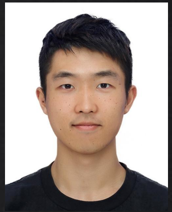
DL Chatbot seminar
Day 02

Text Classification with CNN / RNN

HELLO!

I am Jaemin Cho



- Vision & Learning Lab @ SNU
- NLP / ML / Generative Model
- Looking for Ph.D. / Research programs

You can find me at:

- heythisischo@gmail.com
- j-min
- J-min Cho
- Jaemin Cho

Today We will cover

- ✘ CNN for Text Classification
  - PyTorch Tutorial
- ✘ RNN for Text Classification
- ✘ Advanced CNN/RNN Architectures for NLP

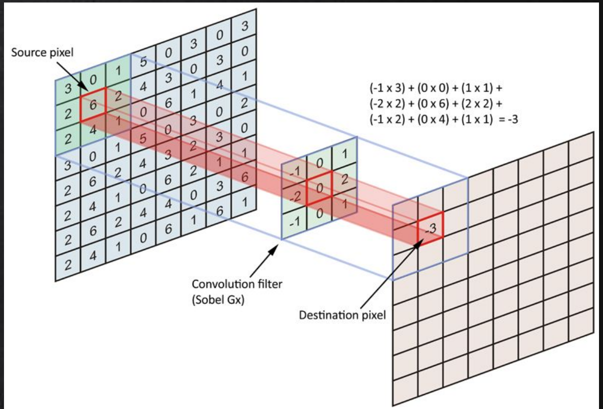
1.

CNN for Text Classification

Word-CNN
Dynamic-CNN
Char-CNN
Very Deep CNN

Convolutional Neural Networks

- Convolution 연산
  - 영상 처리 분야에서 가장 뛰어난 feature extractor
    - Activation이 시신경과 비슷한 Gabor Filter 와 유사
  - Parameter Sharing



- 3가지 특징
  - Local Connectivity
    - 얼굴을 볼 때 눈, 코, 입 등 각 부분을 둘러보고 정보를 종합함
    - 문장을 읽을 때 각 단어들의 의미 및 문맥을 파악 후 내용을 종합함
  - Few Parameters
    - Convolution Filter 재활용 => Fully Connected Network 보다 훨씬 적은 Parameter
  - Parallelization
    - 문장의 각 단어에 대해 독립적으로 연산 가능
    - 순차적으로 적용해야 하는 RNN / Viterbi 계열 연산에 비해 효율적

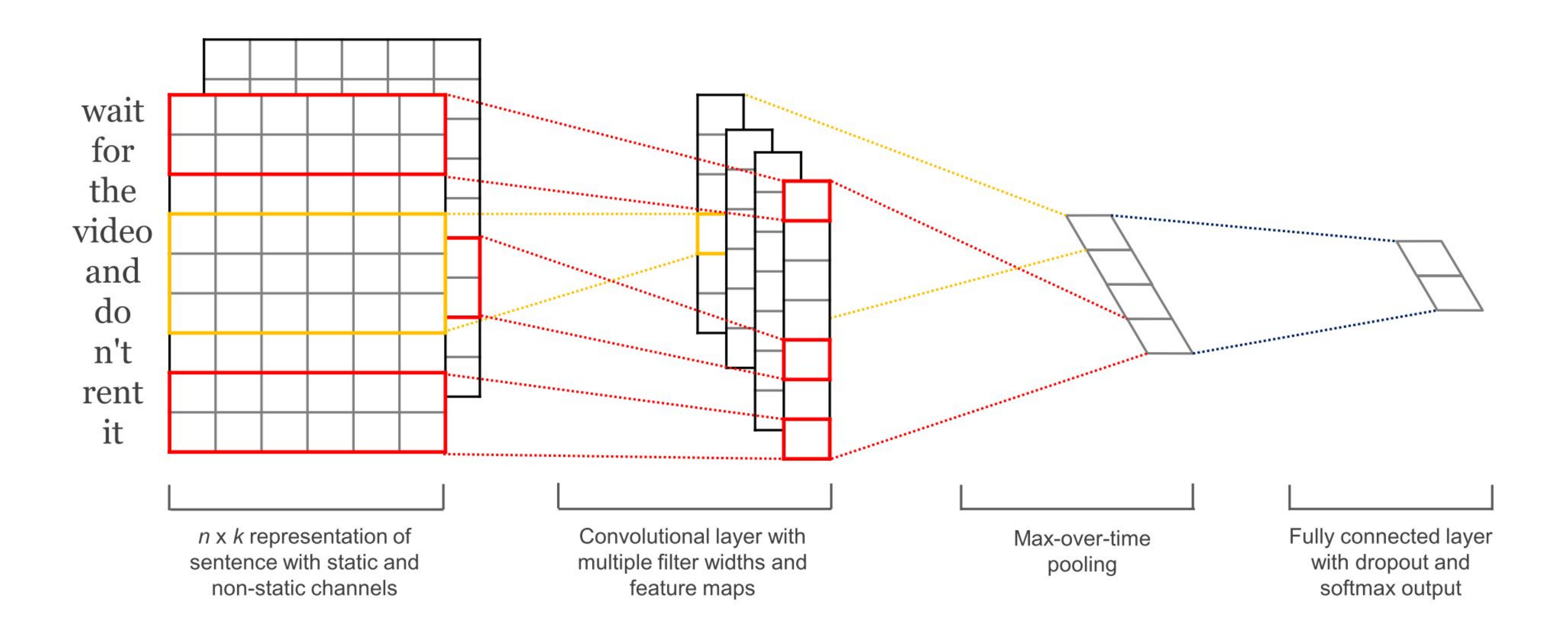
CNN IN NLP

- N-gram 모델링
  - 단어 벡터들의 평균으로 문장 벡터 생성
    - 단어 순서 무시 => (축구가 야구보다 재밌다 vs 야구가 축구보다 재미있다)

- CNN filter를 이용해서 단어 벡터들로 문장 의미 합성 모델링
  - f(n개의 단어) = 문장
  - 다양한 크기의 filter를 사용해서 두 단어 / 세 단어 / 네 단어 등을 모델링
  - 각 filter는 특정한 feature 를 추출한다고 생각할 수 있음
    - ex) 이 3단어에 긍정적인 의미가 있는가? / 이 4 단어에서 제품 이름이 나타나는가?

