多种，Transformer 使用不同频率的正余弦函数：

$$\text{PE}(\text{pos}, 2i) = \sin\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right) \tag{6.43}$$

$$\text{PE}(\text{pos}, 2i + 1) = \cos\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right) \tag{6.44}$$

式中 $\text{PE}(\cdot)$ 表示位置编码的函数， pos 表示单词的位置， i 代表位置编码向量中的第几维， d_{model} 是 Transformer 的一个基础参数，表示每个位置的隐层大小。因为，正余弦函数的编码各占一半，因此当位置编码的维度为 512 时， i 的范围是 0-255。在 Transformer 中，位置编码的维度和词嵌入向量的维度相同（均为 d_{model} ），模型通过将二者相加作为模型输入，如图6.43所示。

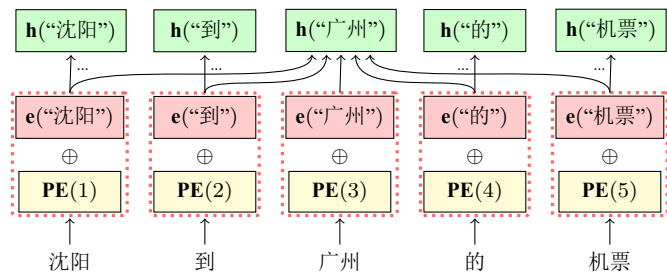


图 6.43: 位置编码与词编码的组合

那么为什么通过这种计算方式可以很好的表示位置信息？有几方面原因。首先，

正余弦函数是具有上下界的周期函数，用正余弦函数可将长度不同的序列的位置编码的范围都固定到 $[-1,1]$ ，这样在与词的编码进行相加时，不至于产生太大差距。另外位置编码的不同维度对应不同的正余弦曲线，这为多维的表示空间赋予一定意义。最后，根据三角函数的性质：

$$\begin{aligned}\sin(\alpha + \beta) &= \sin\alpha \cdot \cos\beta + \cos\alpha \cdot \sin\beta \\ \cos(\alpha + \beta) &= \cos\alpha \cdot \cos\beta - \sin\alpha \cdot \sin\beta\end{aligned}\tag{6.45}$$

可以得到 “ $pos + k$ ” 的位置编码为：

$$\begin{aligned}\text{PE}(pos + k, 2i) &= \text{PE}(pos, 2i) \times \text{PE}(k, 2i + 1) + \\ &\quad \text{PE}(pos, 2i + 1) \times \text{PE}(k, 2i)\end{aligned}\tag{6.46}$$

$$\begin{aligned}\text{PE}(pos + k, 2i + 1) &= \text{PE}(pos, 2i + 1) \times \text{PE}(k, 2i + 1) - \\ &\quad \text{PE}(pos, 2i) \times \text{PE}(k, 2i)\end{aligned}\tag{6.47}$$

即对于任意固定的偏移量 k ， $\text{PE}(pos + k)$ 能被表示成 $\text{PE}(pos)$ 的线性函数，换句话说，位置编码可以表示词之间的距离。在实践中发现，位置编码对 Transformer 系统的性能有很大影响。对其进行改进也会带来进一步的性能提升 [255]。

6.4.4 基于点乘的注意力机制

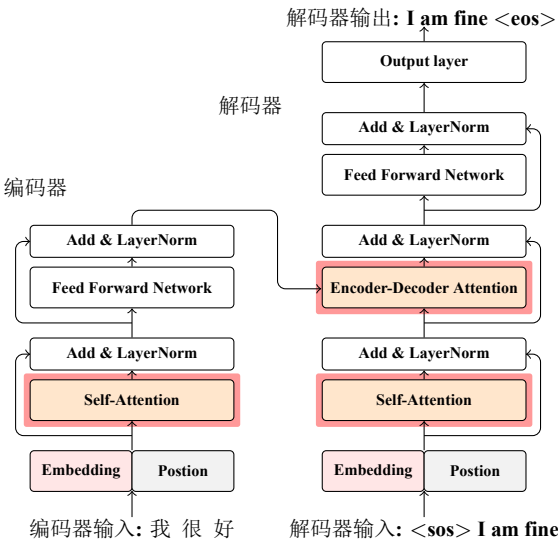


图 6.44: 自注意力机制在模型中的位置

Transformer 模型摒弃了循环单元和卷积等结构，完全基于注意力机制来构造模型，其中包含着大量的注意力计算。比如，可以通过自注意力机制对源语言和目标

语言序列进行信息提取，并通过编码-解码注意力对双语句对之间的关系进行提取。图6.44中红色方框部分是 Transformer 中使用自注意力机制的模块。

在6.4.1节中已经介绍，自注意力机制中，至关重要的是获取相关性系数，也就是在融合不同位置的表示向量时各位置的权重。在6.3节基于循环神经网络的机器翻译模型中，注意力机制的相关性系数有很多种计算方式，如余弦相似度等。而在 Transformer 模型中，则采用了一种基于点乘的方法来计算相关性系数。这种方法也称为**点乘注意力**（Scaled Dot-Product Attention）机制。它的运算并行度高，同时并不消耗太多的存储空间。

具体来看，在注意力机制的计算过程中，包含三个重要的参数，分别是 Query，Key 和 Value。在下面的描述中，分别用 \mathbf{Q} ， \mathbf{K} ， \mathbf{V} 对它们进行表示，其中 \mathbf{Q} 和 \mathbf{K} 的维度为 $L \times d_k$ ， \mathbf{V} 的维度为 $L \times d_v$ 。这里， L 为序列的长度， d_k 和 d_v 分别表示每个 Key 和 Value 的大小，通常设置为 $d_k = d_v = d_{model}$ 。

在自注意力机制中， \mathbf{Q} 、 \mathbf{K} 、 \mathbf{V} 都是相同的，对应着源语言或目标语言的表示。而在编码-解码注意力机制中，由于要对双语之间的信息进行建模，因此，将目标语每个位置的表示视为编码-解码注意力机制的 \mathbf{Q} ，源语言句子的表示视为 \mathbf{K} 和 \mathbf{V} 。

在得到 \mathbf{Q} ， \mathbf{K} 和 \mathbf{V} 后，便可以进行注意力机制的运算，这个过程可以被形式化为：

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \text{Mask}\right)\mathbf{V} \quad (6.48)$$

首先，通过对 \mathbf{Q} 和 \mathbf{K} 的转置进行点乘操作，计算得到一个维度大小为 $L \times L$ 的相关性矩阵，即 $\mathbf{Q}\mathbf{K}^T$ ，它表示一个序列上任意两个位置的相关性。再通过系数 $1/\sqrt{d_k}$ 进行放缩操作，放缩可以尽量减少相关性矩阵的方差，具体体现在运算过程中实数矩阵中的数值不会过大，有利于模型训练。

在此基础上，通过对相关性矩阵累加一个掩码矩阵，来屏蔽掉矩阵中的无用信息。比如，在编码端对句子的补齐，在解码端则屏蔽掉未来信息，这一部分内容将在下一小节进行详细介绍。随后，使用 Softmax 函数对相关性矩阵在行的维度上进行归一化操作，这可以理解为对第 i 行进行归一化，结果对应了 \mathbf{V} 中的不同位置上向量的注意力权重。对于 value 的加权求和，可以直接用相关性系数和 \mathbf{V} 进行矩阵乘法得到，即 $\text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \text{Mask}\right)$ 和 \mathbf{V} 进行矩阵乘。最终得到自注意力的输出，它和输入的 \mathbf{V} 的大小是一模一样的。图6.45展示了点乘注意力计算的全过程。

下面举个简单的例子介绍点乘注意力的具体计算过程。如图6.46所示，用黄色、蓝色和橙色的矩阵分别表示 \mathbf{Q} 、 \mathbf{K} 和 \mathbf{V} 。 \mathbf{Q} 、 \mathbf{K} 和 \mathbf{V} 中的每一个小格都对应一个单词在模型中的表示（即一个向量）。首先，通过点乘、放缩、掩码等操作得到相关性矩阵，即粉色部分。其次，将得到的中间结果矩阵（粉色）的每一行使用 Softmax 激活函数进行归一化操作，得到最终的权重矩阵，也就是图中的红色矩阵。红色矩阵中的每一行都对应一个注意力分布。最后，按行对 \mathbf{V} 进行加权求和，便得到了每个

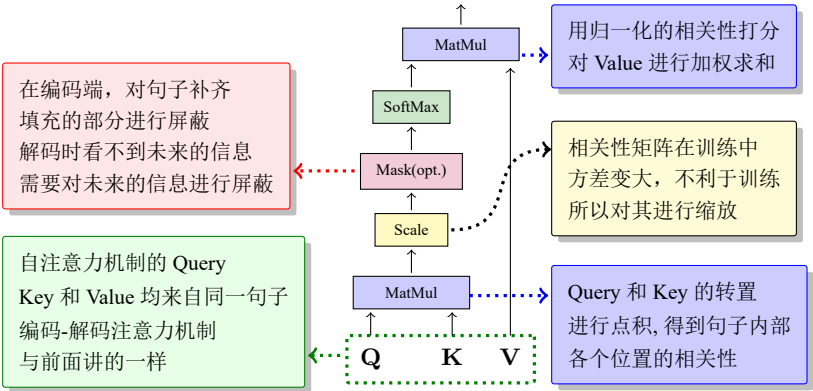


图 6.45: 点乘注意力模型

单词通过点乘注意力机制计算得到的表示。这里面，主要的计算消耗是两次矩阵乘法，即 Q 与 K^T 的乘法、相关性矩阵和 V 的乘法。这两个操作都可以在 GPU 上高效地完成，因此可以一次性计算出序列中所有单词之间的注意力权重，并完成所有位置表示的加权求和过程，这样大大提高了模型的计算速度。

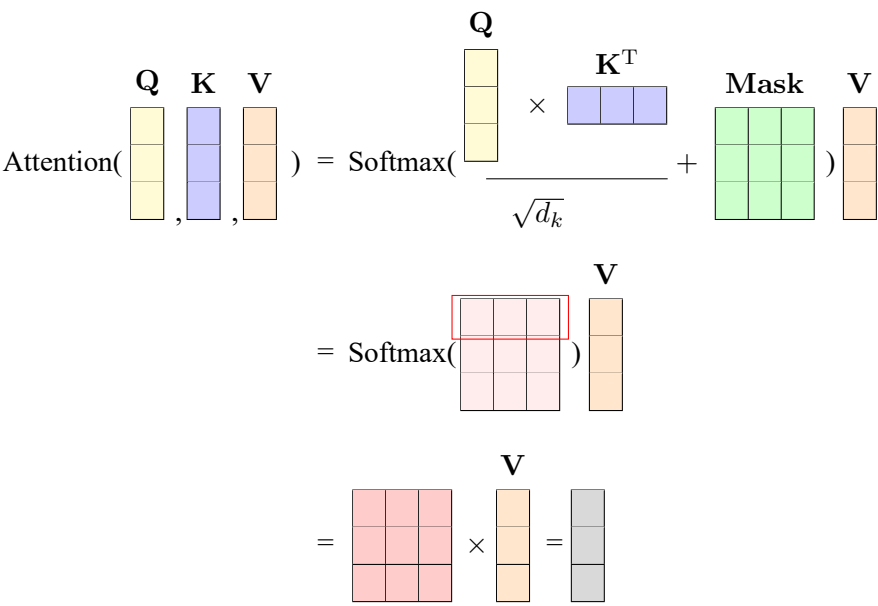


图 6.46: 式6.48的执行过程示例

6.4.5 掩码操作

在公式6.48中提到了 Mask（掩码），它的目的是对向量中某些值进行掩盖，避免无关位置的数值对运算造成影响。Transformer 中的 Mask 主要应用在注意力机制中的相关性系数计算，具体方式是在相关性系数矩阵上累加一个 Mask 矩阵。该矩阵在

需要 Mask 的位置的值为负无穷 $-\text{inf}$ （具体实现时是一个非常小的数，比如 $-1\text{e-}9$ ），其余位置为 0，这样在进行了 Softmax 归一化操作之后，被掩码掉的位置计算得到的权重便近似为 0，也就是说对无用信息分配的权重为 0，从而避免了其对结果产生影响。Transformer 包含两种 Mask：

- **Padding Mask**。在批量处理多个样本时（训练或解码），由于要对源语言和目标语言的输入进行批次化处理，而每个批次内序列的长度不一样，为了方便对批次内序列进行矩阵表示，需要进行对齐操作，即在较短的序列后面填充 0 来占位（padding 操作）。而这些填充的位置没有意义，不参与注意力机制的计算，因此，需要进行 Mask 操作，屏蔽其影响。
- **Future Mask**。对于解码器来说，由于在预测的时候是自左向右进行的，即第 t 时刻解码器的输出只能依赖于 t 时刻之前的输出。且为了保证训练解码一致，避免在训练过程中观测到目标语端每个位置未来的信息，因此需要对未来信息进行屏蔽。具体的做法是：构造一个上三角值全为 $-\text{inf}$ 的 Mask 矩阵，也就是说，在解码端计算中，在当前位置，通过 Future Mask 把序列之后的信息屏蔽掉了，避免了 t 之后的位置对当前的计算产生影响。图6.47给出了一个具体的实例。

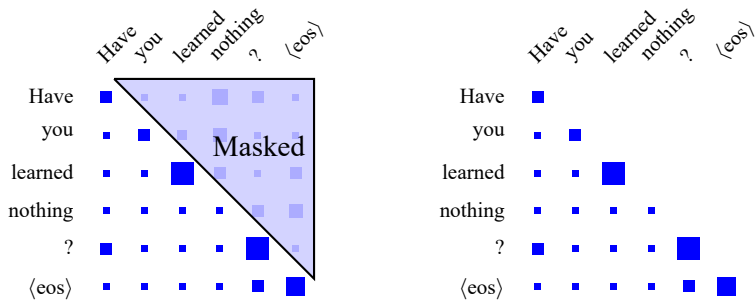


图 6.47: Transformer 中对于未来位置进行的屏蔽的 Mask 实例

6.4.6 多头注意力

Transformer 中使用的另一项重要技术是**多头注意力**（Multi-head Attention）。“多头”可以理解成将原来的 \mathbf{Q} 、 \mathbf{K} 、 \mathbf{V} 按照隐层维度平均切分成多份。假设切分 h 份，那么最终会得到 $\mathbf{Q} = \{\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_h\}$ ， $\mathbf{K} = \{\mathbf{k}_1, \mathbf{k}_2, \dots, \mathbf{k}_h\}$ ， $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_h\}$ 。多头注意力机制就是用每一个切分得到的 \mathbf{q} ， \mathbf{k} ， \mathbf{v} 独立的进行注意力计算。即第 i 个头的注意力计算结果 $\text{head}_i = \text{Attention}(\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i)$ 。

下面根据图6.48详细介绍多头注意力的计算过程：

- 首先将 \mathbf{Q} 、 \mathbf{K} 、 \mathbf{V} 分别通过线性变换的方式映射为 h 个子集（机器翻译任务中， h 一般为 8）。即 $\mathbf{q}_i = \mathbf{Q}\mathbf{W}_i^Q$ 、 $\mathbf{k}_i = \mathbf{K}\mathbf{W}_i^K$ 、 $\mathbf{v}_i = \mathbf{V}\mathbf{W}_i^V$ ，其中 i 表示第 i 个头， $\mathbf{W}_i^Q \in \mathbb{R}^{d_{model} \times d_k}$ ， $\mathbf{W}_i^K \in \mathbb{R}^{d_{model} \times d_k}$ ， $\mathbf{W}_i^V \in \mathbb{R}^{d_{model} \times d_v}$ 是参数矩阵； $d_k = d_v =$

d_{model}/h ，对于不同的头采用不同的变换矩阵，这里 d_{model} 是 Transformer 的一个参数，表示每个隐层向量的维度；

- 其次对每个头分别执行点乘注意力操作，并得到每个头的注意力操作的输出 head_i ；
- 最后将 h 个头的注意力输出在最后一维 d_v 进行拼接（Concat）重新得到维度为 $h \times d_v$ 的输出，并通过对其左乘一个权重矩阵 \mathbf{W}^o 进行线性变换，从而对多头计算得到的信息进行融合，且将多头注意力输出的维度映射为模型的隐层大小（即 d_{model} ），这里参数矩阵 $\mathbf{W}^o \in \mathbb{R}^{h \times d_v \times d_{model}}$ 。

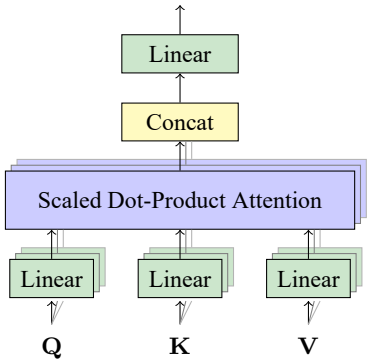


图 6.48: 多头注意力模型

多头机制具体的计算公式如下：

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) \mathbf{W}^o \tag{6.49}$$

$$\text{head}_i = \text{Attention}(\mathbf{QW}_i^Q, \mathbf{KW}_i^K, \mathbf{VW}_i^V) \tag{6.50}$$

多头机制的好处是允许模型在不同的表示子空间里学习。在很多实验中发现，不同表示空间的头捕获的信息是不同的，比如，在使用 Transformer 处理自然语言时，有的头可以捕捉句法信息，有头可以捕捉词法信息。

6.4.7 残差网络和层正则化

Transformer 编码器、解码器分别由多层网络组成（通常为 6 层），每层网络又包含多个子层（自注意力网络、前馈神经网络）。因此 Transformer 实际上是一个很深的网络结构。再加上前面介绍的点乘注意力机制，包含很多线性和非线性变换；另外，注意力函数 $\text{Attention}(\cdot)$ 的计算也涉及多层网络，整个网络的信息传递非常复杂。从反向传播的角度来看，每次回传的梯度都会经过若干步骤，容易产生梯度爆炸或者消失。

解决这个问题的一种办法就是使用**残差连接**[101]。残差连接是一种用来训练深层网络的技术，其结构如图6.49，即在子层之前通过增加直接连接的方式，将底层信

息直接传递给上层。

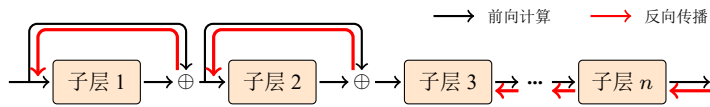


图 6.49: 残差网络结构

残差连接从广义上讲也叫**短连接**（Short-cut Connection），指的是这种短距离的连接。它的思想很简单，就是把层和层之间的距离拉近。如图6.49所示，子层 1 通过残差连接跳过了子层 2，直接和子层 3 进行信息传递。使信息传递变得更高效，有效解决了深层网络训练过程中容易出现的梯度消失/爆炸问题，使得深层网络的训练更加容易。其计算公式为：

$$x_{l+1} = x_l + \mathcal{F}(x_l) \tag{6.51}$$

其中 $\mathcal{F}(x_l)$ 是子层运算。如果 $l = 2$ ，那么公式6.51可以解释为，第 3 层的输出等于第 2 层的输出加上第二层的输入。图6.50中的红色方框展示了 Transformer 中残差连接的位置。

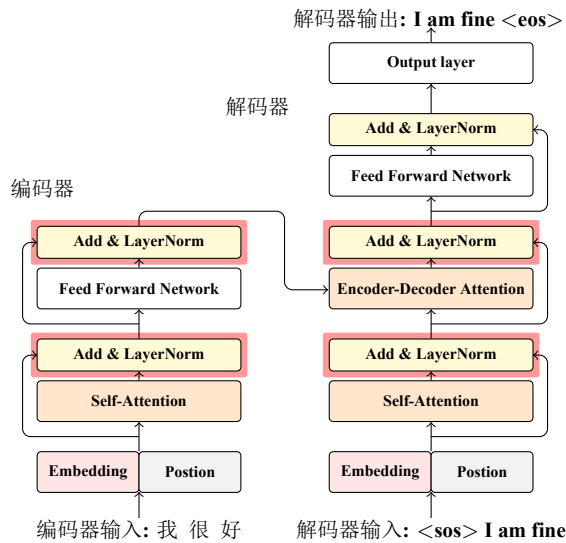


图 6.50: 残差和层正则化在模型中的位置

在 Transformer 的训练过程中，由于引入了残差操作，将前面所有层的输出加到一起。这样会导致不同层（或子层）的结果之间的差异性很大，造成训练过程不稳定、训练时间较长。为了避免这种情况，在每层中加入了层正则化操作 [5]。层正则化的计算公式如下：

$$\text{LN}(x) = g \cdot \frac{x - \mu}{\sigma} + b \tag{6.52}$$

该公式使用均值 μ 和方差 σ 对样本进行平移缩放，将数据规范化为均值为 0，方差为 1 的标准分布。 g 和 b 是可学习的参数。

在 Transformer 中经常使用的层正则化操作有两种结构，分别是**后正则化**（Post-norm）和**前正则化**（Pre-norm），结构如图6.51所示。后正则化中先进行残差连接再进行层正则化，而前正则化则是在子层输入之前进行层正则化操作。在很多实践中已经发现，前正则化的方式更有利于信息传递，因此适合训练深层的 Transformer 模型 [299]。

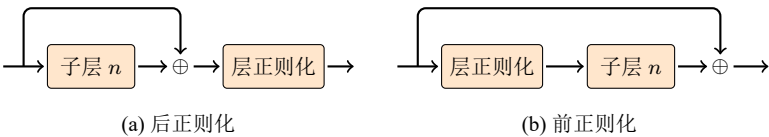


图 6.51: 不同正则化方式

6.4.8 前馈全连接网络子层

在 Transformer 的结构中，每一个编码层或者解码层中都包含一个前馈神经网络，它在模型中的位置如图6.52中红色方框所示。

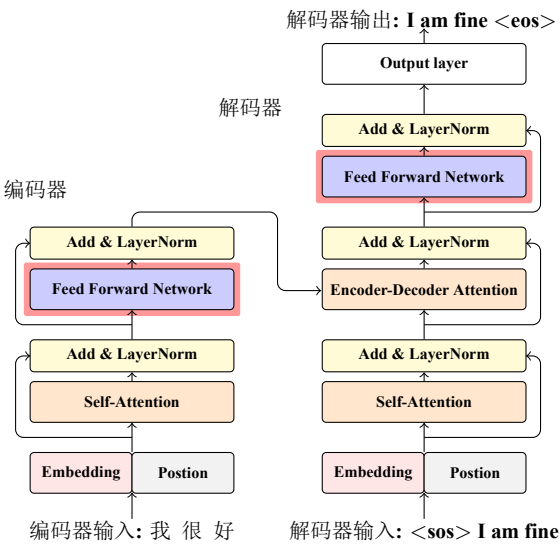


图 6.52: 前馈神经网络在模型中的位置

Transformer 使用了全连接网络。全连接网络的作用主要体现在将经过注意力操作之后的表示映射到新的空间中，新的空间会有利于接下来的非线性变换等操作。实验证明，去掉全连接网络会对模型的性能造成影响。Transformer 的全连接前馈神经网络包含两次线性变换和一次非线性变换（ReLU 激活函数: $\text{ReLU}(x) = \max(0, x)$ ），

每层的前馈神经网络参数不共享，计算公式如下：

$$\text{FFN}(x) = \max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \quad (6.53)$$

其中， \mathbf{W}_1 、 \mathbf{W}_2 、 \mathbf{b}_1 和 \mathbf{b}_2 为模型的参数。通常情况下，前馈神经网络的隐层维度要比注意力部分的隐层维度大，而且研究人员发现这种设置对 Transformer 是至关重要的。比如，注意力部分的隐层维度为 512，前馈神经网络部分的隐层维度为 2048。当然，继续增大前馈神经网络的隐层大小，比如设为 4096，甚至 8192，还可以带来性能的增益，但是前馈部分的存储消耗较大，需要更大规模 GPU 设备的支持。因此在具体实现时，往往需要在翻译准确性和存储/速度之间找到一个平衡。

6.4.9 训练

与前面介绍的神经机器翻译模型的训练一样，Transformer 的训练流程为：首先对模型进行初始化，然后在编码器输入包含结束符的源语言单词序列。前面已经介绍过，解码端每个位置单词的预测都要依赖已经生成的序列。在解码端输入包含起始符号的目标语序列，通过起始符号预测目标语的第一个单词，用真实的目标语的第一个单词去预测第二个单词，以此类推，然后用真实的目标语序列和预测的结果比较，计算它的损失。Transformer 使用了**交叉熵损失**（Cross Entropy Loss）函数，损失越小说明模型的预测越接近真实输出。然后利用反向传播来调整模型中的参数。由于 Transformer 将任意时刻输入信息之间的距离拉近为 1，摒弃了 RNN 中每一个时刻的计算都要基于前一时刻的计算这种具有时序性的训练方式，因此 Transformer 中训练的不同位置可以并行化训练，大大提高了训练效率。

需要注意的是，Transformer 也包含很多工程方面的技巧。首先，在训练优化器方面，需要注意以下几点：

- Transformer 使用 Adam 优化器优化参数，并设置 $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon = 10^{-9}$ 。
- Transformer 在学习率中同样应用了学习率**预热**（Warmup）策略，其计算公式如下：

$$lrate = d_{model}^{-0.5} \cdot \min(step^{-0.5}, step \cdot warmup_steps^{-1.5}) \quad (6.54)$$

其中， $step$ 表示更新的次数（或步数）。通常设置网络更新的前 4000 步为预热阶段即 $warmup_steps = 4000$ 。Transformer 的学习率曲线如图 6.53 所示。在训练初期，学习率从一个较小的初始值逐渐增大（线性增长），当到达一定的步数，学习率再逐渐减小。这样做可以减缓在训练初期的不稳定现象，同时在模型达到相对稳定之后，通过逐渐减小的学习率让模型进行更细致的调整。这种学习率的调整方法是 Transformer 的一个很大的工程贡献。

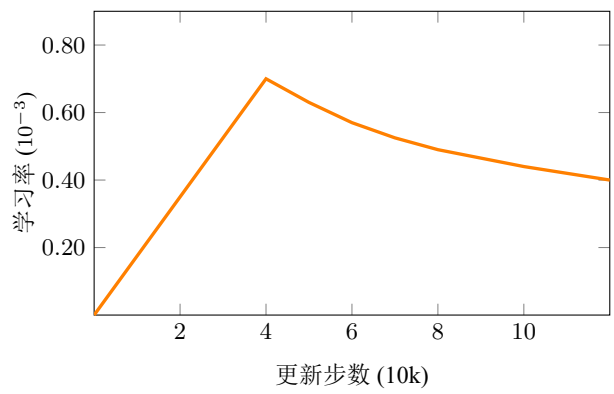


图 6.53: Transformer 模型的学习率曲线

另外，Transformer 为了提高模型训练的效率和性能，还进行了以下几方面的操作：

- **小批量训练 (Mini-batch Training)**：每次使用一定数量的样本进行训练，即每次从样本中选择一小部分数据进行训练。这种方法的收敛较快，同时易于提高设备的利用率。批次大小通常设置为 2048/4096（token 数即每个批次中的单词个数）。每一个批次中的句子并不是随机选择的，模型通常会根据句子长度进行排序，选取长度相近的句子组成一个批次。这样做可以减少 padding 数量，提高训练效率，如图6.54。

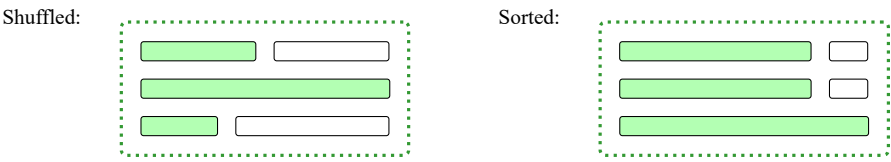


图 6.54: batch 中 padding 数量对比（白色部分为 padding）

- **Dropout**：由于 Transformer 模型网络结构较为复杂，会导致过度拟合训练数据，从而对未见数据的预测结果变差。这种现象也被称作**过拟合 (Over Fitting)**。为了避免这种现象，Transformer 加入了 Dropout 操作 [264]。Transformer 中这四个地方用到了 Dropout：词嵌入和位置编码、残差连接、注意力操作和前馈神经网络。Dropout 比例通常设置为 0.1。
- **标签平滑 (Label Smoothing)**：在计算损失的过程中，需要用预测概率去拟合真实概率。在分类任务中，往往使用 One-hot 向量代表真实概率，即真实答案位置那一维对应的概率为 1，其余维为 0，而拟合这种概率分布会造成两个问题：1) 无法保证模型的泛化能力，容易造成过拟合；2) 1 和 0 概率鼓励所属类别和其他类别之间的差距尽可能加大，会造成模型过于相信预测的类别。因此 Transformer 里引入标签平滑 [278] 来缓解这种现象，简单的说就是给正确答案

以外的类别分配一定的概率，而不是采用非 0 即 1 的概率。这样，可以学习一个比较平滑的概率分布，从而提升泛化能力。

不同的 Transformer 可以适应不同的任务，常见的 Transformer 模型有 Transformer Base、Transformer Big 和 Transformer Deep[288, 299]，具体设置如下：

- Transformer Base: 标准的 Transformer 结构，解码器编码器均包含 6 层，隐层维度为 512，前馈神经网络维度为 2048，多头注意力机制为 8 头，Dropout 设为 0.1。
- Transformer Big: 为了提升网络的容量，使用更宽的网络。在 Base 的基础上增大隐层维度至 1024，前馈神经网络的维度变为 4096，多头注意力机制为 16 头，Dropout 设为 0.3。
- Transformer Deep: 加深编码器网络层数可以进一步提升网络的性能，它的参数设置与 Transformer Base 基本一致，但是层数增加到 48 层，同时使用 Pre-Norm 作为层正则化的结构。

在 WMT’16 数据上的实验对比如表 6.13 所示。可以看出，Transformer Base 的 BLEU 得分虽不如另外两种模型，但其参数量是最少的。而 Transformer Deep 的性能整体好于 Transformer Big。

表 6.13: 三种 Transformer 模型的对比

系统	BLEU[%]		# of params
	EN-DE	EN-FR	
Transformer Base	27.3	38.1	65×10^6
Transformer Big	28.4	41.8	213×10^6
Transformer Deep(48 层)	30.2	43.1	194×10^6

6.4.10 推断

Transformer 解码器生成目标语的过程和前面介绍的循环网络翻译模型类似，都是从左往右生成，且下一个单词的预测依赖已经生成的上一个单词。其具体推断过程如图 6.55 所示，其中 C_i 是编码-解码注意力的结果，解码器首先根据 “<eos>” 和 C_1 生成第一个单词 “how”，然后根据 “how” 和 C_2 生成第二个单词 “are”，以此类推，当解码器生成 “<eos>” 时结束推断。

但是，Transformer 在推断阶段无法对所有位置进行并行化操作，因为对于每一个目标语单词都需要对前面所有单词进行注意力操作，因此它推断速度非常慢。可以采用的加速手段有：低精度 [50]、Cache（缓存需要重复计算的变量）[340]、共享注意力网络等 [316]。关于 Transformer 模型的推断技术将会在第七章进一步深入介绍。

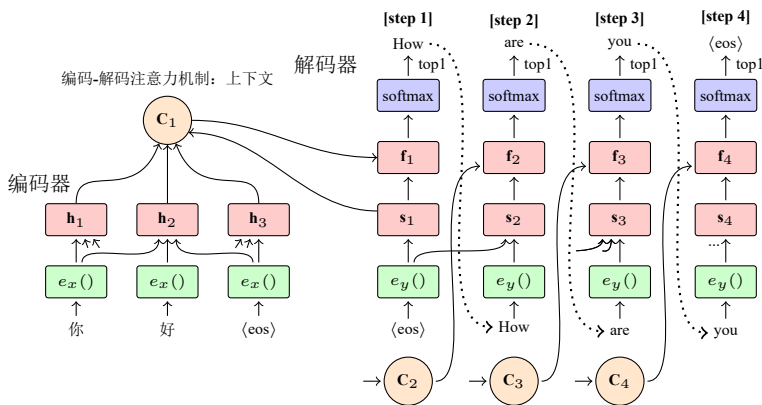


图 6.55: Transformer 推断过程示例

6.5 序列到序列问题及应用

虽然翻译的目的是进行自然语言文字的转化，但是并不需要限制机器翻译只能进行两种语言之间的转换。从某种意义上讲，一个输入序列转化到一个输出序列的过程都可以被看作“翻译”。这类问题通常被称作**序列到序列的转换/生成问题**（Sequence-to-Sequence Problem）。而机器翻译模型也是一种典型的序列到序列模型。

实际上，很多自然语言处理问题都可以被看作是序列到序列的任务。比如，在自动问答中，可以把问题看作是输入序列，把回答看作是输出序列；在自动对联生成中，可以把上联看作是输入序列，把下联看作是输出序列。这样的例子还有很多。对于这类问题，都可以使用神经机器翻译来进行建模。比如，使用编码器-解码器框架对输入和输出的序列进行建模。下面就来看几个序列到序列的问题，以及如何使用神经机器翻译类似的思想对它们进行求解。

6.5.1 自动问答

自动问答，即能够根据给定的问题和与该问题有关的文档，生成问题所对应的答案。自动问答的应用场景很多，智能语音助手、自动客服都是自动问答的典型应用。自动问答系统需要根据输入问题的文字序列，匹配相关文档，然后整合问题和相关知识，输出答案的文字序列。这也可以被看作是一个编码-解码的过程。普遍的做法是，通过编码器对问题编码，然后融合外部知识后，通过解码器生成答案。具体的实现机制有很多，比如基于阅读理解、基于知识库的方法等。但是，不论是何种类型的问答任务，本质上还是要找到问题和答案之间的对应关系，因此可以直接使用类似于神经机器翻译的这种序列到序列模型对其进行求解。

6.5.2 自动文摘

自动文本摘要，即在不改变文本原意的情况下，自动生成文本的主要内容。自动文本摘要技术被广泛应用于新闻报道、信息检索等领域。文本自动摘要是根据输

入的文档得到摘要，因此可以把原始文档看作输入序列，把得到的摘要看作输出序列。常见的解决思路有：抽取式文摘和生成式文摘。前者试图从输入的文本中抽取能表达原文主要内容的句子，进行重新组合、提炼；后者则试图让计算机“理解”并“表达”出原文的主要内容。生成式文摘也可以用端到端框架实现。比如，可以利用编码器将整个输入序列编码成一个具有输入序列信息的固定维度向量，然后利用解码器对这个向量解码，获取所需要文本摘要 [244]。图6.56展示了一个文本自动摘要的例子 [227]。

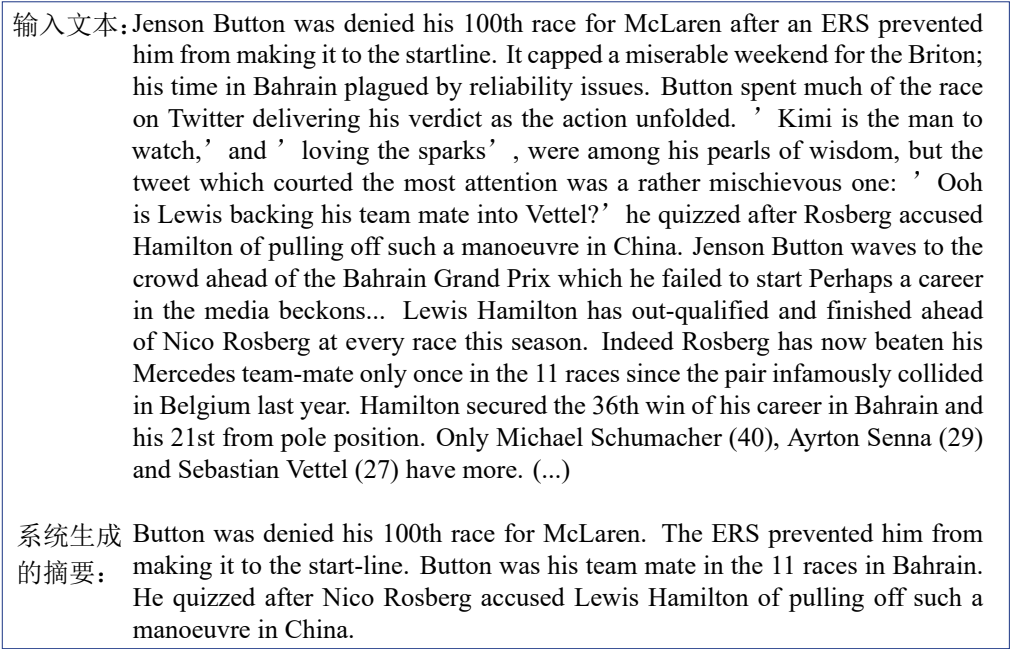


图 6.56: 文本自动摘要实例

6.5.3 文言文翻译

文言文翻译，即能够根据输入的文言文，输出相应的现代汉语翻译。中国几千年的文化都是用文言文记载的，不同时代的古文之间存在巨大差异，普通人在阅读古籍时会面临很大困难。为此，有研究者致力于将古籍翻译成现代汉语。实际上，文言文翻译也是机器翻译的一个典型应用。想要训练一个文言文翻译系统并不难，只需要将文言文看作源语言，将现代文看作目标语言，并送入机器翻译模型，就可以获得一个古文翻译系统。不过，由于古文短，现代文长，过翻译或欠翻译等问题会在古文翻译中表现得更为突出。在此，如果想要获得一个性能优异的文言文翻译系统，就需要考虑如何对长度进行更精准的建模。另外，不同时代语言差异性大，因此还需要能够进行自动适应和风格迁移。图6.57展示了使用神经机器翻译模型得到的文言文翻译系统的实例。输入古文，系统就能生成其现代文翻译。当然，也可以用类似的方法训练现代文-古文的翻译系统。

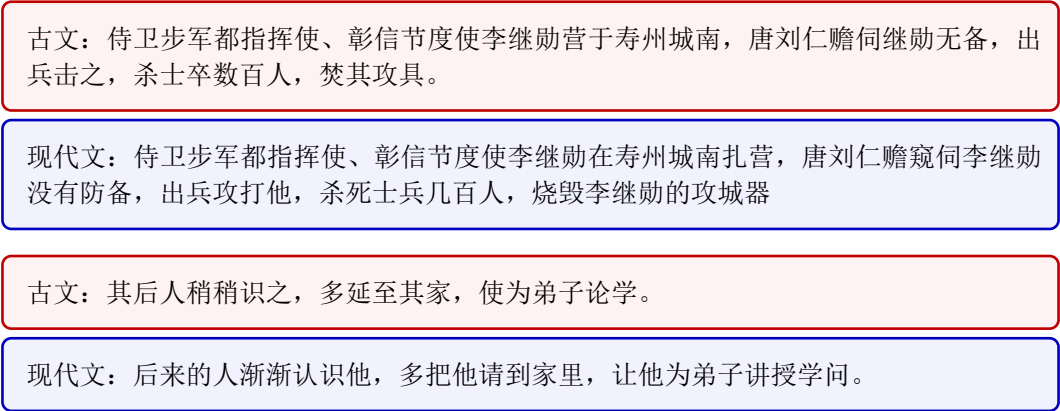


图 6.57: 文言文自动翻译实例

6.5.4 对联生成

对联生成，即能够根据输入的上联，生成与之匹配的下联。春节、结婚、寿诞、乔迁、丧事等都用到对联。对联非常符合序列到序列问题的定义。比如，只需要把整理好的对联数据（上下联对应）送给神经机器翻译系统，系统就可以学习上下联之间的对应关系。当用户输入新的上联，系统可以自动“翻译”出下联。图6.58展示了几个使用神经机器翻译技术生成的对联。当然，对联也有自身特有的问题。比如，对联的上下联有较严格的长度、押韵和词义的对应要求。如何让系统学习到这些要求，并且能够“灵活”运用是很有挑战的。除此之外，由于缺乏数据，如何生成高度概括上下联内容的横批，难度也很大。这些都是值得探索的研究方向。

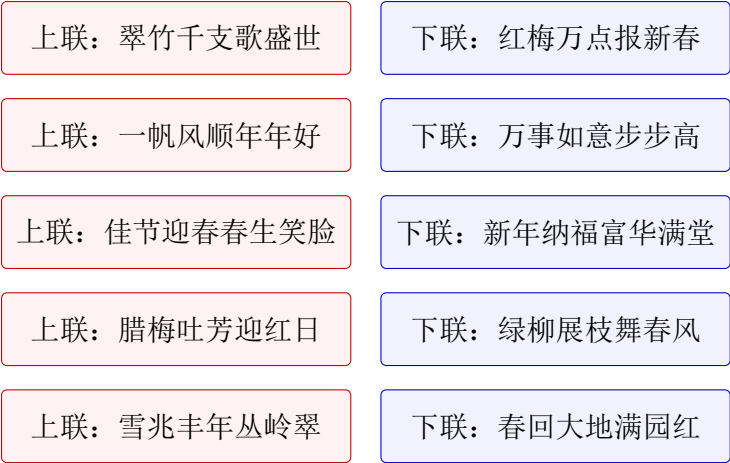


图 6.58: 对联自动生成实例（人工给定上联）

6.5.5 古诗生成

古诗生成，即能够根据输入的关键词，输出与关键词相关的古诗。古诗生成也是大家喜闻乐见的一种传统文化体验形式。古诗生成问题也可以用神经机器翻译技术解决。比如，可以把一些关键词作为输入，把古诗作为输出。那写一个藏头诗呢？对于藏头诗生成，本质上对应了机器翻译中的一类经典问题，即使用约束干预机器翻译结果。当然这里不会开展深入的讨论。可以使用一种简单的方法，使用强制解码技术，在生成过程中排除掉所有不满足藏头诗约束的候选。当然，藏头诗生成系统还面临很多困难，比如如何体现关键词的意境等。这些都需要设计独立的模块进行建模。图6.59展示了藏头诗生成系统的简单框架。

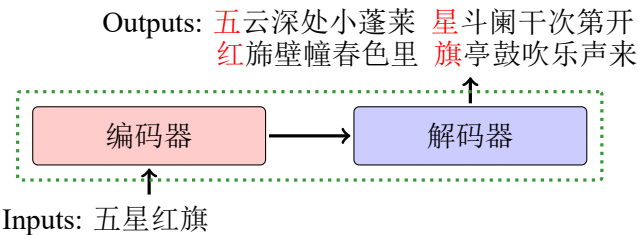


图 6.59: 基于编码器-解码器框架的古诗自动生成

6.6 小结及深入阅读

神经机器翻译是近几年的热门方向。无论是前沿性的技术探索，还是面向应用落地的系统研发，神经机器翻译已经成为当下最好的选择之一。研究人员对神经机器翻译的热情使得这个领域得到了快速的发展。本章作为神经机器翻译的入门章节，对神经机器翻译的建模思想和基础框架进行了描述。同时，对常用的神经机器翻译架构 - 循环神经网络和 Transformer - 进行了讨论与分析。下一章会对神经机器翻译中的一些常用技术和前沿方法进行进一步介绍。

经过几年的积累，神经机器翻译的细分方向已经十分多样，由于篇幅所限，这里也无法覆盖所有内容（虽然笔者尽所能全面介绍相关的基础知识，但是难免会有疏漏）。很多神经机器翻译的模型和方法值得进一步学习和探讨：

- 无论是循环神经网络还是 Transformer 都有很多变种结构。比如，除了 RNN、LSTM、GRU，还有其他改进的循环单元结构，如 LRN[338]、SRU[164]、ATR[341]。Transformer 是近些年的热门，它也衍生出很多的改进版本，如相对位置编码 [255]、局部注意力机制 [329]、多层信息交互 [300]、深层网络 [299]。此外，其他神经网络架构，如卷积神经网络，也是研发神经机器翻译系统很好的选择 [84][310]。最近，也有一些研究者探索异构系统，使用不同的神经网络结构搭建编码器和解码器 [34]，比如，编码端使用性能更强的 Transformer，而解码端使用速度更快的循环神经网络。

- 注意力机制的使用是机器翻译乃至整个自然语言处理近几年获得成功的重要因素之一 [179][289][207]。早期，有研究者尝试将注意力机制和统计机器翻译的词对齐进行统一 [303]。近两年，也有研究已经发现注意力模型可以捕捉一些语言现象 [296]，比如，在 Transformer 的多头注意力中，不同头往往会捕捉到不同的信息，比如，有些头对低频词更加敏感，有些头更适合词意消歧，甚至有些头可以捕捉句法信息。此外，由于注意力机制增加了模型的复杂性，而且随着网络层数的增多，神经机器翻译中也存在大量的冗余，因此研发轻量的注意力模型也是具有实践意义的方向 [316]。
- 一般来说，神经机器翻译的计算过程是没有人工干预的，翻译流程也无法用人类的知识直接进行解释，因此一个有趣的方向是在神经机器翻译中引入先验知识，使得机器翻译的行为更“像”人。比如，可以使用句法树来引入人类的语言学知识 [305, 330]，基于句法的神经机器翻译也包含大量的树结构的神经网络建模 [95, 304]。此外，也可以把用户定义的词典或者翻译记忆加入到翻译过程来 [54, 347]，使得用户的约束可以直接反映到机器翻译的结果上来。先验知识的种类还有很多，包括词对齐 [170, 345]、篇章信息 [295, 308] 等等，都是神经机器翻译中能够使用的信息。
- 神经机器翻译依赖成本较高的 GPU 设备，因此对模型的裁剪和加速也是很多系统研发人员所感兴趣的方向。比如，从工程上，可以考虑减少运算强度，比如使用低精度浮点数或者整数进行计算，或者引入缓存机制来加速模型的推断 [19, 50]；也可以通过对模型参数矩阵的剪枝，甚至对模块的剪枝，来减小整个模型的体积 [248, 349]；另一种方法是知识精炼。利用大模型训练小模型，这样往往可以得到比单独训练小模型更好的效果 [36, 109, 275]。