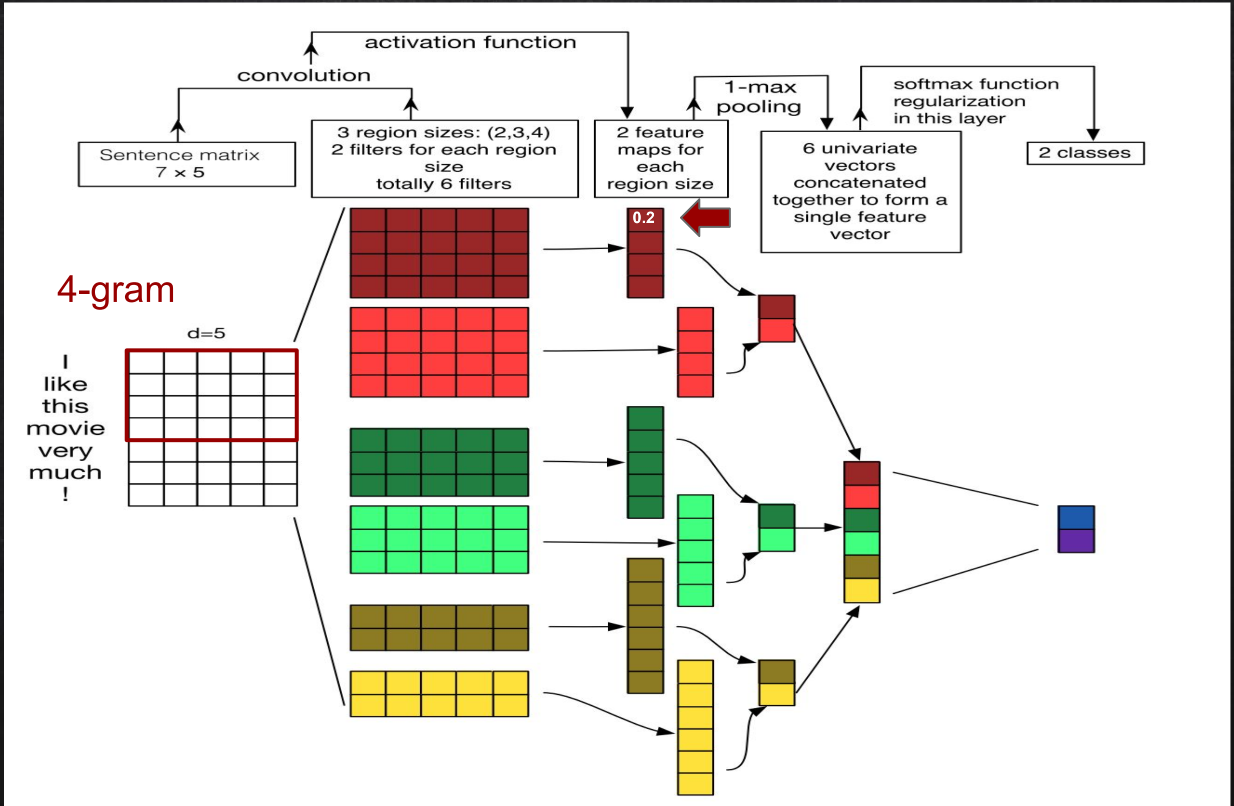
- Padding / Pooling 을 통해 다양한 길이의 문장을 한번에 처리
  - Batch 내의 가장 긴 문장 길이에 맞게 padding => 문장 representation의 길이 통일
  - Convolution 연산 후 필요에 따라 Max-pooling

Word-CNN



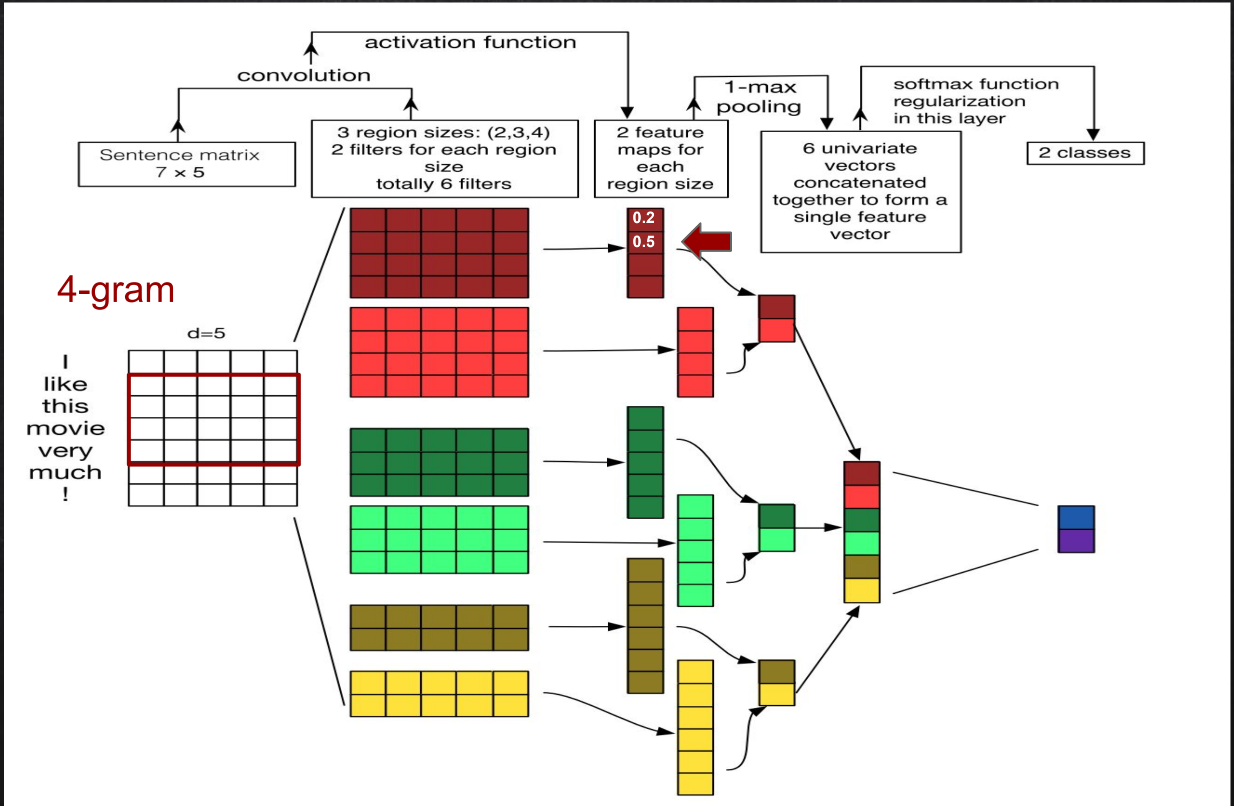
"Convolutional neural networks for sentence classification" (2014)

Word-CNN



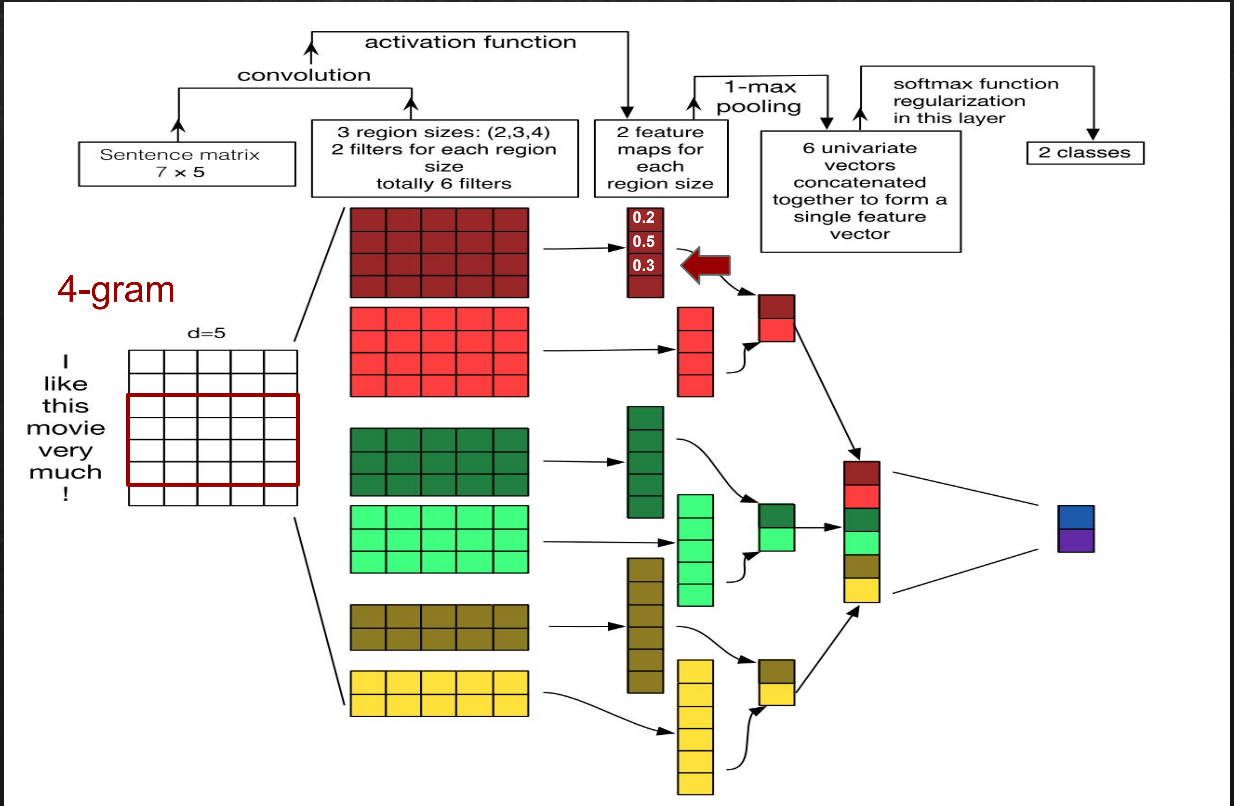
"A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification" (2015)