7. 神经机器翻译实战

作为机器翻译的前沿方向，神经机器翻译方法是近些年来最受关注的热点之一。凭借其高品质的译文，神经机器翻译的身影在各种机器翻译比赛和产品中随处可见。第六章已经介绍了神经机器翻译的基础模型，包括：基于循环神经网络的翻译模型、基于 Transformer 的翻译模型、注意力机制等等。但是，简单使用这些模型和方法显然无法取得最好的结果。实际上，先进的系统往往依赖多种技术的综合运用，是一项庞大的系统工程。

本章将沿着神经机器翻译框架继续探讨：如何研发性能更为突出的机器翻译系统。这里将介绍若干常用的提升神经机器翻译系统品质和速度的方法。同时，也会讨论一些开放性的前沿课题，旨在使机器翻译达到更加先进的水平。本章的绝大部分内容都经过了笔者所在团队的实验，具有实践方面的参考价值。正如本章的副标题一样，希望这里所讨论的内容可以帮助读者了解如何开发出一套足以参加高水平机器翻译比赛的系统，为相关研究建立更加科学、合理的基线，同时为机器翻译的应用提供一些具体可落地的思路。

7.1 神经机器翻译并不简单

同统计机器翻译一样，高品质神经机器翻译系统的开发并没有想象的那么简单。虽然有很多开源系统可以参考，但是系统实践者仍然有大量的工作需要完成。在神经机器翻译系统研发方面，有若干问题需要注意：

- **神经网络模型很脆弱。**神经机器翻译对超参数和训练策略的选择以及网络结构的细微差别都非常敏感。比如，学习率、Dropout 比率上下浮动一点点都会给翻译结果带来明显区别。这也导致系统研发人员需要花费大量的时间来寻找合理的系统配置。虽然也有一些研究工作探讨自动化调参和结构设计（如：AutoML），但是为了确保翻译品质，现在普遍的做法仍然是“人肉”搜索最佳的网络架构和系统配置。
- **神经机器翻译需要对不同翻译任务进行适应。**理想中一套“包打天下”的模型和设置是不存在的。针对不同语种、不同领域，机器翻译系统都需要进行调整。比如，一个在新闻数据上训练的系统在专利数据上的表现往往不会很好。这里并不否认在很多研究工作中为了缩短研究周期，可以用同一套系统及设置在所有任务上进行实验。但是，具体到每个翻译任务上，高质量翻译都离不开对系统细致地调整，比如，与语言相关的数据加工等。
- **神经机器翻译的“最后一公里”仍然很长。**无论是使用开源系统，还是从论文中进行复现，都可以很容易的得到一个基础版本的机器翻译系统。但是这样的系统离真正的 state-of-the-art 往往会有距离，离实用系统的距离可能更远。更具挑战的是，很多高水平系统中所使用的技巧甚至都没有被开源或者以论文的形式发表，这大大增加了普通研究者挑战前沿的难度。
- **优秀系统的研发需要长时间的打磨，但是很多时候我们仅仅是在搭建快速原型。**这不是一个技术问题。因为如果有足够的时间，所有人都可以把任何事情做到极致。但是，不论是为了毕业、提职，还是希望在领域占据一席之地，我们总是希望在尽可能短的时间内把系统研发出来，把结果报道出来。这种做法无可厚非，因为科学研究需要对更本质的科学问题进行探索，而非简单的工程开发与调试。但是，对一个初级的系统进行研究往往会掩盖掉“真正的问题”，因为很多问题在更先进的系统中根本就不存在。

这里并不是要对神经机器翻译产生“畏难情绪”，更不是要对机器翻译繁荣的景象泼冷水。这里只是希望可以冷静的看待神经机器翻译技术的发展现状，让相关研究和系统研发更加扎实。少一些对技术的过度吹捧，多一些脚踏实地的工匠精神。

7.1.1 影响神经机器翻译性能的因素

一般来说，有三方面因素会影响神经机器翻译系统的性能：

- **技术的先进性和成熟度。**选择什么样的神经网络模型、什么样的训练方法、什么样的模型初始化策略等等，都会对最终的系统产生影响。显然，更加先进、成熟度更高的技术会带来性能的提升。比如，在最近，Transformer 架构就受到了很多人的青睐，Adam 等训练策略在神经机器翻译中也很流行。当然，未来一定会有更加强大的模型和更加先进的技术被提出。因此系统研发者也需要紧跟技术的前沿，这样才能保证系统的先进性。

- **数据的质量和规模。**数据驱动的方法对数据的依赖性不言而喻。虽然技术日新月异，但是“更好更多的数据”是一直被广泛接受的“真理”。所谓数据质量和规模一般都是针对任务而言，因此如何（或者是否可以）获取适合目标任务的高质量、大规模的数据是实践中需要考虑的因素。在机器翻译系统研发的初级阶段，特别是在数据规模受限的情况下，增加高质量数据带来的性能提升往往更容易体现出来（如图7.1）。

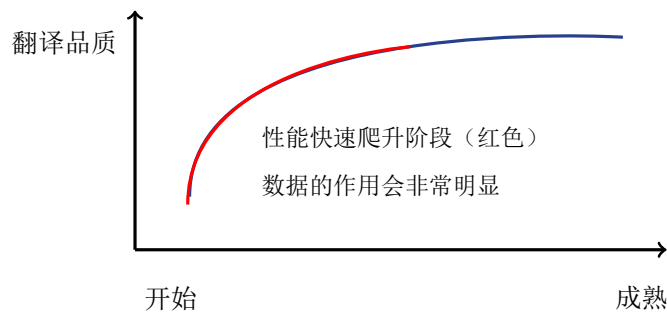


图 7.1: 机器翻译性能曲线

- **系统的打磨。**容易被忽视的是系统的工程打磨，包括对数据的细致处理、参数的精细调整。更重要的是，在应用的过程中需要对发现的问题不断进行修正，这种日积月累的改变最终会带来翻译品质的显著提升。不过，系统打磨所需要的毅力与投入是十分巨大的。甚至从整个系统研发的进程来看，打磨的时间往往会占据主要部分。这对系统研发者来说也是一种考验。

从应用的角度，评价系统的维度有很多，因此研发系统所使用的策略也会有所不同。比如，如果希望让机器翻译能够在小型离线设备上运行，这时可能需要同时关心模型的体积和翻译的速度；如果是为了做非实时的文本翻译，翻译品质就是最关键的。

7.1.2 搭建神经机器翻译系统的步骤

构建一个机器翻译系统的主要步骤如图7.2所示，包含三个方面：

- **数据处理：**机器翻译的训练数据、测试数据都需要进行加工和处理，包括，数据的清洗、翻译单元（字词）的切分、对特殊字符的处理等。数据处理的好坏对机器翻译系统的品质影响很大。
- **建模和架构选择：**确定了翻译任务，下一步需要设计机器翻译模型。核心是神经网络架构的选择，比如，是使用循环神经网络还是 Transformer。在确定架构之后，需要进一步设计细致的网络结构，比如，注意力机制、深层网络等等。神经网络结构的设计也会影响模型训练和推断策略的选择。
- **训练和推断：**模型设计好之后，需要实现训练和推断模块。这部分会涉及损失函数的设计、超参数的调整、推断中搜索算法的设计等等。如果对翻译速度有

要求，模型的加速和压缩策略往往也需要被考虑进来。

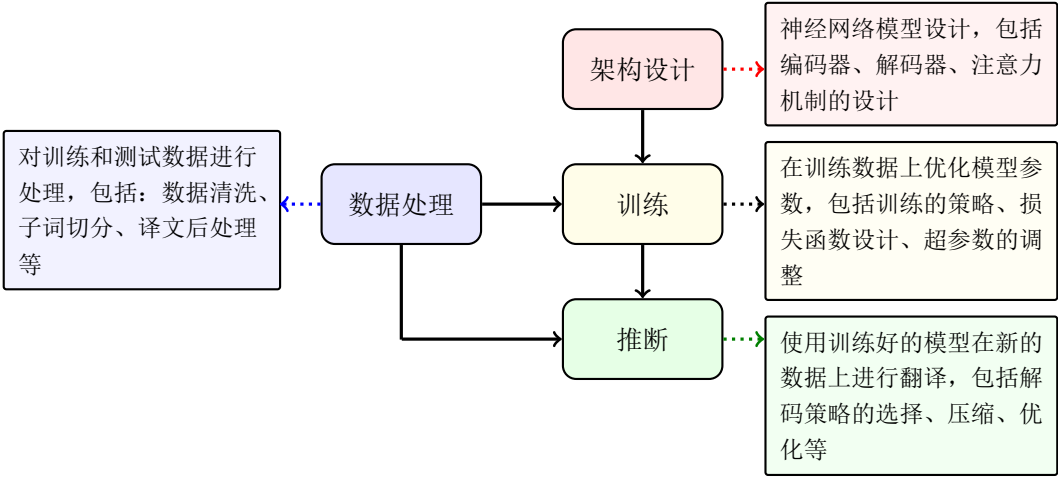


图 7.2: 构建神经机器翻译系统的主要步骤

在实际开发中，还有很多因素会影响技术方案的选择，比如，开发的时间周期、设备的成本等因素¹。不过，本章着重关注那些简单有效的技术方法。

7.1.3 架构选择

神经机器翻译模型的架构有很多，比如，循环神经网络、卷积神经网络、Transformer 都是可以考虑的架构。这里，我们会以 Transformer 为基础模型展开讨论。选择 Transformer 的主要原因是其在很多机器翻译任务上表现非常出色，比如在最近一届的 WMT 和 CCMT 的新闻翻译评测中，绝大多数冠军系统都是基于 Transformer 架构。现在的很多线上机器翻译服务也都采用了 Transformer。因此 Transformer 在一定程度上代表了当今最先进的架构。

此外，Transformer 使用了自注意力机制，因此训练时模型并行度较高，模型训练的时间较短。这为系统研发节省了时间。特别是面临大量超参数调整时，较短的训练周期会大大加速优化的进程。

7.2 数据处理

同统计机器翻译一样，神经机器翻译也需要对输入和输出的句子进行分词，目的是得到翻译的最基本单元。但是，这里所说的单词并不是语言学上的单词，更多的是指面向机器翻译任务的最小翻译片段。比如，可以复用第二章中的自动分词系统对句子进行切分，之后在切分得到的“词”序列上完成翻译建模。

¹有些方法需要大量的并行运算设备，比如超大模型的训练。

自然语言的表达非常丰富，因此需要很多的单词才能表达不同的语义。但是，神经机器翻译系统对大词表的处理效率很低，比如，输出层在大规模词表上进行预测会有明显的速度下降，甚至无法进行计算。因此，在神经机器翻译中会使用受限的词表，比如包含 30000-50000 个单词的词表。另一方面，翻译新的句子时，受限词表会带来大量的**未登录词**（Out of Vocabulary Word, OOV Word），系统无法对其进行翻译。实际上，产生未登录词一方面的原因是词表大小受限，另一方面的原因在于分词的颗粒度过大。对于后者，一种解决方法是进一步对“单词”进行切分，以得到更小的单元，这样可以大大缓解单词颗粒度过大造成的数据稀疏问题。这个过程通常被称作**子词切分**（Sub-word Segmentation）。比如，以 BPE 为代表的子词切分方法已经成为了当今神经机器翻译所使用的标准方法，翻译效果显著超越基于传统分词的系统。

此外，机器翻译依赖高质量的训练数据。在神经机器翻译时代，模型对训练数据很敏感。由于神经机器翻译的模型较为复杂，因此数据中的噪声会对翻译系统产生较大的影响。特别是在实际应用中，数据的来源繁杂，质量参差不齐。因此，往往需要对原始的训练集进行**标准化**（Normalization）和**数据清洗**（Data Cleaning），从而获得高质量的双语数据用于模型训练。

以上这些内容统称为数据处理。神经机器翻译的训练和测试数据都需要经过数据处理。图7.3展示了神经机器翻译中常见的数据处理流程，主要步骤包括分词、标准化、数据过滤和子词切分。此外，由于人类语言丰富多样，不同的语种在处理细节上可能不会完全一致。下面以中英翻译为例介绍数据处理的主要步骤。

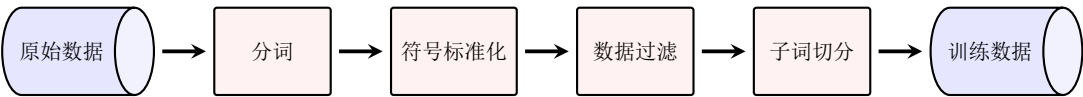


图 7.3: 机器翻译数据处理流程

7.2.1 分词

分词是数据处理的第一步。这部分技术在第二章已经进行了讨论。对于像中文这样没有单词边界的语言，分词的策略通常比较复杂。现在常用的一些中文分词工具有 NLTK[20]、jieba²等。而像英文这种有单词边界的语言，分词要简单许多，比如，Moses 工具包就有可以处理绝大多数拉丁语系语言的分词脚本 [150]。图7.4展示了一个经过分词后的中英文双语数据对照图。

分词系统的效率是一个容易被忽略的问题，也是研发机器翻译系统需要考虑的因素。比如，对上亿、甚至几十亿个句子进行分词的时候，分词系统过慢会拖慢整个训练进程。因此，在实践中，对于大规模的分词，可以考虑使用多线程或者其它分布式计算技术，比如 Map-Reduce，进行并行处理。

²<https://github.com/fxsjy/jieba>

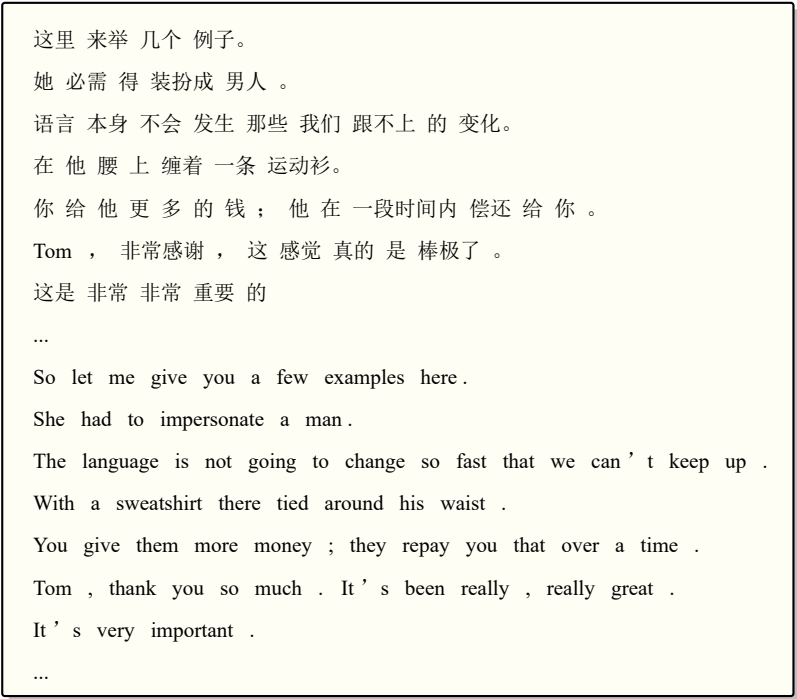


图 7.4: 中英文分词结果示例

此外，在很多翻译场景下，需要对特殊的翻译单元单独处理。比如，在金融领域中，需要对企业名、时间、金额等信息进行准确翻译。这时，对分词系统的优化就十分必要，比如利用正则表达式保证特定类型的单词不会被切割得太碎；再比如，专利翻译中，一些化学式的表达、技术专有名词也需要单独处理。特别是，当用户希望通过预定义的词典干预翻译结果时，分词系统需要能配合相应的处理。比如，希望对一个片段强制使用用户指定的译文，这时就要求分词系统也能把这个单元准确的切分出来。这些问题的解决也依赖大量的工程优化。

7.2.2 标准化

在机器翻译的数据处理过程中，标准化是指对数据中的字符表示或者大小写等进行统一，具体包括符号标准化，大小写转换和中文的简繁体转化等。由于数据来源多样，不同的数据集中可能使用不同的符号标准或者大小写规范等。同一个符号，由于使用的编码不同，计算机也会认为是不同的符号或单词。此外，这种多样性会变相地导致数据中各种符号相对稀疏，增大了模型的学习负担。通过标准化，可以将功能相同的符号或者单词表示进行统一，去除其中的噪音，减小词表规模。

符号标准化，主要指的是全角或者半角符号的统一。如表7.1所示，虽然其中的百分号、字母‘A’和数字‘9’表示的含义没有变，但是在 Unicode 标准中存在不同的编码表示。因此，需要将不同的编码进行统一。在中英翻译中，通常会根据映射规则将符号全部统一成半角符号。

表 7.1: 同一个字符的多种编码表示

字符	Unicode 编码 16 进制
%	FF05
%	FF6A
%	25
A	41
A	FF21
9	39
9	FF19

在英语等一些大小写敏感的语言中，一些专有名词和有特殊用法的单词，以及每个句子的首字母都需要进行大写。此外，训练数据中也会包括一些大小写错误的用法。这导致许多单词由于大小写的区分存在多种形式。一种简单的做法是将数据全部进行小写化，这样可以使所有的单词进行统一，大大提升模型预测的准确性。然而，用小写化数据训练的模型翻译结果也都是小写的，需要额外的还原模型对结果进行处理。

现在更常用的做法是保留句子中每个单词的正确大小写形式。但是对于句子的首字母，需将其转换成这个单词最常见的形式，如图7.5所示。通过这种方式，训练数据中只包含单词的正确大小写形式，大写单词只存在于一些专有名词或者有特殊用法的单词中，在一定程度上减小了词表大小，同时，也去除了一部分数据中由于错误大小写形式所产生的噪音。在翻译结束后，对首字母进行大写就能得到大小写合理的翻译结果。另外，中文存在简繁体两种形式的汉字，训练数据中可能会同时包含这两种形式。因此通常也会考虑把繁体中文转化为简体中文，以统一汉字的编码。

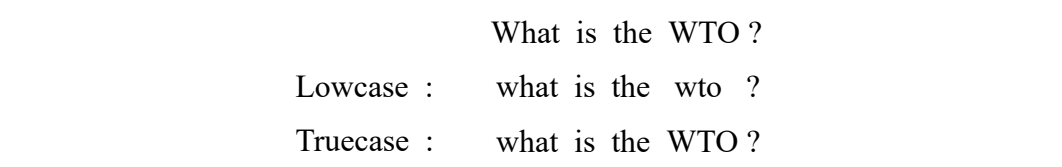


图 7.5: 英文小写化（lowercase）与保留大小写（truecase）处理结果的对比

7.2.3 数据清洗

数据清洗的本质是让机器翻译系统更有效的从数据中进行学习。有两种观点：

- 应该选择性的使用对系统性能提升最大的数据，即**数据选择**（Data Selection）。
- 应该除去低质量的或者有噪声的数据，即**数据过滤**（Data Filtering）。

数据选择认为所有样本都是有用的，只是作用大小不同。因此，如果可以更充分

的利用对机器翻译帮助更大的那部分数据，系统性能应该可以得到提升 [301] 比如，很多比赛系统中会将测试数据与训练数据（源语言部分）进行匹配，选择一部分与测试集更相关的数据，之后用这部分数据微调系统 [165, 297]；也可以对不同训练数据的集合进行加权，之后进行权重敏感的训练，以期望权重大的数据对模型产生更大的影响 [301]。

数据过滤则认为数据中存在不太多的噪声，可以通过去除这种噪声提高数据整体的质量，进而提升训练效果。有很多方法，比如：过滤掉非对齐的样本、翻译质量极低的样本、重复样本等等。图7.6展示了数据过滤的实例。通常数据过滤需要很多工程手段的综合运用，因此也非常考验系统研发团队对系统打磨的能力。

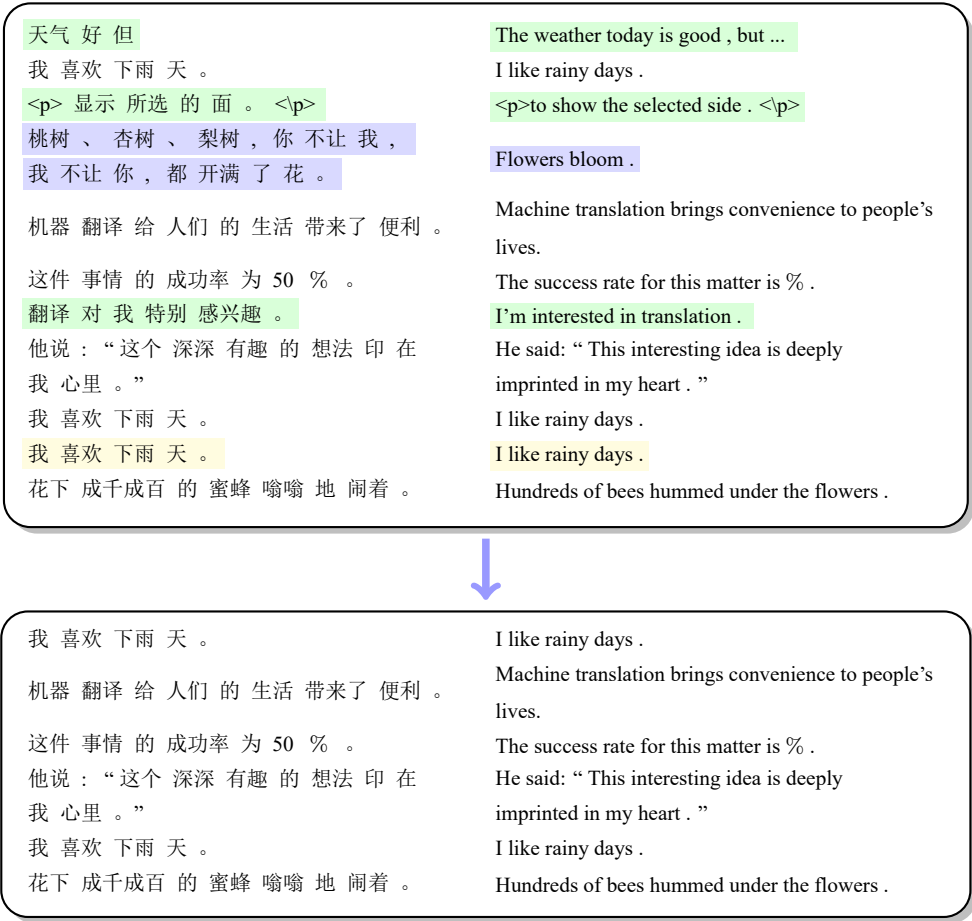


图 7.6: 数据清洗示意图

下面会简单介绍数据过滤的基础方法，因为它们在系统实践中已经被证明是十分有效的。由于原始双语数据往往是基于篇章或者段落，在语料库的构建过程中，需要对原始数据进行句对齐，从而获得训练机器翻译系统所需要的数据。由于句对齐算法并不完美，以及原始数据本身可能存在错误，会导致得到的数据中存在一些非对齐的句对。比如，处理数据时经常会遇到一句话只翻译了一半的情况。更极端一

些的例子是，整个源语言和目标语言句子完全不对应。下面的例子展示了一些典型的非对齐汉英双语句子。显然，数据清洗需要过滤掉这样的数据。

双语句对实例 1:

中文：今天 天气 不错 。

英文：Let's go !

双语句对实例 2:

中文：桃树 、 杏树 、 梨树 、 你 不 让 我 、 我 不 让 你 、 都 开 满 了 花 。

英文：Flowers bloom .

对于有明显问题或者低质量的数据，一般采用启发性的数据过滤方法，有几方面的因素可以考虑：

- 翻译长度比是否合理。有缺失成分的句子，会造成译文和源语言句子的长度比过大或者过小，因此可以设定一个区间，过滤掉长度落在这个区间外的所有句子。
- 单词对应是否合理。两个互为译文的句子之间的单词应该具有较好的对应关系。因此，可以考虑使用自动词对齐的结果，对句对的对齐质量进行评价。统计机器翻译中，这类方法已经被广泛用于短语翻译质量的度量，这里可以直接复用类似的方法（见第四章）。
- 译文是否流畅。译文的流畅度是评价译文质量的重要指标。通常，在数据过滤中也会使用语言模型对译文流畅度进行评价。

以上这些因素可以被看作是不同的特征。可以使用额外的模型组合这些特征，最终得到一个数据过滤的参考分数。当然，数据过滤的策略很多，不同语言的处理策略也不相同。系统研发人员需要大量尝试才能得到一套完整的数据过滤方法。

另外有一点需要注意，数据质量和数据覆盖度是衡量一个语料质量的两个维度，二者通常是矛盾的。很多时候，在获得高质量数据的同时，数据的多样性也降低了。一个极端的例子是数据集中只包含一个翻译很好的双语句子，这个语料的质量是完美的，但是由于它只能覆盖单一的语言现象，无法训练出一个很好的翻译模型。在实践中，往往需要平衡数据的质量和数据的覆盖度。有些情况下为了覆盖更多样的样本，也可以适当保留一些质量不太高的数据 [363]。这是由于，即使这些数据不完美，但是仍然可以提供部分正确的翻译信息，让模型从中学习有用的知识。

7.2.4 子词切分

人类表达语言的方式是十分多样的，这也体现在单词的构成上，甚至我们都无法想象数据中存在的不同单词的数量。比如，如果使用简单的分词策略，WMT、CCMT 等评测数据的英文词表大小都会在 100 万以上。当然，这里面也包括很多的数字和字母的混合，还有一些组合词。不过，如果不加限制，机器翻译所面对的词表确实很

“大”。这也会导致系统速度变慢，模型变大。更严重的问题是，测试数据中的一些单词根本就没有在训练数据中出现过，这时会出现 OOV 翻译问题，即系统无法对未见单词进行翻译。在神经机器翻译中，通常会考虑使用更小的翻译单元来缓解以上问题。

大词表和 OOV 问题

首先来具体看一看神经机器翻译的大词表问题。神经机器翻译模型训练和解码都依赖于源语言和目标语言的词表。在建模中，词表中的每一个单词都会被转换为分布式（向量）表示，即词嵌入。这些向量会作为模型的输入（见第六章）。如果每个单词都对应一个向量，那么单词的各种变形（时态、语态等）都会导致词表和相应的向量数量的增加。图7.7展示了一些英语单词的时态语态变化。

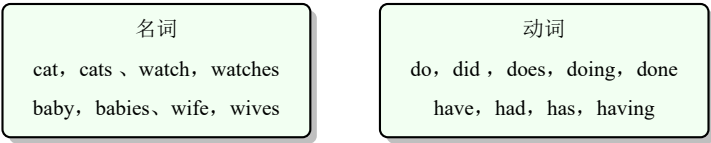


图 7.7: 单词时态、语态、单复数的变化

如果要覆盖更多的翻译现象，词表会不断膨胀，并带来两个问题：

- 数据稀疏。很多不常见的低频词包含在词表中，而这些低频词的分布式表示很难得到充分学习；
- 词向量矩阵的增大。这会增加计算和存储的负担。

理想情况下，机器翻译应该是一个**开放词表**（Open-Vocabulary）的翻译任务。也就是，不论测试数据中包含什么样的词，机器翻译系统都应该能够正常翻译。但是，现实的情况是，即使不断扩充词表，也不可能覆盖所有可能的单词。这时就会出现 OOV 问题（集外词问题）。这个问题在使用受限词表时会更加严重，因为低频词和未见过的词都会被看作 OOV 单词。这时会将这些单词用 <UNK> 代替。通常，数据中 <UNK> 的数量会直接影响翻译性能，过多的 <UNK> 会造成欠翻译、结构混乱等问题。因此神经机器翻译需要额外的机制解决大词表和 OOV 问题。

子词

一种解决开放词表翻译问题的方法是改造输出层结构 [83, 127]，比如，替换原始的 Softmax 层，用更加高效的神经网络结构进行超大规模词表上的预测。不过这类方法往往需要对系统进行修改，由于模型结构和训练方法的调整使得系统开发与调试的工作量增加。而且这类方法仍然无法解决 OOV 问题。因此在实用系统中并不常用。

另一种思路是不改变机器翻译系统，而是从数据处理的角度来缓解 OOV 问题。既然使用单词会带来数据稀疏问题，那么自然会想到使用更小的单元。比如，把字

符作为最小的翻译单元³——也就是基于字符的翻译模型 [163]。以英文为例，只需要构造一个包含 26 个英文字母、数字和一些特殊符号的字符表，便可以表示所有的单词。

但是字符级翻译也面临着新的问题——使用字符增加了系统捕捉不同语言单元之间搭配的难度。假设平均一个单词由 5 个字符组成，所处理的序列长度便增大 5 倍。这使得具有独立意义的不同语言单元需要跨越更远的距离才能产生联系。此外，基于字符的方法也破坏了单词中天然存在的构词规律，或者说破坏了单词内字符的局部依赖。比如，英文单词 “telephone” 中的 “tele” 和 “phone” 都是有具体意义的词缀，但是如果把它们打散为字符就失去了这些含义。

那么有没有一种方式能够兼顾基于单词和基于字符方法的优点呢？常用的手段包括两种，一种是采用字词融合的方式构建词表，将未知单词转换为字符的序列并通过特殊的标记将其与普通的单词区分开来 [187]。而另一种方式是将单词切分为**子词**（Sub-word），它是介于单词和字符中间的一种语言单元表示形式。比如，将英文单词 “doing” 切分为 “do”+“ing”。对于形态学丰富的语言来说，子词体现了一种具有独立意义的构词基本单元。比如，如图 7.8，子词 “do”，和 “new” 在可以用于组成其他不同形态的单词。

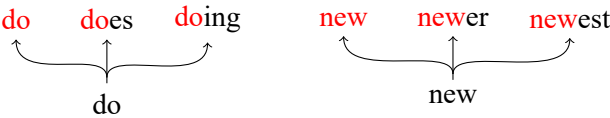


图 7.8: 不同单词共享相同的子词（前缀）

在极端一些的情况下，子词仍然可以包含所有的字母和数字。这样，理论上，所有的单词都可以用子词进行组装。当然，理想的状况是：在子词词表不太大的情况下，使用尽可能少的子词单元拼装出每个单词。在神经机器翻译中，基于子词的切分是很常用的数据处理方法，称为子词切分。主要包括三个步骤：

- 对原始数据进行分词操作；
- 构建子词词表；
- 通过子词词表重新对数据中的单词进行切分。

这里面的核心是如何构建子词词表，下面对一些典型方法进行介绍。

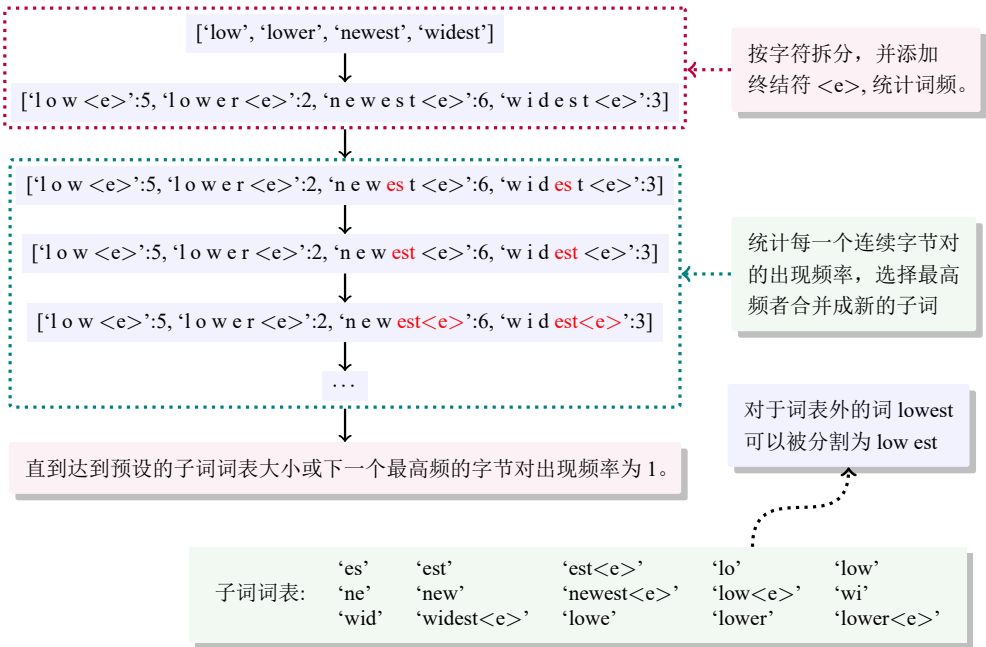
双字节编码（BPE）

字节对编码或**双字节编码**（Byte Pair Encoding, BPE）是一种常用的子词词表构建方法 [252]。BPE 方法最早用于数据压缩，该方法将数据中常见的连续字符串替换为一个不存在的字符，之后通过构建一个替换关系的对应表，对压缩后的数据进行还

³中文中字符可以被看作是汉字。

原。机器翻译借用了这种思想，把子词切分看作是学习对自然语言句子进行压缩编码表示的问题 [79]。其目的是，保证编码后的结果（即子词切分）占用的字节尽可能少。这样，子词单元会尽可能被不同单词复用，同时又不会因为使用过小的单元造成子词切分序列过长。使用 BPE 算法构建子词词表可以分为如下几个步骤：

- 对每个句子进行分词；
- 将分词后的每个单词进行进一步切分，划分为字符序列。同时，在每个单词结尾添加结束符 <e> 用于标记单词的边界。之后，统计该单词在数据中出现的次数。例如单词 low 在数据中出现了 5 次，可以将其记为 'low <e>':5。
- 对得到的字符集合进行统计，统计每个单词中 2-gram 符号出现的频次⁴。之后，选择最高频的 2-gram 符号，将其合并为新的符号，即新的子词。例如“A”和“B”连续出现的频次最高，则以“AB”替换所有单词内连续出现的“A”和“B”并将其加入子词词表。这样，“AB”会被作为一个整体，在之后的过程中可以与其他符号进一步合并。需要注意的是替换和合并不会跨越单词的边界，即只对单个单词进行替换和合并。
- 不断重复上一步骤，直到子词词表大小达到预定的大小或者下一个最高频的 2-gram 字符的频次为 1。子词词表大小是 BPE 的唯一的参数，它用来控制上述子词合并的规模。



⁴发生合并前，一个字符便是一个符号

图7.9给出了 BPE 算法执行的实例。在执行合并操作时，需要考虑不同的情况。假设词表中存在子词“ab”和“cd”，此时要加入子词“abcd”。可能会出现如下的情况：

- 若“ab”、“cd”、“abcd”完全独立，彼此的出现互不影响，将“abcd”加入词表，词表数目 +1；
- 若“ab”和“cd”必同时出现则词表中加入“abcd”，去除“ab”和“cd”，词表数目 -1。这个操作是为了较少词表中的冗余；
- 若出现“ab”，其后必出现“cd”，但是“cd”却可以作为独立的子词出现，则将“abcd”加入词表，去除“ab”，反之亦然，词表数目不变。

在得到了子词词表后，便需要对单词进行切分。BPE 要求从较长的子词开始替换。首先，对子词词表按照字符长度从大到小进行排序。然后，对于每个单词，遍历子词词表，判断每个子词是不是当前词的子串，若是则进行替换切分。将单词中所有的子串替换为子词后，如果仍有子串未被替换，则将其用 <UNK> 代替，如图7.10。

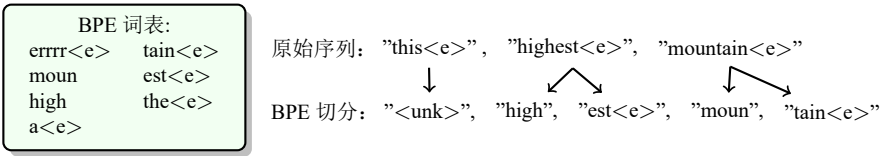


图 7.10: BPE 中的子词切分过程

由于模型的输出也是子词序列，因此需要对最终得到的翻译结果进行子词还原，即将子词形式表达的单元重新组合为原本的单词。这一步操作也十分简单，只需要不断的将每个子词向后合并，直至遇到表示单词边界的结束符 <e>，便得到了一个完整的单词。

使用 BPE 方法的策略有很多。不仅可以单独对源语言和目标语言进行子词的切分，也可以联合源语言和目标语言，共同进行子词切分，被称作 Joint-BPE[252]。单语 BPE 比较简单直接，而 Joint-BPE 则可以增加两种语言子词切分的一致性。对于相似语系中的语言，如英语和德语，常使用 Joint-BPE 的方法联合构建词表。而对于中英这些差异比较大的语种，则需要独立的进行子词切分。

BPE 还有很多变种方法。在进行子词切分时，BPE 从最长的子词开始进行切分。这个启发性规则可以保证切分结果的唯一性，实际上，在对一个单词用同一个子词词表切分时，可能存在多种切分方式，如 hello，我们可以分割为“hell”和“o”，也可以分割为“h”和“ello”。这种切分的多样性可以用来提高神经机器翻译系统的健壮性[158]。而在 T5 等预训练模型中 [238] 则使用了基于字符级别的 BPE。此外，尽管 BPE 被命名为字节对编码，实际上一般处理的是 Unicode 编码，而不是字节。在预训练模型 GPT2 中，也探索了字节级别的 BPE，在机器翻译、问答等任务中取得了很好的效果 [237]。

其他方法

与基于统计的 BPE 算法不同, 基于 Word Piece 和 1-gram Language Model (ULM) 的方法则是利用语言模型进行子词词表的构造 [158]。本质上, 基于语言模型的方法和基于 BPE 的方法的思路是一样的, 即通过合并字符和子词不断生成新的子词。它们的区别仅在于合并子词的方式不同。基于 BPE 的方法选择出现频次最高的连续字符 2-gram 合并为新的子词, 而基于语言模型的方法则是根据语言模型输出的概率选择要合并哪些子词。

具体来说, 基于 Word Piece 的方法首先将句子切割为字符表示的形式 [246], 并利用该数据训练一个 1-gram 语言模型, 记为 $\log P(\cdot)$ 。假设两个相邻的子词单元 a 和 b 被合并为新的子词 c , 则整个句子的语言模型得分的变化为 $\Delta = \log P(c) - \log P(a) - \log P(b)$ 。这样, 可以不断的选择使 Δ 最大的两个子词单元进行合并, 直到达到预设的词表大小或者句子概率的增量低于某个阈值。而 ULM 方法以最大化整个句子的概率为目标构建词表 [158], 具体实现上也不同于基于 Word Piece 的方法, 这里不做详细介绍。

使用子词表示句子的方法可以有效的平衡词汇量, 增大对未见单词的覆盖度。像英译德、汉译英任务, 使用 16k 或者 32k 的子词词表大小便能取得很好的效果。

7.3 建模与训练

7.3.1 正则化

正则化 (Regularization) 是机器学习中的经典技术, 通常用于缓解**过拟合问题** (The Overfitting Problem)。正则化的概念源自线性代数和代数几何。在实践中, 它更多的是指对**反问题** (The Inverse Problem) 的一种求解方式。假设输入 x 和输出 y 之间存在一种映射 f

$$y = f(x) \quad (7.1)$$

反问题是指: 当观测到 y 时, 能否求出 x 。反问题对应了很多实际问题, 比如, 可以把 y 看作经过美化的图片, x 看作原始的图片, 反问题就对应了图片还原。机器翻译的训练也是一种反问题, 因为可以把 y 看作是正確的译文, x 看作是输入句子或者模型参数⁵。

理想的情况下, 我们希望反问题的解是**适定的** (Well-posed)。所谓适定解, 需要满足三个条件: 解是存在的、解是唯一的、解是稳定的 (即 y 微小的变化会导致 x 微小的变化, 也被称作解连续)。所有不存在唯一稳定解的问题都被称作**不适定问题** (Ill-posed Problem)。对于机器学习问题, 解的存在性比较容易理解。解的唯一性大多由问题决定。比如, 如果把描述问题的函数 $f(\cdot)$ 看作一个 $n \times n$ 矩阵 \mathbf{A} , x 和 y 都

⁵在训练中, 如果把源语言句子看作是不变的量, 这时 f 的输入只有模型参数。

看作是 n 维向量。那么 x 不唯一的原因在于 \mathbf{A} 不满秩（非奇异矩阵）。不过，存在性和唯一性并不会对机器学习方法造成太大困扰，因为在实践中往往会找到近似的解。

但是，解的稳定性却给神经机器翻译带来了很大的挑战。因为神经机器翻译模型非常复杂，里面存在大量的矩阵乘法和非线性变化。这导致 $f(\cdot)$ 往往是不稳定的，也就是说，神经机器翻译中输出 y 的微小变化会导致输入 x 的巨大变化。比如，在系统研发中经常会发现，即使训练样本发生很小的变化，模型训练得到的参数都会有非常明显的区别。不仅如此，神经机器翻译模型参数解的稳定性还存在两方面问题：

- 观测数据不充分。由于语言现象的多样性，训练样本只能覆盖非常有限的翻译现象。从样本的表示空间上看，对于没有观测样本的区域，根本无法知道真实解的样子，更不用说稳定性训练了。
- 数据中存在噪声。噪声问题是稳定性训练最大的挑战之一。因为，即使是很小的噪声，也可能会导致解的巨大变化。

以上问题体现出来的现象就是过拟合。因为训练数据有限同时存在噪声，因此模型参数会过分拟合噪声数据。而且，这样的模型参数又与真实（理想）的模型参数相差很远。正则化正是针对这个问题。有时候，正则化也被称作**降噪**（Denoising），虽然它的出发点并不只是去除噪声的影响。图7.11对比了不同函数对二维空间中一些数据点的拟合情况。在过拟合现象中，函数可以完美的拟合所有的数据点，即使有些数据点是噪声。

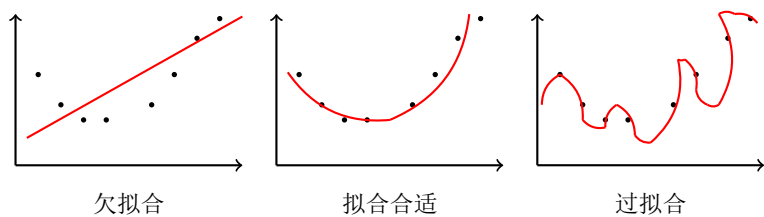


图 7.11: 欠拟合 vs 过拟合

正则化的一种实现是在训练目标中引入一个正则项。在神经机器翻译中，引入正则项的训练目标为：

$$\hat{\mathbf{w}} = \arg \max_{\mathbf{w}} L(\mathbf{w}) + \lambda R(\mathbf{w}) \tag{7.2}$$

其中， $L(\mathbf{w})$ 是损失函数， $R(\mathbf{w})$ 是正则项， λ 是正则项的系数，用于控制正则化对训练影响的程度。 $R(\mathbf{w})$ 通常也可以被看作是一种先验，因为在数据不充分且存在噪声的情况下，可以根据一些先验知识让模型偏向正确的方向一些，而不是一味地根据受噪声影响的不准确的 $L(\mathbf{w})$ 进行优化。相应的，引入正则化后的模型可以获得更好的**泛化**（Generalization）能力，即模型在新的未见数据上表现会更好。

实践中已经证明，正则化方法有助于像神经机器翻译这样复杂的模型获得稳定

的模型参数。甚至有些情况下，如果不引入正则化，训练得到的翻译模型根本无法使用。

L1/L2 正则化

L1/L2 正则化是常用的正则化方法。它们分别对应正则项是 L1 和 L2 范数的情況。具体来说，L1 正则化是指

$$\begin{aligned} R(\mathbf{w}) &= \|\mathbf{w}\|_1 \\ &= \sum_{w_i} |w_i| \end{aligned} \quad (7.3)$$

L2 正则化是指

$$\begin{aligned} R(\mathbf{w}) &= (\|\mathbf{w}\|_2)^2 \\ &= \sum_{w_i} w_i^2 \end{aligned} \quad (7.4)$$

从几何的角度看，L1 和 L2 正则项都是有物理意义的。二者都可以被看作是空间上的一个区域，比如，在二维平面上，L1 范数表示一个以 0 点为中心的矩形，L2 范数表示一个以 0 点为中心的圆。因此，优化问题可以被看作是在两个区域（ $L(\mathbf{w})$ 和 $R(\mathbf{w})$ ）叠加在一起所形成的区域上进行优化。由于 L1 和 L2 正则项都是在 0 点（坐标原点）附近形成的区域，因此优化的过程可以确保参数不会偏离 0 点太多。也就是说，L1 和 L2 正则项引入了一个先验：模型的解不应该离 0 点太远。而 L1 和 L2 正则项实际上是在度量这个距离。

那为什么要用 L1 和 L2 正则项惩罚离 0 点远的解呢？这还要从模型复杂度谈起。实际上，对于神经机器翻译这样的模型来说，模型的容量是足够的。所谓容量可以被简单的理解为独立参数的个数⁶。也就是说，理论上存在一种模型可以完美的描述问题。但是，从目标函数拟合的角度来看，如果一个模型可以拟合很复杂的目标函数，那模型所表示的函数形态也会很复杂。这往往体现在模型中参数的值“偏大”。比如，用一个多项式函数拟合一些空间中的点，如果希望拟合得很好，各个项的系数往往是非零的。而且为了对每个点进行拟合，通常需要多项式中的某些项具有较大的系数，以获得函数在局部有较大的斜率。显然，这样的模型是很复杂的。而模型的复杂度可以用函数中的参数（比如多项式中各项的系数）的“值”进行度量，体现出来就是模型参数的范数。

因此，L1 和 L2 正则项的目的是防止模型为了匹配少数（噪声）样本而导致模型的参数过大。反过来说，L1 和 L2 正则项会鼓励那些参数值在 0 点附近的情况。从实践的角度看，这种方法可以很好的对统计模型的训练进行校正，得到泛化能力更强的模型。

⁶关于模型容量，在 7.3.2 节会有进一步讨论。

标签平滑

神经机器翻译在每个目标语位置 j 会输出一个分布 y_j ，这个分布描述了每个目标语言单词出现的可能性。在训练时，每个目标语言位置上的答案是一个单词，也就对应了 One-hot 分布 \tilde{y}_j ，它仅仅在正确答案那一维为 1，其它维均为 0。模型训练可以被看作是一个调整模型参数让 y_j 逼近 \tilde{y}_j 的过程。但是， \tilde{y}_j 的每一个维度是一个非 0 即 1 的目标，这样也就无法考虑类别之间的相关性。具体来说，除非模型在答案那一维输出 1，否则都会得到惩罚。即使模型把一部分概率分配给与答案相近的单词（比如同义词），这个相近的单词仍被视为完全错误的预测。

标签平滑（Label Smoothing）的思想很简单 [278]：答案所对应的单词不应该“独享”所有的概率，其它单词应该有机会作为答案。这个观点与第二章中语言模型的平滑非常类似。在复杂模型的参数估计中，往往需要给未见或者低频事件分配一些概率，以保证模型具有更好的泛化能力。具体实现时，标签平滑使用了一个额外的分布 q ，它是在词汇表 V 上的一个均匀分布，即 $q(k) = \frac{1}{|V|}$ ，其中 $q(k)$ 表示分布的第 k 维。然后，答案分布被重新定义为 \tilde{y}_j 和 q 的线性插值：

$$y_j^{ls} = (1 - \alpha) \cdot \tilde{y}_j + \alpha \cdot q \tag{7.5}$$

这里 α 表示一个系数，用于控制分布 q 的重要性。 y_j^{ls} 会被作为最终的答案分布用于模型的训练。

标签平滑实际上定义了一种“软”标签，使得所有标签都可以分到一些概率。一方面可以缓解数据中噪声的影响，另一方面目标分布会更合理（显然，真实的分布不应该是 One-hot 分布）。图7.12展示了标签平滑前后的损失函数计算结果的对比。

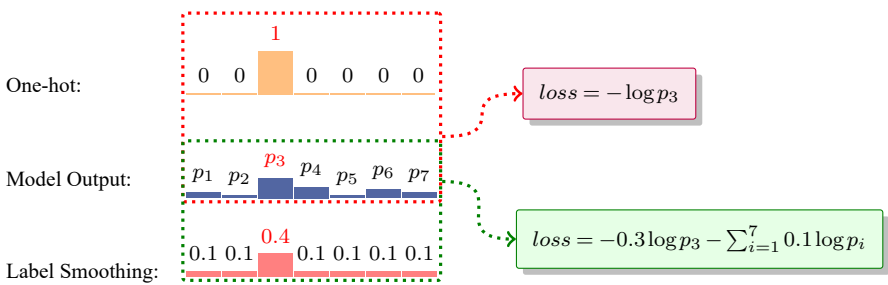


图 7.12: 不使用标签平滑 vs 使用标签平滑

标签平滑也可以被看作是对损失函数的一种调整，并引入了额外的先验知识（即与 q 相关的部分）。只不过这种先验知识并不是通过线性插值的方式与原始损失函数进行融合（公式7.2）。

Dropout

神经机器翻译模型是一种典型的多层神经网络模型。每一层网络都包含若干神经元，负责接收前一层所有神经元的输出，并进行诸如乘法、加法等变换，并有选择

的使用非线性的激活函数，最终得到当前层每个神经元的输出。从模型最终预测的角度看，每个神经元都在参与最终的预测。理想的情况下，我们希望每个神经元都能相互独立的做出“贡献”。这样的模型会更加健壮，因为即使一部分神经元不能正常工作，其它神经元仍然可以独立做出合理的预测。但是，随着每一层神经元数量的增加以及网络结构的复杂化，研究者发现神经元之间会出现**相互适应**（Co-Adaptation）的现象。所谓相互适应是指，一个神经元对输出的贡献与同一层其它神经元的行为是相关的，也就是说这个神经元已经适应到它周围的“环境”中。

相互适应的好处在于神经网络可以处理更加复杂的问题，因为联合使用两个神经元要比单独使用每个神经元的表示能力强。这也类似于传统机器学习任务中往往会设计一些高阶特征，比如自然语言序列标注中对 **bi-gram** 和 **tri-gram** 的使用。不过另一方面，相互适应会导致模型变得更加“脆弱”。因为相互适应的神经元可以更好的描述训练数据中的现象，但是在测试数据上，由于很多现象是未见的，细微的扰动会导致神经元无法适应。具体体现出来就是过拟合问题。

Dropout 是解决这个问题的一种常用方法 [108]。方法很简单，在训练时随机让一部分神经元停止工作，这样每次参数更新中每个神经元周围的环境都在变化，它就不会过分适应到环境中。图7.13中给出了某一次参数训练中使用 **Dropout** 之前和之后的状态对比。

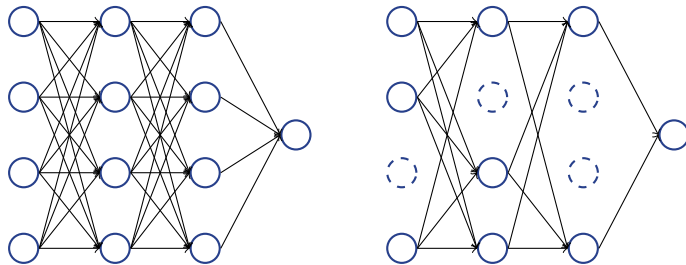


图 7.13: 使用 **Dropout** 之前（左）和之后（右）神经网络状态的对比，其中虚线圆圈表示不工作的神经元，实线圆圈表示工作的神经元。

具体实现时，可以设置一个参数 $p \in (0,1)$ 。在每次参数更新所使用的前向和反向计算中，每个神经元都以概率 p 停止工作。相当于每层神经网络会有以 p 为比例的神经元被“屏蔽”掉。每一次参数更新中会随机屏蔽不同的神经元。图7.14给出了 **Dropout** 方法和传统方法计算方式的对比。

对于新的样本，可以使用 **Dropout** 训练之后的模型对其进行推断，但是每个神经元的输出要乘以 $1 - p$ ，以保证每层神经元输出的期望和训练时是一样的。另一种常用的做法是，在训练时对每个神经元的输出乘以 $\frac{1}{1-p}$ ，然后在推断时神经网络可以不经任何调整就直接使用。

Dropout 方法的另一种解释是，训练中屏蔽掉一些神经元相当于从原始的神经网络中抽取出一个子网络。这样，每次训练都在一个随机生成的子网络上进行，而不同子网络之间的参数是共享的。在推断时，则把所有的子网络集成到一起。这种

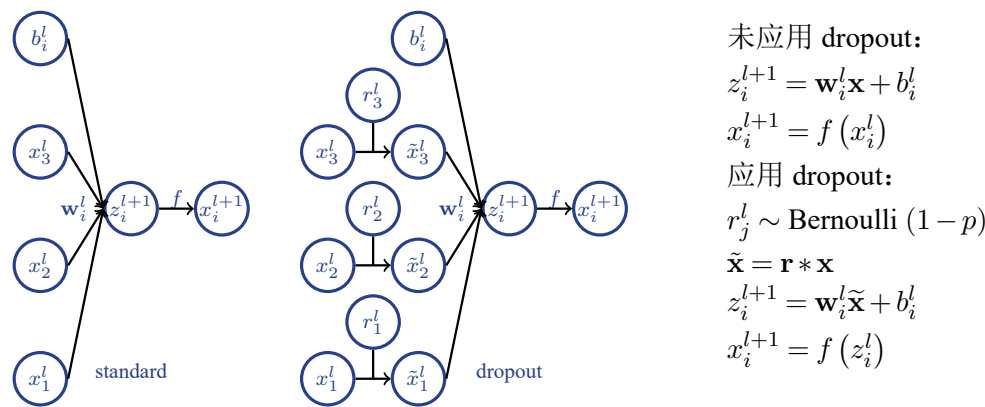


图 7.14: 使用 Dropout 之前（左）和之后（右）一层神经网络的计算

思想也有一些**集成学习**（Ensemble Learning）的味道。只不过 Dropout 中子模型（或子网络）是在指数级空间中采样出来的。由于 Dropout 可以很好的缓解复杂神经模型的过拟合问题，因此也成为了大多数神经机器翻译系统的标配。

Layer Dropout

随时网络层数的增多，相互适应也会出现在不同层之间。特别是在引入残差网络之后，不同层的输出可以进行线性组合，因此不同层之间的相互影响会更加直接。对于这个问题，也可以使用 Dropout 的思想对不同层进行屏蔽。比如，可以使用一个开关来控制一个层能否发挥作用，这个开关以概率 p 被随机关闭，即该层有为 p 的可能性不工作。图 7.15 展示了 Transformer 多层网络引入 Layer Dropout 前后的情况。可以看到，使用 Layer Dropout 后，开关 M 会被随机打开或者关闭，以达到屏蔽某一层计算的目的。由于使用了残差网络，关闭每一层相当于“跳过”这一层网络，因此 Layer Dropout 并不会影响神经网络中数据流的传递。

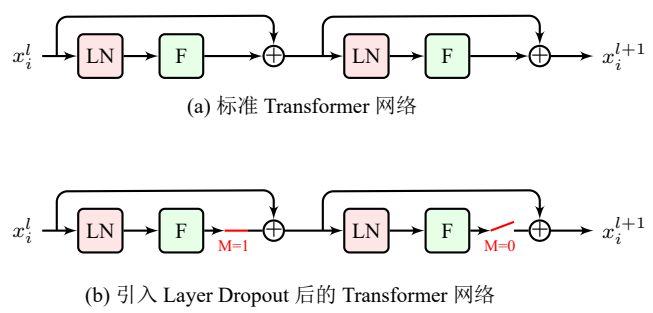


图 7.15: 标准 Transformer 网络 (a) vs 引入 Layer Dropout 后的 Transformer 网络 (b)

Layer Dropout 可以被理解为一个在深网络（即原始网络）中随机采样出一个由若干层网络构成的“浅”网络。不同“浅”网络所对应的同一层的模型参数是共享的。这也达到了对指数级子网络高效训练的目的。需要注意的是，在推断阶段，每层的

输出需要乘以 $1 - p$ ，确保训练时每层输出的期望和解码是一致的。Layer Dropout 可以非常有效的缓解深层网路中的过拟合问题。在7.3.1节还会看到 Layer Dropout 可以成功地帮助我们训练 Deep Transformer 模型。

7.3.2 增大模型容量

神经机器翻译是一种典型的多层神经网络。一方面，可以通过设计合适的网络连接方式和激活函数来捕捉复杂的翻译现象；另一方面，越来越多的可用数据让模型能够得到更有效的训练。在训练数据较为充分的情况下，设计更加“复杂”的模型成为了提升系统性能的有效手段。比如，Transformer 模型有两个常用配置 Transformer-Base 和 Transformer-Big。其中，Transformer-Big 比 Transformer-Base 使用了更多的神经元，相应的翻译品质更优 [288]。

那么是否还有类似的方法可以改善系统性能呢？答案显然是肯定的。这里，把这类方法统称为基于大容量模型的方法。在传统机器学习的观点中，神经网络的性能不仅依赖于架构设计，同样与容量密切相关。那么什么是模型的**容量**（Capacity）？简单理解，容量是指神经网络的参数量，即神经元之间连接权重的个数。另一种定义是把容量看作神经网络所能表示的假设空间大小 [162]，也就是神经网络能表示的不同函数所构成的空间。

而学习一个神经网络就是要找到一个“最优”的函数，它可以准确地拟合数据。当假设空间变大时，训练系统有机会找到更好的函数，但是同时也需要依赖更多的训练样本才能完成最优函数的搜索。相反，当假设空间变小时，训练系统会更容易完成函数搜索，但是很多优质的函数可能都没有被包含在假设空间里。这也体现了一种简单的辩证思想：如果训练（搜索）的代价高，会有更大的机会找到更好的解；另一方面，如果想少花力气进行训练（搜索），那就设计一个小一些的假设空间，在小一些规模的样本集上进行训练，当然搜索到的解可能不是最好的。

在很多机器翻译任务中，训练数据是相对充分的。这时增加模型容量是提升性能的一种很好的选择。常见的方法有三种：

- 增加隐藏层维度：即增加网络宽度，加强每一层网络的线性拟合能力。
- 增加网络的整体层数：即增加网络深度，利用更多的线性和非线性变换来获得更复杂的特征抽取能力。
- 增大输入层和输出层的维度：即增强模型对词表中每个词的表示能力。

宽网络

宽网络通常指隐藏层维度更大的网络，目前在图像处理领域和自然语言处理领域被广泛地使用。第五章已经验证了包含足够多神经元的多层前馈神经网络可以无限逼近任意复杂的连续函数 [114]，这也在一定程度上说明了神经网络建模中神经元数目的重要性。

增大隐藏层神经元的数目是网络变宽的基本方式之一。例如，图像处理领域中

提出的**宽残差网络**（Wide Residual Network）使用更大的卷积核来提高每次卷积计算的精度 [335]；神经机器翻译中，Transformer-Big 模型广受研究人员的认可 [288]，它同样是一个典型的宽网络。对比基线模型 Transformer-Base，Transformer-Big 通过扩大隐藏层维度与滤波器（Filter）维度，取得了显著的翻译性能提升。表7.2是相应的参数设置。

表 7.2: 基线网络与宽网络的参数设置

	Transformer-Base	Transformer-Big
词向量维度	512	1024
注意力头数	8	16
隐藏层维度	512	1024
FFN 子层映射维度	2048	4096

值得注意的是，Transformer 模型中的前馈神经网络子层将隐藏层表示映射到更高维度的空间（通过一个 Filter），之后经过激活函数 Relu 后再映射回原来的维度大小。这个操作对翻译模型的性能有明显的正向作用。从表7.2中可以看出，Filter 的维度是普通隐藏层维度的四倍。通过增大 Filter 大小可以有效地扩展网络的宽度，比如，有些情况下可以将 Filter 增大到 8192 甚至更大。

但伴随着模型变宽，网络的整体参数量会显著增长⁷。同时，宽网络需要更长的训练时间才能达到稳定的收敛状态。此外，训练宽网络时通常需要对一些超参数进行相应的调整，例如 Dropout 的大小，学习率的峰值等。

深网络

虽然，理论上宽网络有能力拟合任意的函数，但是获得这种能力的代价是非常高的。在实践中，往往需要增加相当的宽度，以极大的训练代价才能换来少量的性能提升。当神经网络达到一定宽度后这种现象更为严重。“无限”增加宽度显然是不现实的。

因此，另一种思路是使用更深的网络以增加模型的容量。深网络是指包含更多层的神经网络。相比宽网络的参数量随着宽度呈平方增长，深网络的参数量随着深度呈线性增长。这带给深网络一个优点：在同样参数量下可以通过更多的非线性变换来对问题进行描述。这也赋予了深网络对复杂问题建模的能力。比如，在图像识别领域，很多先进的系统都是基于很深的神经网络，甚至在一些任务上最好的的结果需要 1000 层以上的神经网络。

宽网络和深网络是增加模型表示能力的两个维度。宽网络相当于增强了模型线性变换的能力，将模型的输入在更高维度的空间上进行抽象；深网络通过引入更多的层构建了多个表示空间，通过逐层的变换，在多个表示空间上对输入进行多次抽象。二者在有些情况下甚至可以相互转换。

⁷在一个全连接神经网络中，参数的数量与各层宽度呈平方关系。

除了数学上的解释，深度神经网络也可以给分析、理解现实世界的问题提供有效的手段。很多时候，可以把一个多层神经网络看作是对一个复杂问题的拆解，每层（或每几层）是在处理一个子问题。例如，在人脸识别任务中，一个 3 层的神经网络中第一层主要提取低层次的简单特征，即边缘特征；第二层将简单的特征组合成更为复杂的特征，如器官特征；第三层针对第二层的输出进行进一步的抽象得到人脸的面部特征。这样，深网络通过不同层的逐层特征抽象可以在人脸识别数据集上超越人类的精度 [101]。

类似的现象也出现在基于语言模型的预训练任务中。比如，研究人员通过使用**探测任务**（Probing Task）来分析 12 层的 BERT 模型中的不同层所表示的含义 [74, 126]：

- 浅层网络表示：网络的底层部分更擅长处理词串的**表面信息**（Surface Information），例如词性选择、词义消歧等。
- 中间层的表示：中间层部分更关注于**语法信息**（Syntactic Information）
- 顶层网络的表示：上层部分更擅长处理**语义信息**（Semantic Information）

目前在神经机器翻译领域，研究人员发现编码端的表示能力对翻译性能有较大的影响，因此加深编码网络是一种有效的改进系统的手段（如图 7.16）。而且，增加编码端的深度对模型推断的速度并没有较大影响，因为整个序列可以通过 GPU 进行并行计算。

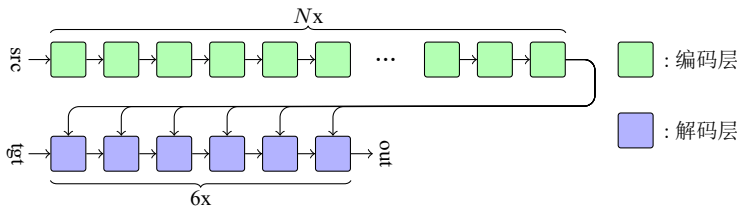


图 7.16: 增大神经机器翻译的编码端来增大模型的容量

不过，深网络容易发生梯度消失和梯度爆炸问题。因此在使用深网络时，训练策略的选择是至关重要的。实际上，标准的 Transformer 模型已经是不太“浅”的神经网络，因此里面使用了残差连接来缓解梯度消失等问题。此外，为了避免过拟合，深层网络的训练也要与 Dropout 等正则化策略相配合，并且需要设计恰当的参数初始化方法和学习率调整策略。关于构建深层神经机器翻译的方法，本章 7.5.1 节还会有进一步讨论。

增大输入层和输出层表示能力

如前所述，神经机器翻译的原始输入是单词序列，包括源语言端和目标语言端。模型中的输入层将这种离散的单词表示转换成实数向量的表示，也就是常说的**词嵌入**（Embedding）。从实现的角度来看，输入层其实就是从一词嵌入矩阵中提取对应的词向量表示，这个矩阵两个维度大小分别对应着词表大小和词嵌入的维度。词

嵌入的维度也代表着模型对单词刻画的能力。因此适当增加词嵌入的维度也是一种增加模型容量的手段。通常，词嵌入和隐藏层的维度是一致的，这种设计也是为了便于系统实现。

当然，并不是说词嵌入的维度一定越大就越好。本质上，词嵌入是要在一个多维空间上有效的区分含有不同语义的单词。如果词表较大，更大的词嵌入维度会更有意义，因为需要更多的“特征”描述更多的语义。当词表较小时，增大词嵌入维度可能不会带来增益，相反会增加系统计算的负担。另一种策略是，动态选择词嵌入维度，比如，对于高频词使用较大的词嵌入维度，而对于低频词则使用较小的词嵌入维度 [6]。这种方法可以用同样的参数量处理更大的词表。

大模型的分布式计算

伴随着模型容量的增大，复杂模型可能无法在单 GPU 上完成训练。比如，即使是不太复杂的 Transformer-Base 模型在很多任务上也需要在 8 张 GPU 进行训练。如何利用多个设备进行大模型的并行训练是一个很现实的问题。比较简单的策略是使用**数据并行**（Data Parallelism），即把一个批次分到多个 GPU 上进行训练，之后对多个 GPU 上的梯度进行汇总，并更新参数。不过，当模型规模增大到一定程度时，单 GPU 可能仍然无法处理。这个问题在 GPU 显存较小的时候会非常突出。这时需要考虑**模型并行**（Model Parallelism）。模型并行是指将模型分割成不同的部分，在不同的 GPU 上运行其中的一部分。例如，在训练深层 LSTM 模型时可以将不同层放置在不同 GPU 上，这种方式一定程度上能够加速模型的训练。对于更大的模型，如参数量为 10 亿的 BERT-Large 模型 [56]，同样可以使用这种策略。不过，模型并行中不同设备传输的延时会大大降低模型运行的效率，因此很多时候要考虑训练效率和模型性能之间的平衡。

7.3.3 大批量训练

在第六章已经介绍了神经机器翻译模型需要使用梯度下降方法进行训练。其中，一项非常重要的技术就是**小批量训练**（Mini-batch Training），即每次使用多个样本来获取梯度并对模型参数进行更新。这里将每次参数更新使用的多个样本集合称为批次，将样本的数量称作批次的大小。在机器翻译中，通常用批次中的源语言/目标语言单词数或者句子数来表示批次大小。理论上，过小的批次会带来训练的不稳定，而且参数更新次数会大大增加。因此，很多研究者尝试增加批次大小来提高训练的稳定性。在 Transformer 模型中，使用更大的批次已经被验证是有效的。这种方法也被称作大批量训练。不过，这里所谓“大”批量是一个相对的概念。下面就一起看一看如何使用合适的批次大小来训练神经机器翻译模型。

为什么需要大批量训练

在模型训练的过程中，训练批次的大小对模型的收敛状态有很大影响，合理选择批次的大小往往会取得事半功倍的效果。较大的批次可以保证模型每次更新时梯

度是在较多的样本上计算，其梯度结果更准确，有效地缓解训练中出现的性能震荡的问题。此外，较大的批次可以让模型在更新次数更少的情况下遍历整个数据集。同时大矩阵运算在 GPU 上的并行度更高，提高了设备的利用率。

然而，批次也不是越大越好，要根据训练数据集的规模与模型的容量做出合理的选择。此外，GPU 显存的大小也对批次大小有约束，因为过大的批次可能无法放入 GPU 显存中。另一方面，在一些场景例中，如增量式训练、领域迁移，往往需要对模型进行微调。这时，常用的做法是使用小批次并固定较小的学习率，防止现有的模型参数发生较大的偏离。

大多数情况下，批次的大小是指单独一块 GPU 上的数据量，然而考虑到利用数据并行进行分布式训练时，批次大小等于所有 GPU 中批次大小的和。下面以 Transformer 模型为例。通常 Transformer-Base 模型使用 4096 词/GPU 的批次在 8 张 GPU 上进行训练，此时真实批次大小为 $4096 \times 8 = 32768$ 词。伴随着模型容量的进一步增加，例如 Transformer-Big 模型，由于训练过程中网络的中间表示要消耗大量的 GPU 显存，可能会考虑减小批次的大小并使用累计梯度的方式来保证稳定的训练。累计梯度是一种大批量训练的常用手段，即按照一定频率缓存多个相邻批次的梯度后再进行参数的更新。比如，为了获取大批次，可以考虑将累计的更新频率设置为 2 或 4。图7.17给出了累计梯度的参数更新方法，可以看到使用累积梯度的方式可以减少设备的空闲时间。

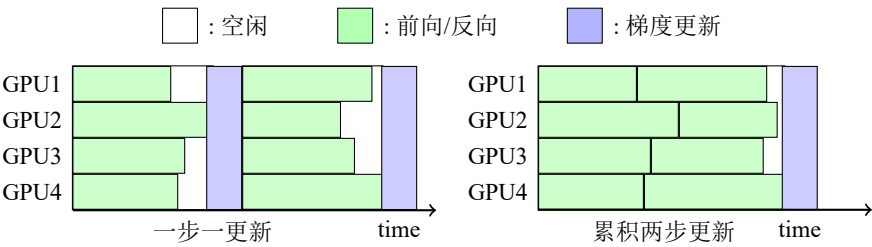


图 7.17: 生成批次的方式

此外，前人工作表明，使用大批量训练复杂网络结构时要配合略大一些的学习率，加快模型在梯度方向上的更新速度，进而达到更优的翻译性能 [225]。例如，深层网络也需要对学习率进行适当的调整才能发挥较好的性能。表7.3展示了 30 层网络在不同批次大小和学习率峰值的条件下的 BLEU 值（WMT14 En-De）⁸。可以发现，在固定学习率峰值的条件下增大批次大小并不能带来性能上的增益，必须同时调整学习率的峰值。也有研究团队验证了 Transformer-Big 模型在 128 张 GPU 上进行分布式训练时，适当的增大学习率会带来明显的 BLEU 提升 [225]。

⁸学习率峰值是指 Transformer 模型训练的预热阶段，学习率所到达的最高值。

表 7.3: 深层 Transformer 不同批次大小和学习率峰值设置下的 BLEU 值

batch	lr	BLEU
4096	0.01	29.15
8192	0.01	29.06
8192	0.02	29.49

如何构建批次

由于不同句子之间的长度有明显的差异，这时批次所占用的显存/内存大小由其中最长句子的长度决定。通常使用**填充**（Padding）的方式对一个批次中句长不足的部分用空白填充（见第六章）。但由于生成批次的方式不同，最终批次内的 Padding 数量各不相同，因此合理选择生成批次的方式也是至关重要的。通常有几种方法：

- 随机生成：最简单的方式是从整个数据集中随机生成批次。这种方式可以有效地保证样本间的随机性，但随机生成的批次中不同句子之间的长度会有较大区别，因此 Padding 数量较多会导致显卡的利用率较低。
- 按句长排序：为了减少显卡利用率低的问题，可以根据源语言或者目标语言的句子长度进行排序，让相邻句长的样本更为相近（图7.18）。这样在同一个批次中不会因为句长差异过大造成设备利用率的降低。

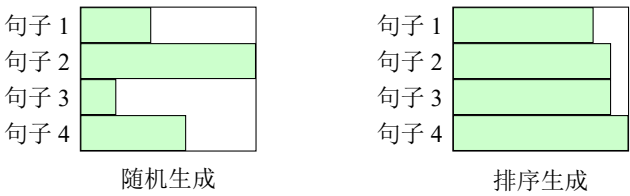


图 7.18: 批次生成 - 随机生成 vs 按句长生成

- 按词数构建批次：对比按照句长生成批次，按词数生成批次可以防止某些批次中句子整体长度特别长或者特别短的情况，保证不同批次之间整体的词数处于大致相同的范围，这样所得到的梯度也是可比较的。通常的做法是根据源语言词数、目标语言词数，或者源语言词数与目标语言词数的最大值等指标生成批次。
- 按课程学习的方式：考虑样本的“难度”也是生成批次的一种策略。比如，可以使用**课程学习**（Curriculum Learning）的思想 [15]，让系统先学习“简单”的样本，之后逐渐增加样本的难度，达到循序渐进的学习。具体来说，可以利用句子长度、词频等指标计算每个批次的“难度”，记为 d 。之后，选择满足 $d \leq c$ 的样本构建一个批次。这里， c 表示难度的阈值，它可以随着训练的执行不断增大。

7.4 推断

7.4.1 推断优化

自统计机器翻译时代开始，推断装置一直是机器翻译系统研发的核心模块。所谓**推断**（Inference）是指，利用训练好的模型对新的句子进行翻译的过程，这本质上是一个搜索过程。在机器翻译中，这个过程也被称作**解码**（Decoding）。但是为了避免与神经机器翻译的解码端造成概念上的混淆，这里统一把翻译新句子的操作称作推断。

神经机器翻译的推断通常包含三方面问题：

- 搜索的**准确性**（Accuracy），也就是搜索到结果的好坏；
- 搜索的**时延**（Latency），也就是翻译速度；
- 搜索所需要的**存储**（Memory），也就是模型运行时的内存或显存消耗。

准确性通常是研究人员最关心的问题。作为一个搜索过程，需要依据某种指标（如模型得分）找到样本空间中的一个或者若干个样本。不过，机器翻译是一个 NP 难问题，对整个样本空间进行全搜索显然是十分困难的 [145]。因此，需要优化搜索算法，并配合剪枝等策略，最终得到一个尽可能逼近全局最优解的搜索结果。

如果搜索算法没有找到全局最优解，这时称系统出现了**搜索错误**（Search Error）。如果模型打分不准确造成没有把最好的翻译排序到第一，这时称系统出现了**模型错误**（Modeling Error）。模型错误是由建模和模型训练等因素决定的，而搜索错误一般是由搜索算法决定的。在早期的机器翻译研究中，搜索错误是机器翻译问题的主要来源之一。不过随着技术的进步，研究者逐渐发现，机器翻译系统的错误更多的集中在模型错误上 [266]。特别是在神经机器翻译时代，绝大多数研究工作都是在解决模型错误。

当然，这里并不是说搜索不重要。相反，在很多应用场景中，搜索算法的效率会起到决定性作用，比如，机器同传 [190]。而且，往往需要在翻译精度和速度之间找到一种折中。下面将对神经机器翻译推断的精度和效率进行讨论。

推断系统的架构

对于神经机器翻译，推断的时间主要消耗在解码端，也就是对源语言进行编码后，解码器生成译文单词序列的过程。因此，神经机器翻译的推断装置的开发大多集中在设计解码器及搜索策略上。通用的神经机器翻译搜索算法包括如下几步：

- 对输入的源语言句子进行编码⁹；
- 使用源语言句子的编码结果，在目标语言端自左向右逐词生成译文；
- 在目标语言的每个位置计算模型得分，同时进行剪枝；

⁹由于整个源语言句子是可见的，因此这个步骤可以一步执行完毕

- 当满足某种条件时终止搜索。

上面这个算法与统计机器翻译中自左向右翻译的算法基本上是一样的（见第四章），即，在每一个目标语位置，根据已经生成的译文和源语言的信息，生成下一个译文单词。这个过程可以由两个模块实现 [268]：

- 预测模块，也就是根据已经翻译的历史和源语言句子，预测下一个要生成单词的概率¹⁰。因此预测模块实际上就是一个模型打分装置；
- 搜索模块，它会利用预测结果，对当前的翻译假设进行打分，并根据模型得分对翻译假设进行排序和剪枝。

预测模块是由模型决定的，而搜索模块可以与模型无关。也就是说，不同的模型可以共享同一个搜索模块完成推断，系统的代码有更好的复用性。比如，对于 RNN 模型，预测模块需要读入前一个状态的信息和前一个位置的译文单词，然后预测当前位置的单词概率；对于 Transformer，预测模块需要对前面的所有位置做注意力运算，之后预测当前位置的单词概率。不过，这两个模型都可以使用同一个搜索模块。图7.19给出了这种架构的示意图。

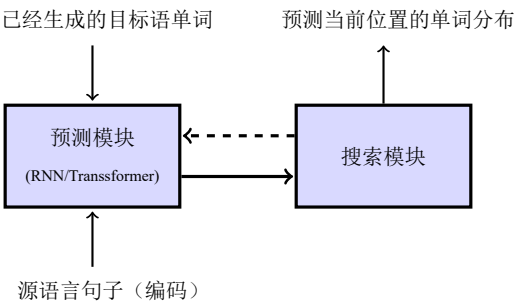


图 7.19: 神经机器翻译推断系统结构

在具体实现时，还有很多细节需要注意，比如：词汇表大小、束搜索的宽度、解码终止条件等等。不同的设置对翻译的品质和速度都会有影响。这些内容可以参考第六章中的相关介绍，在本章的7.4.1节也会有进一步进行讨论。

自左向右推断 vs 自右向左推断

自左向右翻译符合人类对语言使用的规律，因为在翻译一个句子时，我们总是习惯从句子开始的部分往后生成。不过，有时候人也会使用当前单词后面的译文信息。也就是说，翻译也需要“未来”的上下文。于是很容易想到使用自右向左的方法对译文进行生成。

需要注意的是，自右向左的翻译需要对模型进行调整。比较简单的方式是直接

¹⁰在统计机器翻译中，翻译的每一步也可以预测短语。在神经机器翻译中也有类似于生成短语的方法，但是主流的方法还是按单词为单位进行生成。

将训练用双语数据中的目标语句子进行反向，之后仍然使用原始的模型进行训练即可。在测试的时候，推断生成的目标语词串也需要进行反向得到最终的译文。

不过，简单的使用自右向左的翻译方式并不会取得更好的效果。大多数情况下，自右向左比自左向右推断的翻译质量更差。因此很多时候是对二者进行融合，而不是简单的相互替换。有两种思路：

- **重排序 (Re-ranking)**。可以用一个基础模型（比如自左向右的模型）得到每个源语言句子的 n -best 结果，之后同时用基础模型的得分和自右向左模型对 n -best 结果进行重排序 [165, 250]。由于这种方法不会改变基础模型的翻译过程，因此相对“安全”，不会对系统性能造成副作用。特别是对于基于循环神经网络的翻译系统，利用自右向左的翻译模型进行重排序往往会取得较好的效果。
- **双向推断 (Bidirectional Inference)**。另一种方法是，让自左向右和自右向左模型同步进行，也就是同时考虑译文左侧和右侧的上下文 [350]。例如，可以同时对左边和右边生成的译文进行注意力计算，得到当前位置的单词预测结果。这种方法能够更加充分的融合双向翻译的优势，不过需要对训练和推断系统进行修改，因此也引入了额外的开发和调试工作。

不论是自左向右还是自右向左翻译，本质上都是在对上下文信息进行建模。最近，以 BERT 为代表的预训练语言模型已经证明，一个单词的“历史”和“未来”信息对于生成当前单词都是有帮助的 [56]。类似的观点也在神经机器翻译编码器设计中得到验证。比如，在基于循环神经网络的模型中，经常同时使用自左向右和自右向左的方式对源语言句子进行编码，还有，Transformer 编码会使用整个句子的信息对每一个源语言位置进行表示。因此，在神经机器翻译的解码端采用类似的策略是有其合理性的。

推断加速

很多时候，我们需要在翻译速度和翻译精度之间进行平衡。即使是以提升翻译品质为目标的任务（如用 BLEU 评价），也不得不考虑翻译速度的影响。比如，在 WMT 和 CCMT 的一些任务中可能会使用反向翻译构造伪数据，需要大量的机器翻译；无指导机器翻译中也会频繁地使用神经机器翻译系统构造训练数据。如果翻译速度过慢会增大实验的周期。从应用的角度看，在很多场景下翻译速度甚至比品质更重要。比如，在线翻译和一些小设备上的机器翻译系统都需要保证相对低的翻译延时，以满足用户体验的最基本要求。虽然，我们希望能有一套又好又快的翻译系统，但是现实的情况是：往往需要通过牺牲一些翻译品质来换取速度的提升。

下面就列举一些常用的神经机器翻译加速方法。这些方法通常是应用在解码端，因为相比编码端，神经机器翻译的解码端是推断过程中最耗时的部分。

a) 输出层的词汇选择

神经机器翻译需要对输入和输出的单词进行分布式表示，比如，每一个单词都

用一个 512 维向量进行表示。但是，由于真实的词表通常很大，因此计算并保存这些单词向量表示就会消耗较多的计算和存储资源。特别是对于基于 Softmax 的输出层，使用大词表往往会占用较多的系统运算时间。虽然可以通过 BPE 和限制词汇表规模的方法降低输出层计算的负担，但是为了获得可接受的翻译品质，词汇表也不能过小（比如小于 10000），因此输出层仍然十分耗时。

对于这个问题，可以通过改变输出层的网络结构进行缓解 [127]。一种比较简单的方法是对可能输出的单词进行筛选，简称词汇选择。这里，可以利用类似于统计机器翻译的翻译表，获得每个源语言单词最可能的译文。在翻译过程中，利用注意力机制找到每个目标语位置对应的源语言位置，之后获得这些源语言单词最可能的翻译候选。之后，Softmax 只需要在这个有限的翻译候选单词集合上计算，大大降低了输出层的计算量。尤其是对于 CPU 上的系统，这个方法往往会带来明显的速度提升，同时保证翻译品质。图7.20给出了词汇选择方法的示意图。

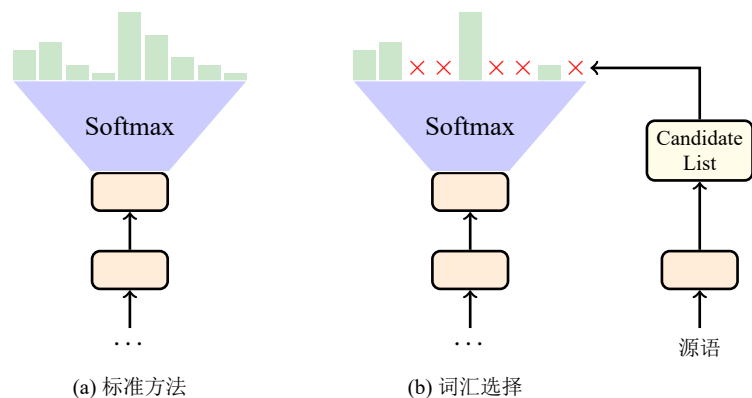


图 7.20: 标准 Softmax vs 基于词汇选择的 Softmax

b) 消除冗余计算

消除不必要的计算是加速机器翻译的常用技术。比如，在统计机器翻译时代，假设重组就是一种典型的避免冗余计算的手段（第四章）。对于神经机器翻译中的 Transformer 模型，一种简单有效的方法是对解码端的注意力结果进行缓存。在生成每个目标语译文时，Transformer 模型会对当前位置之前所有位置进行自注意力操作，但是这些计算里只有和当前位置相关的计算是“新”的，前面位置之间的注意力结果已经在之前的解码步骤里计算过，因此可以对其进行缓存。

此外，由于 Transformer 模型较为复杂，还存在很多冗余。比如，Transformer 的每一层会包含自注意力机制、层正则化、残差连接、前馈神经网络等多种不同的结构。同时，不同结构之间还会包含一些线性变换。多层 Transformer（通常为 6 层）模型会更加复杂。但是，这些层可能在做相似的事情，甚至有些计算根本就是重复的。图7.21中展示了解码端自注意力和编码-解码注意力中不同层的注意力权重的相似性，这里的相似性利用 Jensen-Shannon 散度进行度量 [177]。可以看到，自注意力中，2-5

层之间的注意力权重的分布非常相似。编码-解码注意力也有类似的现象，临近的层之间有非常相似的注意力权重。这个现象说明：在多层神经网络中有些计算是冗余的，因此很自然的想法是消除这些冗余使得机器翻译变得更“轻”。

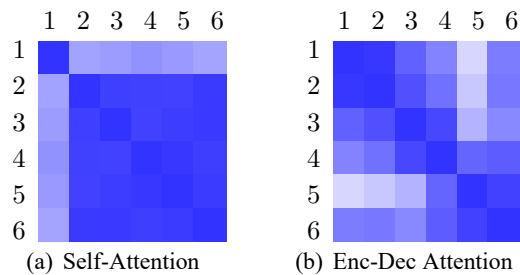


图 7.21: 自注意力和编码-解码注意力中不同层之间注意力权重的相似性（深色表示相似）

一种方法是将不同层的注意力权重进行共享，这样上层的注意力权重可以复用下层的注意力权重。在编码-解码注意力中，由于注意力机制中输入的 **Value** 都是一样的¹¹，我们甚至可以直接复用前一层注意力计算的结果。图7.22给出了不同方法的对比，其中 *S* 表示注意力权重，*A* 表示注意机制的输出。可以看到，使用共享的思想，可以大大减少冗余的计算。当然，这种方法也可能会带来翻译精度的损失。在 WMT 的多个任务上发现，使用层共享的方法可以带来 20% 以上的提速，同时 BLEU 的浮动在 0.2 个点左右 [316]。

另一种方法是对不同层的参数进行共享。这种方法虽然不能带来直接的提速，但是可以大大减小模型的体积。比如，可以重复使用同一层的参数完成多层的计算。极端一些的情况下，六层网络可以只使用一层网络的参数 [53]。不过，在深层模型中（层数 > 20），浅层部分的差异往往较大，而深层（远离输出）之间的相似度会更高。这时可以考虑对深层的部分进行更多的共享。

c) 轻量解码端及小模型

在推断时，神经机器翻译的解码端是最耗时的，因为每个目标语位置需要单独输出单词的分布，同时在搜索过程中每一个翻译假设都要被扩展成多个翻译假设，进一步增加了计算量。因此，另一种加速系统的思路是使用更加轻量的解码器神经网络。

比较简单的做法是把解码端的网络变得更“浅”、更“窄”。所谓浅网络是指使用更少的层构建神经网络，比如，使用 3 层，甚至 1 层网络的 Transformer 解码器。所谓窄网络是指将网络中某些层中神经元的数量减少。不过，直接训练这样的小模型会带来翻译品质的下降。这时会考虑使用知识精炼（见 7.5.3 节）或深层编码器（见 7.3.1 节）配合基于小模型的解码神经网络一起使用。

¹¹ 在 Transformer 解码端，编码-解码注意力输入的 **Value** 是编码端的输出，因此是相同的（详见第六章关于 Transformer 模型的内容）。

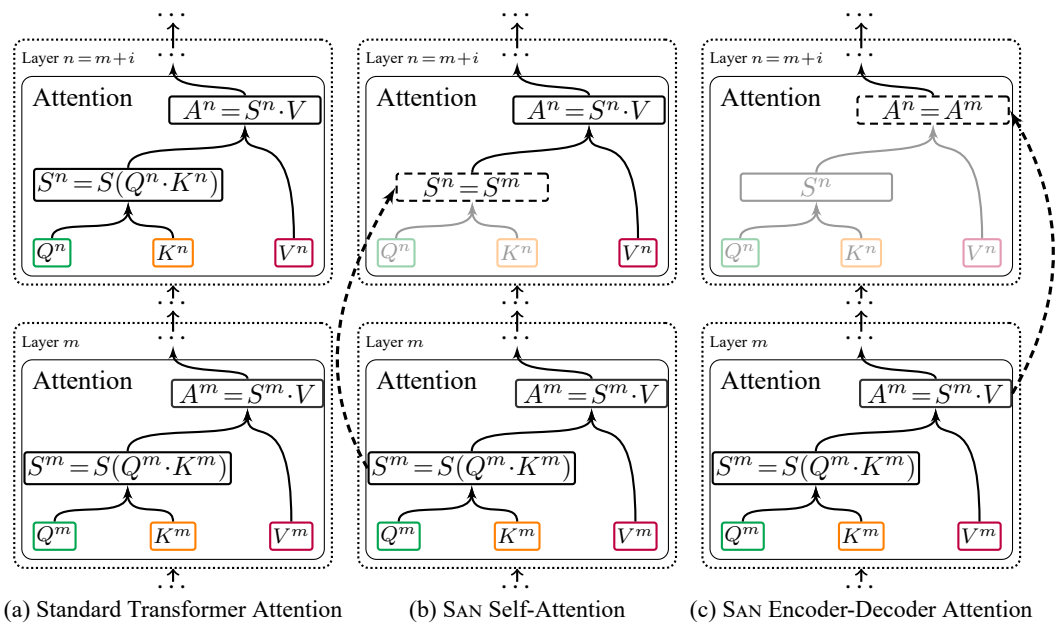


图 7.22: 标准的多层自注意力、共享自注意力、共享编码-解码注意力方法的对比 [316]

另一种思路是化简 Transformer 的解码端神经网络。比如，可以使用平均注意力机制代替原始的 Transformer 自注意力机制 [340]，也可以使用运算更轻的卷积操作代替注意力模块 [310]。前面提到的基于共享注意力机制的模型也是一种典型的轻量模型 [316]。