Word-CNN



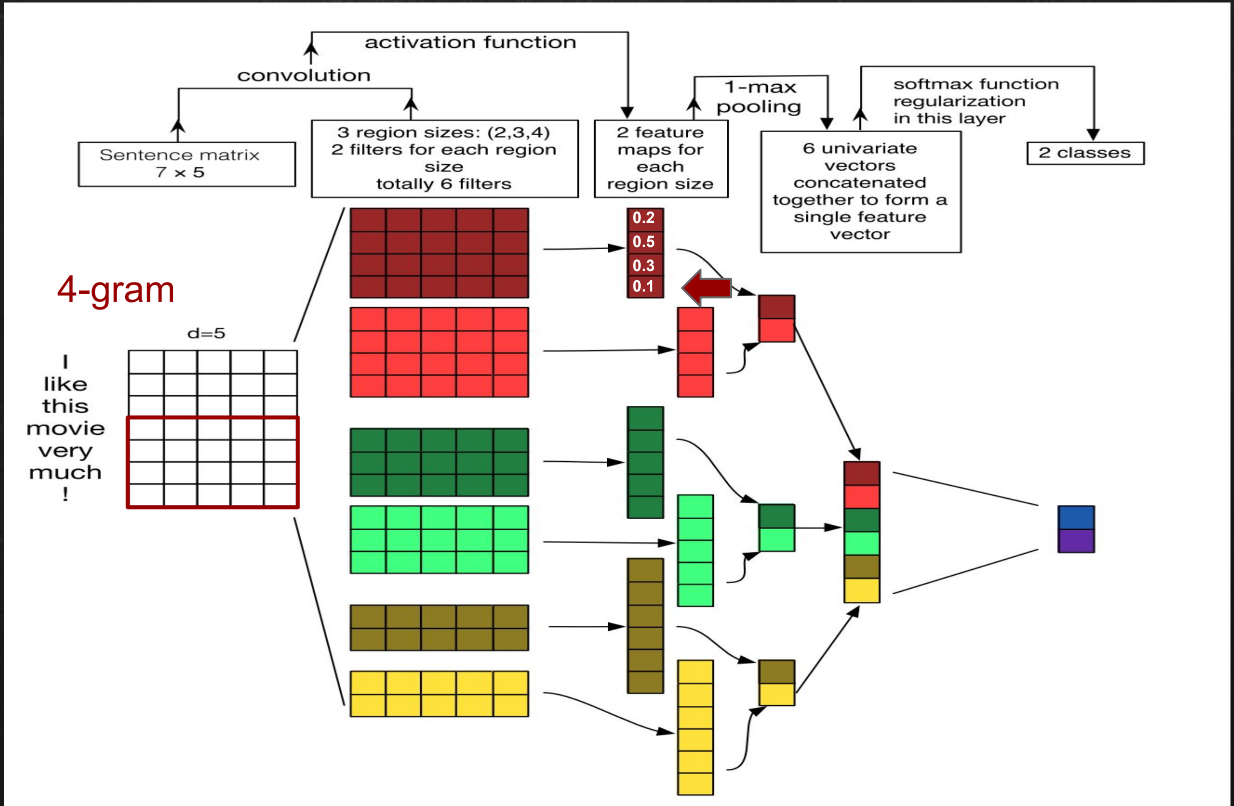
"A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification" (2015)

Word-CNN



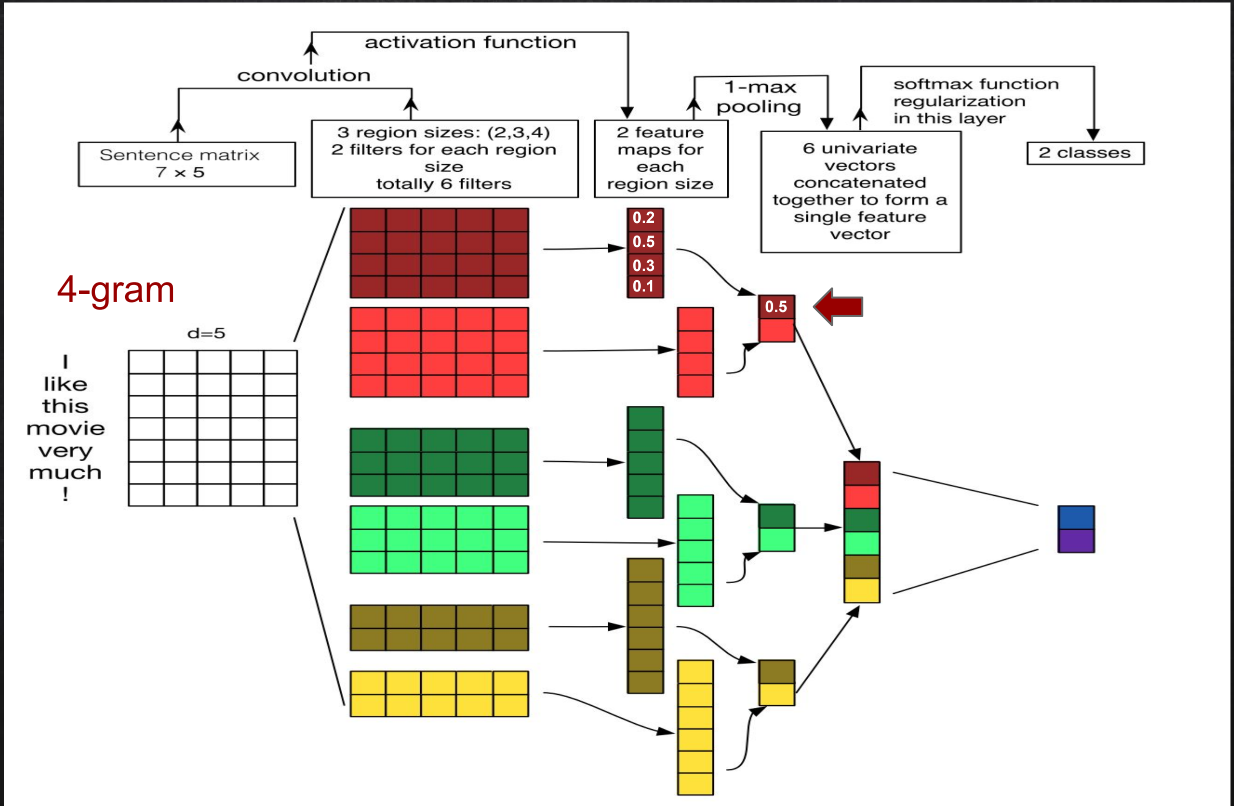
"A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification" (2015)