此外，使用异构神经网络也是一种平衡精度和速度的有效方法。在很多研究中发现，基于 Transformer 的编码器对翻译品质的影响更大，而解码端的作用会小一些。因此，一种想法是用更快速的解码端结构，比如，用基于循环神经网络的解码端替换基于 Transformer 的解码端 [34]。这样，既能发挥 Transformer 在编码上的优势，同时也能利用循环神经网络在解码端速度上的优势。使用类似的思想，也可以用卷积网络等结构进行解码端网络的设计。

d) 批量推断

深度学习时代下，使用 GPU（图形处理单元）已经成为绝大多数神经网络模型研究的基本要求。特别是对于机器翻译这样的复杂任务，GPU 的并行运算能力会带来明显的速度提升。为了充分利用 GPU 的并行能力，可以同时多个句子进行翻译，即**批量推断**（Batch Inference）。

在第六章已经介绍了神经机器翻译中**批量处理**（Batching）的基本概念。其实现并不困难，不过有两方面问题需要注意：

- 批次生成策略。对于源语言文本预先给定的情况，通常是按句子长度组织每个批次，即：把长度相似的句子放到一个批次里。这样做的好处是可以尽可能保

证一个批次中的内容是“满”的，否则如果句长差异过大会造成批次中有很多位置用占位符填充，产生无用计算。对于实时翻译的情况，批次的组织较为复杂。由于有翻译延时的限制，可能无法等到有足够多的句子就要进行翻译。常见的做法是，设置一个等待的时间，在同一个时间段中的句子可以放到一个批次中（或者几个批次中）。对于高并发的情况，也可以考虑使用不同的 Bucket 保存不同长度范围的句子，之后将同一个 Bucket 中的句子进行批量推断。

- 批次大小的选择。一个批次中的句子数量越多，GPU 设备的利用率越高，系统吞吐越大。但是，一个批次中所有句子翻译结束后才能拿到翻译结果，因此批次中有些句子即使已经翻译结束也要等待其它没有完成的句子。也就是说，从单个句子来看，批次越大翻译的延时越长，这也导致在翻译实时性要求较高的场景中，不能使用过大的批次。而且，大批次对 GPU 显存的消耗更大，因此也需要根据具体任务合理选择批次大小。为了说明这些问题，图7.23展示了不同批次大小下的吞吐、延时和显存消耗。

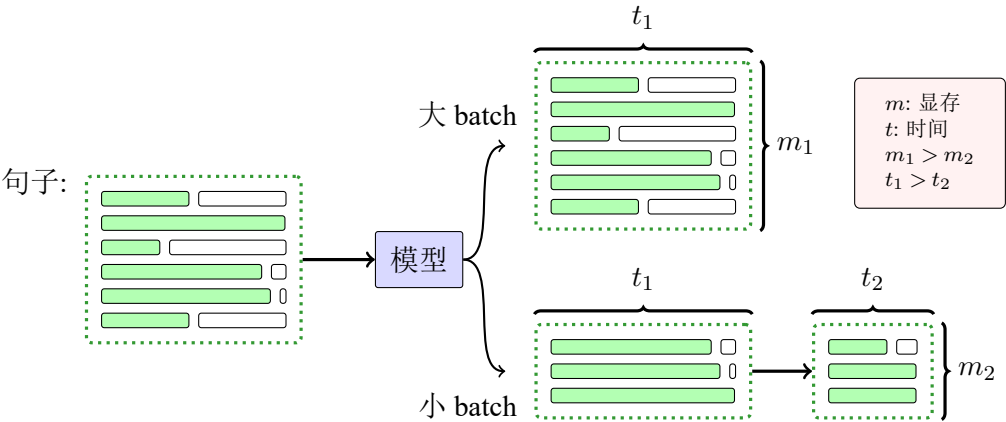


图 7.23: Transformer 系统在不同批次大小下的吞吐、延时和显存消耗

e) 低精度运算

降低运算强度也是计算密集型任务的加速手段之一。标准的神经机器翻译系统大多基于单精度浮点运算。从计算机的硬件发展看，单精度浮点运算还是很“重”的。当计算能容忍一些精度损失的时候，可以考虑降低运算精度来达到加速的目的。比如：

- 半精度运算。半精度运算是随着近几年 GPU 技术发展而逐渐流行的一种运算方式。简单来说，半精度的表示要比单精度需要更少的存储单元，所表示的浮点数范围也相应的变小。不过，实践中已经证明神经机器翻译中的许多运算用半精度计算就可以满足对精度的要求。因此，直接使用半精度运算可以大大加速系统的训练和推断进程，同时对翻译品质的影响很小。不过，需要注意的是，

在分布式训练的时候，由于参数服务器需要对多个计算节点上的梯度进行累加，因此保存参数的部分仍然会使用单精度浮点以保证多次累加之后不会造成精度的损失。

- 整型运算。整数运算是一种比浮点运算“轻”很多的运算。无论是芯片占用面积、能耗还是处理单次运算的时钟周期数，整数运算相比浮点运算都有着明显的优势。因此，使用整数运算也是很有潜力的加速手段。不过，整数的表示和浮点数有着很大的不同。一个基本的问题是，整数是不连续的，因此无法准确的刻画浮点数中很小的数。对于这个问题，一种解决方法是利用“量化 + 反量化 + 缩放”的策略让整数运算近似浮点运算的效果 [19, 125, 234])。所谓“量化”就是把一个浮点数离散化为一个整数，“反量化”是这个过程的逆过程。由于浮点数可能超出整数的范围，因此会引入一个缩放因子。在量化前将浮点数缩放到整数可以表示的范围，反量化前再缩放回原始浮点数的表示范围。这种方法在理论上可以带来很好的加速效果。不过由于量化和反量化的操作本身也有时间消耗，而且在不同处理器上的表现差异较大。因此不同的实现方式带来的加速效果并不相同，需要通过实验测算。
- 低精度整型运算。使用更低精度的整型运算是进一步加速的手段之一。比如使用 16 位整数、8 位整数，甚至 4 位整数在理论上都会带来速度的提升（表 7.4）。不过，并不是所有处理器都支持低精度整数的运算。开发这样的系统，一般需要硬件和特殊低精度整数计算库的支持。而且相关计算大多是在 CPU 上实现，应用会受到一定的限制。

表 7.4: 不同计算精度的芯片的运算速度对比¹²

指标	FP32	INT32	INT16	INT8	INT4
速度	1×	3~4×	≈4×	4~6×	≈8×

实际上，低精度表示的另一个好处是可以减少模型存储的体积。比如，如果要把机器翻译模型作为软件的一部分打包存储，这时可以考虑用低精度的方式保存模型参数，使用时再恢复成原始精度的参数。值得注意的是，参数的离散化表示（比如整型表示）的一个极端例子是**二值网络**（Binarized Neural Networks）[121]，即只用 -1 和 +1 表示网络的每个参数。二值化可以被看作是一种极端的量化手段。不过，这类方法还没有在机器翻译中得到大规模验证。

f) 非自回归翻译

神经机器翻译的推断是一种**自回归翻译**（Autoregressive Translation）过程。所谓

¹²表中比较了几种通用数据类型的乘法运算速度，不同硬件和架构上不同类型的数据的计算速度略有不同。总体来看整型数据类型和浮点型数据相比具有显著的计算速度优势，INT4 相比于 FP32 数据类型的计算最高能达到 8 倍的速度提升。

自回归是一种描述时间序列生成的方式。对于序列 $\{x_1, \dots, x_m\}$ ，自回归模型假设 i 时刻状态 x_i 的生成依赖于之前的状态 $\{x_1, \dots, x_{i-1}\}$ ，而且 x_i 与 $\{x_1, \dots, x_{i-1}\}$ 构成线性关系。神经机器翻译借用了这个概念，但是并不要求线性模型。对于输入的源语言序列 $\mathbf{x} = \{x_1, \dots, x_m\}$ ，用自回归翻译模型生成译文序列 $\mathbf{y} = \{y_1, \dots, y_n\}$ 的概率可以被定义为：

$$P(\mathbf{y}|\mathbf{x}) = \prod_{j=1}^n P(y_j|\mathbf{y}_{<j}, \mathbf{x}) \quad (7.6)$$

即译文单词 y_j 依赖前面已经生成的单词 $\mathbf{y}_{<j} = \{y_1, \dots, y_{j-1}\}$ 和源语言句子 \mathbf{x} 。显然，这个模型中，每一个目标语位置 j 都需要等待前面 $j-1$ 个位置输出的结果。因此，自回归翻译会阻碍不同译文单词生成的并行化。特别是在 GPU 上，翻译的自回归性会大大降低计算的并行度，导致推断过程的效率不高。

对于这个问题，研究者也考虑移除翻译的自回归性 [96]，进行**非自回归翻译**（Non-Autoregressive Translation）。非自回归翻译可以用如下公式描述：

$$P(\mathbf{y}|\mathbf{x}) = \prod_{j=1}^n P(y_j|\mathbf{x}) \quad (7.7)$$

其中，位置 j 上的输出 y_j 只依赖于输入句子 \mathbf{x} ，与其它位置上的输出无关。于是，所有位置上 y_j 都可以并行生成，大大提高了 GPU 等并行运算设备的利用率。这种方式一般可以带来几倍的速度提升。

不过非自回归翻译也面临一些挑战。最主要的问题在于，系统无法靠生成 $\langle \text{eos} \rangle$ 等结束符来判断翻译是否结束，也就是，非自回归翻译不知道译文的长度。对于这个问题，可以考虑借用统计机器翻译中**繁衍率**（Fertility）的概念（见第三章）。对于每个源语言单词预测所对应的目标语单词的个数，即繁衍率。这样，目标语句子的长度可以由这个源语言的繁衍率序列决定。

繁衍率模型也可以被看作是一种基于隐含变量的非自回归模型。也可以考虑使用其它的先验知识设计隐含变量，比如，用句法结构指导非自回归翻译 [96]。不过，非自回归模型的翻译品质与自回归模型还有一些距离，特别是在保证翻译速度的情况下，非自回归翻译会带来一定的性能损失。如何融合自回归和非自回归翻译各自的优点也是值得探索的方向。

g) 其它方法

还有很多方法可以用于神经机器翻译的推断加速。比如：

- 更多的剪枝。可以使用更小的束宽度进行搜索，或者直接使用贪婪的方法进行搜索，即搜索的每个步骤中只保留最好的一个结果。这种方法基本上不用修改代码，比较适合对系统的快速验证和调试。

- 新的搜索终止条件。影响推断时间的一个因素是搜索的终止条件。为了保证搜索到高质量的译文，推断系统往往会多“看”一部分样本，比如，会在一个目标语长度范围内生成译文，但是这个范围都是凭经验设置，甚至无法保证最佳译文的长度会落在这个范围内。理想的情况下，当最佳结果出现的时候就可以停止搜索。因此，可以考虑设计更加合理的搜索终止条件减少不必要的计算[118]。比如，可以考虑当前最优翻译假设的得分和其它翻译假设得分的差距，如果差距过大就可以提前终止。

实际上，推断速度是机器翻译能否应用的重要因素之一。虽然，在一些研究性课题中，人们可能并不太在乎系统翻译有多“快”。但是，很多场景中速度是必须要考虑的问题，例如：

- 需要大量自生成数据的情况。比如，需要利用机器翻译生成大量的伪数据的情况。在无指导神经机器翻译训练中，数据的生成也非常频繁。
- 交互式翻译。机器翻译的一个应用场景就是交互式机器翻译 [59, 210, 229]，即机器翻译会根据用户的行为实时进行调整，这时机器翻译的延时会影响用户体验。
- 互联网机器翻译服务和产品。在大并发时如何保证翻译的低延时也是开发这类应用的过程中必须要考虑的。
- 机器同声传译。同声传译是机器翻译的一个重要的应用场景。由于同传对实时性的要求，机器翻译的速度是影响整个系统的关键要素。此外，为了保证更好的用户体验，往往需要在讲话者说完前就开始翻译，也就是根据一句话的前缀进行翻译，当得到后面的内容后再对翻译进行调整。这些都对机器翻译提出了新的要求 [190]。
- 小设备上的机器翻译。在手机或者专用翻译设备上的机器翻译对速度也有很高的要求，同时需要考虑设备存储的限制。尤其，CPU 上的推断加速是这类场景中需要关注的。

7.4.2 译文长度控制

机器翻译的一个特点是译文长度需要额外的机制进行控制。这是因为机器翻译在建模时仅考虑了将训练样本（即标准答案）上的损失最小化，但是推断的时候会看到从未见过的现象，而且这些未见样本占据了样本空间的绝大多数。这时，模型会产生 **偏置**（Bias），也就是模型仅仅能够对见过的样本进行准确建模，而对于未见样本的建模并不准确。所导致的一个现象是：直接使用训练好的模型会翻译出长度短的离谱的译文。这是一个典型的 **退化**（Degenerate）问题，即：在推断时，模型倾向于使用更少的步骤生成结果。神经机器翻译模型使用单词概率的乘积表示整个句子的翻译概率，它天然就倾向生成短译文，因为短译文会使用更少的概率因式相乘。

在使用极大似然估计进行模型训练时，这个问题会更加严重，因为模型只关心每个目标语位置的正确预测，对于译文长度没有考虑。此外，在机器翻译中译文长度如此重要的另一个原因在于：以 BLEU 为代表的评价指标对译文长度是非常敏感的，如果译文长度偏离参考答案过多，BLEU 的结果会非常低。

译文长度不合理的问题也出现在统计机器翻译模型中，当时的策略是在推断过程中引入译文长度控制机制。神经机器翻译也借用了类似的思想，有几种简单的方法。

长度惩罚因子

最常用的方法是直接对翻译概率进行正规化，也就是用译文长度来归一化翻译概率。第六章已经对长度归一化方法进行过介绍。为了保证内容的连贯性，这里再简单回顾一下相关内容。令源语言句子为 $\mathbf{x} = \{x_1, \dots, x_m\}$ ，译文为 $\mathbf{y} = \{y_1, \dots, y_n\}$ ，于是翻译模型得分 $\text{score}(\mathbf{x}, \mathbf{y})$ 可以被定义为：

$$\text{score}(\mathbf{x}, \mathbf{y}) = \log(P(\mathbf{y}|\mathbf{x})) \tag{7.8}$$

因为 $P(\mathbf{y}|\mathbf{x}) = \prod_{j=1}^n P(y_j|y_{<j}, \mathbf{x})$ ， $\text{score}(\mathbf{x}, \mathbf{y})$ 的值会随着译文 \mathbf{y} 的变长而减小。于是可以直接对 $\text{score}(\mathbf{x}, \mathbf{y})$ 按 \mathbf{y} 的长度进行归一化。令 $\text{lp}(\mathbf{y})$ 表示长度惩罚因子，归一化后的模型得分可以被定义为：

$$\text{score}(\mathbf{x}, \mathbf{y}) = \frac{\log(P(\mathbf{y}|\mathbf{x}))}{\text{lp}(\mathbf{y})} \tag{7.9}$$

$\text{lp}(\mathbf{y})$ 的物理意义是要给短译文一些惩罚，反映到公式7.9上就是对 $\log(P(\mathbf{y}|\mathbf{x}))$ 进行正规化。 $\text{lp}(\mathbf{y})$ 的定义方式很多，比如表7.5就列出了一些常用的形式。

表 7.5: 长度惩罚因子 $\text{lp}(\mathbf{y})$ 的定义（ $|\mathbf{y}|$ 表示译文长度）

名称	$\text{lp}(\mathbf{y})$
句子长度	$\text{lp}(\mathbf{y}) = \mathbf{y} ^\alpha$
GNMT 惩罚因子	$\text{lp}(\mathbf{y}) = \frac{(5+ \mathbf{y})^\alpha}{(5+1)^\alpha}$
指数化长度惩罚因子	$\text{lp}(\mathbf{y}) = \alpha \cdot \log(\mathbf{y})$

在推断时，直接采用公式7.9定义的模型进行搜索即可。 α 是一个超参数，需要经验性设置。有时候为了调节译文的长度，可以手工调整 α ，比如，增大 α 可以使译文更长一些。在有些场景下，调整 α 也是一种快速响应问题的手段。

译文长度范围约束

神经机器翻译系统进行推断时，可以依靠生成结束符 $\langle \text{eos} \rangle$ 来判断一个句子是否翻译完毕。不过，经常会出现生成了很长的译文仍然没有出现 $\langle \text{eos} \rangle$ 的问题。另

一种极端的情况是，刚刚生成了几个词，`<eos>` 就出现了。这两种情况对应了译文过长或者过短的情况。为了让译文的长度落在合理的范围，神经机器翻译的推断也会有一个译文长度约束。令 $[a, b]$ 表示一个长度范围，可以定义：

$$a = \omega_{\text{low}} \cdot |\mathbf{x}| \quad (7.10)$$

$$b = \omega_{\text{high}} \cdot |\mathbf{x}| \quad (7.11)$$

其中， ω_{low} 和 ω_{high} 是两个参数，用来表示译文长度的下限和上限。比如，很多系统中有 $\omega_{\text{low}} = \frac{1}{2}$ ， $\omega_{\text{high}} = 2$ ，表示译文至少有源语言句子一半长，最多有源语言句子两倍长。

值得注意的是， ω_{low} 和 ω_{high} 的设置对推断效率影响很大。 ω_{high} 可以被看作是一个推断的终止条件，最理想的情况是 $\omega_{\text{high}} \cdot |\mathbf{x}|$ 恰巧就等于最佳译文的长度，这时没有任何计算的浪费。反过来的一种情况， $\omega_{\text{high}} \cdot |\mathbf{x}|$ 远大于最佳译文的长度，这时很多计算都是无用的。实际上，译文长度预测是机器翻译的核心问题，而且十分困难。包括非自回归翻译在内的很多方法都依赖对目标语译文长度的建模（见7.4.1节）。为了找到长度预测的准确率和召回率之间的平衡，一般需要大量的实验最终确定 ω_{low} 和 ω_{high} 。当然，利用统计模型预测 ω_{low} 和 ω_{high} 也是非常值得探索的方向，比如基于产出率的模型。

覆盖度模型

上面提到的译文长度过长或过短的问题，本质上对应着**过翻译**（Over Translation）和**欠翻译**（Under Translation）的问题。具体来说，过翻译的一种表现是重复翻译，比如，神经机器翻译系统的输出中有时会莫名其妙的将同一个片段重复生成多次；欠翻译的主要表现就是删词，也就是源语言句子中的一些内容根本就没在译文中得到体现。这两个问题都会带来用户体验的下降。过翻译和欠翻译问题出现的原因在于：

- 机器翻译自动评价指标对过翻译和欠翻译并不敏感。众所周知，在机器翻译系统开发和调试过程中，使用较多的是 BLEU 等自动评价指标。但是，过翻译和欠翻译在 BLEU 这样的指标中是没有明确体现的¹³。一个典型的例子是实词漏翻。在人翻译一个句子的时候，如果漏掉一个实词会带来很差甚至不正确的翻译结果，但是在 BLEU 等指标中可能只是影响几个 n -gram 的匹配。特别是，极大似然训练会使模型对过翻译和欠翻译“更加”不敏感，因为目标函数对这两个问题没有显性的考虑。
- 神经机器翻译没有对过翻译和欠翻译建模。在统计机器翻译中，由于覆盖度模型会保证所有单词都可以被翻译，且只被翻译一次，因此过翻译和欠翻译等问题很少出现。这也对应了翻译**充分性**（Adequacy）的问题，也就是机器翻译结

¹³BLEU 中也有准确率和长度惩罚因子，但是并没有考虑源语言句子和译文之间的对应关系，因此无法捕捉过翻译和欠翻译。

果能否准确、完整地表达源文的意思。而在神经机器翻译中这个问题没有明确的模型与之对应。

对于第一个问题，可以通过引入人工评价和基于代价的评价方式进行缓解。机器翻译研究者更多的是关注第二个问题，即机器翻译覆盖度问题 [286]。其核心思想是对每个源语言单词被翻译的“程度”进行建模，一般来说希望每个单词被翻译的“程度”接近 1。这个模型可以从训练数据中自动学习，不过这样会增加系统的复杂性。一种更加常用的方法是在推断的过程中引入一个度量覆盖度的模型。比如，使用 GNMT 覆盖度模型 [312]，其中翻译模型得分被定义为：

$$\text{score}(\mathbf{x}, \mathbf{y}) = \frac{\log P(\mathbf{y}|\mathbf{x})}{\text{lp}(\mathbf{y})} + \text{cp}(\mathbf{x}, \mathbf{y}) \quad (7.12)$$

$$\text{cp}(\mathbf{x}, \mathbf{y}) = \beta \cdot \sum_{i=1}^{|\mathbf{x}|} \log(\min(\sum_j^{|\mathbf{y}|} a_{ij}, 1)) \quad (7.13)$$

$\text{cp}(\mathbf{x}, \mathbf{y})$ 表示覆盖度模型，它度量了译文对源语言每个单词覆盖的好坏。 $\text{cp}(\mathbf{x}, \mathbf{y})$ 的定义中， a_{ij} 表示源语言第 i 个位置与目标语第 j 个位置的注意力权重，这样 $\sum_j^{|\mathbf{y}|} a_{ij}$ 就可以被当作是源语言第 i 个单词被翻译了“多少”，如果它大于 1，表明翻译多了；如果小于 1，表明翻译少了。公式 7.13 会惩罚那些欠翻译的翻译假设。

覆盖度模型的一种改进形式是 [172]：

$$\text{cp}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{|\mathbf{x}|} \log(\max(\sum_j^{|\mathbf{y}|} a_{ij}, \beta)) \quad (7.14)$$

公式 7.14 将公式 7.13 中的向下截断方式换为了向上截断。这样，模型可以对过翻译（或重复翻译）有更好的建模能力。不过，这个模型需要在开发集上细致地调整 β ，也带来了一定的额外工作量。

7.4.3 多模型集成

在机器学习领域，把多个模型融合成一个模型是提升系统性能的一种有效的方法。比如，在经典的 AdaBoost 方法中 [78]，用多个“弱”分类器构建的“强”分类器可以使模型在训练集上的分类错误率无限接近 0。类似的思想也被应用到机器翻译 [241, 242, 259, 325]，被称为**系统融合**（System Combination）。在各种机器翻译比赛中，系统融合已经成为经常使用的技术之一。

广义上来讲，使用多个特征组合的方式都可以被看作是一种模型的融合。融合多个神经机器翻译系统的方法有很多，可以分为如下几类。

假设选择

假设选择（Hypothesis Selection）是最简单的系统融合方法 [63]。其思想是：给定一个翻译假设集合，综合多个模型对每一个翻译假设进行打分，之后选择得分最高的假设作为结果输出。其中包含两方面问题：

- 假设生成。构建翻译假设集合是假设选择的第一步，也是最重要的一步。理想的情况下，我们希望这个集合尽可能包含更多高质量的翻译假设，这样后面有更大的几率选出更好的结果。不过，由于单个模型的性能是有上限的，因此无法期望这些翻译假设的品质超越单个模型的上限。研究人员更加关心的是翻译假设的**多样性**（Diversity），因为已经证明多样的翻译假设非常有助于提升系统融合的性能 [167, 319]。为了生成多样的翻译假设，通常有两种思路：1）使用不同的模型生成翻译假设；2）使用同一个模型的不同参数和设置生成翻译假设。图7.24展示了二者的区别。

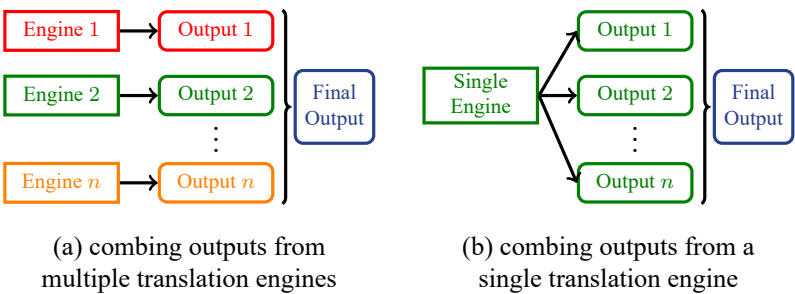


图 7.24: 多模型翻译假设生成 vs 单模型翻译假设生成

比如，可以使用基于 RNN 的模型和 Transformer 模型生成不同的翻译假设，之后都放入集合中；也可以只用 Transformer 模型，但是用不同的模型参数构建多个系统，之后分别生成翻译假设。在神经机器翻译中，经常采用的是第二种方式，因为系统开发的成本更低。比如，很多研究工作都是基于一个基础模型，用不同的初始参数、不同层数、不同解码方式生成多个模型进行翻译假设生成。

- 选择模型。所谓假设选择实际上就是要用一个更强的模型在候选中进行选择。这个“强”模型一般是由更多、更复杂的子模型组合而成。常用的方法是直接使用翻译假设生成时的模型构建“强”模型。比如，我们使用了两个模型生成了翻译假设集合，之后对所有翻译假设都分别用这两个模型进行打分。最后，综合两个模型的打分（如线性插值）得到翻译假设的最终得分，并进行选择。当然，也可以使用更强大的统计模型对多个子模型进行组合（如使用多层神经网络）。

假设选择也可以被看作是一种简单的投票模型。对所有的候选用多个模型投票，选出最好的结果输出。包括**重排序**（Re-ranking）在内的很多方法也是假设选择的一种特例。比如，在重排序中，可以把生成 n -best 列表的过程看作是翻译假设生成过

程，而重排序的过程可以被看作是融合多个子模型进行最终结果选择的过程。

局部预测融合

神经机器翻译模型对每个目标端位置 j 的单词分布进行预测，即对于目标语言词汇表中的每个单词 y_j ，需要计算 $P(y_j|\mathbf{y}_{<j}, \mathbf{x})$ 。假设有 K 个神经机器翻译系统，那么每个系统 k 都可以独立的计算这个概率，记为 $P_k(y_j|\mathbf{y}_{<j}, \mathbf{x})$ 。于是，可以用如下方式融合这 K 个系统的预测：

$$P(y_j|\mathbf{y}_{<j}, \mathbf{x}) = \sum_{k=1}^K \gamma_k \cdot P_k(y_j|\mathbf{y}_{<j}, \mathbf{x}) \tag{7.15}$$

其中 γ_k 表示第 k 个系统的权重，且满足 $\sum_{k=1}^K \gamma_k = 1$ 。公式7.15是一种线性模型。权重 $\{\gamma_k\}$ 可以在开发集上自动调整，比如，使用最小错误率训练得到最优的权重（见第四章）。不过在实践中发现，如果这 K 个模型都是由一个基础模型衍生出来的，权重 $\{\gamma_k\}$ 对最终结果的影响并不大。因此，有时候也简单的将权重设置为 $\gamma_k = \frac{1}{K}$ 。图7.25展示了对三个模型预测结果的集成。

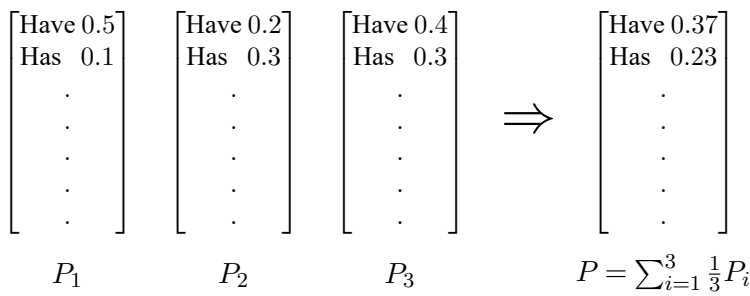


图 7.25: 基于三个模型预测结果的集成

公式7.15是一种典型的线性插值模型，这类模型在语言建模等任务中已经得到成功应用。从统计学习的角度，多个模型的插值可以有效的降低经验错误率。不过，多模型集成依赖一个假设：这些模型之间需要有一定的互补性。这种互补性有时也体现在多个模型预测的上限上，称为 Oracle。比如，可以把这 K 个模型输出中 BLEU 最高的结果作为 Oracle，也可以选择每个预测结果中使 BLEU 达到最高的译文单词，这样构成的句子作为 Oracle。当然，并不是说 Oracle 提高，模型集成的结果一定会变好。因为 Oracle 是最理想情况下的结果，而实际预测的结果与 Oracle 往往有很大差异。如何使用 Oracle 进行模型优化也是很多研究者在探索的问题。

此外，如何构建集成用的模型也是非常重要的，甚至说这部分工作会成为模型集成方法中最困难的部分。绝大多数时候，模型生成并没有固定的方法。系统研发者大多也是“八仙过海、各显神通”。一些常用的方法有：

- 改变模型宽度和深度，即用不同层数或者不同隐藏层大小得到多个模型；

- 不同的参数初始化，即用不同的参数初始化随机种子训练多个模型；
- 不同模型 (局部) 架构的调整，比如，使用不同的位置编码模型 [255]、多层融合模型 [299] 等。

译文重组

假设直接从已经生成的译文中进行选择，因此无法产生“新”的译文，也就是它的输出只能是某个单模型的输出。另一方面，预测融合需要同时使用多个模型进行推断，对计算和内存消耗较大。而且这两种方法有一个共性问题：搜索都是基于一个个字符串，相比指数级的译文空间，所看到的结果还是非常小的一部分。对于这个问题，一种方法是利用更加紧凑的数据结构对指数级的译文串进行表示。比如，可以使用格 (lattice) 对多个译文串进行表示 [284]。图7.26展示了基于 *n*-best 词串和基于 lattice 的表示方法的区别。可以看到，lattice 中从起始状态到结束状态的每一条路径都表示一个译文，不同译文的不同部分可以通过 lattice 中的节点得到共享¹⁴。理论上，lattice 可以把指数级数量的词串用线性复杂度的结构表示出来。

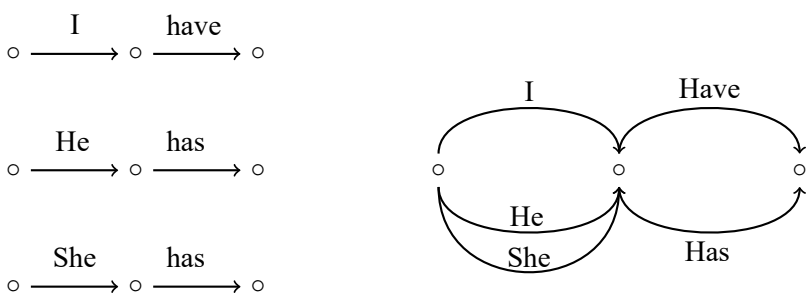


图 7.26: *n*-best 词串表示 vs lattice 词串表示

有了 lattice 这样的结构，多模型融合又有了新的思路。首先，可以将多个模型的译文融合为 lattice。注意，这个 lattice 会包含这些模型无法生成的完整译文句子。之后，用一个更强的模型在 lattice 上搜索最优的结果。这个过程有可能找到一些“新”的译文，即结果可能是从多个模型的结果中重组而来的。lattice 上的搜索模型可以基于多模型的融合，也可以使用一个简单的模型，这里需要考虑的是将神经机器翻译模型适应到 lattice 上进行推断 [271]。其过程基本与原始的模型推断没有区别，只是需要把模型预测的结果附着到 lattice 中的每条边上，再进行推断。

图7.27对比了不同模型集成方法的区别。从系统开发的角度看，假设选择和模型预测融合的复杂度较低，适合快速原型，而且性能稳定。译文重组需要更多的模块，系统调试的复杂度较高，但是由于看到了更大的搜索空间，因此系统性能提升的潜力较大¹⁵。

¹⁴本例中的 lattice 也是一个混淆网络 (Confusion Network)
¹⁵一般来说 lattice 上的 Oracle 要比 *n*-best 译文上的 Oracle 的质量高。

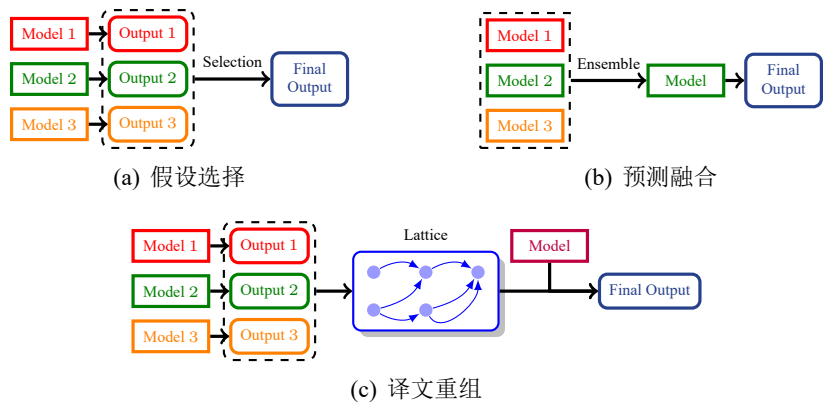


图 7.27: 不同的模型集成方法对比

7.5 进阶技术

这里继续对神经机器翻译的进阶技术进行介绍和讨论，内容涉及：深层模型、单语数据使用、知识精炼、双向训练等。

7.5.1 深层模型

7.3.2节已经指出：增加神经网络的深度有助于对句子进行更充分的表示、同时增加模型的容量。但是，简单地堆叠很多层 Transformer 网络并不能带来性能上的提升，反而会面临更加严重的梯度消失/梯度爆炸的问题。这是由于伴随神经网络变深，梯度无法有效地从输出层回传到底层网络，造成网络浅层部分的参数无法得到充分训练 [299, 334]。针对这些问题，已经有研究者开始尝试进行求解，并取得了很好的效果。比如，设计更有利于深层信息传递的网络连接和恰当的参数初始化方法等 [10, 299, 339]。

但是，如何设计一个足够“深”的机器翻译模型仍然是业界关注的热点问题之一。此外，伴随着网络的继续变深，将会面临一些新的问题，例如，如何加速深层网络的训练，如何解决深层网络的过拟合问题等。下面将会对以上问题展开讨论。

Post-Norm vs Pre-Norm

为了探究为何深层的 Transformer 模型很难直接训练，首先对 Transformer 的模型结构进行简单的回顾。以 Transformer 的编码端为例，在多头自注意力网络和前馈神经网络中间，Transformer 模型利用残差连接和层正则化操作来提高信息的传递效率。Transformer 模型大致分为图7.28中两种结构——后作方式的残差单元（Post-Norm）和前作方式的残差单元（Pre-Norm）。

令 x_l 和 x_{l+1} 表示第 l 子层的输入和输出¹³， y_l 表示中间的临时输出； $\text{LN}(\cdot)$ 表示

¹³这里沿用 Transformer 中的定义，每一层 (Layer) 包含多个子层 (Sub-layer)。比如，对于 Transformer 编码器，每一层包含一个自注意力子层和一个前馈神经网络子层。所有子层都需要进行层归一化和残

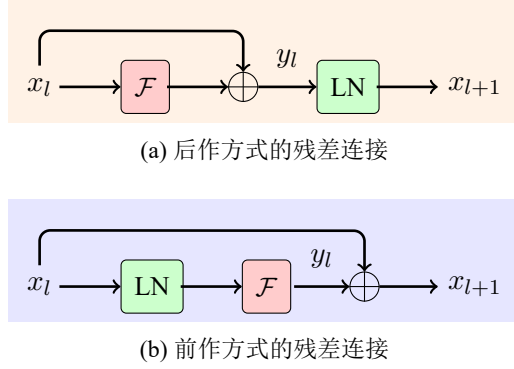


图 7.28: Post-Norm Transformer vs Pre-Norm Transformer

层归一化操作 [5], 帮助减少子层输出分布的方差。从而让训练变得更稳定; $\mathcal{F}(\cdot)$ 表示子层所对应的函数, 比如前馈神经网络、自注意力网络等。下面分别对 Post-Norm 和 Pre-Norm 进行简单的描述。

- **Post-Norm:** 早期的 Transformer 遵循的是 Post-Norm 结构 [288]。也就是, 层正则化作用于每一子层的输入和输出的残差结果上, 如图7.28(a) 所示。可以表示如下:

$$x_{l+1} = \text{LN}(x_l + \mathcal{F}(x_l; \theta_l)) \quad (7.16)$$

其中, θ_l 是子层 l 的参数。

- **Pre-Norm:** 通过调整层正则化的位置, 将其放置于每一子层的输入之前, 得到了 Pre-Norm 结构, 如图7.28(b) 所示。其思想与 He 等人的思想一致 [102], 也被广泛应用于最新的 Transformer 开源系统中 [143, 224, 287], 公式如下:

$$x_{l+1} = x_l + \mathcal{F}(\text{LN}(x_l); \theta_l) \quad (7.17)$$

从上述公式可以看到 Pre-Norm 结构可以通过残差路径将底层网络的输出直接暴露给上层网络; 另一方面从反向传播的角度看, 使用 Pre-Norm 结构, 顶层的梯度可以更容易地反馈到底层网络。这里以一个含有 L 个子层的结构为例。令 $Loss$ 表示整个神经网络输出上的损失, x_L 为顶层的输出。对于 Post-Norm 结构, 根据链式法则, 损失 $Loss$ 相对于 x_l 的梯度可以表示为:

$$\frac{\partial Loss}{\partial x_l} = \frac{\partial Loss}{\partial x_L} \times \prod_{k=l}^{L-1} \frac{\partial \text{LN}(y_k)}{\partial y_k} \times \prod_{k=l}^{L-1} \left(1 + \frac{\partial \mathcal{F}(x_k; \theta_k)}{\partial x_k}\right) \quad (7.18)$$

其中 $\prod_{k=l}^{L-1} \frac{\partial \text{LN}(y_k)}{\partial y_k}$ 表示在反向传播过程中经过层正则化得到的复合函数导数, $\prod_{k=l}^{L-1} (1 + \frac{\partial \mathcal{F}(x_k; \theta_k)}{\partial x_k})$ 代表每个子层间残差连接的导数。

类似的, 也能得到 Pre-Norm 结构的梯度计算结果, 如下式所示:

$$\frac{\partial \text{Loss}}{\partial x_l} = \frac{\partial \text{Loss}}{\partial x_L} \times (1 + \sum_{k=l}^{L-1} \frac{\partial \mathcal{F}(\text{LN}(x_k); \theta_k)}{\partial x_l}) \tag{7.19}$$

对比公式7.18和公式7.19可以明显发现 Pre-Norm 结构直接把顶层的梯度 $\frac{\partial \text{Loss}}{\partial x_L}$ 传递给下层, 也就是 $\frac{\partial \text{Loss}}{\partial x_l}$ 中直接含有 $\frac{\partial \text{Loss}}{\partial x_L}$ 的部分。这个性质弱化了梯度计算对模型深度 L 的依赖; 而 Post-Norm 结构会导致一个与 L 相关的多项导数的积 (见公式7.18右侧), 伴随着 L 的增大更容易发生梯度消失和梯度爆炸问题。因此, Pre-Norm 结构更适于堆叠多层神经网络的情况。比如, 使用 Pre-Norm 结构可以很轻松的训练一个 30 层 (60 个子层) 的 Transformer 编码器网络, 并带来可观的 BLEU 提升。这个结果相当于标准 Transformer 编码器深度的 6 倍 [299]。相对的, 用 Pre-Norm 结构训练深网络的时候, 训练结果很不稳定, 甚至有时候无法完成有效训练。这里把使用 Pre-Norm 的深层 Transformer 称为 Transformer-Deep。

另一个有趣的发现是, 使用深层网络后, 训练模型收敛的时间大大缩短。相比于 Transformer-Big 等宽网络, Transformer-Deep 并不需要太大的隐藏层大小就可以取得相当甚至更优的翻译品质。也就是说, Transformer-Deep 是一个更“窄”更“深”的网络。这种结构的参数量比 Transformer-Big 少, 系统运行效率更高。表7.6对比了不同模型的参数量和训练/推断时间。

表 7.6: 不同 Transformer 结构的训练/推断时间对比 (WMT14 英德任务)

系统	参数量	训练时间	推断时间
Base	63M	4.6h	19.4s
Big	210M	36.1h	29.3s
DLCL-30	137M	9.8h	24.6s

还有一个有趣的发现是, 当编码器端使用深层网络之后, 解码器端使用更浅的网络依然能够维持相当的翻译品质。这是由于解码器端的计算仍然会有对源语言端信息的加工和抽象, 当编码器变深之后, 解码器对源语言端的加工不那么重要了, 因此可以减少解码网络的深度。这样做的一个直接好处是: 可以通过减少解码器的深度加速翻译系统。对于一些延时敏感的场景, 这种架构是极具潜力的。

层聚合

尽管使用 Pre-Norm 结构可以很容易地训练深层 Transformer 模型, 但从信息传递的角度看, Transformer 模型中第 n 层的输入仅仅依赖于前一层的输出。虽然残差

连接可以将信息跨层传递，但是对于很深的网络，整个模型的输入和输出之间仍需要很多次残差连接才能进行有效的传递。为了使上层的网络可以更加方便地访问下层网络的信息，一种方法是直接引入更多跨层的连接。最简单的一种方法是直接将所有层的输出都连接到最上层，达到聚合多层信息的目的 [10, 300]。另一种更加有效的方式是使用**动态线性层聚合方法**（Dynamic Linear Combination of Layers, DLCL）。在每一层的输入中不仅考虑前一层的输出，而是将前面所有层的中间结果（包括词嵌入）进行线性聚合，理论上等价于常微分方程中的高阶求解方法 [299]。以 Pre-Norm 结构为例，具体做法如下：

- 对于每一层的输出 x_{l+1} ，对其进行层正则化，得到每一层的信息的表示

$$z_l = \text{LN}(x_{l+1}) \tag{7.20}$$

注意， z_0 表示词嵌入层的输出， $z_l(l > 0)$ 表示 Transformer 网络中最终的各层输出。

- 定义一个维度为 $(L + 1) \times (L + 1)$ 的权值矩阵 \mathbf{W} ，矩阵中每一行表示之前各层对当前层计算的贡献度，其中 L 是编码端（或解码端）的层数。令 $\mathbf{W}_{l,i}$ 代表权值矩阵 \mathbf{W} 第 l 行第 i 列的权重，则层聚合的输出为 z_i 的线性加权和：

$$g_l = \sum_{i=0}^l z_i \times \mathbf{W}_{l,i} \tag{7.21}$$

g_l 会作为输入的一部分送入第 $l + 1$ 层。其网络的结构如图7.29所示

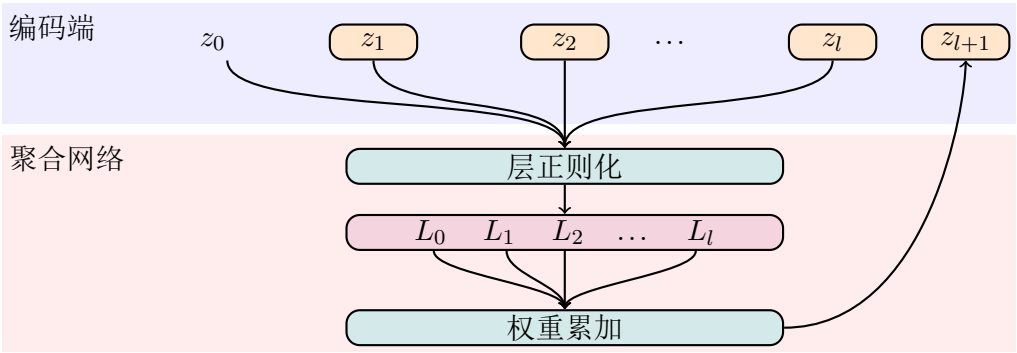


图 7.29: 动态线性层聚合网络结构图

可以看到，权值矩阵 \mathbf{W} 是一个下三角矩阵。开始时，对矩阵参数的每行进行平均初始化，即初始化矩阵 \mathbf{W}_0 的每一行各个位置的值为 $1/M, M \in (1, 2, 3 \cdots L + 1)$ 。伴随着神经网络的训练，网络通过反向传播算法来不断更新 \mathbf{W} 中每一行不同位置权重的大小。

动态线性层聚合的一个好处是，系统可以自动学习不同层对当前层的贡献度。在

实验中也发现，离当前层更近的部分贡献度（权重）会更大，这也是符合直觉的。

深层模型的训练加速

尽管训练这种窄而深的神经网络对比宽网络有更快的收敛速度，但伴随着训练数据的增加，以及模型进一步的加深，神经网络的训练代价成为不可忽视的问题。例如，在几千万甚至上亿的双语平行语料上训练一个 48 层的 Transformer 模型需要将近几周的时间能达到收敛¹⁴。因此，在保证模型精度不变的前提下如何高效地完成深层网络的训练也是至关重要的。在实践中能够发现，深层网络中相邻层之间具有一定的相似性。因此，一个想法是：能否通过不断复用浅层网络的参数来初始化更深层的网络，渐进式的训练深层网络，避免从头训练整个网络，进而达到加速深层网络训练的目的。

渐进式训练

所谓渐进式训练是指从浅层网络开始，在训练过程中逐渐增加训练的深度。一种比较简单的方法是将网络分为浅层部分和深层部分，之后分别进行训练，最终达到提高模型翻译性能的目的 [311]。

另一种方式是动态构建深层网络，并尽可能复用浅层网络的训练结果。假设开始的时候模型包含 h 层网络，然后训练这个模型至收敛。之后，直接拷贝这 h 层网络（包括参数），并堆叠出一个 $2h$ 层的模型。之后继续训练，重复这个过程。进行 n 次之后就得到了 $n \times h$ 层的模型。图7.30给出了在编码端使用渐进式训练的示意图。

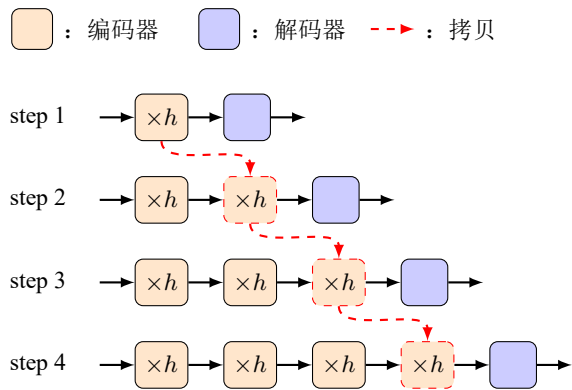


图 7.30: 渐进式深层网络训练过程

渐进式训练的好处在于深层模型并不是从头开始训练。每一次堆叠，都相当于利用“浅”模型给“深”模型提供了一个很好的初始点，这样深层模型的训练会更加容易。

¹⁴训练时间的估算是在单台 8 卡 Titan V GPU 服务器上得到的

分组稠密连接

很多研究者已经发现深层网络不同层之间的稠密连接能够很明显地提高信息传递的效率 [61, 115, 299, 311]。与此同时，对之前层信息的不断复用有助于得到更好的表示，但随之而来的是网络计算代价过大的问题。由于动态线性层聚合方法（DLCL）在每一次聚合时都需要重新计算之前每一层表示对当前层网络输入的贡献度，因此伴随着编码端整体深度的不断增加，这部分的计算代价变得不可忽略。例如，一个基于动态层聚合的 48 层 Transformer 模型的训练时间比不使用动态层聚合慢近 1.9 倍。同时，缓存中间结果也增加了显存的使用量，尽管使用了 FP16 计算，每张 12G 显存的 GPU 上计算的词也不能超过 2048 个，这导致训练开销急剧增大。

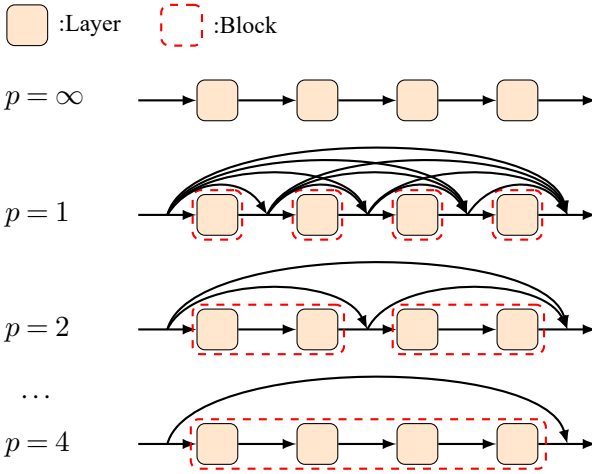


图 7.31: 不同组之间的稀疏连接

缓解这个问题的一种方法是使用更稀疏的层间连接方式。其核心思想与动态线性层聚合是类似的，不同点在于可以通过调整层之间连接的稠密程度来降低训练代价。比如，可以将每 p 层分为一组，之后动态线性层聚合只在不同组之间进行。这样，通过调节 p 值的大小可以控制网络中连接的稠密程度，作为一种训练代价与翻译性能之间的权衡。显然，标准的 Transformer 模型 [288] 和 DLCL 模型 [299] 都可以看作是该方法的一种特例。如图7.31所示：当 $p = 1$ 时，每一个单独的块被看作一个独立的组，这等价于基于动态层聚合的 DLCL 模型；当 $p = \infty$ 时，这等价于正常的 Transformer 模型。值得注意的是，如果配合渐进式训练。在分组稠密连接中可以设置 $p = h$ 。

学习率重置策略

尽管渐进式训练策略与分组稠密连接结构可以加速深层网络的训练，但使用传统的学习率衰减策略会导致堆叠深层模型时的学习率较小，因此模型无法快速地达到收敛状态，同时也影响最终的模型性能。

图7.32中的红色曲线描绘了标准的 Transformer 模型的学习率曲线（WMT 英德任务），可以看到当模型训练到 40k 步时，网络的学习率值对比峰值有明显的差距，而此时刚开始训练最终的深层模型，过小的学习率并不利于后期深层网络的充分训练。

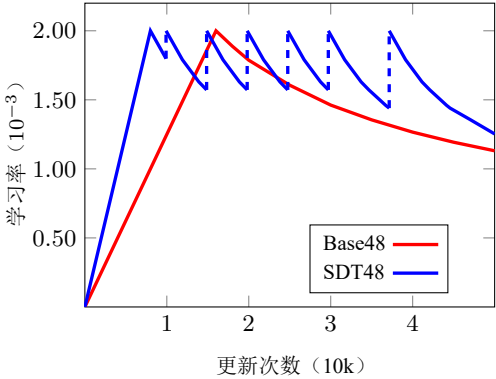


图 7.32: 学习率重置 vs 从头训练的学习率曲线

针对该问题的一个解决方案是修改学习率曲线的衰减策略。图中蓝色的曲线是修改后的学习率曲线。首先在训练的初期让网络快速的达到学习率的峰值（线性递增），之后的每一次从 p 层网络变为 $2p$ 层网络时，都会将当前的学习率值重置到峰值点。之后，根据训练的步数对其进行相应的衰减。具体的步骤如下：

- 在训练的初期，模型先经历一个学习率预热的过程：

$$lr = d_{model}^{-0.5} \cdot step_num \cdot warmup_steps^{-0.5} \tag{7.22}$$

这里， $step_num$ 表示参数更新的次数， $warmup_step$ 表示预热的更次次数， $d_{model}^{-0.5}$ 表示 Transformer 模型隐层大小， lr 是学习率。

- 在之后的迭代训练过程中，每当进行新的迭代，学习率都会重置到峰值，之后进行相应的衰减：

$$lr = d_{model}^{-0.5} \cdot step_num^{-0.5} \tag{7.23}$$

这里 $step_num$ 代表学习率重置后更新的步数。

综合使用渐进式训练、分组稠密连接、学习率重置策略可以在保证翻译品质不变的前提下，缩减近 40% 的训练时间（40 层编码器）。同时，加速比伴随着模型的加深与数据集的增大会进一步地扩大。

深层模型的鲁棒性训练

伴随着网络的加深，还会面临另外一个比较严峻的问题——过拟合。由于参数量的增大，深层网络的输入与输出分布之间的差异也会越来越大，然而不同子层之间的**相互适应**（Co-adaptation）也会更加的明显，导致任意子层网络对其他子层的依赖过大。这对于训练阶段是有帮助的，因为不同子层可以协同工作从而更好地拟合训练数据。然而这种方式也降低了模型的泛化能力，即深层网络更容易陷入过拟合问题。

通常，可以使用 Dropout 手段用来缓解过拟合问题（见7.3.1节）。不幸的是，尽管目前 Transformer 模型使用了多种 Dropout 手段（如 Residual Dropout、Attention Dropout、ReLU Dropout 等），过拟合问题在深层网络中仍然存在。从图7.33中可以看到，深层网络对比浅层网络在训练集和校验集的困惑度上都有显著的优势，然而网络在训练一段时间后出现校验集困惑度上涨的现象，说明模型已经过拟合于训练数据。

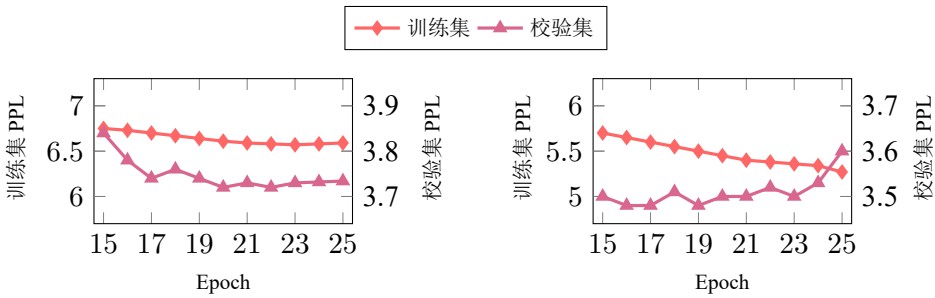


图 7.33: 浅层网络 (左) 与深层网络 (右) 在 WMT16 英德的校验集与训练集的困惑度

在7.3.1节提到的 Layer Dropout 方法可以有效地缓解这个问题。以编码端为例，Layer Dropout 的过程可以被描述为：在训练过程中，对自注意力子层或前馈神经网络子层进行随机丢弃，以减少不同子层之间的相互适应。这里选择 Pre-Norm 结构作为基础架构，它可以被描述为：

$$x_{l+1} = \mathcal{F}(\text{LN}(x_l)) + x_l \tag{7.24}$$

其中 $\text{LN}(\cdot)$ 表示层正则化函数， $\mathcal{F}(\cdot)$ 表示自注意力机制或者前馈神经网络， x_l 表示第 l 个子层的输出。之后，使用一个掩码 M （值为 0 或 1）来控制每一子层是正常计算还是丢弃。于是，该子层的计算公式可以被重写为：

$$x_{l+1} = M \cdot \mathcal{F}(\text{LN}(x_l)) + x_l \tag{7.25}$$

$M = 0$ 代表该子层被丢弃，而 $M = 1$ 代表正常进行当前子层的计算。图7.34展示了这个方法与标准 Pre-Norm 结构之间的区别。

除此之外，有研究者已经发现残差网络中底层的子网络通过对输入进行抽象得到的表示对最终的输出有很大的影响，上层网络通过对底层网络得到的表示不断修

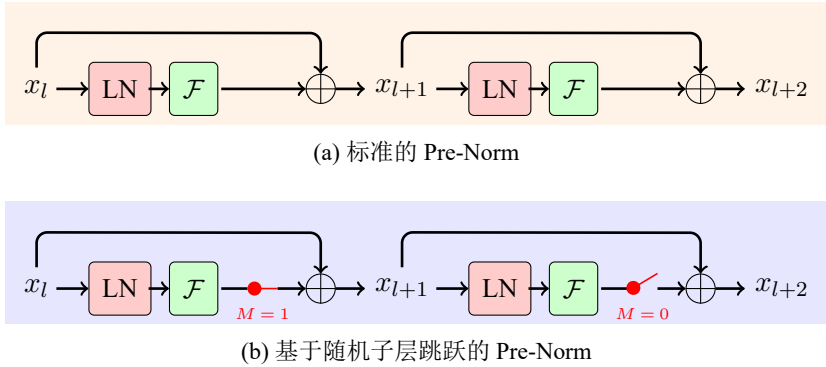


图 7.34: 标准的 Pre-Norm 结构与基于随机跳跃子层的 Pre-Norm 结构

正来拟合训练目标 [92]。该结论同样适用于 Transformer 模型，比如，在训练中，残差支路以及底层的梯度范数通常比较大，这也间接表明底层网络在整个优化的过程中需要更大的更新。考虑到这个因素，在设计每一个子层被丢弃的概率时可以采用自底向上线性增大的策略，保证底层的网络相比于顶层更容易保留下来。这里用 L 来代表编码端块的个数， l 代表当前的子层的编号，那么 M 可以通过以下方式得到：

$$M = \begin{cases} 0 & P \leq p_l \\ 1 & P > p_l \end{cases} \tag{7.26}$$

其中， P 是服从伯努利分布的随机变量， p_l 指每一个子层被丢弃的概率，具体计算方式如下：

$$p_l = \frac{l}{2L} \cdot \varphi \tag{7.27}$$

这里， $1 \leq l \leq 2L$ ，且 φ 是预先设定的超参数。

在 Layer Dropout 中，一个由 $2L$ 个子层构成的残差网络，其顶层的输出相当于 是 2^{2L} 个子网络的聚合结果。通过随机丢弃 n 个子层，则会屏蔽掉 2^n 个子网络的输出，将子网络的总体数量降低至 2^{2L-n} 。如图7.35所示的残差网络展开图，当有 3 个子层时，从输入到输出共存在 8 条路径，当删除子层 sublayer2 后，从输入到输出路径的路径则会减少到 4 条。

7.5.2 单语数据的使用

在统计机器翻译时代，使用单语数据训练语言模型就是构建机器翻译系统的关键步骤。好的语言模型往往会带来性能的增益。而这个现象在神经机器翻译中似乎并不明显，因为在大多数神经机器翻译的范式中，并不要求使用大规模单语数据来帮助机器翻译系统。甚至，连语言模型都不会作为一个独立的模块。这一方面是由

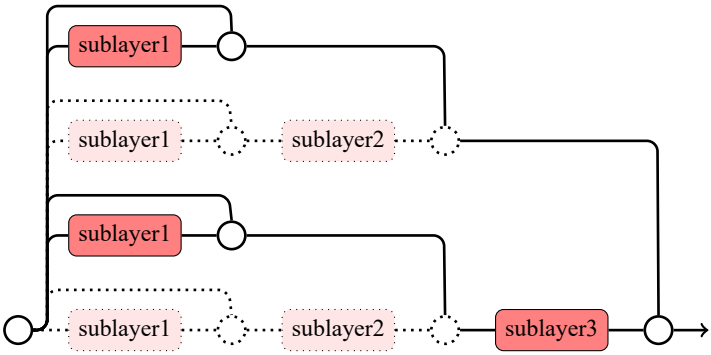


图 7.35: Layer Dropout 中残差网络的展开图

于神经机器翻译系统的解码端本身就起着语言模型的作用，另一方面是由于数据的增多使得翻译模型可以更好的捕捉目标语言的规律。

但是，双语数据总是有限的。很多场景下，单语数据的规模会远大于双语数据。比如，在专利翻译的很多细分领域中，双语数据的规模十分有限，但是有大量的和领域相关的单语数据。如果能够让这些单语数据发挥作用，显然是一种非常好的选择。

在神经机器翻译中使用单语数据面临这两方面问题：

- 从单语数据中学习什么样的知识？
- 如何在神经机器翻译中集成单语数据的知识？

下面将从数据增强、预训练、联合训练等方面对这两个问题展开讨论。

伪数据

通常，有两种思路使用单语数据：

- 将单语数据的建模作为神经机器翻译模型的一部分，比如，语言模型；
- 不改变神经机器翻译系统，让单语数据适配到翻译模型的学习过程中。

使用单语数据构建（双语）伪数据属于后者，它也是一种典型的**数据增强**（Data Augmentation）方法。一种常用做法是**回译**（Back Translation）[68, 251]：训练一个从目标语翻译到源语的系统，也就是一个反向翻译系统；之后，用这个系统翻译目标语言单语数据；最后将单语数据（目标语言）和翻译的结果（源语言）作为训练数据，送入源语言到目标语言的翻译系统。这种做法不需要更改任何模型结构，就能很好的利用单语数据，因此也被广泛采用。图7.36给出了回译方法的一个简要流程。

在理想情况下，生成的伪数据和真实数据分布越接近越好。不过，在实践中发现，即使一些简单的策略也能带来性能的增长。比如，在一些低资源的语种，仅仅通过将目标语句子复制到源语端构造的伪数据都能为模型带来增益 [51]。相比这些简单的构造策略，利用目标语言单语数据进行回译可以获得更高质量的伪数据。因为

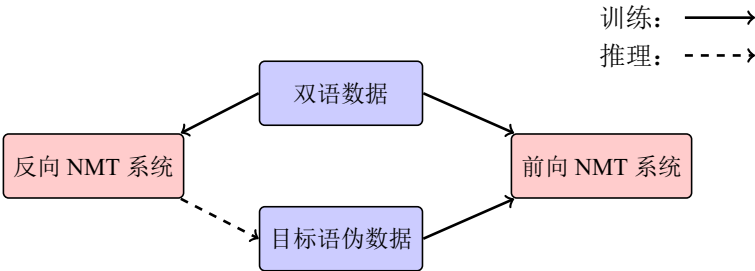


图 7.36: 回译方法的流程

目标语是正确的句子，这种方法可以保证译文的流畅度。这也间接达到了对目标语言进行语言建模的目的。在富资源的语种中，通常对回译产生的源语句子添加一些噪音，比如随机删除、替换一些词，或者交换两个词的位置，这样可以为模型提供一些训练噪声。而在低资源的语种上，由于双语数据稀缺，模型需要更多的高质量双语数据，不加噪音反而具有更好的效果。

回译方法的一个问题是：反向翻译模型的训练只依赖于有限的双语数据，生成的源语言端伪数据的质量难以保证。为此，可以采用**迭代式回译**（Iterative Back Translation）的方法，同时利用源语端和目标语端的单语数据，不断通过回译的方式来提升前向和反向翻译模型的性能。图7.37展示了迭代式回译的框架。首先使用双语数据训练一个前向翻译模型，然后利用源语言单语数据通过回译的方式来提升反向翻译模型的性能，最后由反向翻译模型和目标端单语数据生成的伪数据来提升前向翻译模型的性能。可以看出，这个往复的过程是闭环的，因此可以一直进行下去，直到两个翻译模型的性能不再提升。

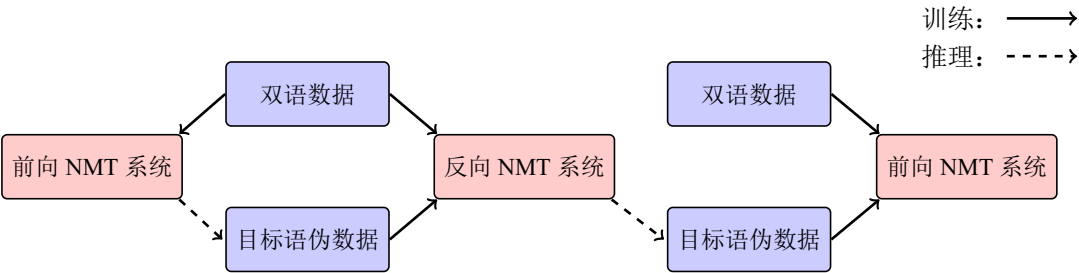


图 7.37: 迭代式回译方法的流程

与回译的方法类似，源语言的单语数据也可以通过一个双语数据训练的翻译模型获得对应的目标语，构造**前向翻译**（Forward Translation）的伪数据。与回译方法相反，前向翻译伪数据中源语端是真实的，而目标语端是生成的，构造的伪数据对译文的流畅性并没有太大帮助，其主要作用是丰富了训练数据中源语的表达，提升翻译模型中编码器的性能。大多数情况下，前向翻译方法带来的性能提升效果要弱于回译。

预训练

编码器-解码器框架天然就包含了对输入（源语言）和输出（目标语言）进行表示学习的过程。比如，在编码端需要学习一种分布式表示（Distributed Representation）来表示源语言句子的信息，这种分布式表示既包含单词的表示也包括整个序列的表示。因此，可以使用更大规模的源语言单语数据完成编码器的训练。

实现上述想法的一种手段是**预训练**（Pre-training）。常用的方法是将机器翻译模型中的一部分（比如，编码器）单独提抽取出来，之后用语言建模等方式在大规模单语数据上进行训练。得到优化后的参数后，将其重新放入神经机器翻译模型中，作为模型的初始值。最后，神经机器翻译模型在双语数据上进行**微调**（Fine-tuning），以得到最终的翻译模型。图7.38给出了机器翻译编码器预训练流程的示意图。

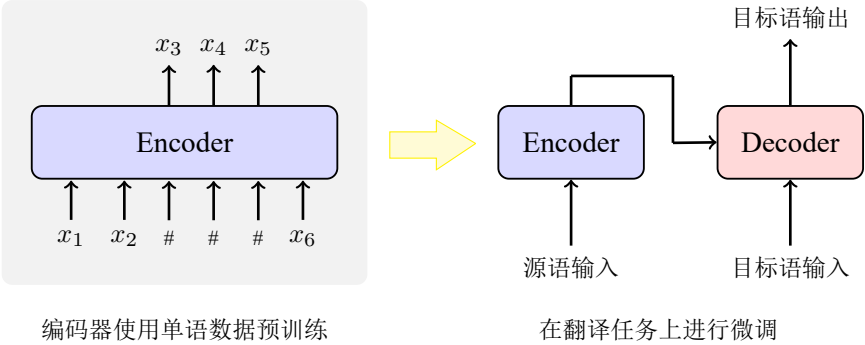


图 7.38: 机器翻译编码器预训练流程

预训练的做法相当于对目标任务进行了简化，将表示模型的学习任务从目标任务中分离出来了。这样，可以学习一种更加通用的模型，具有更好的泛化能力。此外，预训练的任务相比于机器翻译要简单许多，比如语言模型或者句子调序等。将预训练任务的结果作为机器翻译模型的初始值可以减轻目标任务上的学习负担。在第六章中已经介绍了几种基于预训练的模型，如 ELMO、GPT 和 BERT 等。这些模型的结构和神经机器翻译是兼容的，比如，BERT 使用的就是 Transformer 模型。因此可以直接使用这些模型进行面向机器翻译的预训练。

词嵌入预训练

词嵌入可以被看作是对每个独立单词进行的表示学习，在自然语言处理的众多任务中都扮演着重要角色 [2]。因此，可以使用第五章介绍的词嵌入方法，在外部单语数据上训练得到词嵌入，并把它作为神经机器翻译系统的词嵌入输入。

需要注意的是，在神经机器翻译中使用预训练的词嵌入有两种方法。一种方法是直接将词嵌入作为固定的输入，也就是在训练机器翻译模型的过程中，并不调整词嵌入的参数。这样做的目的是完全将词嵌入模块独立出来，机器翻译可以被看作是在固定的词嵌入输入上进行的建模。另一种方法是仍然遵循“预训练 + 微调”的策略，将词嵌入作为翻译模型的初始值。之后在机器翻译训练过程中，词嵌入模型结果会被进一步更新。近些年，在词嵌入预训练的基础上进行微调的方法受到研究者

越来越多的青睐。

编码器预训练

编码器在神经机器翻译中的作用是对源语句子中的信息进行抽象和提取，将离散的词序列编码成一组上下文相关的向量表示，本质上就是一个源语端的句子表示模型。因此，可以使用预训练好的句子级表示模型（比如，BERT 和 XLM 等），来初始化编码器参数。然而在实践中发现，这种参数初始化的方法在一些富资源语种上提升效果并不明显，甚至反而有些下降 [352]。原因可能在于预训练模型和编码器虽然都是对句子进行表示，但是由于目标任务不一致，二者的参数状态还是存在区别的。因此，也有一些做法将预训练模型和翻译模型在结构上进行融合，将预训练句子模型作为一个独立的模块来为编码器或者解码器提供句子级表示信息 [352]。

序列到序列预训练

传统的预训练模型都是针对自然语言理解任务设计的，比如情感分类和命名实体识别等任务。其目的是获得更好的句子表示结果，并用于下游任务。而在机器翻译和文本摘要等序列到序列的语言生成任务中，不只包含源语言表示学习的问题，还有序列到序列的映射，以及目标端序列生成的问题，这些知识是无法通过（源语言）单语数据学习到的。为了能够在序列到序列任务上更好地使用单语数据，可以同时使用编码器和解码器的结构完成对单语数据的预训练。

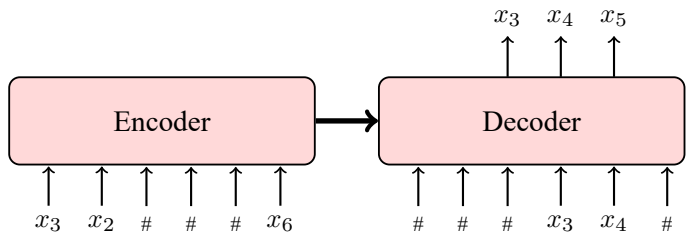


图 7.39: MASS 预训练方法

以 MASS 方法为例 [262]，可以直接对整个编码器-解码器的结构进行预训练。训练中采用掩码的方式，将源语词序列中的片段替换成特殊词 <mask>，然后在解码器端预测这个未知片段，如图7.39所示，# 号表示特殊词 <mask>。这种做法可以使得编码器端捕捉上下文信息，同时迫使解码器依赖于编码器，学习编码器和解码器之间的注意力进行预训练。而解码器端片段的预测也使得解码器能够学习到向前依赖的上下文表示。

联合训练

多任务学习（Multitask Learning）是机器学习的一个子领域，是指同时学习多个独立但是相关的任务 [243]。多任务学习通过模型共享的方式，对多个模型进行学习，而这些模型都对应不同的任务，这样不同模型可以互相“促进”。在神经机器翻译中，为了使用单语数据，可以将翻译任务作为主任务，同时设置一些仅使用单语数据的

子任务，通过这些子任务来捕捉单语数据中的语言知识 [58]。

语言模型是使用目标端单语数据最直接的方式，但是翻译模型作为一个受限的语言模型，还需要依赖于源语，并不能直接进行多任务学习。针对这个问题，对原有翻译模型结构进行了修改，在解码器中增加了一个语言模型子层，将这个子层用于语言模型任务（图7.40）。在训练过程中，分别将双语数据和单语数据送入翻译模型和语言模型进行计算，得到的损失相加用于整体模型参数的梯度计算和参数更新，其中语言模型的参数是翻译模型的一部分。

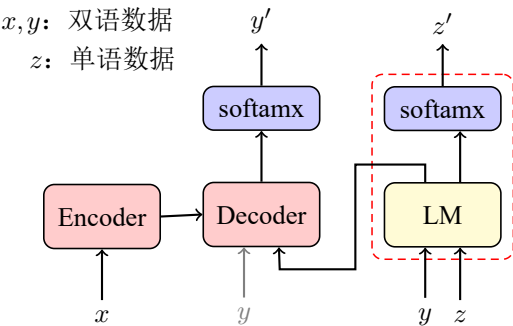


图 7.40: 机器翻译中的单任务学习和多任务学习

除了多任务学习，还有一些方法将前向模型和反向模型一起训练，在训练过程中同时使用源语言端和目标语言端的单语数据来提升模型性能，双向训练的内容会在7.5.4节中进行介绍。

7.5.3 知识精炼

理想的机器翻译系统应该是品质好、速度快、存储占用少。不过现实的机器翻译系统往往需要用运行速度和存储空间来换取翻译品质，比如，7.3.2节提到的增大模型容量的方法就是通过增加模型参数量来达到更好的函数拟合效果，但是这也导致系统变得更加笨拙。在很多场景下，这样的模型甚至无法使用。比如，Transformer-Big等“大”模型通常在专用 GPU 服务器上运行，在手机等受限环境下仍很难应用。

另一方面，直接训练“小”模型的效果往往并不理想，其翻译品质与“大”模型相比仍有比较明显的差距。比如，在 Transformer 中，使用一个 48 层的编码器要比传统的 6 层编码器在 BLEU 上高出 1-2 个点，而且两者翻译结果的人工评价的区别也十分明显。

面对小模型难以训练的问题，一种有趣的想法是把“大”模型的知识传递给“小”模型，让“小”模型可以更好的进行学习。这类似于，教小孩子学习数学，是请一个权威数学家（数据中的标准答案），还是请一个小学数学教师（“大”模型）。这就是知识精炼的基本思想。

什么是知识精炼

通常，知识精炼可以被看作是一种知识迁移的手段 [109]。如果把“大”模型的知识迁移到“小”模型，这种方法的直接结果就是**模型压缩**（Model Compression）。当然，理论上也可以把“小”模型的知识迁移到“大”模型，比如，将迁移后得到的“大”模型作为初始状态，之后继续训练该模型，以期取得加速收敛的效果。不过，在实践中更多是使用“大”模型到“小”模型的迁移，这也是本节讨论的重点。

知识精炼基于两个假设：

- “知识”在模型间是可迁移的。也就是说，一个模型中蕴含的规律可以被另一个模型使用。最典型的例子就是预训练模型（见7.5.2）。使用单语数据学习到的表示模型，在双语的翻译任务中仍然可以发挥很好的作用。也就是，把单语语言模型学习到的知识迁移到双语翻译中对句子表示的任务中；
- 模型所蕴含的“知识”比原始数据中的“知识”更容易被学习到。比如，机器翻译中大量使用的回译（伪数据）方法，就把模型的输出作为数据让系统进行学习。

这里所说的第二个假设对应了机器学习中的一大类问题——**学习难度**（Learning Difficulty）。所谓难度是指：在给定一个模型的情况下，需要花费多少代价对目标任务进行学习。如果目标任务很简单，同时模型与任务很匹配，那学习难度就会降低。如果目标任务很复杂，同时模型与其匹配程度很低，那学习难度就会很大。在自然语言处理任务中，这个问题的一种表现是：在很好的数据中学习的模型的翻译质量可能仍然很差。即使训练数据是完美的，但是模型仍然无法做到完美的学习。这可能是因为建模的不合理，导致模型无法描述目标任务中复杂的规律。也就是纵然数据很好，但是模型学不到其中的“知识”。在机器翻译中这个问题体现的尤为明显。比如，在机器翻译系统 n -best 结果中挑选最好的译文（成为 Oracle）作为训练样本让系统重新学习，系统仍然达不到 Oracle 的水平。

知识精炼本身也体现了一种“自学习”的思想。即利用模型（自己）的预测来教模型（自己）。这样既保证了知识可以向更轻量的模型迁移，同时也避免了模型从原始数据中学习难度大的问题。虽然“大”模型的预测中也会有错误，但是这种预测是更符合建模的假设的，因此“小”模型反倒更容易从不完美的信息中学习¹⁵到更多的知识。类似于，刚开始学习围棋的人从职业九段身上可能什么也学不到，但是向一个业余初段的选手学习可能更容易入门。另外，也有研究表明：在机器翻译中，相比于“小”模型，“大”模型更容易进行优化，也更容易找到更好的模型收敛状态。因此在需要一个性能优越，存储较小的模型时，也会考虑将大模型压缩得到更轻量模型的手段 [176]。

通常把“大”模型看作的传授知识的“教师”，被称作**教师模型**（Teacher Model）；

¹⁵很多时候，“大”模型和“小”模型都是基于同一种架构，因此二者对问题的假设和模型结构都是相似的。

把“小”模型看作是接收知识的“学生”，被称作**学生模型**（Student Model）。比如，可以把 Transformer-Big 看作是教师模型，把 Transformer-Base 看作是学生模型。

知识精炼的基本方法

知识精炼的基本思路是让学生模型所表示的函数尽可能去拟合教师模型所表示的函数 [109]。通常有两种实现方式 [139]:

- **基于单词的知识精炼**（Word-level Knowledge Distillation）。该方法的目标是使得学生模型的预测（分布）尽可能逼近教师模型的预测（分布）。令 $\mathbf{x} = \{x_1, \dots, x_m\}$ 和 $\mathbf{y} = \{y_1, \dots, y_n\}$ 分别表示输入和输出（数据中的答案）序列， V 表示目标语言词表， n 表示译文序列的长度，则基于单词的知识精炼的损失函数被定义为：

$$L_{\text{word}} = - \sum_{j=1}^n \sum_{y_j \in V} P_t(y_j|\mathbf{x}) \log P_s(y_j|\mathbf{x}) \quad (7.28)$$

这里， $P_s(y_j|\mathbf{x})$ 和 $P_t(y_i|\mathbf{x})$ 分别表示学生模型和教师模型在 j 位置的输出的概率。公式7.28实际上在最小化教师模型和学生模型输出分布之间的交叉熵。

- **基于序列的知识精炼**（Sequence-level Knowledge Distillation）。除了单词一级的拟合，基于序列的知识精炼希望在序列整体上进行拟合。其损失函数被定义为：

$$L_{\text{seq}} = - \sum_{\mathbf{y}} P_t(\mathbf{y}|\mathbf{x}) \log P_s(\mathbf{y}|\mathbf{x}) \quad (7.29)$$

公式7.29要求遍历所有可能的译文序列，并进行求和，当词表大小为 V ，序列长度为 L 时，则可能的序列的数量有 V 的 L 次幂，这么多的译文将消耗大量的计算资源。因此，会考虑用教师模型的真实输出序列 $\hat{\mathbf{y}}$ 来代替整个空间，即假设 $P_t(\hat{\mathbf{y}}|\mathbf{x}) = 1$ 。于是，目标函数变为：

$$L_{\text{seq}} = -\log P_s(\hat{\mathbf{y}}|\mathbf{x}) \quad (7.30)$$

这样的损失函数带来最直接的好处是，知识精炼的流程会非常简单。因为只需要利用教师模型将训练数据（源语言）翻译一遍，之后把它的输出替换为训练数据的目标语言部分。之后，利用得到的新的双语数据训练学生模型即可，图7.41展示了简化后词级和序列级的不同，其中词级知识精炼的解码端输入为真实双语数据的目标语言，并以 teacher 模型输出的概率分布作为学习目标，而序列级则直接将 teacher 推断后得到的结果作为解码端的输入，并将解码结果的 One-hot 向量作为学习目标。

本质上，基于单词的知识精炼和传统的语言模型等问题的建模方式是一致的。在传统方法中，训练数据中的答案会被看作是一个 One-hot 分布，之后让模型去尽可能

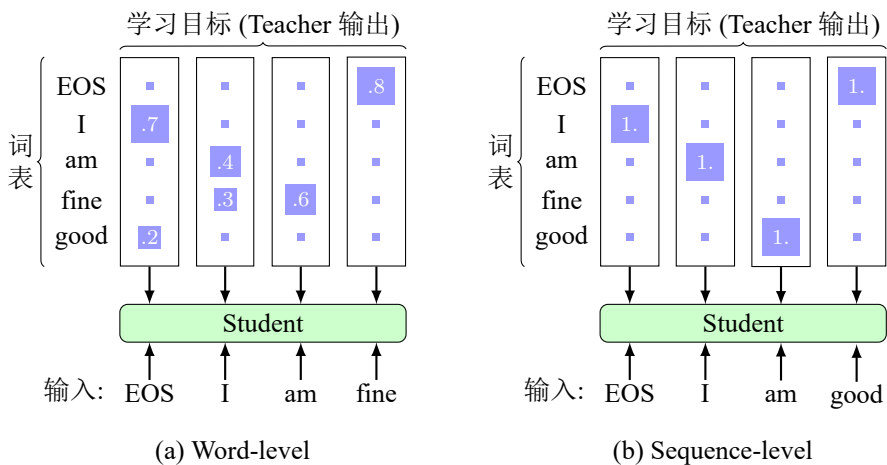


图 7.41: 词级和序列级知识精炼的差异

拟合这种分布。而这里，答案不再是一个 One-hot 分布，而是由教师模型生成的真实分布，但是损失函数的形式是一模一样的。在具体实现时，一个容易出现的问题是在词级别的知识精炼中，teacher 模型的 Softmax 可能会生成非常尖锐的分布。这时需要考虑对分布进行平滑，提高模型的泛化能力¹⁶。

除了在模型最后输出的分布上进行知识精炼，同样可以使用教师模型对学生模型的**中间层输出**（Hint-based Knowledge Transfer）和**注意力分布**（Attention To Attention Transfer）进行约束。而对翻译常用的 Transformer 架构，也有研究者使用更精细的精炼方式对模型各个位置的知识重新设计了知识迁移的方法 [131]。

机器翻译中的知识精炼

在神经机器翻译中，通常使用公式7.30的方法进行知识精炼，即通过教师模型构造伪数据，之后让学生模型从伪数据中学习。这样做的好处在于，系统研发人员不需要对系统进行任何修改，整个过程只需要调用教师模型和学生模型标准的训练和推断模块即可。

另一个问题是如何构造教师模型和学生模型。以 Transformer 为例，通常有两种思路：

- 固定教师模型，通过减少模型容量的方式设计学生模型。比如，可以使用容量较大的模型作为教师模型（如：Transformer-Big 或 Transformer-Deep），然后将神经网络变“窄”、变“浅”的方式得到学生模型。我们可以用 Transformer-Big 做教师模型，然后把 Transformer-Big 的解码器变为一层网络，作为学生模型。
- 固定学生模型，通过模型集成的方式设计教师模型。可以组合多个模型生成更

¹⁶比如，可以在 Softmax 函数中加入一个参数 α ，如 $\text{Softmax}(s_i) = \frac{\exp(s_i/\alpha)}{\sum_j \exp(s_j/\alpha)}$ 。这样可以通过 α 控制分布的平滑程度。

高质量的译文（见7.4.3节）。比如，融合多个 Transformer-Big 模型（不同参数初始化方式），之后学习一个 Transformer-Base 模型。

此外还可以采用迭代的知识精炼的方式。首先，通过模型集成得到较强的教师模型，再将知识迁移到不同的学生模型上，随后继续使用这些学生模型集成新的教师模型。不断的重复上述过程可以逐步提升集成模型的性能，如图7.42所示。值得注意的是，随着迭代次数的增加，集成所带来的收益也会随着子模型之间差异性的减小而减少。

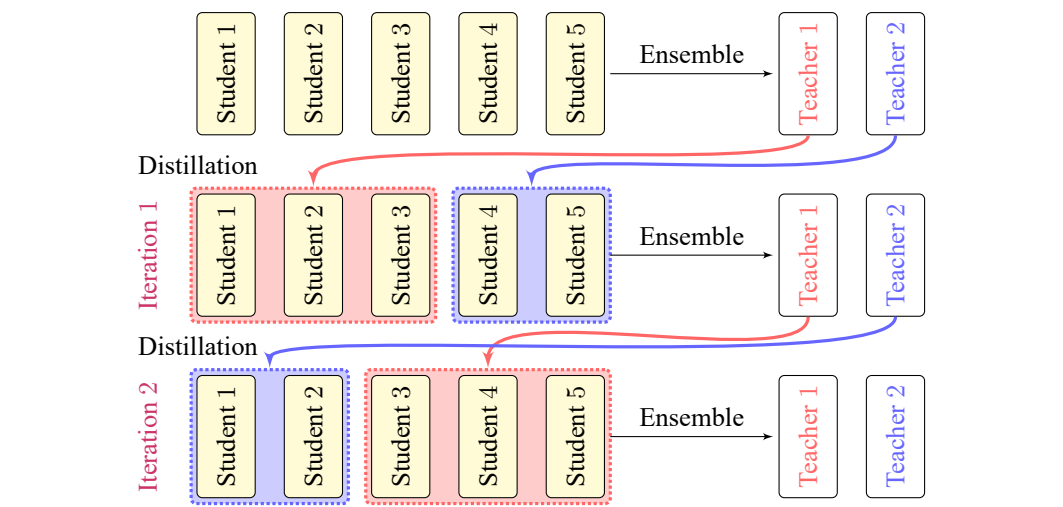


图 7.42: 迭代式知识精炼

如果倾向于使用更少的存储，更快的推理速度，则可以使用更小的学生模型。值得注意的是，对于 Transformer 模型来说，减少解码端的层数会给推理速度带来巨大的提升。特别是对于基于深层编码器的 Transformer-Deep，适当减少解码端层数往往不会带来翻译品质的下降。可以根据不同任务的需求，选择适当大小的学生模型，来平衡存储、推断速度和模型品质之间的关系。

7.5.4 双向训练

到目前为止，神经机器翻译系统都是每次一次只训练一个方向的模型。比如，给定中英的双语数据，一次只训练一个中到英或者英到中的翻译系统。既然两个方向的系统都使用同样的双语数据进行训练，那么是否可以一次训练同时得到两个方向的翻译系统呢？

回顾神经机器翻译系统的建模过程，给定一个互译的句对 (s, t) ，一个从源语言句子 s 到目标语言句子 t 的翻译被表示为求条件概率 $P(t|s)$ 的问题。类似地，一个从目标语言句子 t 到源语言句子 s 的翻译可以表示为 $P(s|t)$ 。通常来说，神经机器翻译的训练一次只得到一个方向的模型，也就是 $P(t|s)$ 或者 $P(s|t)$ 。这意味着 $P(t|s)$ 和 $P(s|t)$ 之间是互相独立的。 $P(t|s)$ 和 $P(s|t)$ 是否真的没有关系呢？比如， s 和 t 是相

同大小的向量，且 \mathbf{s} 到 \mathbf{t} 的变换是一个线性变换，也就是与一个方阵 \mathbf{W} 做矩阵乘法：

$$\mathbf{t} = \mathbf{s} \cdot \mathbf{W} \quad (7.31)$$

这里可以把 \mathbf{s} 和 \mathbf{t} 都看作分布式的向量表示； \mathbf{W} 应当是一个满秩矩阵，否则对于任意一个 \mathbf{s} 经过 \mathbf{W} 变换得到的 \mathbf{t} 只落在所有可能的 \mathbf{t} 的一个子空间内，即在给定 \mathbf{W} 的情况下有些 \mathbf{t} 不能被任何一个 \mathbf{s} 表达，而这不符合常识，因为不管是什么句子，我们总能找到它的一种译文。若 \mathbf{W} 是满秩矩阵说明 \mathbf{W} 可逆，也就是给定 \mathbf{s} 到 \mathbf{t} 的变换 \mathbf{W} 下， \mathbf{t} 到 \mathbf{s} 的变换必然是 \mathbf{W} 的逆而不是其他矩阵。这个例子说明 $P(\mathbf{t}|\mathbf{s})$ 和 $P(\mathbf{s}|\mathbf{t})$ 直觉上应当存在联系。当然， \mathbf{s} 和 \mathbf{t} 之间是否存在简单的线性变换关系并没有结论，但是上面的例子给出了一种对源语言句子和目标语言句子进行相互转化的思路。实际上，研究人员已经通过一些数学技巧用目标函数来把 $P(\mathbf{t}|\mathbf{s})$ 和 $P(\mathbf{s}|\mathbf{t})$ 联系起来，这样训练神经机器翻译系统一次就可以同时得到两个方向的翻译模型，使得训练变得更加高效 [99]。

有监督对偶学习

除了用条件概率 $P(\mathbf{t}|\mathbf{s})$ 建模翻译问题，还可以使用联合分布 $P(\mathbf{s}, \mathbf{t})$ 进行建模 [314]。根据条件概率的定义，有：

$$\begin{aligned} P(\mathbf{s}, \mathbf{t}) &= P(\mathbf{s})P(\mathbf{t}|\mathbf{s}) \\ &= P(\mathbf{t})P(\mathbf{s}|\mathbf{t}) \end{aligned} \quad (7.32)$$

公式7.32很自然地把两个方向的翻译模型 $P(\mathbf{t}|\mathbf{s})$ 和 $P(\mathbf{s}|\mathbf{t})$ 以及两个语言模型 $P(\mathbf{s})$ 和 $P(\mathbf{t})$ 联系起来： $P(\mathbf{s})P(\mathbf{t}|\mathbf{s})$ 应该与 $P(\mathbf{t})P(\mathbf{s}|\mathbf{t})$ 接近，因为它们都表达了同一个联合分布 $P(\mathbf{s}, \mathbf{t})$ 。因此，在构建训练两个方向的翻译模型的目标函数时，除了它们单独训练时各自使用的极大似然估计目标函数，可以额外增加一个目标项来鼓励两个方向的翻译模型去满足公式7.32：

$$\mathcal{L} = (\log P(\mathbf{s}) + \log P(\mathbf{t}|\mathbf{s}) - \log P(\mathbf{t}) - \log P(\mathbf{s}|\mathbf{t}))^2 \quad (7.33)$$

其中 $P(\mathbf{s})$ 和 $P(\mathbf{t})$ 这两个语言模型是预先训练好的，并不参与翻译模型的训练。可以看到，对于单独的一个模型来说，其目标函数增加了与另外一个方向的模型相关的项。这样的形式与 L1/L2 正则化非常类似（见7.3.1节），因此可以把这个方法看作是一种任务特定的正则化的手段（由翻译任务本身的性质所启发而来）。由于两个方向的翻译模型和语言模型相互影响，这种方法能得到比基于单个方向训练效果更好的模型。

无监督对偶学习

在有监督对偶学习对联合分布 $P(s, t)$ 建模的基础上，如果把 t 看作一个隐变量，那么可以得到边缘分布 $P(s)$ ，也就是关于 s 的语言模型：

$$\begin{aligned} P(s) &= \sum_t P(s, t) \\ &= \sum_t P(s|t)P(t|s) \end{aligned} \quad (7.34)$$

公式7.34假设 $P(s|t) = P(s|s, t)$ 。这个假设显然是成立的，因为当知道一个句子的译文时，并不需要知道它的源文就可以把它翻译回去。如果直接优化（最大化）公式7.34右侧，相当于对这个等式 $P(s|t)$ 和 $P(t|s)$ 施加了**循环一致性**（Circle Consistency）的约束 [353]，也就是对于一个句子 s ，通过 $P(t|s)$ 把它翻译成 t 后，根据 $P(s|t)$ 应该能重新翻译出 s ，如图7.43所示。公式7.34给出了同时优化 $P(s|t)$ 和 $P(t|s)$ 的一个目标函数形式。这个目标函数的一个额外的好处是它本质上是在学习一个由 $P(s|t)$ 和 $P(t|s)$ 组成的语言模型 $P(s)$ ，而 $P(s)$ 的学习依赖于单语数据，这意味着这个目标函数可以很自然地直接使用大量单语数据来同时训练两个翻译模型。相同的结论可以推广到 $P(t)$ 上 [100]。