Word-CNN



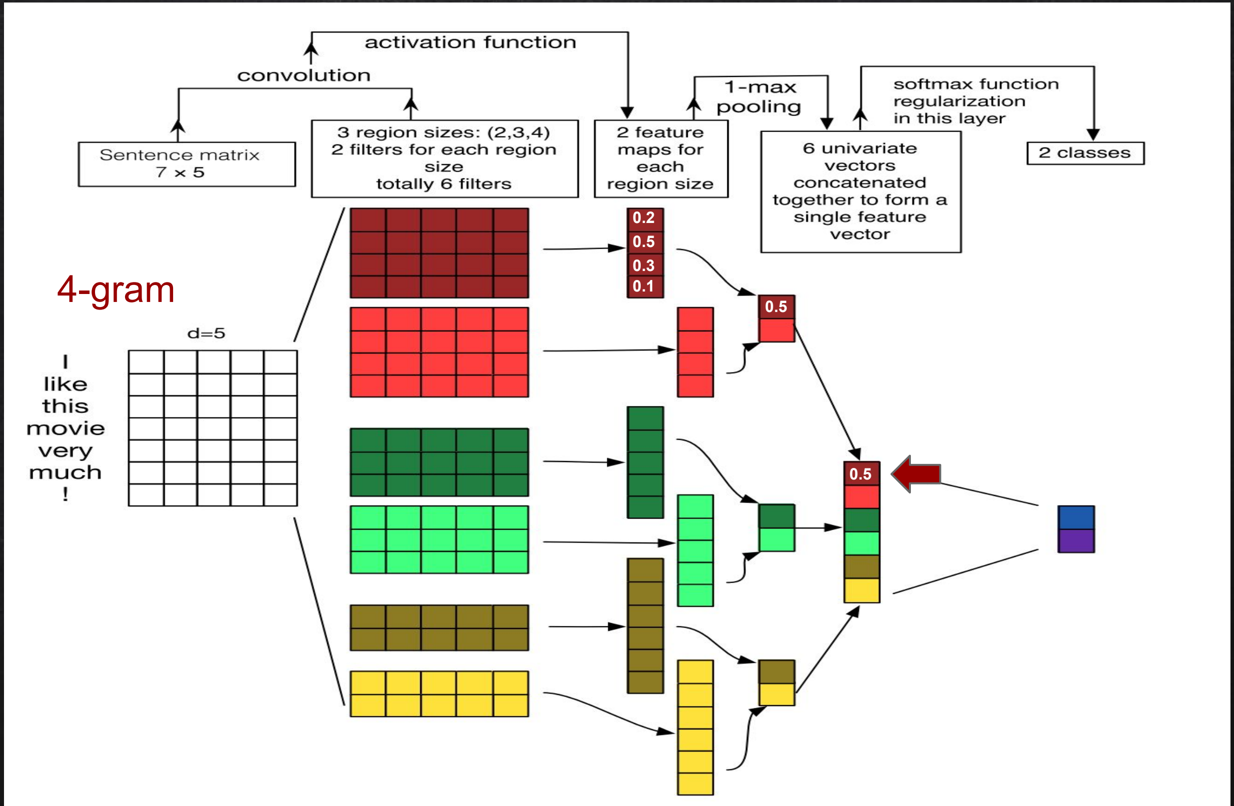
"A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification" (2015)

Word-CNN



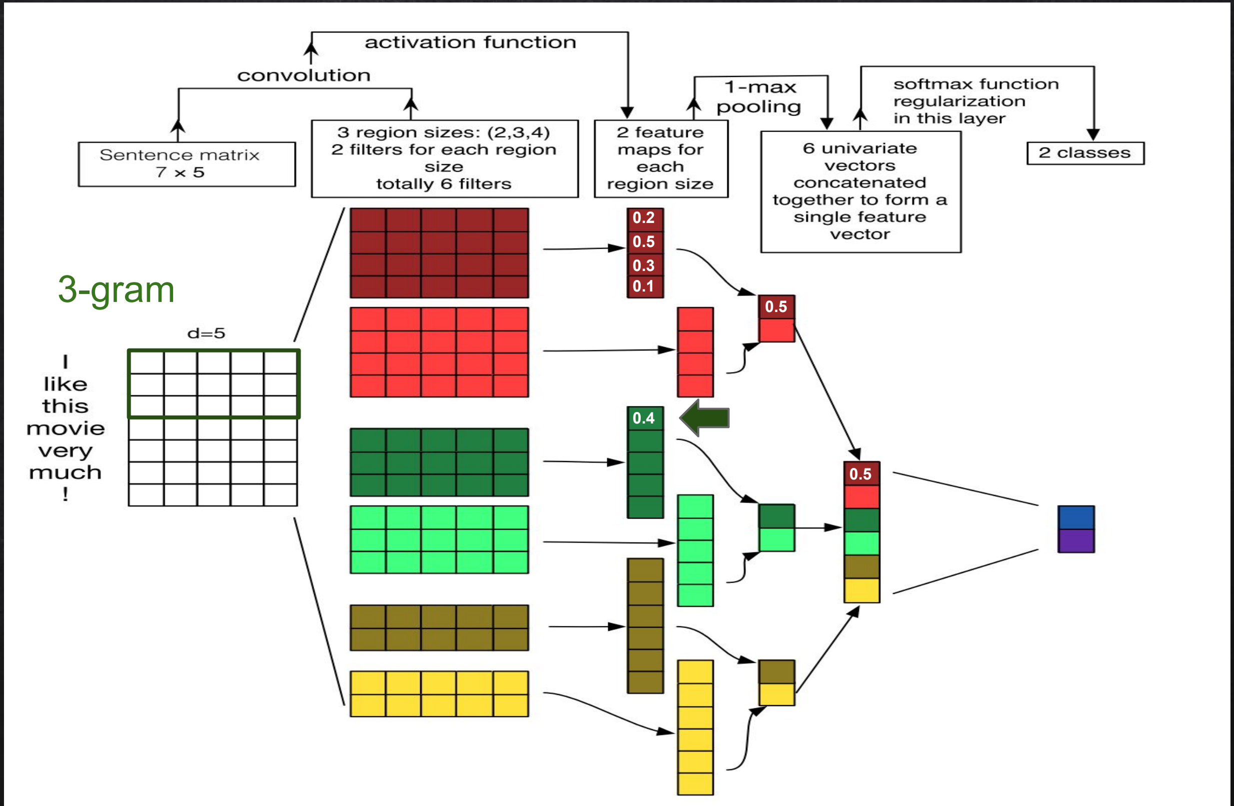
"A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification" (2015)