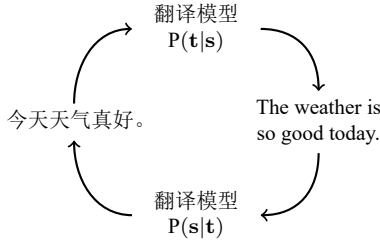


图 7.43: 循环一致性

但是直接使用公式7.34作为目标函数需要解决两个问题：

- 计算公式7.34要枚举所有可能的隐变量 t 的取值，也就是所有可能产生的目标语句子，而这是不可能的，因此一般会通过平均多个随机产生的 t 对应的损失来近似真正的目标函数值；
- 从公式7.34可以看到，在 $P(s)$ 上计算完目标函数值后，得到的梯度首先传递给 $P(s|t)$ ，然后通过 $P(s|t)$ 传递给 $P(t|s)$ 。由于 $P(s|t)$ 的输入 t 由 $P(t|s)$ 采样得到，而采样操作不可导，导致梯度的传播在 $P(t|s)$ 的输出处断开了，因此 $P(t|s)$ 接收不到任何梯度来进行更新。常见的解决方案是使用策略梯度 [277]。它把 $P(t|s)$ 采样得到的 t 当成 $P(t|s)$ 的目标来学习，并使用 $\log P(s|t)$ 对 $P(t|s)$ 的损失进行加权。但是由于仅使用少量样本来近似真正的目标函数，得到的策略梯度方差非常大，系统无法稳定学习，特别是训练的初期，因此通常会需要先使

用双语数据预训练两个方向的翻译模型，然后把公式7.34作为正常训练的一个正则化项使用。

翻译中回译

重新回顾公式7.34对应的目标函数，无监督对偶学习跟回译（假设现在只在一个句对 (s, t) 上做回译）之间有着很深的内在联系：给定一个句子 s ，无监督对偶学习和回译都首先用 $P(t|s)$ 把 s 翻译成 t ，然后无监督对偶学习最大化 $P(s|t)P(t|s)$ ，而回译则是最大化 $P(s|t)$ 。可以看到，当无监督对偶学习假设 $P(t|s)$ 是一个完美的翻译模型的时候，它与回译是等价的。此外，在共享两个方向的模型参数 θ 的情况下，可以看到无监督对偶学习的梯度为 $\frac{\partial P(s)}{\partial \theta} = P(t|s) \frac{\partial P(s|t)}{\partial \theta} + P(s|t) \frac{\partial P(t|s)}{\partial \theta}$ ，而回译的梯度为 $\frac{\partial P(s|t)}{\partial \theta}$ 。从这个角度出发，无监督对偶学习与回译都在优化语言模型 $P(s)$ 这个目标函数，只不过回译使用对 θ 有偏的梯度估计。

这个事实说明对回译进行适当的增广后应该能取得与无监督对偶学习相似的结果。**翻译中回译**（On-the-fly Back-translation）就是这样一个例子。一般回译的过程是先把数据集里所有 s 都翻译出来，然后只训练 $P(s|t)$ 。区别于回译，从数据集中采集到一个 s 之后，翻译中回译立刻把 s 翻译成 t ，然后训练 $P(s|t)$ ，并且在下一步迭代中采集一个 t 然后训练 $P(t|s)$ ，这样交替更新 $P(s|t)$ 和 $P(t|s)$ 。尽管翻译中回译无法像无监督对偶学习那样在一个样本里通过梯度把 $P(s|t)$ 的信息传到 $P(t|s)$ ，但是它交替更新 $P(s|t)$ 和 $P(t|s)$ 的策略允许 $P(s|t)$ 在两个样本间通过其产生的输出 s 来把信息传递到 $P(t|s)$ ，因此也能获得相近的效果，并且在实现和计算上都非常高效。翻译中回译已经在无监督神经机器翻译系统训练中被广泛使用 [49]。

7.6 小结及深入阅读

神经机器翻译的模型和技术方法已经十分丰富，无论是对基础问题的研究，还是研发实际可用的系统，人们都会面临很多选择。本章，从构建一个足以参加机器翻译比赛的系统出发，对神经机器翻译的数据处理、建模与训练、推断等基本问题进行了介绍。其中的许多方法已经在实践中得到验证，具有较好的参考意义。此外，本章也对一些前沿方法进行了讨论，旨在挖掘更具潜力的方向。

除了以上内容，还有一些方向值得关注：

- 无指导机器翻译。无指导机器翻译由于其不需要双语语料即可训练翻译模型的特性，在稀缺资源机器翻译的场景中有非常大的潜力而得到广泛的关注。目前无指导机器翻译主要有两种范式：第一种先得到词典的翻译，然后得到短语表的翻译和相应的统计机器翻译系统，最后使用统计机器翻译系统生成伪双语平行语料训练神经机器翻译系统 [3]；第二种是先预训练语言模型来初始化神经机器翻译系统的编码器和解码器，然后使用翻译中回译以及降噪自编码器来训练神经机器翻译系统 [49]。尽管目前无指导机器翻译在富资源的语种上取得了

很大进展,但是离实际应用还有很远距离。比如,目前无指导系统都依赖于大量单语数据,而实际上稀缺资源的语种不但双语语料少,单语语料也少;此外,这些系统还无法在远距离如中英这些字母表重合少,需要大范围调序的语种对上取得可接受的结果;使用大量单语训练无指导系统还面临数据来自于不同领域的问题 [194]。设计更鲁棒,使用单语数据更高效的无指导机器翻译方法乃至新范式会是未来的趋势。

- 图片翻译。由于人类语言潜在的歧义性,传统的神经机器翻译在单句翻译中可能会出现歧义。为此,一些研究工作在翻译过程中尝试引入更多的上下文信息,比如多模态翻译、基于树的翻译或者篇章级翻译。比如,图片翻译的目标就是在给定一个图片和其源语描述的情况下,生成目标语言的描述。一般做法就是通过一个额外的编码器来提取图像特征 [71, 110],然后通过权重门控机制、注意力网络等融合到系统中 [119]。
- 基于树的翻译。这类方法在翻译模型中引入句法结构树或依存树,从而引入更多的句法信息。一种常用的做法是将句法树进行序列化,从而保留序列到序列的模型结构 [52, 245]。在此基础上,一些研究工作引入了更多的解析结果 [189, 337]。同时,也有一些研究工作直接使用 **Tree-LSTMs** 等网络结构 [257, 279] 来直接表示树结构,并将其应用到神经机器翻译模型中 [33, 73, 330]。
- 篇章级翻译。可以通过引入篇章级上下文信息,来处理篇章翻译中译文不连贯,主谓不一致等问题。为此,一些研究人员针对该问题进行了改进,主要可以分为两类方法:一种是将当前句子与上下文进行句子级的拼接,不改变模型的结构 [282],另外一种是采用额外的编码器来捕获篇章信息 [128, 295, 346]。编码器的结构除了传统的 RNN、自注意力网络,还有利用层级注意力来编码之前的多句上文 [280, 308],使用可选择的稀疏注意力机制对整个文档进行篇章建模 [198],使用记忆网络、缓存机制等对篇章中的关键词进行提取 [156, 285] 或者采用两阶段解码的方式 [294, 327]。除了从建模角度引入上下文信息,也有一些工作使用篇章级修正模型 [293] 或者语言模型 [333] 对句子级翻译模型的译文进行修正,或者通过自学习在解码过程中保持翻译连贯性 [193]。
- 语音翻译。在日常生活中,语音翻译也是有很大的需求。针对语音到文本翻译的特点,最简单的做法是使用自动语音识别 (ASR) 将语音转换成文本,然后送入文本翻译模型进行翻译 [199, 212]。然而为了避免流水线中的错误传播和高延迟问题,现在通常采用端到端的建模做法 [18, 65]。同时,针对语音翻译数据稀缺的问题,一些研究工作采用各种方法来进行缓解,包括预训练 [9]、多任务学习 [17, 307]、课程学习 [136]、注意力传递 [263] 和知识精炼 [129, 184]。
- 多语言翻译。神经机器翻译模型经过训练,通常可以将一种固定的源语言翻译成另一种固定的目标语言,但考虑到世界上有成千上万种语言,为每种语言对训练一个单独的模型非常耗资源 [132]。多语言神经机器翻译旨在训练涵盖多


种语言翻译的单一模型。多语言神经机器翻译系统可以根据它们在不同翻译语言对之间共享的组件进行分类。一种常见的做法是通过语言标签指定源语言和目标语言的同时，共享整个神经网络结构（编码器和解码器）[98, 132]。除此之外，还可以共享部分网络结构。比如共享编码器的一对多模型[60]，共享解码器的多对一模型[77, 358]，以及共享注意力机制的多对多模型[76, 186]。多语言神经机器翻译不仅可以减少训练单一语言对神经机器翻译的训练代价，还可以有效的解决零资源神经机器翻译[132]以及多源神经机器翻译问题[219]。

- 结构搜索。除了由研究人员手工设计神经网络结构之外，近些年**网络结构搜索技术**（Neural Architecture Search; NAS）也逐渐在包括机器翻译在内的自然语言处理任务中得到广泛关注[72]。不同于前文提到的基于循环神经网络、Transformer结构的机器翻译模型，网络结构搜索旨在通过自动的方式根据提供的训练数据自动学习到最适合于当前任务的神经网络模型结构。目前而言，网络结构搜索的方法已经在自然语言处理的各项任务中崭露头角，在语言模型、命名实体识别等任务中获得优异的成绩[130, 173, 359]，但对于机器翻译任务而言，由于其任务的复杂性，网络结构的搜索空间往往比较大，很难直接对其空间进行搜索，因此研究人员更倾向于对基于现有经验设计的模型结构进行改良。谷歌大脑团队在 The Evolved Transformer 文章中提出使用进化算法，在 Transformer 结构基础上对模型结构进行演化，得到更加高效且建模能力更强的机器翻译模型。微软团队也在 Neural Architecture Optimization[185] 论文中提出 NAO 的方法，通过将神经网络结构映射到连续空间上进行优化来获得优于初始结构的模型，NAO 方法在 WMT19 机器翻译评测任务中也进行了使用，在英语-芬兰语以及芬兰语-英语的任务上均取得了优异的成绩。
- 与统计机器翻译的结合。尽管神经机器翻译在自动评价和人工评价上都取得比统计机器翻译优异的结果，但神经机器翻译仍然面临一些统计机器翻译没有的问题[152]，如神经机器翻译系统会产生漏译的现象，也就是源语句子的一些短语甚至从句没有被翻译，而统计机器翻译因为是把源语里所有短语都翻译出来后进行拼装，因此不会产生这种译文对原文的忠实度低的问题。一个解决的思路就是把统计机器翻译系统和神经机器翻译系统进行结合。目前的方法主要分为两种，一种是模型的改进，比如在神经机器翻译里建模统计机器翻译的概念或者使用统计机器翻译系统的模块，如词对齐，覆盖度等等[104]，或者是把神经机器翻译系统结合到统计机器翻译系统中，如作为一个特征[211]；第二种是系统融合，在不改变模型的情况下，把来自神经机器翻译系统的输出和统计机器翻译系统的输出进行融合，得到更好的结果，如使用重排序[138, 211]，后处理[215]，或者把统计机器翻译系统的输出作为神经机器翻译系统解码的约束条件等等[267]。除此之外，也可以把神经机器翻译与翻译记忆相融合[103, 313]，这在机器翻译应用中也是非常有趣的方向。



附录

A	附录 A	407
A.1	基准数据集	
A.2	平行语料	
A.3	相关工具	
B	附录 B	411
B.1	IBM 模型 3 训练方法	
B.2	IBM 模型 4 训练方法	
B.3	IBM 模型 5 训练方法	



A. 附录 A

在构建机器翻译系统的过程中，数据是必不可少的，尤其是现在主流的神经机器翻译系统，系统的性能往往受限于语料库规模和质量。所幸的是，随着语料库语言学的发展，一些主流语种的相关语料资源已经十分丰富。

为了方便读者进行相关研究，我们汇总了几个常用的基准数据集，这些数据集已经在机器翻译领域中被广泛使用，有很多之前的相关工作可以进行复现和对比。同时，我们收集了一下常用的平行语料，方便读者进行一些探索。

A.1 基准数据集

表 A.1: 基准数据集

任务	语种	领域	描述	数据集地址
WMT	En Zh De Ru 等	新闻、医学、翻译	以英语为核心的多语种机器翻译数据集，涉及多种任务	http://www.statmt.org/wmt19/
IWSLT	En De Fr Cs Zh 等	口语翻译	文本翻译数据集来自 TED 演讲，数据规模较小	https://wit3.fbk.eu/
NIST	Zh-En 等 Cs Zh 等	新闻翻译	评测集包括 4 句参考译文，质量较高	https://www ldc.upenn.edu/collaborations/evaluations/nist

任务	语种	领域	描述	数据集地址
TVsub	Zh-En	字幕翻译	数据抽取自电视剧字幕，用于对话中长距离上下文研究	https://github.com/longyuewan/gdcu/tvsub
Flickr30K	En-De	多模态翻译	31783 张图片，每张图片 5 个语句标注	http://shannon.cs.illinois.edu/DenotationGraph/
Multi30K	En-De En-Fr	多模态翻译	31014 张图片，每张图片 5 个语句标注	http://www.statmt.org/wmt16/multimodal-task.html
IAPRTC-12	En-De	多模态翻译	20000 张图片及对应标注	https://www.imageclef.org/photodata
IKEA	En-De En-Fr	多模态翻译	3600 张图片及对应标注	https://github.com/sampalomad/IKEA-Dataset.git

A.2 平行语料

神经机器翻译系统的训练需要大量的双语数据，这里我们汇总了一些公开的平行语料，方便读者获取。

- **News Commentary Corpus:** 包括汉语、英语等 12 个语种，64 个语言对的双语数据，爬取自 Project Syndicate 网站的政治、经济评论。URL: <http://www.casmacat.eu/corpus/news-commentary.html>
- **CWMT Corpus:** 中国计算机翻译研讨会社区收集和共享的中英平行语料，涵盖多种领域，例如新闻、电影字幕、小说和政府文档等。URL: <http://nlp.nju.edu.cn/cwmt-wmt/>
- **Common Crawl corpus:** 包括捷克语、德语、俄语、法语 4 种语言到英语的双语数据，爬取自互联网网页。URL: <http://www.statmt.org/wmt13/training-parallel-commoncrawl.tgz>
- **Europarl Corpus:** 包括保加利亚语、捷克语等 20 种欧洲语言到英语的双语数据，来源于欧洲议会记录。URL: <http://www.statmt.org/europarl/>
- **ParaCrawl Corpus:** 包括 23 种欧洲语言到英语的双语语料，数据来源于网络爬取。URL: <https://www.paracrawl.eu/index.php>
- **United Nations Parallel Corpus:** 包括阿拉伯语、英语、西班牙语、法语、俄语、汉语 6 种联合国正式语言，30 种语言对的双语数据，来源于联合国公共领域的官方记录和其他会议文件。URL: <https://conferences.unite.un.org/UNCorpus/>
- **TED Corpus:** TED 大会演讲在其网站公布了自 2007 年以来的演讲字幕，以

及超过 100 种语言的翻译版本。WIT 收集整理这些数据，以方便科研工作者使用，同时，会为每年的 IWSLT 评测比赛提供评测数据集。URL: <https://wit3.fbk.eu/>

- OpenSubtile: 由 P. Lison 和 J. Tiedemann 收集自 opensubtitles 电影字幕网站，包含 62 种语言、1782 个语种对的平行语料，资源相对比较丰富。URL: <http://opus.nlpl.eu/OpenSubtitles2018.php>
- Wikititles Corpus: 包括古吉拉特语等 14 个语种，11 个语言对的双语数据，数据来源于维基百科的标题。URL: <http://data.statmt.org/wikititles/v1/>
- CzEng: 捷克语和英语的平行语料，数据来源于欧洲法律、信息技术和小说领域。URL: <http://ufal.mff.cuni.cz/czeng/czeng17>
- Yandex Corpus: 俄语和英语的平行语料，爬取自互联网网页。URL: <https://translate.yandex.ru/corpus>
- Tilde MODEL Corpus: 欧洲语言的多语言开放数据，包含多个数据集，数据来自于经济、新闻、政府、旅游等门户网站。URL: https://tilde-model.s3-eu-west-1.amazonaws.com/Tilde_MODEL_Corpus.html
- Setimes Corpus: 包括克罗地亚语、阿尔巴尼亚等 9 种巴尔干语言，72 个语言对的双语数据，来源于东南欧时报的新闻报道。URL: <http://www.statmt.org/setimes/>
- TVsub: 收集自电视剧集字幕的中英文对话语料库，包含超过 200 万的句对，可用于对话领域和长距离上下文信息的研究。URL: <https://github.com/longyuewangdcu/tvsub>
- Recipe Corpus: 由 Cookpad 公司创建的日英食谱语料库，包含 10 万多的句对。URL: <http://lotus.kuee.kyoto-u.ac.jp/WAT/recipe-corpus/>

A.3 相关工具

A.3.1 数据预处理工具

数据处理是搭建神经机器翻译系统的重要步骤，这里我们提供了一些开源工具供读者进行使用。


- Moses: Moses 提供了很多数据预处理的脚本和工具，被机器翻译研究者广泛使用。其中包括符号标准化、分词、大小写转换和长度过滤等。URL: <https://github.com/moses-smt/mosesdecoder/tree/master/scripts>
- Jieba: 常用的中文分词工具。URL: <https://github.com/fxsjy/jieba>

- Subword-nmt: 基于 BPE 算法的子词切分工具。URL: <https://github.com/rsennrich/subword-nmt>

A.3.2 评价工具

机器翻译领域已经有多种自动评价指标，包括 BLEU、TER 和 METEOR 等，这里我们提供了一些自动评价指标的工具，方便读者使用。

- Moses: 其中包括了通用的 BLEU 评测脚本。URL: <https://github.com/moses-smt/mosesdecoder/tree/master/scripts/generic>
- Tercom: 自动评价指标 TER 的计算工具，只有 java 版本。URL: <http://www.cs.umd.edu/~snover/tercom/>
- Meteor: 自动评价指标 METEOR 的实现。URL: <https://www.cs.cmu.edu/~alavie/METEOR/>



B. 附录 B

B.1 IBM 模型 3 训练方法

模型 3 的参数估计与模型 1 和模型 2 采用相同的方法。这里直接给出辅助函数。

$$\begin{aligned} h(t, d, n, p, \lambda, \mu, \nu, \zeta) = & P_{\theta}(\mathbf{s}|\mathbf{t}) - \sum_t \lambda_t \left(\sum_s t(s|t) - 1 \right) \\ & - \sum_i \mu_{iml} \left(\sum_j d(j|i, m, l) - 1 \right) \\ & - \sum_t \nu_t \left(\sum_{\varphi} n(\varphi|t) - 1 \right) - \zeta(p^0 + p^1 - 1) \end{aligned} \quad (\text{B.1})$$

由于篇幅所限这里略去了推导步骤直接给出一些用于参数估计的等式。

$$c(s|t, \mathbf{s}, \mathbf{t}) = \sum_{\mathbf{a}} [P_{\theta}(\mathbf{s}, \mathbf{a}|\mathbf{t}) \times \sum_{j=1}^m (\delta(s_j, s) \cdot \delta(t_{a_j}, t))] \quad (\text{B.2})$$

$$c(j|i, m, l; \mathbf{s}, \mathbf{t}) = \sum_{\mathbf{a}} [P_{\theta}(\mathbf{s}, \mathbf{a}|\mathbf{t}) \times \delta(i, a_j)] \quad (\text{B.3})$$

$$c(\varphi|t; \mathbf{s}, \mathbf{t}) = \sum_{\mathbf{a}} [P_{\theta}(\mathbf{s}, \mathbf{a}|\mathbf{t}) \times \sum_{i=1}^l \delta(\varphi, \varphi_i) \delta(t, t_i)] \quad (\text{B.4})$$

$$c(0|\mathbf{s}, \mathbf{t}) = \sum_{\mathbf{a}} [\mathbf{P}_{\theta}(\mathbf{s}, \mathbf{a}|\mathbf{t}) \times (m - 2\varphi_0)] \quad (\text{B.5})$$

$$c(1|\mathbf{s}, \mathbf{t}) = \sum_{\mathbf{a}} [\mathbf{P}_{\theta}(\mathbf{s}, \mathbf{a}|\mathbf{t}) \times \varphi_0] \quad (\text{B.6})$$

进一步，对于由 K 个样本组成的训练集，有：

$$t(s|t) = \lambda_t^{-1} \times \sum_{k=1}^K c(s|t; \mathbf{s}^{[k]}, \mathbf{t}^{[k]}) \quad (\text{B.7})$$

$$d(j|i, m, l) = \mu_{iml}^{-1} \times \sum_{k=1}^K c(j|i, m, l; \mathbf{s}^{[k]}, \mathbf{t}^{[k]}) \quad (\text{B.8})$$

$$n(\varphi|t) = \nu_t^{-1} \times \sum_{s=1}^K c(\varphi|t; \mathbf{s}^{[k]}, \mathbf{t}^{[k]}) \quad (\text{B.9})$$

$$p_x = \zeta^{-1} \sum_{k=1}^K c(x; \mathbf{s}^{[k]}, \mathbf{t}^{[k]}) \quad (\text{B.10})$$

在模型 3 中，因为产出率的引入，并不能像模型 1 和模型 2 那样，在保证正确性的情况下加速参数估计的过程。这就使得每次迭代过程中，都不得不面对大小为 $(l+1)^m$ 的词对齐空间。遍历所有 $(l+1)^m$ 个词对齐所带来的高时间复杂度显然是不能被接受的。因此就要考虑能否仅利用词对齐空间中的部分词对齐对这些参数进行估计。比较简单且直接的方法就是仅利用 Viterbi 对齐来进行参数估计¹。遗憾的是，在模型 3 中并没有方法直接获得 Viterbi 对齐。这样只能采用一种折中的策略，即仅考虑那些使得 $\mathbf{P}_{\theta}(\mathbf{s}, \mathbf{a}|\mathbf{t})$ 达到较高值的词对齐。这里把这部分词对齐组成的集合记为 S 。式 B.2 可以被修改为：

$$c(s|t, \mathbf{s}, \mathbf{t}) \approx \sum_{\mathbf{a} \in S} [\mathbf{P}_{\theta}(\mathbf{s}, \mathbf{a}|\mathbf{t}) \times \sum_{j=1}^m (\delta(s_j, \mathbf{s}) \cdot \delta(t_{a_j}, \mathbf{t}))] \quad (\text{B.11})$$

同理可以获得式 B.3-B.6 的修改结果。进一步，在 IBM 模型 3 中，可以定义 S 如下：

$$S = N(b^{\infty}(V(\mathbf{s}|\mathbf{t}; 2))) \cup (\cup_{ij} N(b_{i \leftrightarrow j}^{\infty}(V_{i \leftrightarrow j}(\mathbf{s}|\mathbf{t}, 2)))) \quad (\text{B.12})$$

为了理解这个公式，先介绍几个概念。

- $V(\mathbf{s}|\mathbf{t})$ 表示 Viterbi 词对齐， $V(\mathbf{s}|\mathbf{t}, 1)$ 、 $V(\mathbf{s}|\mathbf{t}, 2)$ 和 $V(\mathbf{s}|\mathbf{t}, 3)$ 就分别对应了模型

¹Viterbi 词对齐可以被简单的看作搜索到的最好词对齐。

1、2 和 3 的 Viterbi 词对齐；

- 把那些满足第 j 个源语言单词对应第 i 个目标语言单词 ($a_j = i$) 的词对齐构成的集合记为 $\mathbf{A}_{i \leftrightarrow j}(\mathbf{s}, \mathbf{t})$ 。通常称这些对齐中 j 和 i 被“钉”在了一起。在 $\mathbf{A}_{i \leftrightarrow j}(\mathbf{s}, \mathbf{t})$ 中使 $P(\mathbf{a}|\mathbf{s}, \mathbf{t})$ 达到最大的那个词对齐被记为 $V_{i \leftrightarrow j}(\mathbf{s}, \mathbf{t})$ ；
- 如果两个词对齐，通过交换两个词对齐连接就能互相转化，则称它们为邻居。一个词对齐 \mathbf{a} 的所有邻居记为 $N(\mathbf{a})$ 。

公式 B.12 中, $b^\infty(V(\mathbf{s}|\mathbf{t}; 2))$ 和 $b_{i \leftrightarrow j}^\infty(V_{i \leftrightarrow j}(\mathbf{s}|\mathbf{t}, 2))$ 分别是对 $V(\mathbf{s}|\mathbf{t}; 3)$ 和 $V_{i \leftrightarrow j}(\mathbf{s}|\mathbf{t}, 3)$ 的估计。在计算 S 的过程中, 需要知道一个对齐 \mathbf{a} 的邻居 \mathbf{a}' 的概率, 即通过 $P_\theta(\mathbf{a}, \mathbf{s}|\mathbf{t})$ 计算 $p_\theta(\mathbf{a}', \mathbf{s}|\mathbf{t})$ 。在模型 3 中, 如果 \mathbf{a} 和 \mathbf{a}' 仅区别于某个源语单词对齐到的目标位置上 ($a_j \neq a'_j$), 那么

$$P_\theta(\mathbf{a}', \mathbf{s}|\mathbf{t}) = P_\theta(\mathbf{a}, \mathbf{s}|\mathbf{t}) \cdot \frac{\varphi_{i'} + 1}{\varphi_i} \cdot \frac{n(\varphi_{i'} + 1|t_{i'})}{n(\varphi_{i'}|t_{i'})} \cdot \frac{n(\varphi_i - 1|t_i)}{n(\varphi_i|t_i)} \cdot \frac{t(s_j|t_{i'})}{t(s_j|t_i)} \cdot \frac{d(j|i', m, l)}{d(j|i, m, l)} \quad (\text{B.13})$$

如果 \mathbf{a} 和 \mathbf{a}' 区别于两个位置 j_1 和 j_2 的对齐上, $a_{j_1} = a'_{j_2}$ 且 $a_{j_2} = a'_{j_1}$, 那么

$$P_\theta(\mathbf{a}', \mathbf{s}|\mathbf{t}) = P_\theta(\mathbf{a}, \mathbf{s}|\mathbf{t}) \cdot \frac{t(s_{j_2}|t_{a_{j_2}})}{t(s_{j_1}|t_{a_{j_1}})} \cdot \frac{d(j_2|a_{j_2}, m, l)}{d(j_1|a_{j_1}, m, l)} \quad (\text{B.14})$$

相比整个词对齐空间, S 只是一个非常小的子集, 因此运算复杂度可以被大大降低。可以看到, 模型 3 的参数估计过程是建立在模型 1 和模型 2 的参数估计结果上的。这不仅是因为模型 3 要利用模型 2 的 Viterbi 对齐, 而且还因为模型 3 参数的初值也要直接利用模型 2 的参数。从这个角度说, 模型 1, 2, 3 是有序的且向前依赖的。单独的对模型 3 的参数进行估计是极其困难的。实际上 IBM 的模型 4 和模型 5 也具有这样的性质, 即它们都可以利用前一个模型参数估计的结果作为自身参数的初始值。

B.2 IBM 模型 4 训练方法

模型 4 的参数估计基本与模型 3 一致。需要修改的是扭曲度的估计公式, 对于目标语第 i 个 cept. 生成的第一单词, 可以得到 (假设有 K 个训练样本):

$$d_1(\Delta_j|ca, cb; \mathbf{s}, \mathbf{t}) = \mu_{1cacb}^{-1} \times \sum_{k=1}^K c_1(\Delta_j|ca, cb; \mathbf{s}^{[k]}, \mathbf{t}^{[k]}) \quad (\text{B.15})$$

其中,

$$c_1(\Delta_j|ca, cb; \mathbf{s}, \mathbf{t}) = \sum_{\mathbf{a}} [\mathbf{P}_{\theta}(\mathbf{s}, \mathbf{a}|\mathbf{t}) \times s_1(\Delta_j|ca, cb; \mathbf{a}, \mathbf{s}, \mathbf{t})] \quad (\text{B.16})$$

$$s_1(\Delta_j|ca, cb; \mathbf{a}, \mathbf{s}, \mathbf{t}) = \sum_{i=1}^l [\varepsilon(\phi_i) \cdot \delta(\pi_{i1} - \odot_i, \Delta_j) \cdot \delta(A(t_{i-1}), ca) \cdot \delta(B(\tau_{i1}), cb)] \quad (\text{B.17})$$

且

$$\varepsilon(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases} \quad (\text{B.18})$$

对于目标语第 i 个 **cept.** 生成的其他单词 (非第一个单词), 可以得到:

$$d_{>1}(\Delta_j|cb; \mathbf{s}, \mathbf{t}) = \mu_{>1cb}^{-1} \times \sum_{k=1}^K c_{>1}(\Delta_j|cb; \mathbf{s}^{[k]}, \mathbf{t}^{[k]}) \quad (\text{B.19})$$

其中,

$$c_{>1}(\Delta_j|cb; \mathbf{s}, \mathbf{t}) = \sum_{\mathbf{a}} [\mathbf{p}_{\theta}(\mathbf{s}, \mathbf{a}|\mathbf{t}) \times s_{>1}(\Delta_j|cb; \mathbf{a}, \mathbf{s}, \mathbf{t})] \quad (\text{B.20})$$

$$s_{>1}(\Delta_j|cb; \mathbf{a}, \mathbf{s}, \mathbf{t}) = \sum_{i=1}^l [\varepsilon(\phi_i - 1) \sum_{k=2}^{\phi_i} \delta(\pi_{[i]k} - \pi_{[i]k-1}, \Delta_j) \cdot \delta(B(\tau_{[i]k}), cb)] \quad (\text{B.21})$$

这里, ca 和 cb 分别表示目标语言和源语言的某个词类。模型 4 需要像模型 3 一样, 通过定义一个词对齐集合 S , 使得每次迭代都在 S 上进行, 进而降低运算量。模型 4 中 S 的定义为:

$$S = N(\tilde{b}^{\infty}(V(\mathbf{s}|\mathbf{t}; 2))) \cup (\bigcup_{ij} N(\tilde{b}_{i \leftrightarrow j}^{\infty}(V_{i \leftrightarrow j}(\mathbf{s}|\mathbf{t}, 2)))) \quad (\text{B.22})$$

对于一个对齐 \mathbf{a} , 可用模型 3 对它的邻居进行排名, 即按 $\mathbf{P}_{\theta}(b(\mathbf{a})|\mathbf{s}, \mathbf{t}; 3)$ 排序, 其中 $b(\mathbf{a})$ 表示 \mathbf{a} 的邻居。 $\tilde{b}(\mathbf{a})$ 表示这个排名表中满足 $\mathbf{P}_{\theta}(\mathbf{a}'|\mathbf{s}, \mathbf{t}; 4) > \mathbf{P}_{\theta}(\mathbf{a}|\mathbf{s}, \mathbf{t}; 4)$ 的最

高排名的 \mathbf{a}' 。同理可知 $\tilde{b}_{i \leftrightarrow j}^\infty(\mathbf{a})$ 的意义。这里之所以不用模型 3 中采用的方法直接利用 $b^\infty(\mathbf{a})$ 得到模型 4 中高概率的对齐，是因为模型 4 中，要想获得某个对齐 \mathbf{a} 的邻居 \mathbf{a}' ，必须做很大调整，比如：调整 $\tau_{[i]1}$ 和 \odot_i 等等。这个过程要比模型 3 的相应过程复杂得多。因此在模型 4 中只能借助于模型 3 的中间步骤来进行参数估计。

B.3 IBM 模型 5 训练方法

模型 5 的参数估计过程也与模型 3 的过程基本一致，二者的区别在于扭曲度的估计公式。在模型 5 中，对于目标语第 i 个 **cept.** 生成的第一单词，可以得到（假设有 K 个训练样本）：

$$d_1(\Delta_j|cb;\mathbf{s},\mathbf{t}) = \mu_{1cb}^{-1} \times \sum_{k=1}^K c_1(\Delta_j|cb;\mathbf{s}^{[k]},\mathbf{t}^{[k]}) \quad (\text{B.23})$$

其中，

$$c_1(\Delta_j|cb,v_x,v_y;\mathbf{s},\mathbf{t}) = \sum_{\mathbf{a}} \left[P(\mathbf{s},\mathbf{a}|\mathbf{t}) \times s_1(\Delta_j|cb,v_x,v_y;\mathbf{a},\mathbf{s},\mathbf{t}) \right] \quad (\text{B.24})$$

$$s_1(\Delta_j|cb,v_x,v_y;\mathbf{a},\mathbf{s},\mathbf{t}) = \sum_{i=1}^l \left[\varepsilon(\phi_i) \cdot \delta(v_{\pi_{i1}}, \Delta_j) \cdot \delta(v_{\odot_{i-1}}, v_x) \cdot \delta(v_m - \phi_i + 1, v_y) \cdot \delta(v_{\pi_{i1}}, v_{\pi_{i1}-1}) \right] \quad (\text{B.25})$$

对于目标语第 i 个 **cept.** 生成的其他单词（非第一个单词），可以得到：

$$d_{>1}(\Delta_j|cb,v;\mathbf{s},\mathbf{t}) = \mu_{>1cb}^{-1} \times \sum_{k=1}^K c_{>1}(\Delta_j|cb,v;\mathbf{s}^{[k]},\mathbf{t}^{[k]}) \quad (\text{B.26})$$

其中，

$$c_{>1}(\Delta_j|cb,v;\mathbf{s},\mathbf{t}) = \sum_{\mathbf{a}} \left[P(\mathbf{a},\mathbf{s}|\mathbf{t}) \times s_{>1}(\Delta_j|cb,v;\mathbf{a},\mathbf{s},\mathbf{t}) \right] \quad (\text{B.27})$$

$$s_{>1}(\Delta_j|cb,v;\mathbf{a},\mathbf{s},\mathbf{t}) = \sum_{i=1}^l \left[\varepsilon(\phi_i - 1) \sum_{k=2}^{\phi_i} \left[\delta(v_{\pi_{ik}} - v_{\pi_{[i]k}-1}, \Delta_j) \cdot \delta(B(\tau_{[i]k}), cb) \cdot \delta(v_m - v_{\pi_{i(k-1)}} - \phi_i + k, v) \cdot \delta(v_{\pi_{i1}}, v_{\pi_{i1}-1}) \right] \right] \quad (\text{B.28})$$

从式 (B.24) 中可以看出因子 $\delta(v_{\pi_{i1}}, v_{\pi_{i1}-1})$ 保证了，即使对齐 \mathbf{a} 不合理（一个源

语位置对应多个目标语位置)也可以避免在这个不合理的对齐上计算结果。需要注意的是因子 $\delta(v_{\pi_{p1}}, v_{\pi_{p1-1}})$, 确保了 \mathbf{a} 中不合理的部分不产生坏的影响, 而 \mathbf{a} 中其他正确的部分仍会参与迭代。

不过上面的参数估计过程与 IBM 前 4 个模型的参数估计过程并不完全一样。IBM 前 4 个模型在每次迭代中, 可以在给定 \mathbf{s} 、 \mathbf{t} 和一个对齐 \mathbf{a} 的情况下直接计算并更新参数。但是在模型 5 的参数估计过程中 (如公式 B.24), 需要模拟出由 \mathbf{t} 生成 \mathbf{s} 的过程才能得到正确的结果, 因为从 \mathbf{t} 、 \mathbf{s} 和 \mathbf{a} 中是不能直接得到的正确结果的。具体说, 就是要从目标语言句子的第一个单词开始到最后一个单词结束, 依次生成每个目标语言单词对应的源语言单词, 每处理完一个目标语言单词就要暂停, 然后才能计算式 B.24 中求和符号里面的内容。这也就是说即使给定了 \mathbf{s} 、 \mathbf{t} 和一个对齐 \mathbf{a} , 也不能直接在它们上进行计算, 必须重新模拟 \mathbf{t} 到 \mathbf{s} 的生成过程。

从前面的分析可以看出, 虽然模型 5 比模型 4 更精确, 但是模型 5 过于复杂以至于给参数估计增加了计算量 (对于每组 \mathbf{t} 、 \mathbf{s} 和 \mathbf{a} 都要模拟 \mathbf{t} 生成 \mathbf{s} 的翻译过程)。因此模型 5 的开发对于系统实现是一个挑战。

在模型 5 中同样需要定义一个词对齐集合 S , 使得每次迭代都在 S 上进行。可以对 S 进行如下定义

$$S = N(\tilde{b}^\infty(V(\mathbf{s}|\mathbf{t}; 2))) \cup (\bigcup_{ij} N(\tilde{b}_{i \leftrightarrow j}^\infty(V_{i \leftrightarrow j}(\mathbf{s}|\mathbf{t}, 2)))) \quad (\text{B.29})$$

这里 $\tilde{b}(\mathbf{a})$ 借用了模型 4 中 $\tilde{b}(\mathbf{a})$ 的概念。不过 $\tilde{b}(\mathbf{a})$ 表示在利用模型 3 进行排名的列表中满足 $P_\theta(\mathbf{a}'|\mathbf{s}, \mathbf{t}; 5)$ 的最高排名的词对齐。



Bibliography

- [1] Alfred V. Aho and Jeffrey D. Ullman. *The theory of parsing, translation, and compiling. 2: Compiling*. Prentice-Hall, 1973 (cited on page 83).
- [2] Felipe Almeida and Geraldo Xexéo. “Word Embeddings: A Survey”. In: *CoRR abs/1901.09069* (2019) (cited on page 393).
- [3] Mikel Artetxe, Gorka Labaka, and Eneko Agirre. “An Effective Approach to Unsupervised Machine Translation”. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Association for Computational Linguistics, 2019, pages 194–203 (cited on page 402).
- [4] Philip Arthur, Graham Neubig, and Satoshi Nakamura. “Incorporating Discrete Translation Lexicons into Neural Machine Translation”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*. The Association for Computational Linguistics, 2016, pages 1557–1567 (cited on page 274).
- [5] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. “Layer Normalization”. In: *CoRR abs/1607.06450* (2016) (cited on pages 253, 330, 383).
- [6] Alexei Baevski and Michael Auli. “Adaptive Input Representations for Neural Language Modeling”. In: *7th International Conference on Learning Representations*,

- ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019 (cited on page 363).
- [7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015 (cited on pages 30, 40, 278, 302, 317).
 - [8] B Bangalore, German Bordel, and Giuseppe Riccardi. “Computing consensus translation from multiple machine translation systems”. In: *IEEE Workshop on Automatic Speech Recognition and Understanding, 2001. ASRU’01*. IEEE. 2001, pages 351–354 (cited on page 204).
 - [9] Sameer Bansal, Herman Kamper, Karen Livescu, Adam Lopez, and Sharon Goldwater. “Pre-training on high-resource speech recognition improves low-resource speech-to-text translation”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 2019, pages 58–68 (cited on page 403).
 - [10] Ankur Bapna, Mia Xu Chen, Orhan Firat, Yuan Cao, and Yonghui Wu. “Training Deeper Neural Machine Translation Models with Transparent Attention”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 2018, pages 3028–3033 (cited on pages 382, 385).
 - [11] A. Barbour and Sidney Resnick. “Adventures in Stochastic Processes.” In: *Journal of the American Statistical Association* 88 (Dec. 1993), page 1474 (cited on page 68).
 - [12] Atılım Günes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. “Automatic differentiation in machine learning: a survey”. In: *The Journal of Machine Learning Research* 18.1 (2017), pages 5595–5637 (cited on page 244).
 - [13] Y. Bengio, P. Simard, and P. Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pages 157–166 (cited on page 278).
 - [14] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. “A neural probabilistic language model”. In: *Journal of machine learning research* 3.Feb (2003), pages 1137–1155 (cited on pages 83, 213, 262).

- [15] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. “Curriculum learning”. In: *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*. Volume 382. ACM International Conference Proceeding Series. ACM, 2009, pages 41–48 (cited on page 365).
- [16] Luisa Bentivogli, Arianna Bisazza, Mauro Cettolo, and Marcello Federico. “Neural versus Phrase-Based Machine Translation Quality: a Case Study”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*. The Association for Computational Linguistics, 2016, pages 257–267 (cited on page 281).
- [17] Alexandre Berard, Laurent Besacier, Ali Can Kocabiyikoglu, and Olivier Pietquin. “End-to-End Automatic Speech Translation of Audiobooks”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*. IEEE, 2018, pages 6224–6228 (cited on page 403).
- [18] Alexandre Berard, Olivier Pietquin, Christophe Servan, and Laurent Besacier. “Listen and Translate: A Proof of Concept for End-to-End Speech-to-Text Translation”. In: *CoRR abs/1612.01744* (2016) (cited on page 403).
- [19] Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. “Efficient 8-Bit Quantization of Transformer Neural Machine Language Translation Model”. In: *CoRR abs/1906.00532* (2019) (cited on pages 339, 373).
- [20] Steven Bird. “NLTK: The Natural Language Toolkit”. In: *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17-21 July 2006*. The Association for Computer Linguistics, 2006 (cited on page 345).
- [21] Phil Blunsom, Trevor Cohn, Chris Dyer, and Miles Osborne. “A Gibbs Sampler for Phrasal Synchronous Grammar Induction”. In: *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore*. The Association for Computer Linguistics, 2009, pages 782–790 (cited on page 203).
- [22] Phil Blunsom, Trevor Cohn, and Miles Osborne. “A Discriminative Latent Variable Model for Statistical Machine Translation”. In: *ACL 2008, Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, June 15-20, 2008*,

- Columbus, Ohio, USA*. The Association for Computer Linguistics, 2008, pages 200–208 (cited on page 203).
- [23] Brown, Peter F, Desouza, Peter V, Mercer and Robert L, Pietra, Vincent J Della, Lai, and Jenifer C. “Class-based n-gram models of natural language”. In: *Computational linguistics* 18.4 (1992), pages 467–479 (cited on page 213).
- [24] Peter F. Brown, John Cocke, Stephen Della Pietra, Vincent J. Della Pietra, Frederick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. “A Statistical Approach to Machine Translation”. In: *Computational Linguistics* 16.2 (1990), pages 79–85 (cited on pages 29, 102, 103).
- [25] Peter F. Brown, Stephen Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. “The Mathematics of Statistical Machine Translation: Parameter Estimation”. In: *Computational Linguistics* 19.2 (1993), pages 263–311 (cited on pages 87, 122, 203).
- [26] Ozan Caglayan, Mercedes García-Martínez, Adrien Bardet, Walid Aransa, Fethi Bougares, and Loïc Barrault. “NMTPY: A Flexible Toolkit for Advanced Neural Machine Translation Systems”. In: *Prague Bull. Math. Linguistics* 109 (2017), pages 15–28 (cited on page 41).
- [27] Francisco Casacuberta and Enrique Vidal. “Machine Translation with Inferred Stochastic Finite-State Transducers”. In: *Computational Linguistics* 30.2 (2004), pages 205–225 (cited on page 204).
- [28] Daniel M. Cer, Michel Galley, Daniel Jurafsky, and Christopher D. Manning. “Phrasal: A Statistical Machine Translation Toolkit for Exploring New Model Features”. In: *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 2, 2010, Los Angeles, California, USA - Demonstration Session*. The Association for Computational Linguistics, 2010, pages 9–12 (cited on page 39).
- [29] Eugene Charniak. “Immediate-Head Parsing for Language Models”. In: *Association for Computational Linguistic, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference, July 9-11, 2001, Toulouse, France*. Morgan Kaufmann Publishers, 2001, pages 116–123 (cited on page 204).
- [30] Eugene Charniak, Mark Johnson, Micha Elsner, Joseph Austerweil, David Ellis, Isaac Haxton, Catherine Hill, R. Shrivaths, Jeremy Moore, Michael Pozar, and Theresa Vu. “Multilevel Coarse-to-Fine PCFG Parsing”. In: *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*. 2006, pages 168–175 (cited on page 192).

- [31] Eugene Charniak, Kevin Knight, and Kenji Yamada. “Syntax-based Language Models for Statistical Machine Translation”. In: *In MT Summit IX. Intl. Assoc. for Machine Translation*. 2003 (cited on page 274).
- [32] Danqi Chen and Christopher D. Manning. “A Fast and Accurate Dependency Parser using Neural Networks”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2014, pages 740–750 (cited on page 83).
- [33] Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. “Improved Neural Machine Translation with a Syntax-Aware Encoder and Decoder”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*. Association for Computational Linguistics, 2017, pages 1936–1945 (cited on page 403).
- [34] Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George F. Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. “The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*. Association for Computational Linguistics, 2018, pages 76–86 (cited on pages 338, 371).
- [35] Stanley F. Chen and Joshua Goodman. “An empirical study of smoothing techniques for language modeling”. In: *Computer Speech & Language* 13.4 (1999), pages 359–393 (cited on pages 72, 74).
- [36] Yun Chen, Yang Liu, Yong Cheng, and Victor O. K. Li. “A Teacher-Student Framework for Zero-Resource Neural Machine Translation”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*. Association for Computational Linguistics, 2017, pages 1925–1935 (cited on page 339).
- [37] David Chiang. “A Hierarchical Phrase-Based Model for Statistical Machine Translation”. In: *ACL 2005, 43rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, 25-30 June 2005, University of Michigan, USA*. The Association for Computer Linguistics, 2005, pages 263–270 (cited on page 163).
- [38] David Chiang. “Hierarchical Phrase-Based Translation”. In: *Computational Linguistics* 33.2 (2007), pages 201–228 (cited on pages 138, 163, 178).