Word-CNN



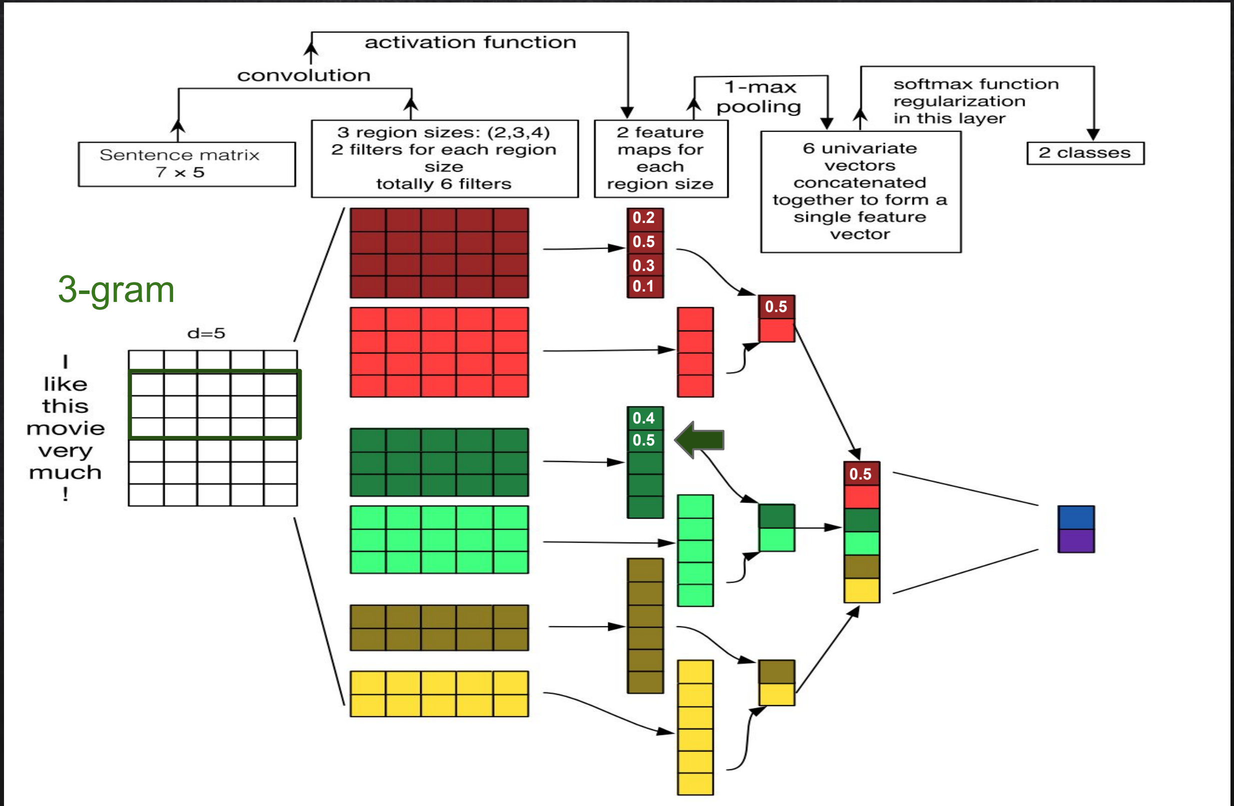
"A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification" (2015)

Word-CNN



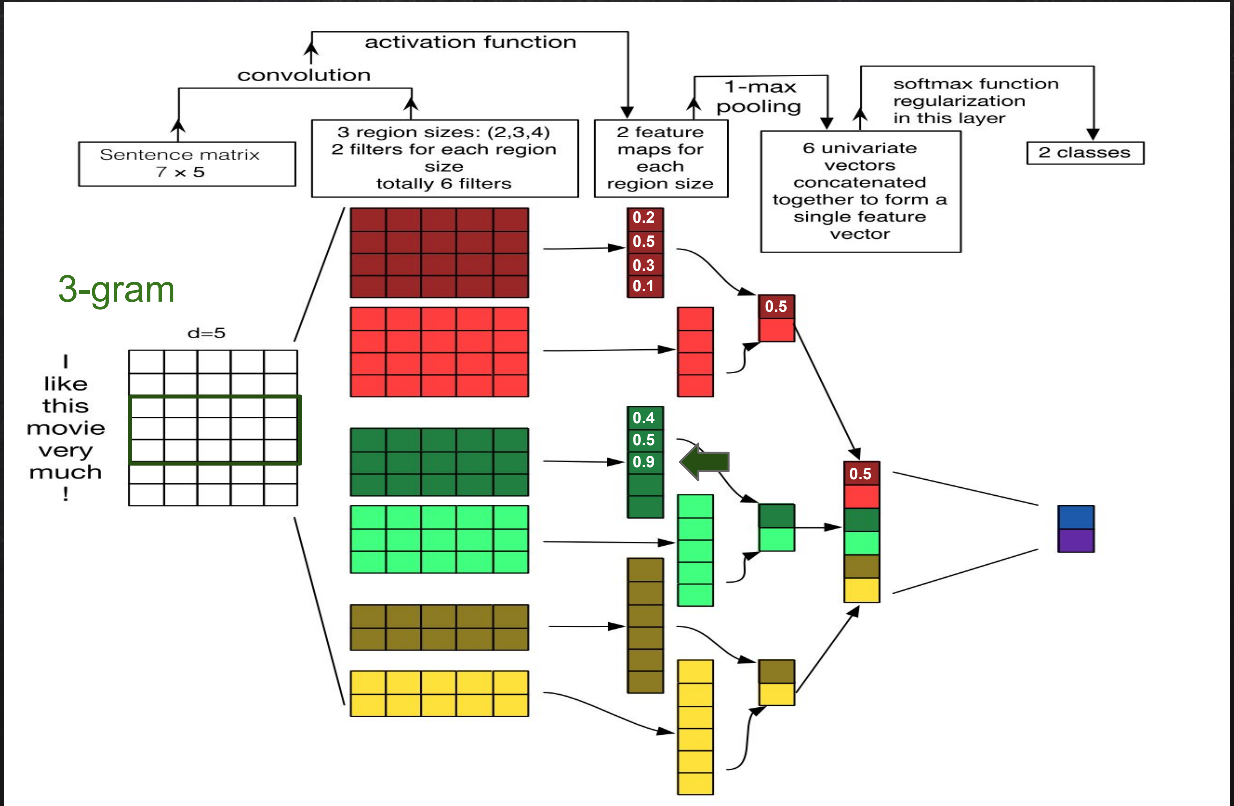
“A Sensitivity Analysis of (and Practitioners’ Guide to) Convolutional Neural Networks for Sentence Classification” (2015)

Word-CNN



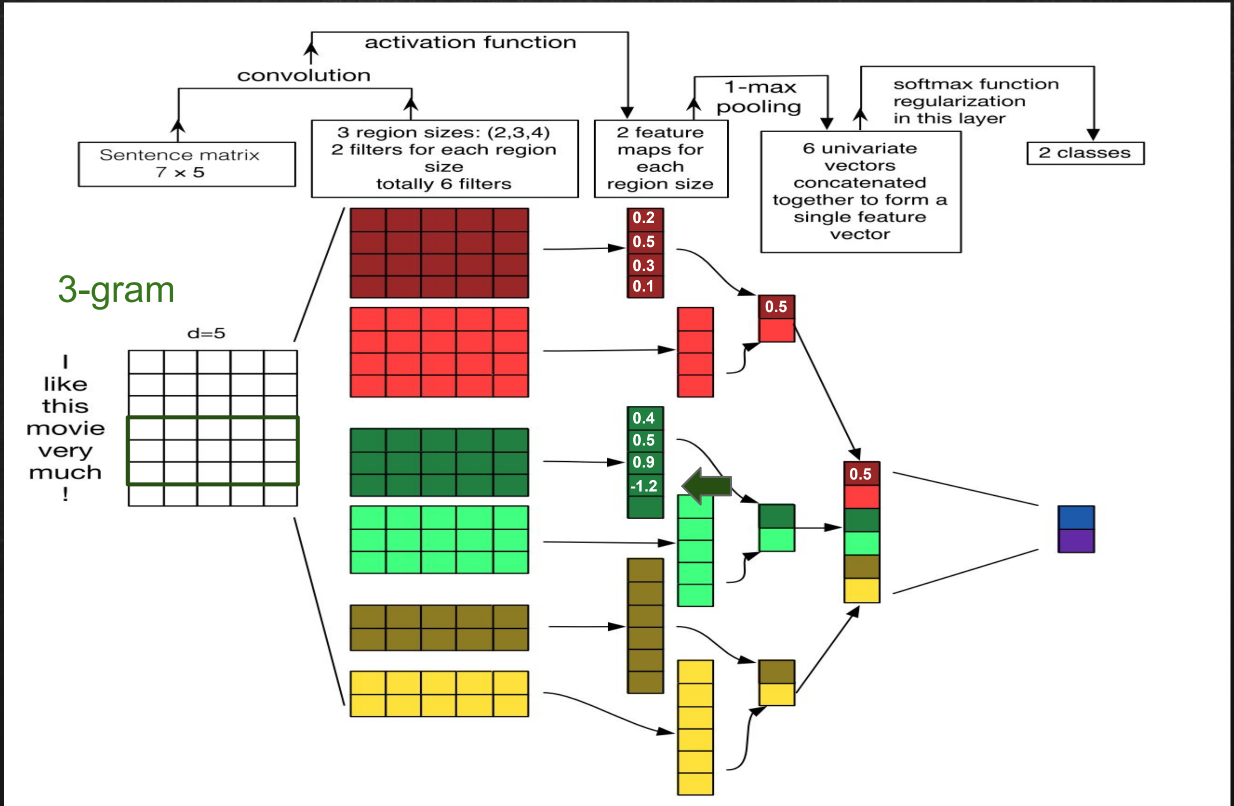
"A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification" (2015)

Word-CNN



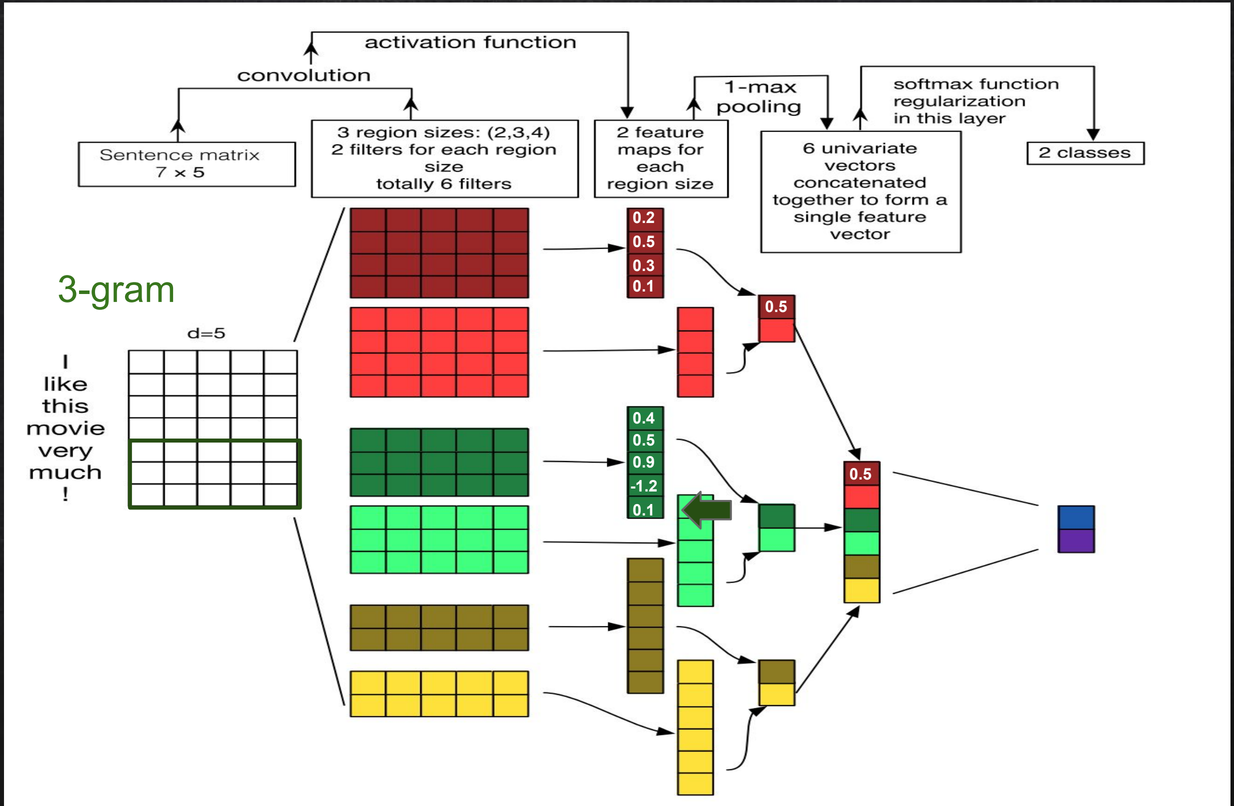
"A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification" (2015)

Word-CNN



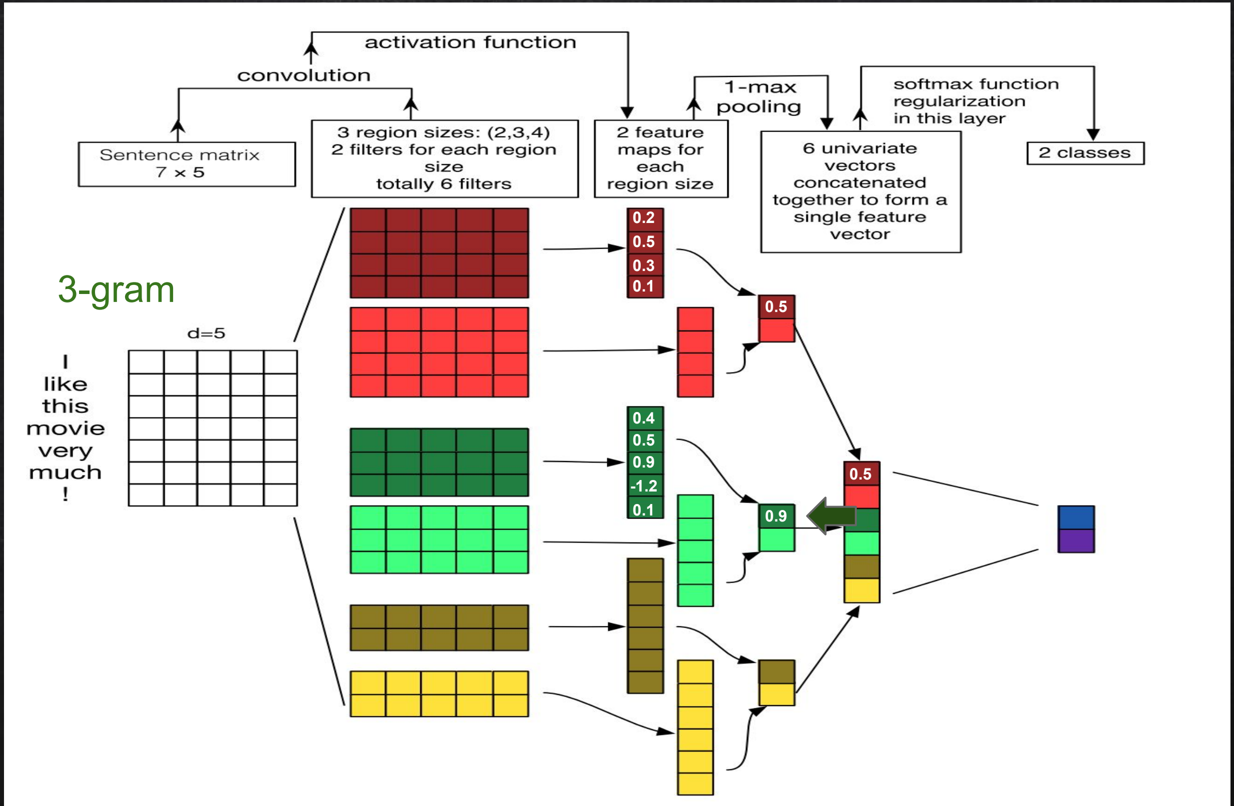
“A Sensitivity Analysis of (and Practitioners’ Guide to) Convolutional Neural Networks for Sentence Classification” (2015)