- [39] David Chiang. “Learning to Translate with Source and Target Syntax”. In: *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden*. The Association for Computer Linguistics, 2010, pages 1443–1452 (cited on page 178).
- [40] David Chiang. “Hope and Fear for Discriminative Training of Statistical Translation Models”. In: *Journal of Machine Learning Research* 13 (2012), pages 1159–1187 (cited on pages 160, 161).
- [41] David Chiang, Kevin Knight, and Wei Wang. “11,001 New Features for Statistical Machine Translation”. In: *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, May 31 - June 5, 2009, Boulder, Colorado, USA*. The Association for Computational Linguistics, 2009, pages 218–226 (cited on page 203).
- [42] David Chiang, Yuval Marton, and Philip Resnik. “Online Large-Margin Training of Syntactic and Structural Translation Features”. In: *2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2008, pages 224–233 (cited on page 155).
- [43] David Chiang, Yuval Marton, and Philip Resnik. “Online Large-Margin Training of Syntactic and Structural Translation Features”. In: *2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2008, pages 224–233 (cited on page 203).
- [44] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2014, pages 1724–1734 (cited on page 298).
- [45] J. Cocke and J.T. Schwartz. *Programming Languages and Their Compilers: Preliminary Notes*. Courant Institute of Mathematical Sciences, New York University, 1970 (cited on page 169).
- [46] Trevor Cohn and Phil Blunsom. “A Bayesian Model of Syntax-Directed Tree to String Grammar Induction”. In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, EMNLP 2009, 6-7 August 2009, Singapore, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2009, pages 352–361 (cited on page 203).

- [47] *Coling 2008: Advanced Dynamic Programming in Computational Linguistics: Theory, Algorithms and Applications - Tutorial notes*. Manchester, UK: Coling 2008 Organizing Committee, 2008 (cited on page 83).
- [48] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. “Natural Language Processing (Almost) from Scratch”. In: *Journal of Machine Learning Research* 12 (2011), pages 2493–2537 (cited on page 274).
- [49] Alexis Conneau and Guillaume Lample. “Cross-lingual Language Model Pretraining”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*. 2019, pages 7057–7067 (cited on pages 274, 402).
- [50] Matthieu Courbariaux and Yoshua Bengio. “BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1”. In: *CoRR abs/1602.02830* (2016) (cited on pages 334, 339).
- [51] Anna Currey, Antonio Valerio Miceli Barone, and Kenneth Heafield. “Copied Monolingual Data Improves Low-Resource Neural Machine Translation”. In: *Proceedings of the Second Conference on Machine Translation, WMT 2017, Copenhagen, Denmark, September 7-8, 2017*. Association for Computational Linguistics, 2017, pages 148–156 (cited on page 391).
- [52] Anna Currey and Kenneth Heafield. “Multi-Source Syntactic Neural Machine Translation”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 2018, pages 2961–2966 (cited on page 403).
- [53] Raj Dabre and Atsushi Fujita. “Recurrent Stacking of Layers for Compact Neural Machine Translation Models”. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pages 6292–6299 (cited on page 370).
- [54] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. “Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context”. In: *CoRR abs/1901.02860* (2019) (cited on page 339).
- [55] John Sturdy DeNero. “Phrase Alignment Models for Statistical Machine Translation”. In: *Ph.D. thesis, UC Berkeley* (2010) (cited on page 204).

- [56] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 2019, pages 4171–4186 (cited on pages 272, 363, 368).
- [57] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard M. Schwartz, and John Makhoul. “Fast and Robust Neural Network Joint Models for Statistical Machine Translation”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*. The Association for Computer Linguistics, 2014, pages 1370–1380 (cited on page 278).
- [58] Tobias Domhan and Felix Hieber. “Using Target-side Monolingual Data for Neural Machine Translation through Multi-task Learning”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. Association for Computational Linguistics, 2017, pages 1500–1505 (cited on page 395).
- [59] Miguel Domingo, álvaro Peris, and Francisco Casacuberta. “Segment-based interactive-predictive machine translation”. In: *Machine Translation 31.4* (2017), pages 163–185 (cited on page 375).
- [60] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. “Multi-Task Learning for Multiple Language Translation”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, July 2015, pages 1723–1732 (cited on page 404).
- [61] Zi-Yi Dou, Zhaopeng Tu, Xing Wang, Shuming Shi, and Tong Zhang. “Exploiting Deep Representations for Neural Machine Translation”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 2018, pages 4253–4262 (cited on page 387).
- [62] Markus Dreyer and Yuanzhe Dong. “APRO: All-Pairs Ranking Optimization for MT Tuning”. In: *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*. The Association for Computational Linguistics, 2015, pages 1018–1023 (cited on page 203).

- [63] Nan Duan, Mu Li, Tong Xiao, and Ming Zhou. “The Feature Subspace Method for SMT System Combination”. In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, EMNLP 2009, 6-7 August 2009, Singapore, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2009, pages 1096–1104 (cited on page 379).
- [64] John C. Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *Journal of Machine Learning Research* 12 (2011), pages 2121–2159 (cited on page 248).
- [65] Long Duong, Antonios Anastasopoulos, David Chiang, Steven Bird, and Trevor Cohn. “An Attentional Model for Speech Translation Without Transcription”. In: *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*. The Association for Computational Linguistics, 2016, pages 949–959 (cited on page 403).
- [66] Chris Dyer, Victor Chahuneau, and Noah A. Smith. “A Simple, Fast, and Effective Reparameterization of IBM Model 2”. In: *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*. The Association for Computational Linguistics, 2013, pages 644–648 (cited on pages 40, 146).
- [67] Chris Dyer, Adam Lopez, Juri Ganitkevitch, Jonathan Weese, Ferhan Türe, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. “cdec: A Decoder, Alignment, and Learning Framework for Finite-State and Context-Free Translation Models”. In: *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden, System Demonstrations*. The Association for Computer Linguistics, 2010, pages 7–12 (cited on page 39).
- [68] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. “Understanding Back-Translation at Scale”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 2018, pages 489–500 (cited on page 391).
- [69] Jason Eisner. “Parameter Estimation for Probabilistic Finite-State Transducers”. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL, 2002, pages 1–8 (cited on page 197).

- [70] Jason Eisner. “Learning Non-Isomorphic Tree Mappings for Machine Translation”. In: *ACL 2003, 41st Annual Meeting of the Association for Computational Linguistics, Companion Volume to the Proceedings, 7-12 July 2003, Sapporo Convention Center, Sapporo, Japan*. The Association for Computer Linguistics, 2003, pages 205–208 (cited on page 178).
- [71] Desmond Elliott, Stella Frank, and Eva Hasler. “Multi-Language Image Description with Neural Sequence Models”. In: *CoRR abs/1510.04709* (2015). arXiv: 1510.04709 (cited on page 403).
- [72] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. “Neural Architecture Search: A Survey”. In: *Journal of Machine Learning Research* 20.55 (2019), pages 1–21 (cited on page 404).
- [73] Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. “Tree-to-Sequence Attentional Neural Machine Translation”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016 (cited on page 403).
- [74] Ethayarajh and Kawin. “How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Nov. 2019, pages 55–65 (cited on page 362).
- [75] Yang Feng, Yang Liu, Haitao Mi, Qun Liu, and Yajuan Lü. “Lattice-based System Combination for Statistical Machine Translation”. In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, EMNLP 2009, 6-7 August 2009, Singapore, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2009, pages 1105–1113 (cited on page 204).
- [76] Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. “Multi-Way, Multilingual Neural Machine Translation with a Shared Attention Mechanism”. In: *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*. The Association for Computational Linguistics, 2016, pages 866–875 (cited on page 404).
- [77] Orhan Firat, Baskaran Sankaran, Yaser Al-onaizan, Fatos T. Yarman Vural, and Kyunghyun Cho. “Zero-Resource Translation with Multi-Lingual Neural Machine Translation”. In: *Proceedings of the 2016 Conference on Empirical Methods in Nat-*

- ural Language Processing*. Association for Computational Linguistics, Nov. 2016, pages 268–277 (cited on page 404).
- [78] Yoav Freund and Robert E. Schapire. “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”. In: *Journal of Computer and System Sciences* 55.1 (1997), pages 119–139 (cited on page 378).
- [79] Philip Gage. “A New Algorithm for Data Compression”. In: *The C Users Journal archive* 12.2 (1994), pages 23–38 (cited on page 352).
- [80] William A. Gale and Geoffrey Sampson. “Good-Turing Frequency Estimation Without Tears”. In: *Journal of Quantitative Linguistics* 2.3 (1995), pages 217–237 (cited on page 71).
- [81] Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. “Scalable Inference and Training of Context-Rich Syntactic Translation Models”. In: *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17-21 July 2006*. The Association for Computer Linguistics, 2006 (cited on pages 178, 184).
- [82] Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. “What’s in a translation rule?” In: *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*. 2004, pages 273–280 (cited on pages 178, 184).
- [83] Mercedes Garcia-Martinez, Loïc Barrault, and Fethi Bougares. “Factored Neural Machine Translation Architectures”. In: *International Workshop on Spoken Language Translation (IWSLT’16)*. 2016 (cited on page 350).
- [84] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. “Convolutional Sequence to Sequence Learning”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Volume 70. Proceedings of Machine Learning Research. PMLR, 2017, pages 1243–1252 (cited on pages 279, 338).
- [85] Daniel Gildea. “Loosely Tree-Based Alignment for Machine Translation”. In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, 7-12 July 2003, Sapporo Convention Center, Sapporo, Japan*. ACL, 2003, pages 80–87 (cited on page 203).

- [86] Adrià de Gispert, Gonzalo Iglesias, Graeme W. Blackwood, Eduardo Rodríguez Banga, and William Byrne. “Hierarchical Phrase-Based Translation with Weighted Finite-State Transducers and Shallow- n Grammars”. In: *Computational Linguistics* 36.3 (2010), pages 505–533 (cited on page 204).
- [87] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*. Volume 9. JMLR Proceedings. JMLR.org, 2010, pages 249–256 (cited on page 308).
- [88] Yoav Goldberg. “Neural network methods for natural language processing”. In: *Synthesis Lectures on Human Language Technologies* 10.1 (2017), pages 1–309 (cited on pages 45, 274).
- [89] Irving J Good. “The population frequencies of species and the estimation of population parameters”. In: *Biometrika* 40.3-4 (1953), pages 237–264 (cited on page 71).
- [90] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016 (cited on pages 45, 274).
- [91] Joshua Goodman. “Semiring Parsing”. In: *Computational Linguistics* 25.4 (1999), pages 573–605 (cited on page 197).
- [92] Klaus Greff, Rupesh Kumar Srivastava, and Jürgen Schmidhuber. “Highway and Residual Networks learn Unrolled Iterative Estimation”. In: *CoRR* abs/1612.07771 (2016) (cited on page 390).
- [93] Jerneja Žganec Gros. “MSD Recombination Method in Statistical Machine Translation”. In: *INTERNATIONAL ELECTRONIC CONFERENCE ON COMPUTER SCIENCE 28 June - 8 July 2007 and 20 November - 10 December 2007*. Volume 1060. AIP Conference Proceedings. American Institute of Physics, 2008, pages 186–189 (cited on page 149).
- [94] Declan Groves, Mary Hearne, and Andy Way. “Robust Sub-Sentential Alignment of Phrase-Structure Trees”. In: *COLING 2004, 20th International Conference on Computational Linguistics, Proceedings of the Conference, 23-27 August 2004, Geneva, Switzerland*. 2004 (cited on page 193).
- [95] Jetic Gu, Hassan S. Shavarani, and Anoop Sarkar. “Top-down Tree Structured Decoding with Syntactic Connections for Neural Machine Translation and Parsing”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 2018, pages 401–413 (cited on page 339).

- [96] Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. “Non-Autoregressive Neural Machine Translation”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018 (cited on pages 314, 374).
- [97] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Gianotti, and Dino Pedreschi. “A Survey of Methods for Explaining Black Box Models”. In: *ACM Computing Surveys* 51.5 (2019), 93:1–93:42 (cited on page 274).
- [98] Thanh-Le Ha, Jan Niehues, and Alexander H. Waibel. “Toward Multilingual Neural Machine Translation with Universal Encoder and Decoder”. In: *CoRR* abs/1611.04798 (2016) (cited on page 404).
- [99] Hany Hassan, Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Xuedong Huang, Marcin Junczys-Dowmunt, William Lewis, Mu Li, Shujie Liu, Tie-Yan Liu, Renqian Luo, Arul Menezes, Tao Qin, Frank Seide, Xu Tan, Fei Tian, Lijun Wu, Shuangzhi Wu, Yingce Xia, Dongdong Zhang, Zhirui Zhang, and Ming Zhou. “Achieving Human Parity on Automatic Chinese to English News Translation”. In: *CoRR* abs/1803.05567 (2018) (cited on pages 281, 400).
- [100] Di He, Yingce Xia, Tao Qin, Liwei Wang, Nenghai Yu, Tie-Yan Liu, and Wei-Ying Ma. “Dual Learning for Machine Translation”. In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016, pages 820–828 (cited on page 401).
- [101] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pages 770–778 (cited on pages 253, 329, 362).
- [102] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Identity Mappings in Deep Residual Networks”. In: *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*. Volume 9908. Springer, 2016, pages 630–645 (cited on page 383).
- [103] Qiuxiang He, Guoping Huang, Lemao Liu, and Li Li. “Word Position Aware Translation Memory for Neural Machine Translation”. In: *Natural Language Processing and Chinese Computing - 8th CCF International Conference, NLPCC 2019, Dunhuang, China, October 9-14, 2019, Proceedings, Part I*. Volume 11838. Lecture Notes in Computer Science. Springer, 2019, pages 367–379 (cited on page 404).

- [104] Wei He, Zhongjun He, Hua Wu, and Haifeng Wang. “Improved Neural Machine Translation with SMT Features”. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. AAAI Press, 2016, pages 151–157 (cited on page 404).
- [105] Xiaodong He, Mei Yang, Jianfeng Gao, Patrick Nguyen, and Robert Moore. “Indirect-HMM-based Hypothesis Alignment for Combining Outputs from Machine Translation Systems”. In: *2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2008, pages 98–107 (cited on page 204).
- [106] Kenneth Heafield. “KenLM: Faster and Smaller Language Model Queries”. In: *Proceedings of the Sixth Workshop on Statistical Machine Translation, WMT@EMNLP 2011, Edinburgh, Scotland, UK, July 30-31, 2011*. Association for Computational Linguistics, 2011, pages 187–197 (cited on page 74).
- [107] Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. “Sockeye: A Toolkit for Neural Machine Translation”. In: *CoRR abs/1712.05690* (2017) (cited on page 41).
- [108] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *CoRR abs/1207.0580* (2012) (cited on page 358).
- [109] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. “Distilling the Knowledge in a Neural Network”. In: *CoRR abs/1503.02531* (2015) (cited on pages 339, 396, 397).
- [110] Julian Hitschler, Shigehiko Schamoni, and Stefan Riezler. “Multimodal Pivots for Image Caption Translation”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016 (cited on page 403).
- [111] Sepp Hochreiter. “The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.2 (1998), pages 107–116 (cited on page 278).
- [112] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-term Memory”. In: *Neural computation* 9 (Dec. 1997), pages 1735–80 (cited on page 295).

- [113] Mark Hopkins and Jonathan May. “Tuning as Ranking”. In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2011, pages 1352–1362 (cited on pages 155, 203).
- [114] K Hornik. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks 2.5* (1989), pages 359–366 (cited on page 360).
- [115] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. “Densely Connected Convolutional Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pages 2261–2269 (cited on page 387).
- [116] Liang Huang and David Chiang. “Better k-best Parsing”. In: *Proceedings of the Ninth International Workshop on Parsing Technology, IWPT 2005, Vancouver, Canada, October 9-10, 2005*. Association for Computational Linguistics, 2005, pages 53–64 (cited on page 172).
- [117] Liang Huang, Kevin Knight, and Aravind Joshi. “Statistical syntax-directed translation with extended domain of locality”. In: *Proceedings of AMTA*. Cambridge, MA. 2006, pages 66–73 (cited on page 178).
- [118] Liang Huang, Kai Zhao, and Mingbo Ma. “When to Finish? Optimal Beam Search for Neural Text Generation (modulo beam size)”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. Association for Computational Linguistics, 2017, pages 2134–2139 (cited on page 375).
- [119] Po-Yao Huang, Frederick Liu, Sz-Rung Shiang, Jean Oh, and Chris Dyer. “Attention-based Multimodal Neural Machine Translation”. In: *Proceedings of the First Conference on Machine Translation, WMT 2016, colocated with ACL 2016, August 11-12, Berlin, Germany*. The Association for Computer Linguistics, 2016, pages 639–645 (cited on page 403).
- [120] Zhiheng Huang, Wei Xu, and Kai Yu. “Bidirectional LSTM-CRF Models for Sequence Tagging”. In: *CoRR abs/1508.01991* (2015) (cited on page 83).
- [121] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. “Binarized Neural Networks”. In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016, pages 4107–4115 (cited on page 373).

- [122] Gonzalo Iglesias, Adrià de Gispert, Eduardo Rodríguez Banga, and William J. Byrne. “Hierarchical Phrase-Based Translation with Weighted Finite State Transducers”. In: *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, May 31 - June 5, 2009, Boulder, Colorado, USA*. The Association for Computational Linguistics, 2009, pages 433–441 (cited on page 39).
- [123] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Volume 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pages 448–456 (cited on page 253).
- [124] Abraham Ittycheriah and Salim Roukos. “A Maximum Entropy Word Aligner for Arabic-English Machine Translation”. In: *HLT/EMNLP 2005, Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference, 6-8 October 2005, Vancouver, British Columbia, Canada*. The Association for Computational Linguistics, 2005, pages 89–96 (cited on page 131).
- [125] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, 2018, pages 2704–2713 (cited on page 373).
- [126] Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. “What Does BERT Learn about the Structure of Language?” In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Association for Computational Linguistics, 2019, pages 3651–3657 (cited on page 362).
- [127] Sébastien Jean, KyungHyun Cho, Roland Memisevic, and Yoshua Bengio. “On Using Very Large Target Vocabulary for Neural Machine Translation”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*. The Association for Computer Linguistics, 2015, pages 1–10 (cited on pages 350, 369).

- [128] Sébastien Jean, Stanislas Lauly, Orhan Firat, and Kyunghyun Cho. “Does Neural Machine Translation Benefit from Larger Context?” In: *CoRR* abs/1704.05135 (2017) (cited on page 403).
- [129] Ye Jia, Melvin Johnson, Wolfgang Macherey, Ron J. Weiss, Yuan Cao, Chung-Cheng Chiu, Naveen Ari, Stella Laurenzo, and Yonghui Wu. “Leveraging Weakly Supervised Data to Improve End-to-end Speech-to-text Translation”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019, Brighton, United Kingdom, May 12-17, 2019*. IEEE, 2019, pages 7180–7184 (cited on page 403).
- [130] Yufan Jiang, Chi Hu, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. “Improved Differentiable Architecture Search for Language Modeling and Named Entity Recognition”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Association for Computational Linguistics, 2019, pages 3583–3588 (cited on page 404).
- [131] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. “TinyBERT: Distilling BERT for Natural Language Understanding”. In: *CoRR* abs/1909.10351 (2019) (cited on page 398).
- [132] Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda B. Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. “Google’s Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation”. In: *Transactions of the Association for Computational Linguistics* 5 (2017), pages 339–351 (cited on pages 403, 404).
- [133] Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Hermann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. “Marian: Fast Neural Machine Translation in C++”. In: *Proceedings of ACL 2018, Melbourne, Australia, July 15-20, 2018, System Demonstrations*. Association for Computational Linguistics, 2018, pages 116–121 (cited on page 41).
- [134] Dan Jurafsky and James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, 2009 (cited on page 74).
- [135] Nal Kalchbrenner and Phil Blunsom. “Recurrent Continuous Translation Models”. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Wash-*

- ington, USA, *A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2013, pages 1700–1709 (cited on page 278).
- [136] Takatomo Kano, Sakriani Sakti, and Satoshi Nakamura. “Structured-Based Curriculum Learning for End-to-End English-Japanese Speech Translation”. In: *Interspeech 2017, 18th Annual Conference of the International Speech Communication Association, Stockholm, Sweden, August 20-24, 2017*. ISCA, 2017, pages 2630–2634 (cited on page 403).
- [137] Tadao Kasami. “An efficient recognition and syntax-analysis algorithm for context-free languages”. In: *Coordinated Science Laboratory Report no. R-257* (1966) (cited on page 169).
- [138] Huda Khayrallah, Gaurav Kumar, Kevin Duh, Matt Post, and Philipp Koehn. “Neural Lattice Search for Domain Adaptation in Machine Translation”. In: *Proceedings of the Eighth International Joint Conference on Natural Language Processing, IJCNLP 2017, Taipei, Taiwan, November 27 - December 1, 2017, Volume 2: Short Papers*. Asian Federation of Natural Language Processing, 2017, pages 20–25 (cited on page 404).
- [139] Yoon Kim and Alexander M. Rush. “Sequence-Level Knowledge Distillation”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*. The Association for Computational Linguistics, 2016, pages 1317–1327 (cited on page 397).
- [140] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015 (cited on page 249).
- [141] Dan Klein and Christopher D. Manning. “Parsing and Hypergraphs”. In: *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT-2001), 17-19 October 2001, Beijing, China*. Tsinghua University Press, 2001 (cited on page 196).
- [142] Dan Klein and Christopher D. Manning. “Accurate Unlexicalized Parsing”. In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, 7-12 July 2003, Sapporo Convention Center, Sapporo, Japan*. Edited by Erhard W. Hinrichs and Dan Roth. ACL, 2003, pages 423–430 (cited on page 192).
- [143] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. “OpenNMT: Open-Source Toolkit for Neural Machine Translation”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics,*

- ACL 2017, Vancouver, Canada, July 30 - August 4, System Demonstrations*. Association for Computational Linguistics, 2017, pages 67–72 (cited on pages 40, 383).
- [144] Reinhard Kneser and Hermann Ney. “Improved backing-off for M-gram language modeling”. In: *1995 International Conference on Acoustics, Speech, and Signal Processing, ICASSP '95, Detroit, Michigan, USA, May 08-12, 1995*. IEEE Computer Society, 1995, pages 181–184 (cited on page 72).
- [145] Kevin Knight. “Decoding Complexity in Word-Replacement Translation Models”. In: *Computational Linguistics* 25.4 (1999), pages 607–615 (cited on pages 100, 155, 366).
- [146] Kevin Knight. “Learning a translation lexicon from monolingual corpora”. In: *In Proceedings of ACL Workshop on Unsupervised Lexical Acquisition*. 2002, pages 9–16 (cited on page 148).
- [147] Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010 (cited on page 44).
- [148] Philipp Koehn. “Neural Machine Translation”. In: *CoRR* abs/1709.07809 (2017) (cited on page 44).
- [149] Philipp Koehn and Hieu Hoang. “Factored Translation Models”. In: *EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic*. ACL, 2007, pages 868–876 (cited on page 146).
- [150] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. “Moses: Open Source Toolkit for Statistical Machine Translation”. In: *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*. The Association for Computational Linguistics, 2007 (cited on pages 38, 345).
- [151] Philipp Koehn and Kevin Knight. “Estimating Word Translation Probabilities from Unrelated Monolingual Corpora Using the EM Algorithm”. In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA*. AAAI Press / The MIT Press, 2000, pages 711–715 (cited on page 146).

- [152] Philipp Koehn and Rebecca Knowles. “Six Challenges for Neural Machine Translation”. In: *Proceedings of the First Workshop on Neural Machine Translation, NMT@ACL 2017, Vancouver, Canada, August 4, 2017*. Association for Computational Linguistics, 2017, pages 28–39 (cited on page 404).
- [153] Philipp Koehn, Franz Josef Och, and Daniel Marcu. “Statistical Phrase-Based Translation”. In: *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL 2003, Edmonton, Canada, May 27 - June 1, 2003*. The Association for Computational Linguistics, 2003 (cited on pages 29, 131, 138, 203).
- [154] Pang Wei Koh and Percy Liang. “Understanding Black-box Predictions via Influence Functions”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Volume 70. Proceedings of Machine Learning Research. PMLR, 2017, pages 1885–1894 (cited on page 274).
- [155] Andre Nikolaevich Kolmogorov and Albert T Bharucha-Reid. *Foundations of the theory of probability: Second English Edition*. Courier Dover Publications, 2018 (cited on page 52).
- [156] Shaohui Kuang, Deyi Xiong, Weihua Luo, and Guodong Zhou. “Modeling Coherence for Neural Machine Translation with Dynamic and Topic Caches”. In: *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*. Association for Computational Linguistics, 2018, pages 596–606 (cited on page 403).
- [157] Oleksii Kuchaiev, Boris Ginsburg, Igor Gitman, Vitaly Lavrukhin, Carl Case, and Paulius Micikevicius. “OpenSeq2Seq: extensible toolkit for distributed and mixed precision training of sequence-to-sequence models”. In: *CoRR abs/1805.10387 (2018)* (cited on page 41).
- [158] Taku Kudo. “Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*. Association for Computational Linguistics, 2018, pages 66–75 (cited on pages 353, 354).
- [159] Shankar Kumar and William J. Byrne. “Local Phrase Reordering Models for Statistical Machine Translation”. In: *HLT/EMNLP 2005, Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference, 6-8 October 2005, Vancouver, British Columbia,*

- Canada. The Association for Computational Linguistics, 2005, pages 161–168. URL: <https://www.aclweb.org/anthology/H05-1021/> (cited on page 151).
- [160] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data”. In: *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*. Morgan Kaufmann, 2001, pages 282–289 (cited on page 83).
- [161] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations”. In: *CoRR* abs/1909.11942 (2019) (cited on page 274).
- [162] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pages 436–444 (cited on page 360).
- [163] Jason Lee, Kyunghyun Cho, and Thomas Hofmann. “Fully Character-Level Neural Machine Translation without Explicit Segmentation”. In: *Trans. Assoc. Comput. Linguistics* 5 (2017), pages 365–378 (cited on page 351).
- [164] Tao Lei, Yu Zhang, and Yoav Artzi. “Training RNNs as Fast as CNNs”. In: *CoRR* abs/1709.02755 (2017) (cited on page 338).
- [165] Bei Li, Yinqiao Li, Chen Xu, Ye Lin, Jiqiang Liu, Hui Liu, Ziyang Wang, Yuhao Zhang, Nuo Xu, Zeyang Wang, Kai Feng, Hexuan Chen, Tengbo Liu, Yanyang Li, Qiang Wang, Tong Xiao, and Jingbo Zhu. “The NiuTrans Machine Translation Systems for WMT19”. In: *Proceedings of the Fourth Conference on Machine Translation, WMT 2019, Florence, Italy, August 1-2, 2019 - Volume 2: Shared Task Papers, Day 1*. Association for Computational Linguistics, 2019, pages 257–266 (cited on pages 348, 368).
- [166] Chi-Ho Li, Xiaodong He, Yupeng Liu, and Ning Xi. “Incremental HMM Alignment for MT System Combination”. In: *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore*. The Association for Computer Linguistics, 2009, pages 949–957 (cited on page 204).
- [167] Jiwei Li, Will Monroe, and Dan Jurafsky. “A Simple, Fast Diverse Decoding Algorithm for Neural Generation”. In: *CoRR* abs/1611.08562 (2016) (cited on page 379).

- [168] Mu Li, Nan Duan, Dongdong Zhang, Chi-Ho Li, and Ming Zhou. “Collaborative Decoding: Partial Hypothesis Re-ranking Using Translation Consensus between Decoders”. In: *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore*. The Association for Computer Linguistics, 2009, pages 585–592 (cited on page 204).
- [169] Peng Li, Yang Liu, Maosong Sun, Tatsuya Izuka, and Dakun Zhang. “A Neural Reordering Model for Phrase-based Translation”. In: *COLING 2014, 25th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, August 23-29, 2014, Dublin, Ireland*. ACL, 2014, pages 1897–1907 (cited on page 151).
- [170] Xintong Li, Guanlin Li, Lemao Liu, Max Meng, and Shuming Shi. “On the Word Alignment from Neural Machine Translation”. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Association for Computational Linguistics, 2019, pages 1293–1303 (cited on page 339).
- [171] Yanyang Li, Qiang Wang, Tong Xiao, Tongran Liu, and Jingbo Zhu. “Neural Machine Translation with Joint Representation”. In: *CoRR abs/2002.06546* (2020) (cited on page 285).
- [172] Yanyang Li, Tong Xiao, Yinqiao Li, Qiang Wang, Changming Xu, and Jingbo Zhu. “A Simple and Effective Approach to Coverage-Aware Neural Machine Translation”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*. Association for Computational Linguistics, 2018, pages 292–297 (cited on pages 316, 378).
- [173] Yinqiao Li, Chi Hu, Yuhao Zhang, Nuo Xu, Yufan Jiang, Tong Xiao, Jingbo Zhu, Tongran Liu, and Changliang Li. “Learning Architectures from an Extended Search Space for Language Modeling”. In: *Association for Computational Linguistics, 2020* (cited on page 404).
- [174] Zhifei Li, Chris Callison-Burch, Chris Dyer, Sanjeev Khudanpur, Lane Schwartz, Wren N. G. Thornton, Jonathan Weese, and Omar Zaidan. “Joshua: An Open Source Toolkit for Parsing-Based Machine Translation”. In: *Proceedings of the Fourth Workshop on Statistical Machine Translation, WMT@EACL 2009, Athens, Greece, March 30-31, 2009*. Association for Computational Linguistics, 2009, pages 135–139 (cited on page 39).

- [175] Zhifei Li and Jason Eisner. “First- and Second-Order Expectation Semirings with Applications to Minimum-Risk Training on Translation Forests”. In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, EMNLP 2009, 6-7 August 2009, Singapore, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2009, pages 40–51 (cited on page 203).
- [176] Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joseph E. Gonzalez. “Train Large, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers”. In: *CoRR* abs/2002.11794 (2020) (cited on page 396).
- [177] J. Lin. “Divergence measures based on the Shannon entropy”. In: *IEEE Transactions on Information Theory* 37.1 (1991), pages 145–151 (cited on page 369).
- [178] Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. “A Structured Self-Attentive Sentence Embedding”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017 (cited on page 320).
- [179] Shikun Liu, Edward Johns, and Andrew J. Davison. “End-To-End Multi-Task Learning With Attention”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pages 1871–1880 (cited on page 339).
- [180] Yang Liu, Qun Liu, and Shouxun Lin. “Tree-to-String Alignment Template for Statistical Machine Translation”. In: *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17-21 July 2006*. The Association for Computer Linguistics, 2006 (cited on page 178).
- [181] Yang Liu, Yajuan Lü, and Qun Liu. “Improving Tree-to-Tree Translation with Packed Forests”. In: *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore*. The Association for Computer Linguistics, 2009, pages 558–566 (cited on pages 178, 192).
- [182] Yang Liu, Haitao Mi, Yang Feng, and Qun Liu. “Joint Decoding with Multiple Translation Models”. In: *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore*. The Association for Computer Linguistics, 2009, pages 576–584 (cited on page 204).

- [183] Yang Liu, Tian Xia, Xinyan Xiao, and Qun Liu. “Weighted Alignment Matrices for Statistical Machine Translation”. In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, EMNLP 2009, 6-7 August 2009, Singapore, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2009, pages 1017–1026 (cited on page 195).
- [184] Yuchen Liu, Hao Xiong, Jiajun Zhang, Zhongjun He, Hua Wu, Haifeng Wang, and Chengqing Zong. “End-to-End Speech Translation with Knowledge Distillation”. In: *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*. ISCA, 2019, pages 1128–1132 (cited on page 403).
- [185] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. “Neural Architecture Optimization”. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*. 2018, pages 7827–7838 (cited on page 404).
- [186] Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. “Multi-task Sequence to Sequence Learning”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 2016 (cited on page 404).
- [187] Minh-Thang Luong and Christopher D. Manning. “Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016 (cited on pages 41, 351).
- [188] Thang Luong, Hieu Pham, and Christopher D. Manning. “Effective Approaches to Attention-based Neural Machine Translation”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*. The Association for Computational Linguistics, 2015, pages 1412–1421 (cited on page 302).
- [189] Chunpeng Ma, Akihiro Tamura, Masao Utiyama, Tiejun Zhao, and Eiichiro Sumita. “Forest-Based Neural Machine Translation”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*. Association for Computational Linguistics, 2018, pages 1253–1263 (cited on page 403).
- [190] Mingbo Ma, Liang Huang, Hao Xiong, Kaibo Liu, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, and Haifeng Wang. “STACL: Simultaneous Translation

- with Integrated Anticipation and Controllable Latency”. In: *CoRR* abs/1810.08398 (2018) (cited on pages 366, 375).
- [191] Xuezhe Ma and Eduard H. Hovy. “End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016 (cited on page 83).
- [192] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999 (cited on pages 44, 126).
- [193] Elman Mansimov, Gábor Melis, and Lei Yu. “Capturing document context inside sentence-level neural machine translation models with self-training”. In: *CoRR* abs/2003.05259 (2020) (cited on page 403).
- [194] Kelly Marchisio, Kevin Duh, and Philipp Koehn. “When Does Unsupervised Machine Translation Work?” In: *CoRR* abs/2004.05516 (2020) (cited on page 403).
- [195] Daniel Marcu and Harold Charles Daume. “Practical structured learning techniques for natural language processing”. In: *Ph.D. thesis, University of Southern California, Los Angeles, CA* (2006) (cited on page 203).
- [196] Daniel Marcu, Wei Wang, Abdessamad Echihabi, and Kevin Knight. “SPMT: Statistical Machine Translation with Syntactified Target Language Phrases”. In: *EMNLP 2006, Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, 22-23 July 2006, Sydney, Australia*. ACL, 2006, pages 44–52 (cited on page 190).
- [197] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. “Building a Large Annotated Corpus of English: The Penn Treebank”. In: *Computational Linguistics* 19.2 (1993), pages 313–330 (cited on page 191).
- [198] Sameen Maruf, André F. T. Martins, and Gholamreza Haffari. “Selective Attention for Context-aware Neural Machine Translation”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 2019, pages 3092–3102 (cited on page 403).
- [199] Evgeny Matusov, Stephan Kanthak, and Hermann Ney. “On the integration of speech recognition and statistical machine translation”. In: *INTERSPEECH 2005 - Eurospeech, 9th European Conference on Speech Communication and Technology, Lisbon, Portugal, September 4-8, 2005*. ISCA, 2005, pages 3177–3180 (cited on page 403).

- [200] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. “Learned in Translation: Contextualized Word Vectors”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 2017, pages 6294–6305 (cited on page 274).
- [201] Merity, tephenn, Keskar, Nitish Shirish, Socher, and Richard. “Regularizing and optimizing LSTM language models”. In: *arXiv: Computation and Language* (2017) (cited on page 213).
- [202] Haitao Mi, Liang Huang, and Qun Liu. “Forest-based translation”. In: *Proceedings of ACL-08: HLT*. 2008, pages 192–199 (cited on page 204).
- [203] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient Estimation of Word Representations in Vector Space”. In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. 2013 (cited on page 274).
- [204] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. “Recurrent neural network based language model”. In: *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*. ISCA, 2010, pages 1045–1048 (cited on pages 83, 213, 264).
- [205] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. “Distributed Representations of Words and Phrases and their Compositionality”. In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. 2013, pages 3111–3119 (cited on pages 269, 274).
- [206] Mikolov, Tomas, Zweig, and Geoffrey. “Context dependent recurrent neural network language model”. In: *2012 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2012, pages 234–239 (cited on page 213).
- [207] Pooya Moradi, Nishant Kambhatla, and Anoop Sarkar. “Interrogating the Explanatory Power of Attention in Neural Machine Translation”. In: *Proceedings of the 3rd Workshop on Neural Generation and Translation@EMNLP-IJCNLP 2019, Hong Kong, November 4, 2019*. Association for Computational Linguistics, 2019, pages 221–230 (cited on page 339).
- [208] Georges Mounin. “Chomsky, Noam: Syntactic Structures”. In: *Babel 7.1* (1961), pages 35–35 (cited on page 76).

- [209] Makoto Nagao. “A framework of a mechanical translation between Japanese and English by analogy principle”. In: *Artificial and human intelligence* (1984), pages 351–354 (cited on page 28).
- [210] Laurent Nepveu, Guy Lapalme, Philippe Langlais, and George F. Foster. “Adaptive Language and Translation Models for Interactive Machine Translation”. In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing, EMNLP 2004, A meeting of SIGDAT, a Special Interest Group of the ACL, held in conjunction with ACL 2004, 25-26 July 2004, Barcelona, Spain*. ACL, 2004, pages 190–197 (cited on page 375).
- [211] Graham Neubig, Makoto Morishita, and Satoshi Nakamura. “Neural Reranking Improves Subjective Quality of Machine Translation: NAIST at WAT2015”. In: *Proceedings of the 2nd Workshop on Asian Translation, WAT 2015, Kyoto, Japan, October 16, 2015*. Workshop on Asian Translation, 2015, pages 35–41 (cited on page 404).
- [212] Hermann Ney. “Speech translation: coupling of recognition and translation”. In: *Proceedings of the 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '99, Phoenix, Arizona, USA, March 15-19, 1999*. IEEE Computer Society, 1999, pages 517–520 (cited on page 403).
- [213] Hermann Ney, Ute Essen, and Reinhard Kneser. “On structuring probabilistic dependencies in stochastic language modelling”. In: *Computer Speech & Language* 8.1 (1994), pages 1–38 (cited on page 74).
- [214] Andrew Y Ng and Michael I Jordan. “On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes”. In: *Advances in neural information processing systems*. 2002, pages 841–848 (cited on page 83).
- [215] Jan Niehues, Eunah Cho, Thanh-Le Ha, and Alex Waibel. “Pre-Translation for Neural Machine Translation”. In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pages 1828–1836 (cited on page 404).
- [216] Franz Josef Och. “Minimum Error Rate Training in Statistical Machine Translation”. In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, 7-12 July 2003, Sapporo Convention Center, Sapporo, Japan*. ACL, 2003, pages 160–167 (cited on page 152).
- [217] Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alexander M. Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, Viren Jain, Zhen Jin, and Dragomir R. Radev. “A Smorgasbord of Features for Statistical Machine Translation”. In: *Human Language Technology Conference of the*

- North American Chapter of the Association for Computational Linguistics, HLT-NAACL 2004, Boston, Massachusetts, USA, May 2-7, 2004*. The Association for Computational Linguistics, 2004, pages 161–168 (cited on pages 203, 204).
- [218] Franz Josef Och and Hermann Ney. “A Comparison of Alignment Models for Statistical Machine Translation”. In: *COLING 2000, 18th International Conference on Computational Linguistics, Proceedings of the Conference, 2 Volumes, July 31 - August 4, 2000, Universität des Saarlandes, Saarbrücken, Germany*. Morgan Kaufmann, 2000, pages 1086–1090 (cited on page 146).
- [219] Franz Josef Och and Hermann Ney. “Statistical multi-source translation”. In: *In MT Summit 2001*. 2001, pages 253–258 (cited on page 404).
- [220] Franz Josef Och and Hermann Ney. “Discriminative Training and Maximum Entropy Models for Statistical Machine Translation”. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL, 2002, pages 295–302. URL: <https://www.aclweb.org/anthology/P02-1038/> (cited on page 143).
- [221] Franz Josef Och and Hermann Ney. “A Systematic Comparison of Various Statistical Alignment Models”. In: *Computational Linguistics* 29.1 (2003), pages 19–51 (cited on pages 40, 131).
- [222] Franz Josef Och and Hermann Ney. “The Alignment Template Approach to Statistical Machine Translation”. In: *Computational Linguistics* 30.4 (2004), pages 417–449 (cited on page 131).
- [223] Franz Josef Och and Hermann Ney. “The Alignment Template Approach to Statistical Machine Translation”. In: *Computational Linguistics* 30.4 (2004), pages 417–449. DOI: 10.1162/0891201042544884. URL: <https://doi.org/10.1162/0891201042544884> (cited on page 151).
- [224] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. “fairseq: A Fast, Extensible Toolkit for Sequence Modeling”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Demonstrations*. Association for Computational Linguistics, 2019, pages 48–53 (cited on pages 40, 383).
- [225] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. “Scaling Neural Machine Translation”. In: *Proceedings of the Third Conference on Machine Translation: Research Papers, WMT 2018, Belgium, Brussels, October 31 - November 1, 2018*. Association for Computational Linguistics, 2018, pages 1–9 (cited on page 364).

- [226] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. “Bleu: a Method for Automatic Evaluation of Machine Translation”. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL, 2002, pages 311–318 (cited on page 33).
- [227] Romain Paulus, Caiming Xiong, and Richard Socher. “A Deep Reinforced Model for Abstractive Summarization”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018 (cited on page 336).
- [228] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “Glove: Global Vectors for Word Representation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2014, pages 1532–1543 (cited on pages 269, 274).
- [229] Alvaro Peris, Miguel Domingo, and Francisco Casacuberta. “Interactive neural machine translation”. In: *Computer Speech & Language* 45.Sep. (2017), pages 201–220 (cited on page 375).
- [230] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. “DeepWalk: online learning of social representations”. In: *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*. ACM, 2014, pages 701–710 (cited on page 274).
- [231] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. “Deep Contextualized Word Representations”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*. Association for Computational Linguistics, 2018, pages 2227–2237 (cited on page 271).
- [232] Barbara Plank and Alessandro Moschitti. “Embedding Semantic Similarity in Tree Kernels for Domain Adaptation of Relation Extraction”. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*. The Association for Computer Linguistics, 2013, pages 1498–1507 (cited on page 274).
- [233] M. J. D. Powell. “An efficient method for finding the minimum of a function of several variables without calculating derivatives”. In: *The Computer Journal* 7.2 (1964), pages 155–162 (cited on page 153).
- [234] Gabriele Prato, Ella Charlaix, and Mehdi Rezagholizadeh. “Fully Quantized Transformer for Improved Translation”. In: *CoRR* abs/1910.10485 (2019) (cited on page 373).

- [235] Ning Qian. “On the momentum term in gradient descent learning algorithms”. In: *Neural Networks* 12.1 (1999), pages 145–151 (cited on page 248).
- [236] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. “Improving language understanding by generative pre-training”. In: *URL <https://s3-us-west-2.amazonaws.com/openai-assets/researchcovers/languageunsupervised/languageunderstandingpaper.pdf>* (2018) (cited on page 271).
- [237] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. “Language models are unsupervised multitask learners”. In: *OpenAI Blog* 1.8 (2019), page 9 (cited on pages 213, 274, 353).
- [238] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *CoRR* abs/1910.10683 (2019) (cited on page 353).
- [239] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermüller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleacher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre Luc Carrier, Kyunghyun Cho, Jan Chorowski, Paul F. Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron C. Courville, Yann N. Dauphin, Olivier Delalleau, et al. “Theano: A Python framework for fast computation of mathematical expressions”. In: *CoRR* abs/1605.02688 (2016) (cited on page 40).
- [240] Antti-Veikko I. Rosti, Necip Fazil Ayan, Bing Xiang, Spyridon Matsoukas, Richard M. Schwartz, and Bonnie J. Dorr. “Combining Outputs from Multiple Machine Translation Systems”. In: *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, April 22-27, 2007, Rochester, New York, USA*. The Association for Computational Linguistics, 2007, pages 228–235 (cited on page 204).
- [241] Antti-Veikko I. Rosti, Spyridon Matsoukas, and Richard M. Schwartz. “Improved Word-Level System Combination for Machine Translation”. In: *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*. The Association for Computational Linguistics, 2007 (cited on page 378).
- [242] Antti-Veikko I. Rosti, Bing Zhang, Spyros Matsoukas, and Richard M. Schwartz. “Incremental Hypothesis Alignment for Building Confusion Networks with Application to Machine Translation System Combination”. In: *Proceedings of the Third*

- Workshop on Statistical Machine Translation, WMT@ACL 2008, Columbus, Ohio, USA, June 19, 2008*. Association for Computational Linguistics, 2008, pages 183–186 (cited on page 378).
- [243] Sebastian Ruder. “An Overview of Multi-Task Learning in Deep Neural Networks”. In: *CoRR* abs/1706.05098 (2017) (cited on page 394).
- [244] Alexander M. Rush, Sumit Chopra, and Jason Weston. “A Neural Attention Model for Abstractive Sentence Summarization”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*. The Association for Computational Linguistics, 2015, pages 379–389 (cited on page 336).
- [245] Danielle Saunders, Felix Stahlberg, Adrià de Gispert, and Bill Byrne. “Multi-representation ensembles and delayed SGD updates improve syntax-based NMT”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*. Association for Computational Linguistics, 2018, pages 319–325 (cited on page 403).
- [246] M. Schuster and K. Nakajima. “Japanese and Korean voice search”. In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012, pages 5149–5152 (cited on page 354).
- [247] Holger Schwenk. “Continuous Space Translation Models for Phrase-Based Statistical Machine Translation”. In: *COLING 2012, 24th International Conference on Computational Linguistics, Proceedings of the Conference: Posters, 8-15 December 2012, Mumbai, India*. Indian Institute of Technology Bombay, 2012, pages 1071–1080 (cited on page 278).
- [248] Abigail See, Minh-Thang Luong, and Christopher D. Manning. “Compression of Neural Machine Translation Models via Pruning”. In: *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*. ACL, 2016, pages 291–301 (cited on page 339).
- [249] Rico Sennrich, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, and Maria Nadejde. “Nematus: a Toolkit for Neural Machine Translation”. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Software Demonstrations*. Association for Computational Linguistics, 2017, pages 65–68 (cited on pages 40, 317).