Word-CNN



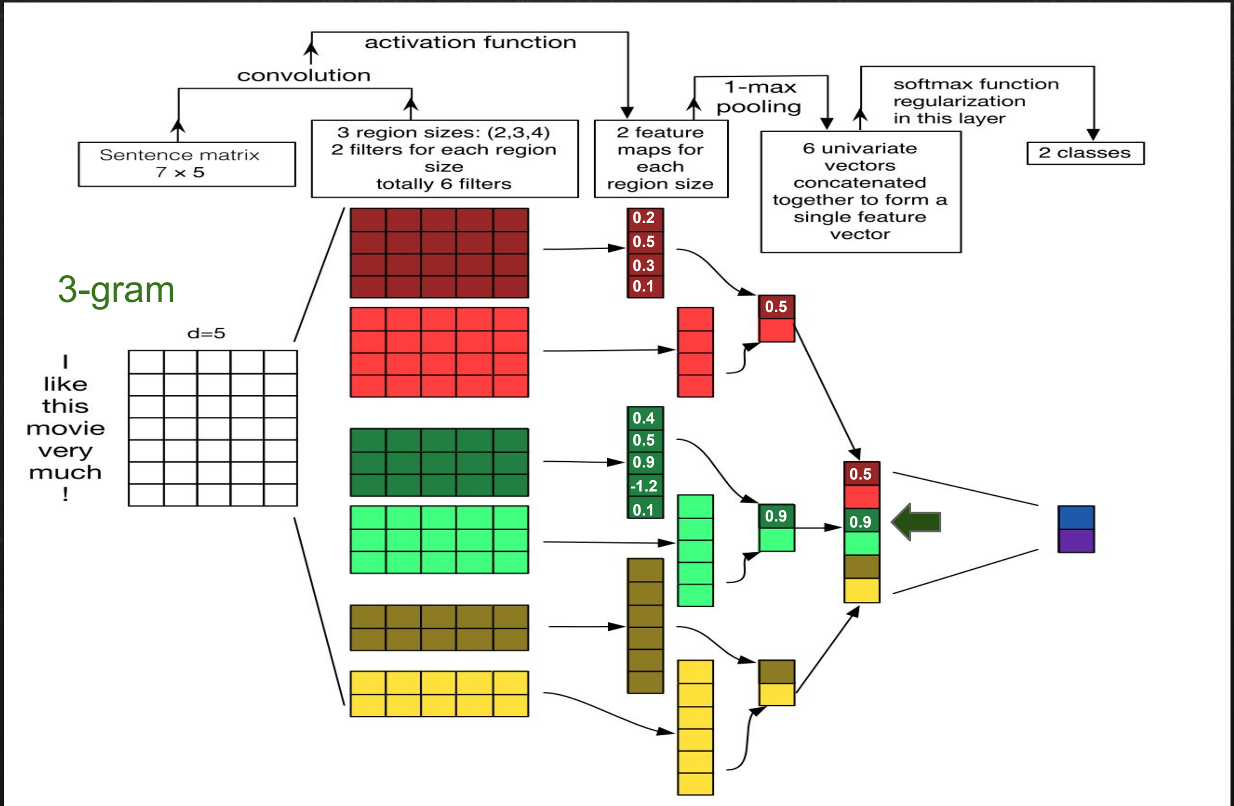
"A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification" (2015)

Word-CNN



"A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification" (2015)

Word-CNN

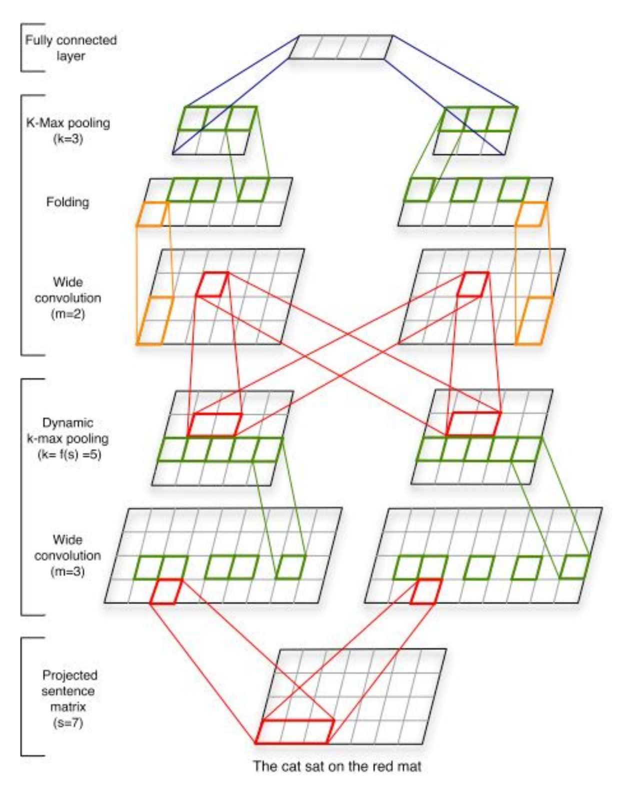


"A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification" (2015)

Dynamic CNN (DCNN)

- Dynamic K-pooling
  - Pooling 시 1 단어만 추출하면 정보 손실이 있을 수 있음
  - 2 단어 이상을 다음 CNN layer로 넘겨주자
  - $$k_l = \max(\, k_{top},\ \lceil \frac{L-l}{L} s \rceil \,)$$
  - PyTorch max-k pooling

- 긴 문장일 때 효과적
  - 단어를 한 번이 아닌 순차적으로 압축



Max-3 pooling

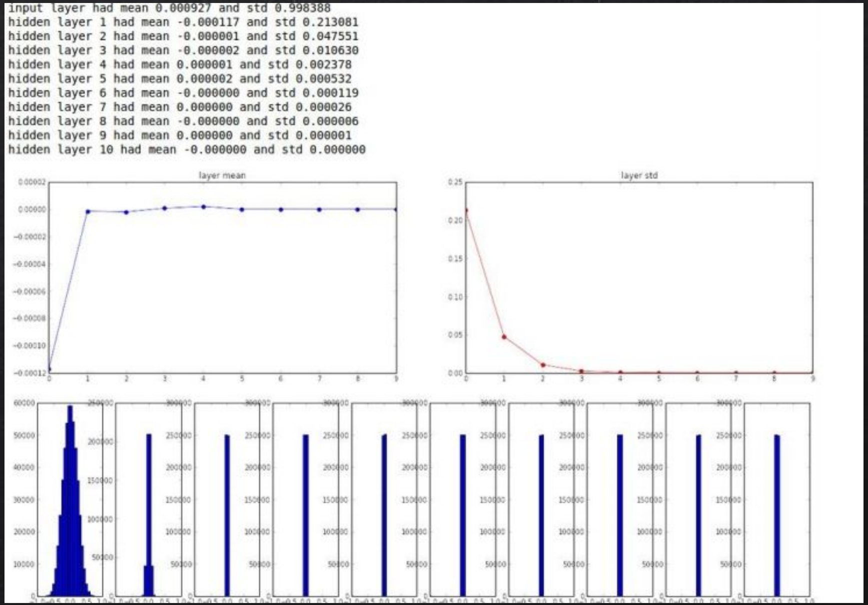
Max-5 pooling

"A Convolutional Neural Network for Modelling Sentences" (2014)

Can we stack CNN deeply?