- [250] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Edinburgh Neural Machine Translation Systems for WMT 16”. In: *Proceedings of the First Conference on Machine Translation, WMT 2016, colocated with ACL 2016, August 11-12, Berlin, Germany*. The Association for Computer Linguistics, 2016, pages 371–376 (cited on page 368).
- [251] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Improving Neural Machine Translation Models with Monolingual Data”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016 (cited on page 391).
- [252] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Neural Machine Translation of Rare Words with Subword Units”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016 (cited on pages 351, 353).
- [253] Claude E Shannon. “A mathematical theory of communication”. In: *Bell system technical journal* 27.3 (1948), pages 379–423 (cited on page 83).
- [254] Claude E Shannon. “Communication theory of secrecy systems”. In: *Bell system technical journal* 28.4 (1949), pages 656–715 (cited on page 102).
- [255] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. “Self-Attention with Relative Position Representations”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*. Association for Computational Linguistics, 2018, pages 464–468 (cited on pages 325, 338, 381).
- [256] Libin Shen, Jinxi Xu, and Ralph M. Weischedel. “A New String-to-Dependency Machine Translation Algorithm with a Target Dependency Language Model”. In: *ACL 2008, Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, June 15-20, 2008, Columbus, Ohio, USA*. The Association for Computer Linguistics, 2008, pages 577–585 (cited on page 204).
- [257] Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron C. Courville. “Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019 (cited on page 403).
- [258] Yu Shiwen. “Automatic evaluation of output quality for machine translation systems”. In: *Machine Translation* 8.1-2 (1993), pages 117–126 (cited on page 35).

- [259] Khe Chai Sim, William J. Byrne, Mark J. F. Gales, Hichem Sahbi, and Philip C. Woodland. “Consensus Network Decoding for Statistical Machine Translation System Combination”. In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2007, Honolulu, Hawaii, USA, April 15-20, 2007*. IEEE, 2007, pages 105–108 (cited on page 378).
- [260] David A. Smith and Jason Eisner. “Minimum Risk Annealing for Training Log-Linear Models”. In: *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17-21 July 2006*. The Association for Computer Linguistics, 2006 (cited on page 203).
- [261] Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. “A Study of Translation Edit Rate with Targeted Human Annotation”. In: *In Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA. 2006*, pages 223–231 (cited on page 34).
- [262] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. “MASS: Masked Sequence to Sequence Pre-training for Language Generation”. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Volume 97. Proceedings of Machine Learning Research. PMLR, 2019, pages 5926–5936 (cited on pages 274, 394).
- [263] Matthias Sperber, Graham Neubig, Jan Niehues, and Alex Waibel. “Attention-Passing Models for Robust and Data-Efficient End-to-End Speech Translation”. In: *Transactions of the Association for Computational Linguistics* 7 (2019), pages 313–325 (cited on page 403).
- [264] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pages 1929–1958 (cited on page 333).
- [265] Felix Stahlberg. “Neural Machine Translation: A Review”. In: *CoRR* abs/1912.02047 (2019) (cited on page 279).
- [266] Felix Stahlberg and Bill Byrne. “On NMT Search Errors and Model Errors: Cat Got Your Tongue?” In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Association for Computational Linguistics, 2019, pages 3354–3360 (cited on page 366).

- [267] Felix Stahlberg, Adrià de Gispert, Eva Hasler, and Bill Byrne. “Neural Machine Translation by Minimising the Bayes-risk with Respect to Syntactic Translation Lattices”. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers*. Association for Computational Linguistics, 2017, pages 362–368 (cited on page 404).
- [268] Felix Stahlberg, Eva Hasler, Danielle Saunders, and Bill Byrne. “SGNMT - A Flexible NMT Decoding Platform for Quick Prototyping of New Models and Search Strategies”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017 - System Demonstrations*. Association for Computational Linguistics, 2017, pages 25–30 (cited on page 367).
- [269] Felix Stahlberg, Eva Hasler, Aurelien Waite, and Bill Byrne. “Syntactically Guided Neural Machine Translation”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*. The Association for Computer Linguistics, 2016 (cited on page 274).
- [270] Andreas Stolcke. “SRILM - an extensible language modeling toolkit”. In: *7th International Conference on Spoken Language Processing, ICSLP2002 - INTERSPEECH 2002, Denver, Colorado, USA, September 16-20, 2002*. ISCA, 2002 (cited on page 74).
- [271] Jinsong Su, Zhixing Tan, Deyi Xiong, Rongrong Ji, Xiaodong Shi, and Yang Liu. “Lattice-Based Recurrent Neural Network Encoders for Neural Machine Translation”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. AAAI Press, 2017, pages 3302–3308 (cited on page 381).
- [272] Jun Sun, Min Zhang, and Chew Lim Tan. “Discriminative Induction of Sub-Tree Alignment using Limited Labeled Data”. In: *COLING 2010, 23rd International Conference on Computational Linguistics, Proceedings of the Conference, 23-27 August 2010, Beijing, China*. Edited by Chu-Ren Huang and Dan Jurafsky. Tsinghua University Press, 2010, pages 1047–1055 (cited on page 193).
- [273] Jun Sun, Min Zhang, and Chew Lim Tan. “Discriminative Induction of Sub-Tree Alignment using Limited Labeled Data”. In: *COLING 2010, 23rd International Conference on Computational Linguistics, Proceedings of the Conference, 23-27 August 2010, Beijing, China*. Tsinghua University Press, 2010, pages 1047–1055 (cited on page 195).

- [274] Jun Sun, Min Zhang, and Chew Lim Tan. “Exploring Syntactic Structural Features for Sub-Tree Alignment Using Bilingual Tree Kernels”. In: *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden*. The Association for Computer Linguistics, 2010, pages 306–315 (cited on page 195).
- [275] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. “Patient Knowledge Distillation for BERT Model Compression”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Association for Computational Linguistics, 2019, pages 4322–4331 (cited on page 339).
- [276] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. 2014, pages 3104–3112 (cited on pages 30, 278).
- [277] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. The MIT Press, 1999, pages 1057–1063 (cited on page 401).
- [278] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. “Rethinking the Inception Architecture for Computer Vision”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pages 2818–2826 (cited on pages 333, 357).
- [279] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. “Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*. The Association for Computer Linguistics, 2015, pages 1556–1566 (cited on page 403).
- [280] Xin Tan, Longyin Zhang, Deyi Xiong, and Guodong Zhou. “Hierarchical Modeling of Global Context for Document-Level Neural Machine Translation”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Process-*

- ing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pages 1576–1585 (cited on page 403).
- [281] Benjamin Taskar, Simon Lacoste-Julien, and Dan Klein. “A Discriminative Matching Approach to Word Alignment”. In: *HLT/EMNLP 2005, Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference, 6-8 October 2005, Vancouver, British Columbia, Canada*. The Association for Computational Linguistics, 2005, pages 73–80 (cited on page 146).
- [282] Jörg Tiedemann and Yves Scherrer. “Neural Machine Translation with Extended Context”. In: *Proceedings of the Third Workshop on Discourse in Machine Translation, DiscoMT@EMNLP 2017, Copenhagen, Denmark, September 8, 2017*. Association for Computational Linguistics, 2017, pages 82–92 (cited on page 403).
- [283] Tijmen Tieleman and Geoffrey Hinton. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural networks for machine learning 4.2* (2012), pages 26–31 (cited on page 248).
- [284] Roy Tromble, Shankar Kumar, Franz Josef Och, and Wolfgang Macherey. “Lattice Minimum Bayes-Risk Decoding for Statistical Machine Translation”. In: *2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2008, pages 620–629 (cited on page 381).
- [285] Zhaopeng Tu, Yang Liu, Shuming Shi, and Tong Zhang. “Learning to Remember Translation History with a Continuous Cache”. In: *Transactions of the Association for Computational Linguistics* 6 (2018), pages 407–420 (cited on page 403).
- [286] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. “Modeling Coverage for Neural Machine Translation”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016 (cited on pages 316, 378).
- [287] Ashish Vaswani, Samy Bengio, Eugene Brevdo, François Chollet, Aidan N. Gomez, Stephan Gouws, Llion Jones, Lukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. “Tensor2Tensor for Neural Machine Translation”. In: *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas, AMTA 2018, Boston, MA, USA, March 17-21, 2018 -*

- Volume 1: Research Papers*. Association for Machine Translation in the Americas, 2018, pages 193–199 (cited on pages 40, 383).
- [288] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 2017, pages 5998–6008 (cited on pages 266, 279, 314, 318, 319, 334, 360, 361, 383, 387).
- [289] Raúl Vázquez, Alessandro Raganato, Jörg Tiedemann, and Mathias Creutz. “Multilingual NMT with a Language-Independent Attention Bridge”. In: *Proceedings of the 4th Workshop on Representation Learning for NLP, RepL4NLP@ACL 2019, Florence, Italy, August 2, 2019*. Association for Computational Linguistics, 2019, pages 33–39 (cited on page 339).
- [290] David Vilar, Daniel Stein, Matthias Huck, and Hermann Ney. “Jane: an advanced freely available hierarchical machine translation toolkit”. In: *Machine Translation 26.3* (2012), pages 197–216 (cited on page 39).
- [291] Andrew J. Viterbi. “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm”. In: *IEEE Trans. Inf. Theory* 13.2 (1967), pages 260–269. DOI: 10.1109/TIT.1967.1054010. URL: <https://doi.org/10.1109/TIT.1967.1054010> (cited on page 142).
- [292] Stephan Vogel, Hermann Ney, and Christoph Tillmann. “HMM-Based Word Alignment in Statistical Translation”. In: *16th International Conference on Computational Linguistics, Proceedings of the Conference, COLING 1996, Center for Sprogteknologi, Copenhagen, Denmark, August 5-9, 1996*. 1996, pages 836–841 (cited on page 125).
- [293] Elena Voita, Rico Sennrich, and Ivan Titov. “Context-Aware Monolingual Repair for Neural Machine Translation”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Association for Computational Linguistics, 2019, pages 877–886 (cited on page 403).
- [294] Elena Voita, Rico Sennrich, and Ivan Titov. “When a Good Translation is Wrong in Context: Context-Aware Machine Translation Improves on Deixis, Ellipsis, and Lexical Cohesion”. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Association for Computational Linguistics, 2019, pages 1198–1212 (cited on page 403).

- [295] Elena Voita, Pavel Serdyukov, Rico Sennrich, and Ivan Titov. “Context-Aware Neural Machine Translation Learns Anaphora Resolution”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*. Association for Computational Linguistics, 2018, pages 1264–1274 (cited on pages 339, 403).
- [296] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. “Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned”. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Association for Computational Linguistics, 2019, pages 5797–5808 (cited on page 339).
- [297] Mingxuan Wang, Li Gong, Wenhuan Zhu, Jun Xie, and Chao Bian. “Tencent Neural Machine Translation Systems for WMT18”. In: *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*. Belgium, Brussels: Association for Computational Linguistics, Oct. 2018, pages 522–527 (cited on page 348).
- [298] Qiang Wang, Bei Li, Jiqiang Liu, Bojian Jiang, Zheyang Zhang, Yinqiao Li, Ye Lin, Tong Xiao, and Jingbo Zhu. “The NiuTrans Machine Translation System for WMT18”. In: *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*. Belgium, Brussels: Association for Computational Linguistics, Oct. 2018, pages 528–534 (cited on page 74).
- [299] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. “Learning Deep Transformer Models for Machine Translation”. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Association for Computational Linguistics, 2019, pages 1810–1822 (cited on pages 282, 331, 334, 338, 381, 382, 384, 385, 387).
- [300] Qiang Wang, Fuxue Li, Tong Xiao, Yanyang Li, Yinqiao Li, and Jingbo Zhu. “Multi-layer Representation Fusion for Neural Machine Translation”. In: *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*. Association for Computational Linguistics, 2018, pages 3015–3026 (cited on pages 338, 385).
- [301] Wang, Rui, Utiyama, Masao, Sumita, and Eiichiro. “Dynamic Sentence Sampling for Efficient Training of Neural Machine Translation”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, July 2018, pages 298–304 (cited on page 348).

- [302] Xiaolin Wang, Masao Utiyama, and Eiichiro Sumita. “CytonMT: an Efficient Neural Machine Translation Open-source Toolkit Implemented in C++”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 2018, pages 133–138 (cited on page 41).
- [303] Xing Wang, Zhengdong Lu, Zhaopeng Tu, Hang Li, Deyi Xiong, and Min Zhang. “Neural Machine Translation Advised by Statistical Machine Translation”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. AAAI Press, 2017, pages 3330–3336 (cited on page 339).
- [304] Xinyi Wang, Hieu Pham, Pengcheng Yin, and Graham Neubig. “A Tree-based Decoder for Neural Machine Translation”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 2018, pages 4772–4777 (cited on page 339).
- [305] Yau-Shian Wang, Hung-yi Lee, and Yun-Nung Chen. “Tree Transformer: Integrating Tree Structures into Self-Attention”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Association for Computational Linguistics, 2019, pages 1061–1070 (cited on page 339).
- [306] Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. “Online Large-Margin Training for Statistical Machine Translation”. In: *EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic*. ACL, 2007, pages 764–773 (cited on page 203).
- [307] Ron J. Weiss, Jan Chorowski, Navdeep Jaitly, Yonghui Wu, and Zhifeng Chen. “Sequence-to-Sequence Models Can Directly Translate Foreign Speech”. In: *INTER-SPEECH*. 2017, pages 2625–2629 (cited on page 403).
- [308] Lesly Miculicich Werlen, Dhananjay Ram, Nikolaos Pappas, and James Henderson. “Document-Level Neural Machine Translation with Hierarchical Attention Networks”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 2018, pages 2947–2954 (cited on pages 339, 403).

- [309] Dekai Wu. “Stochastic Inversion Transduction Grammars and Bilingual Parsing of Parallel Corpora”. In: *Computational Linguistics* 23.3 (1997), pages 377–403 (cited on page 178).
- [310] Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. “Pay Less Attention with Lightweight and Dynamic Convolutions”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019 (cited on pages 338, 371).
- [311] Lijun Wu, Yiren Wang, Yingce Xia, Fei Tian, Fei Gao, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. “Depth Growing for Neural Machine Translation”. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Association for Computational Linguistics, 2019, pages 5558–5563 (cited on pages 386, 387).
- [312] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation”. In: *CoRR* abs/1609.08144 (2016) (cited on pages 279, 288, 317, 318, 378).
- [313] Mengzhou Xia, Guoping Huang, Lemao Liu, and Shuming Shi. “Graph Based Translation Memory for Neural Machine Translation”. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pages 7297–7304 (cited on page 404).
- [314] Yingce Xia, Tao Qin, Wei Chen, Jiang Bian, Nenghai Yu, and Tie-Yan Liu. “Dual Supervised Learning”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Volume 70. Proceedings of Machine Learning Research. PMLR, 2017, pages 3789–3798 (cited on page 400).
- [315] Tong Xiao, Mu Li, Dongdong Zhang, Jingbo Zhu, and Ming Zhou. “Better Synchronous Binarization for Machine Translation”. In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, EMNLP 2009, 6-7*

- August 2009, Singapore, A meeting of SIGDAT, a Special Interest Group of the ACL. ACL, 2009, pages 362–370 (cited on pages 192, 203).*
- [316] Tong Xiao, Yinqiao Li, Jingbo Zhu, Zhengtao Yu, and Tongran Liu. “Sharing Attention Weights for Fast Transformer”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. ijcai.org, 2019, pages 5292–5298 (cited on pages 334, 339, 370, 371).
- [317] Tong Xiao, Derek F. Wong, and Jingbo Zhu. “A Loss-Augmented Approach to Training Syntactic Machine Translation Systems”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 24.11 (2016), pages 2069–2083 (cited on page 203).
- [318] Tong Xiao and Jingbo Zhu. “Unsupervised sub-tree alignment for tree-to-tree translation”. In: *Journal of Artificial Intelligence Research* 48 (2013), pages 733–782 (cited on pages 132, 193–195).
- [319] Tong Xiao, Jingbo Zhu, and Tongran Liu. “Bagging and boosting statistical machine translation systems”. In: *Artificial Intelligence* 195 (2013), pages 496–527 (cited on pages 204, 379).
- [320] Tong Xiao, Jingbo Zhu, Tongran Liu, and Chunliang Zhang. “Fast Parallel Training of Neural Language Models”. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. ijcai.org, 2017, pages 4193–4199 (cited on pages 251, 311).
- [321] Tong Xiao, Jingbo Zhu, Chunliang Zhang, and Tongran Liu. “Syntactic Skeleton-Based Translation”. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. AAAI Press, 2016, pages 2856–2862 (cited on page 204).
- [322] Tong Xiao, Jingbo Zhu, Hao Zhang, and Qiang Li. “NiuTrans: An Open Source Toolkit for Phrase-based and Syntax-based Machine Translation”. In: *The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the System Demonstrations, July 10, 2012, Jeju Island, Korea*. The Association for Computer Linguistics, 2012, pages 19–24 (cited on page 38).
- [323] Tong Xiao, Jingbo Zhu, Hao Zhang, and Muhua Zhu. “An Empirical Study of Translation Rule Extraction with Multiple Parsers”. In: *COLING 2010, 23rd International Conference on Computational Linguistics, Posters Volume, 23-27 August 2010, Beijing, China*. Chinese Information Processing Society of China, 2010, pages 1345–1353 (cited on page 204).

- [324] Tong Xiao, Jingbo Zhu, and Muhua Zhu. “Language Modeling for Syntax-Based Machine Translation Using Tree Substitution Grammars: A Case Study on Chinese-English Translation”. In: *ACM Transactions on Asian Language Information Processing (TALIP)* 10.4 (2011), pages 1–29 (cited on page 204).
- [325] Tong Xiao, Jingbo Zhu, Muhua Zhu, and Huizhen Wang. “Boosting-Based System Combination for Machine Translation”. In: *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden*. The Association for Computer Linguistics, 2010, pages 739–748 (cited on page 378).
- [326] Deyi Xiong, Qun Liu, and Shouxun Lin. “Maximum Entropy Based Phrase Reordering Model for Statistical Machine Translation”. In: *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17-21 July 2006*. The Association for Computer Linguistics, 2006 (cited on page 151).
- [327] Hao Xiong, Zhongjun He, Hua Wu, and Haifeng Wang. “Modeling Coherence for Discourse Neural Machine Translation”. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pages 7338–7345 (cited on page 403).
- [328] Nianwen Xue, Fei Xia, Fu dong Chiou, and Martha Palmer. “Building a large annotated Chinese corpus: the Penn Chinese treebank”. In: *Journal of Natural Language Engineering* 11.2 (2005), pages 207–238 (cited on page 191).
- [329] Baosong Yang, Longyue Wang, Derek F. Wong, Lidia S. Chao, and Zhaopeng Tu. “Convolutional Self-Attention Networks”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 2019, pages 4040–4045 (cited on page 338).
- [330] Baosong Yang, Derek F. Wong, Tong Xiao, Lidia S. Chao, and Jingbo Zhu. “Towards Bidirectional Hierarchical Representations for Attention-based Neural Machine Translation”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, Septem-*

- ber 9-11, 2017. Association for Computational Linguistics, 2017, pages 1432–1441 (cited on pages 339, 403).
- [331] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. “XLNet: Generalized Autoregressive Pretraining for Language Understanding”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*. 2019, pages 5754–5764 (cited on page 274).
- [332] Daniel H. Younger. “Recognition and Parsing of Context-Free Languages in Time n^3 ”. In: *Information and Control* 10.2 (1967), pages 189–208. DOI: 10 . 1016 / S0019 - 9958(67) 80007 - X. URL: [https://doi.org/10.1016/S0019-9958\(67\)80007-X](https://doi.org/10.1016/S0019-9958(67)80007-X) (cited on page 169).
- [333] Lei Yu, Laurent Sartran, Wojciech Stokowiec, Wang Ling, Lingpeng Kong, Phil Blunsom, and Chris Dyer. “Putting Machine Translation in Context with the Noisy Channel Model”. In: *CoRR* abs/1910.00553 (2019) (cited on page 403).
- [334] Xin Yu, Zhiding Yu, and Srikumar Ramalingam. “Learning Strict Identity Mappings in Deep Residual Networks”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, 2018, pages 4432–4440 (cited on page 382).
- [335] Sergey Zagoruyko and Nikos Komodakis. “Wide Residual Networks”. In: *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. BMVA Press, 2016 (cited on page 361).
- [336] Zaremba, Wojciech, Sutskever, Ilya, Vinyals, and Oriol. “Recurrent Neural Network Regularization”. In: *arXiv: Neural and Evolutionary Computing* (2014) (cited on page 213).
- [337] Poorya Zaremoondi and Gholamreza Haffari. “Incorporating Syntactic Uncertainty in Neural Machine Translation with a Forest-to-Sequence Model”. In: *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*. Association for Computational Linguistics, 2018, pages 1421–1429 (cited on page 403).
- [338] Biao Zhang and Rico Sennrich. “A Lightweight Recurrent Network for Sequence Modeling”. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Association for Computational Linguistics, 2019, pages 1538–1548 (cited on page 338).

- [339] Biao Zhang, Ivan Titov, and Rico Sennrich. “Improving Deep Transformer with Depth-Scaled Initialization and Merged Attention”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Association for Computational Linguistics, 2019, pages 898–909 (cited on page 382).
- [340] Biao Zhang, Deyi Xiong, and Jinsong Su. “Accelerating Neural Transformer via an Average Attention Network”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*. Association for Computational Linguistics, 2018, pages 1789–1798 (cited on pages 334, 371).
- [341] Biao Zhang, Deyi Xiong, Jinsong Su, Qian Lin, and Huiji Zhang. “Simplifying Neural Machine Translation with Addition-Subtraction Twin-Gated Recurrent Networks”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 2018, pages 4273–4283 (cited on page 338).
- [342] Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. “Synchronous Binarization for Machine Translation”. In: *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 4-9, 2006, New York, New York, USA*. The Association for Computational Linguistics, 2006. URL: <https://www.aclweb.org/anthology/N06-1033/> (cited on page 192).
- [343] Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. “Synchronous Binarization for Machine Translation”. In: *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 4-9, 2006, New York, New York, USA*. The Association for Computational Linguistics, 2006 (cited on page 203).
- [344] Jiacheng Zhang, Yanzhuo Ding, Shiqi Shen, Yong Cheng, Maosong Sun, Huan-Bo Luan, and Yang Liu. “THUMT: An Open Source Toolkit for Neural Machine Translation”. In: *CoRR abs/1706.06415 (2017)* (cited on page 41).
- [345] Jiacheng Zhang, Yang Liu, Huanbo Luan, Jingfang Xu, and Maosong Sun. “Prior Knowledge Integration for Neural Machine Translation using Posterior Regularization”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*. Association for Computational Linguistics, 2017, pages 1514–1523 (cited on pages 274, 339).

- [346] Jiacheng Zhang, Huanbo Luan, Maosong Sun, Feifei Zhai, Jingfang Xu, Min Zhang, and Yang Liu. “Improving the Transformer Translation Model with Document-Level Context”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 2018, pages 533–542 (cited on page 403).
- [347] Jiajun Zhang and Chengqing Zong. “Bridging Neural Machine Translation and Bilingual Dictionaries”. In: *CoRR* abs/1610.07272 (2016) (cited on page 339).
- [348] Min Zhang, Hongfei Jiang, AiTi Aw, Haizhou Li, Chew Lim Tan, and Sheng Li. “A Tree Sequence Alignment-based Tree-to-Tree Translation Model”. In: *ACL 2008, Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, June 15-20, 2008, Columbus, Ohio, USA*. The Association for Computer Linguistics, 2008, pages 559–567 (cited on page 178).
- [349] Wen Zhang, Liang Huang, Yang Feng, Lei Shen, and Qun Liu. “Speeding Up Neural Machine Translation Decoding by Cube Pruning”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Association for Computational Linguistics, 2018, pages 4284–4294 (cited on page 339).
- [350] Xiangwen Zhang, Jinsong Su, Yue Qin, Yang Liu, Rongrong Ji, and Hongji Wang. “Asynchronous Bidirectional Decoding for Neural Machine Translation”. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. AAAI Press, 2018, pages 5698–5705 (cited on page 368).
- [351] Jingbo Zhu and Tong Xiao. “Improving Decoding Generalization for Tree-to-String Translation”. In: *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA - Short Papers*. The Association for Computer Linguistics, 2011, pages 418–423 (cited on page 201).
- [352] Jinhua Zhu, Yingce Xia, Lijun Wu, Di He, Tao Qin, Wengang Zhou, Houqiang Li, and Tie-Yan Liu. “Incorporating BERT into Neural Machine Translation”. In: *CoRR* abs/2002.06823 (2020) (cited on page 394).
- [353] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. “Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks”. In: *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2017, pages 2242–2251 (cited on page 401).

- [354] Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. “Long Short-Term Memory Over Tree Structures”. In: *CoRR* abs/1503.04881 (2015) (cited on page 83).
- [355] Zilly, Julian, Srivastava, Rupesh Kumar, Koutnik, Jan, Schmidhuber, and Jurgen. “Recurrent Highway Networks”. In: *arXiv: Learning* (2016) (cited on page 213).
- [356] Andreas Zollmann and Ashish Venugopal. “Syntax Augmented Machine Translation via Chart Parsing”. In: *Proceedings on the Workshop on Statistical Machine Translation, WMT@HLT-NAACL 2006, New York City, NY, USA, June 8-9, 2006*. Association for Computational Linguistics, 2006, pages 138–141 (cited on page 274).
- [357] Andreas Zollmann, Ashish Venugopal, Matthias Paulik, and Stephan Vogel. “The Syntax Augmented MT (SAMT) System at the Shared Task for the 2007 ACL Workshop on Statistical Machine Translation”. In: *Proceedings of the Second Workshop on Statistical Machine Translation, WMT@ACL 2007, Prague, Czech Republic, June 23, 2007*. Association for Computational Linguistics, 2007, pages 216–219 (cited on page 39).
- [358] Barret Zoph and Kevin Knight. “Multi-Source Neural Translation”. In: *HLT-NAACL*. 2016 (cited on page 404).
- [359] Barret Zoph and Quoc V. Le. “Neural Architecture Search with Reinforcement Learning”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017 (cited on page 404).
- [360] Barret Zoph, Ashish Vaswani, Jonathan May, and Kevin Knight. “Simple, Fast Noise-Contrastive Estimation for Large RNN Vocabularies”. In: *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*. The Association for Computational Linguistics, 2016, pages 1217–1222 (cited on page 40).
- [361] 刘克. 实用马尔可夫决策过程. 清华大学出版社, 2004 (cited on page 68).
- [362] 周志华. 机器学习. 清华大学出版社, 2016 (cited on page 45).
- [363] 姚树杰, 肖桐, and 朱靖波. “基于句对质量和覆盖度的统计机器翻译训练语料选取”. In: *中文信息学报* 25.2 (2011), pages 72–78 (cited on page 349).
- [364] 宗成庆. 统计自然语言处理. 清华大学出版社, 2013 (cited on page 44).
- [365] 李航. 统计学习方法. 清华大学出版社, 2012 (cited on page 45).
- [366] 姚恺璇 赵军峰. “深化改革探讨创新推进发展——全国翻译专业学位研究生教育 2019 年会综述”. In: *中国翻译* (2019) (cited on page 21).

-
- [367] 邱锡鹏. 神经网络与深度学习. <https://nndl.github.io/>. 2020 (cited on page 45).
- [368] 魏宗舒. 概率论与数理统计教程: 第二版. 北京: 高等教育出版社, 2011 (cited on page 52).



Index

- n -gram Precision, 33
- n -gram 准确率, 33
- n 元语法单元, 33
- Accuracy, 88, 366
- AdaGrad, 248
- Adam, 249
- Additive Smoothing, 70
- Adequacy, 88, 377
- Ambiguity, 79
- Annotated Data, 61
- Annotated Data/Labeled Data, 239
- Artificial Neural Networks, 207
- Asymmetric Word Alignment, 105
- Asynchronous Update, 250
- Attention To Attention Transfer, 398
- Attention Weight, 266, 303
- Automatic Differentiation, 245
- Autoregressive Model, 314
- Autoregressive Translation, 373
- back propagation, 255
- Back Translation, 391
- Backward Mode, 246
- Batch Gradient Descent, 243
- Batch Inference, 371
- Batch Normalization, 253
- Batching, 371
- Bayes' rule, 56
- Beam Pruning, 158, 172
- Beam Search, 314
- Bias, 63, 375
- Bidirectional Inference, 368
- Binarization, 191
- Binarized Neural Networks, 373
- Brevity Penalty, 34
- Byte Pair Encoding, BPE, 351
- Capacity, 360
- Chomsky Normal Form, 169
- Circle Consistency, 401
- Co-Adaptation, 358
- Co-adaptation, 389

- Composition, 198
- Compositional Translation, 134
- Computation Graph, 238
- Concept, 122
- Conditional Probability, 53
- Confusion Network, 381
- Connectionism, 209
- Context-Free Grammar, 76
- Convex function, 128
- Cost Function, 242
- Coverage Model, 156
- Cross Entropy Loss, 332
- Cross-entropy, 58
- Cube Pruning, 174
- Curriculum Learning, 365

- Dada Cleaning, 345
- Data Augmentation, 391
- Data Filtering, 347
- Data Parallelism, 363
- Data Selection, 347
- Data-Driven, 23
- Decay, 248
- Decoding, 91, 99, 366
- Deep Learning, 208
- Deficiency, 124
- Degenerate, 375
- Denoising, 355
- Dependency Parsing, 74
- Derivation, 78, 140, 165
- Directed Hyper-graph, 196
- Disambiguation, 79
- Discriminative Model, 83, 143
- Distortion, 120
- Distributed Representation, 212, 268, 277
- Distributed representation, 209
- Diversity, 379
- Dropout, 333

- Dynamic Linear Combination of Layers, DLCL, 385

- Element-wise Addition, 214
- Element-wise Product, 216
- Embedding, 362
- Encoder-Decoder, 30
- Encoder-decoder Attention Sub-layer, 322
- Encoder-Decoder Paradigm, 283
- End-to-End Learning, 211
- Ensemble Learning, 359
- Entropy, 57
- Estimate, 52
- Euclidean Norm, 218
- Event, 52
- Expectation Maximization, 115
- Expected Count, 115
- Explainable Machine Learning, 274
- Expression Swell, 245

- Feature Engineering, 283
- Feed-forward Neural Network Language Model, 262
- Feed-forward Sub-layer, 322
- Fertility, 118, 374
- Fine-tuning, 393
- First Moment Estimation, 309
- Fluency, 88
- Forward Translation, 392
- Frobenius Norm, 218
- Frobenius 范数, 218
- Frontier Set, 187
- Full Parsing, 75
- Full Search, 313

- Gated Recurrent Unit, GRU, 298
- Generalization, 355
- Generation, 291
- Generative Model, 83, 142

- Glue Rule, 165
- Good-Turing Estimate, 71
- Gradient Clipping, 252
- Gradient Descent Method, 242
- Gradient Explosion, 251
- Gradient Vanishing, 251
- Gradient-based Method, 242
- Gradual Warmup, 310
- Greedy Search, 313
- Grid Search, 154

- Hierarchical Phrase-based Grammar, 164
- Hierarchical Phrase-based Model, 163
- Hint-based Knowledge Transfer, 398
- Histogram Pruning, 158
- Hyper-edge, 196
- Hypothesis Recombination, 158
- Hypothesis Selection, 379

- Ill-posed Problem, 354
- Inference, 61, 91, 366
- Inverse Problem, 354
- Iterative Back Translation, 392

- Joint Probability, 53

- Label Smoothing, 333, 357
- Language, 79
- Language Model, 67
- Language Modeling, 67
- Latency, 366
- Law of Total Probability, 55
- Layer Normalization, 253, 322
- Learning Difficulty, 396
- Learning Rate, 242, 247, 309
- Left-most Derivation, 79
- Lexical Analysis, 59
- Lexical Translation Probability, 147
- Lexicalized Norm Form, 202

- Line Search, 153
- Linear Mapping, 216
- Linear Transformation, 216
- Linearization, 179
- Log-linear Model, 143
- Long Short-Term Memory, 295
- Loss Function, 240

- Machine Translation, 18
- marginal probability, 54
- Matrix, 213
- Memory, 366
- Mini-Batch Gradient Descent, 243
- Mini-batch Training, 333, 363
- Minimal Rules, 188
- Minimum Error Rate Training, 152
- Model Compression, 396
- Model Parallelism, 363
- Model Parameters, 239
- Model Training, 152
- Modeling Error, 366
- Momentum, 247
- Multi-head Attention, 328
- Multitask Learning, 394

- Neural Architecture Search; NAS, 404
- Neural Language Model, 261
- Neural Machine Translation, 275
- Neural Networks, 207
- Noise Channel Model, 102
- Non-autoregressive Model, 314
- Non-Autoregressive Translation, 374
- Non-terminal, 75
- Norm, 217
- Normalization, 345
- Numerical Differentiation, 244

- Objective Function, 240
- On-the-fly Back-translation, 402

- One-hot 编码, 267
Open-Vocabulary, 350
Out of Vocabulary Word, OOV Word, 345
Out-of-Vocabulary Word, OOV Word, 69
Over fitting, 333
Over Translation, 377
Overfitting, 254
Overfitting Problem, 354

Parameter Estimation, 63
Parameter Server, 250
Parsing, 51
Perplexity, 267
Phrasal Segmentation, 139
Phrase Extraction, 144
Phrase Structure Parsing, 74
Phrase Table, 148
Piecewise Constant Decay, 311
Position Embedding, 323
Post-norm, 331
Post-processing, 51
Pre-norm, 331
Pre-processing, 51
Pre-terminal, 75
Pre-training, 393
Probabilistic Context-Free Grammar, 80
Probability, 52
Probing Task, 362
Production Rule, 77
Pruning, 158

Quality Estimation, 32
Quality Evaluation, 32

Random Variable, 52
Rank, 229
Re-ranking, 368, 379
Recurrent Neural Network, 264
Recurrent Neural Network, RNN, 289
Regularization, 254, 354
Relative Entropy, 58
Reordering, 148
Representation Learning, 211, 270, 291
Residual Connection, 322
Residual Networks, 253
RMSprop, 248
RNN Cell, 264
RNNLM, 264
Round-off Error, 244

Scalar, 213
Scalar Multiplication, 215
Scaled Dot-Product Attention, 326
Search Error, 366
Second Moment Estimation, 309
Segmentation, 50, 59
Self-Attention, 320
Self-Attention Mechanism, 266
Self-attention Sub-layer, 321
Self-information, 57
Semantic Information, 362
Semi-ring Parsing, 197
Sentence, 79
Sequence-level Knowledge Distillation, 397
Sequence-to-Sequence Problem, 335
Short-cut Connection, 330
Shortcut Connection, 253
Source Language, 17
Span, 169
Stochastic Gradient Descent, 243
String-based Decoding, 199
Student Model, 397
Sub-word, 351
Sub-word Segmentation, 345
Supervised Training, 239
Surface Information, 362
Symbolic Differentiation, 245

- Symbolicism, 209
Symmetrization, 129
Synchronous Context-free Grammar, 163
Synchronous Tree Substitution Grammar Rule, Word, 59
181
Synchronous Update, 250
Syntactic Information, 362
Syntax, 74
System Bias, 130
System Combination, 378

Target Language, 17
Teacher Model, 396
Tensor, 229
Terminal, 75
The Lagrange Multiplier Method, 113
Threshold Pruning, 158
Top-down Parsing, 170
Training, 61, 91, 239
Training Data Set, 240
Translation Candidate, 89, 156
Translation Hypothesis, 157
Transpose, 214
Tree Fragment, 180
Tree-based Decoding, 199
Tree-to-String Translation Rule, 179
Tree-to-Tree Translation Rule, 179
Treebank, 81
Truncation Error, 244
Tuning Set, 152

Under Translation, 377
Update Rule, 242

Vector, 213
Vectorization, 233

Warmup, 332
weight, 219

Weight Tuning, 152
Well-posed, 354
Wide Residual Network, 361

Word Alignment, 96, 104
Word Embedding, 268, 291, 323
Word-level Knowledge Distillation, 397

一阶矩估计, 309
上下文无关文法, 76
不适定问题, 354
中间层输出, 398
乔姆斯基范式, 169
事件, 52
二值网络, 373
二义化, 191
二阶矩估计, 309
交叉熵, 58
交叉熵损失, 332
产出率, 118
产生式规则, 77
人工神经网络, 207
人工神经网络方法, 68
代价函数, 242
估计, 52
估计值, 52
位置编码, 323
依存分析, 74

假设选择, 379
假设重组, 158
偏置, 63, 375
充分性, 88, 377
全搜索, 313
全概率公式, 55
准确性, 88, 366
凸函数, 128
分布式表示, 209, 212, 268, 277

- 分段常数衰减, 311
分词, 50, 59
判别式模型, 143
判别模型, 83
前向传播, 237
前向翻译, 392
前正则化, 331
前馈神经网络子层, 322
前馈神经网络语言模型, 262
剪枝, 158
加法平滑, 70
动态线性层聚合方法, 385
半环分析, 197
单词, 59
参数估计, 63
参数更新的规则, 242
参数服务器, 250
双向推断, 368
双字节编码, 351
反向传播, 255
反向模式, 246
反问题, 354
古德-图灵估计法, 71
句子, 79
句子的表示, 270
句子表示模型, 270
句法, 74
句法分析, 51
可解释机器学习, 274
同步上下文无关文法, 163
同步更新, 250
同步树替换文法规则, 181
后处理, 51
后正则化, 331
向量, 213
向量化, 233
噪声信道模型, 102
回译, 391
困惑度, 267
基于串的解码, 199
基于单词的知识精炼, 397
基于句法的特征, 195
基于层次短语的文法, 164
基于层次短语的模型, 163
基于序列的知识精炼, 397
基于树的解码, 199
基于梯度的方法, 242
基于短语的特征, 195
多任务学习, 394
多头注意力, 328
多样性, 379
子词, 351
子词切分, 345
字节对编码, 351
学习率, 242, 247, 309
学习难度, 396
学生模型, 397
完全分析, 75
容量, 360
宽残差网络, 361
对数线性模型, 143
对称化, 129
小批量梯度下降, 243
小批量训练, 333, 363
层归一化, 253
层正则化, 322
广播机制, 233
序列到序列的转换/生成问题, 335
序列化, 179
开放词表, 350
异步更新, 250
张量, 229
循环一致性, 401
循环单元, 264

- 循环神经网络, 264, 289
- 循环神经网络语言模型, 264
- 微调, 393
- 成分分析, 75
- 截断误差, 244
- 扭曲度, 120
- 批量处理, 371
- 批量归一化, 253
- 批量推断, 371
- 批量梯度下降, 243
- 拉格朗日乘数法, 113
- 按元素乘积, 216
- 按元素加法, 214
- 损失函数, 240
- 探测任务, 362
- 推导, 78, 140, 165
- 推断, 61, 91, 366
- 搜索错误, 366
- 教师模型, 396
- 数乘, 215
- 数值微分, 244
- 数据增强, 391
- 数据并行, 249, 311, 363
- 数据清洗, 345
- 数据过滤, 347
- 数据选择, 347
- 数据驱动, 23
- 无参考答案的评价, 32
- 时延, 366
- 最小规则, 188
- 最小错误率训练, 152
- 最左优先推导, 79
- 有向超图, 196
- 有指导的训练, 239
- 有标注数据, 239
- 有监督的训练, 239
- 期望最大化, 115
- 期望频次, 115
- 未登录词, 69, 345
- 机器翻译, 18
- 权重, 219
- 权重调优, 152
- 束剪枝, 158, 172
- 束搜索, 314
- 条件概率, 53
- 极大似然估计, 68
- 标准化, 345
- 标注数据, 61
- 标签平滑, 333, 357
- 标量, 213
- 树到串翻译规则, 179
- 树到树翻译规则, 179
- 树库, 81
- 树片段, 180
- 格搜索, 154
- 梯度下降方法, 242
- 梯度消失, 251
- 梯度爆炸, 251
- 梯度裁剪, 252
- 概念, 122
- 概念单元, 122
- 概率, 52
- 概率上下文无关文法, 80
- 概率分布函数, 53
- 概率密度函数, 53
- 模型压缩, 396
- 模型参数, 239
- 模型并行, 312, 363
- 模型训练, 152
- 模型错误, 366
- 欠翻译, 377
- 欧几里得范数, 218
- 正则化, 254, 354
- 歧义, 79

- 残差网络, 253
残差连接, 322, 329
泛化, 355
注意力分布, 398
注意力权重, 266, 303
流畅度, 88
消歧, 79
深度学习, 208
混淆网络, 381
源语言, 17
- 点乘注意力, 326
熵, 57
特征工程, 283
特征提取, 321
独热编码, 267
生成, 291
生成式模型, 142
生成模型, 83
目标函数, 240
目标语言, 17
直方图剪枝, 158
相互适应, 358, 389
相对熵, 58
矩阵, 213
短句惩罚因子, 34
短语, 134
短语切分, 139
短语对, 139
短语抽取, 144
短语结构分析, 74
短语表, 148
短连接, 330
神经机器翻译, 275
神经网络, 207
神经语言模型, 261
空对齐, 105
立方剪枝, 174
- 端到端学习, 211
符号主义, 209
符号微分, 245
系统偏置, 130
系统融合, 378
繁衍率, 118, 374
线性化, 179
线性变换, 216
线性映射, 216
线搜索, 153
组合, 198
组合性翻译, 134
终结符, 75
编码-解码注意力子层, 322
编码器-解码器, 30
编码器-解码器框架, 283
编码器-解码器模型, 283
缺陷, 124
网络结构搜索技术, 404
翻译中回译, 402
翻译候选, 89, 156
翻译假设, 157
- 联合概率, 53
胶水规则, 165
自下而上的分析, 170
自信息, 57
自动微分, 245
自回归模型, 314
自回归翻译, 373
自注意力子层, 321
自注意力机制, 266, 320
舍入误差, 244
范数, 217
表示学习, 211, 270, 291
表达式膨胀, 245
表面信息, 362
衰减, 248

- 覆盖度模型, 156
- 解码, 91, 99, 366
- 计算图, 238
- 训练, 61, 91, 239
- 训练数据集合, 240
- 记忆更新, 296
- 词, 59
- 词对齐, 96, 104
- 词对齐连接, 96
- 词嵌入, 268, 291, 323, 362
- 词汇化翻译概率, 147
- 词法分析, 59
- 语义信息, 362
- 语法信息, 362
- 语言, 79
- 语言建模, 67
- 语言模型, 67
- 课程学习, 365
- 调优集合, 152
- 调序, 148
- 贝叶斯法则, 56
- 质量评价, 32
- 贪婪搜索, 313
- 超边, 196
- 跨度, 169
- 跳接, 253
- 转置, 214
- 输出, 297
- 边缘概率, 54
- 边缘集合, 187
- 过拟合, 254, 333
- 过拟合问题, 354
- 过翻译, 377
- 连接主义, 209
- 迭代式回译, 392
- 退化, 375
- 适定的, 354
- 逐渐预热, 310
- 遗忘, 296
- 重排序, 368, 379
- 长短时记忆, 295
- 门循环单元, 298
- 阈值剪枝, 158
- 阶, 229
- 降噪, 355
- 随机事件, 52
- 随机变量, 52
- 随机梯度下降, 243
- 集成学习, 359
- 非对称的词对齐, 105
- 非终结符, 75
- 非自回归模型, 314
- 非自回归翻译, 374
- 预处理, 51
- 预热, 332
- 预终结符, 75
- 预训练, 393