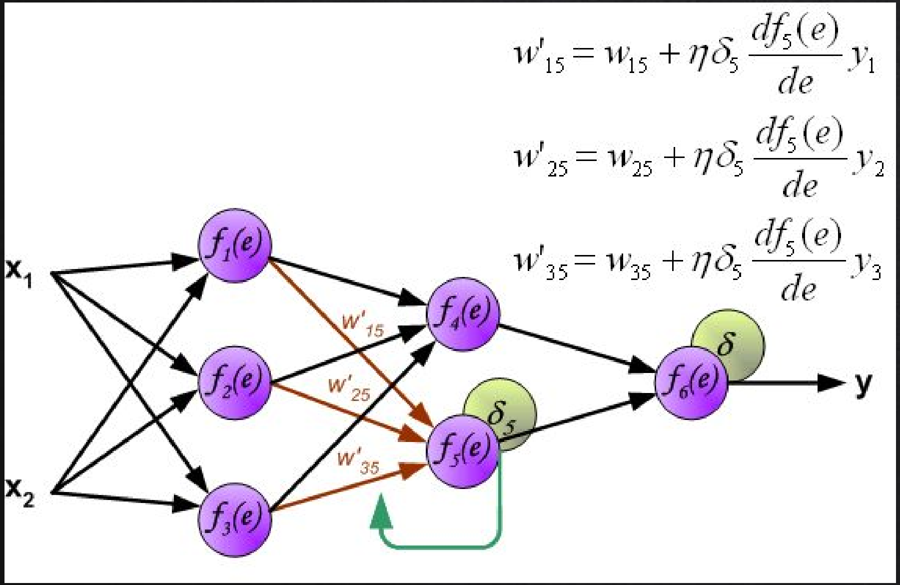
Vanishing / Exploding Gradient

- ✘ 원인: Internal Covariate Shift
  - ○ Network의 각 층의 activation마다 input의 distribution이 달라지는 현상
- ✘ ex) 10-layer NN + sigmoid 의 각 층의 activation



Vanishing / Exploding Gradient

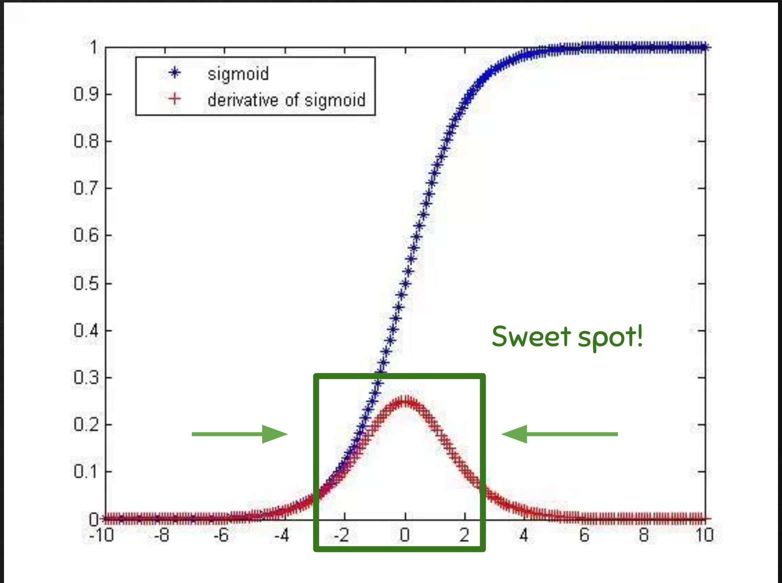
Gradient 전파 => weight 업데이트



Gradient 가 작음 => 업데이트가 일어나지 않음

=> activation의 gradient 를 크게!

=> 평균을 0으로, 분산을 작게!



Gradient 0 Gradient 0.25 Gradient 0

How to avoid Vanishing / Exploding Gradient

- ✘ Proper initialization (weight 초기 값을 적절하게 설정)
  - Initialize weight within proper scale
  - Ex) Xavier initialization

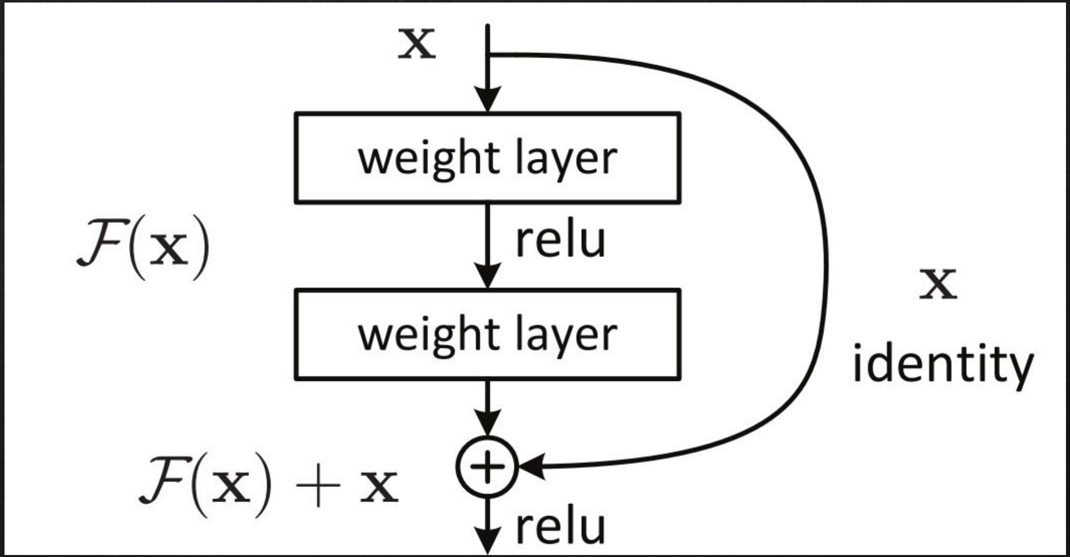
- ✘ Skip Connection (gradient Flow를 여러 갈래로 만들어서 하나가 끊어져도 업데이트가 되게)
  - Loss can be directly propagated to earlier layers
  - Ex) Residual connection, Highway network..

- ✘ Different activation (gradient = 1)
  - Ex) ReLU variants (RELU, ELU, Leaky RELU, SELU..)

- ✘ Normalization (매 layer 의 입력을 직접 정규화)
  - Add extra modules that normalize activations of previous layer
  - Ex) Batch / Layer / Weight/ Instance / Cosine normalizations

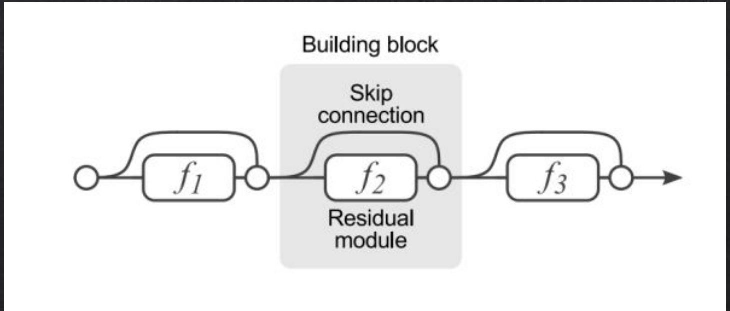
Residual Connection

- Add a skip connection!
- Loss can be back-propagated directly to original input



"Deep Residual Learning for Image Recognition" (2015)

Residual Connection as ensemble



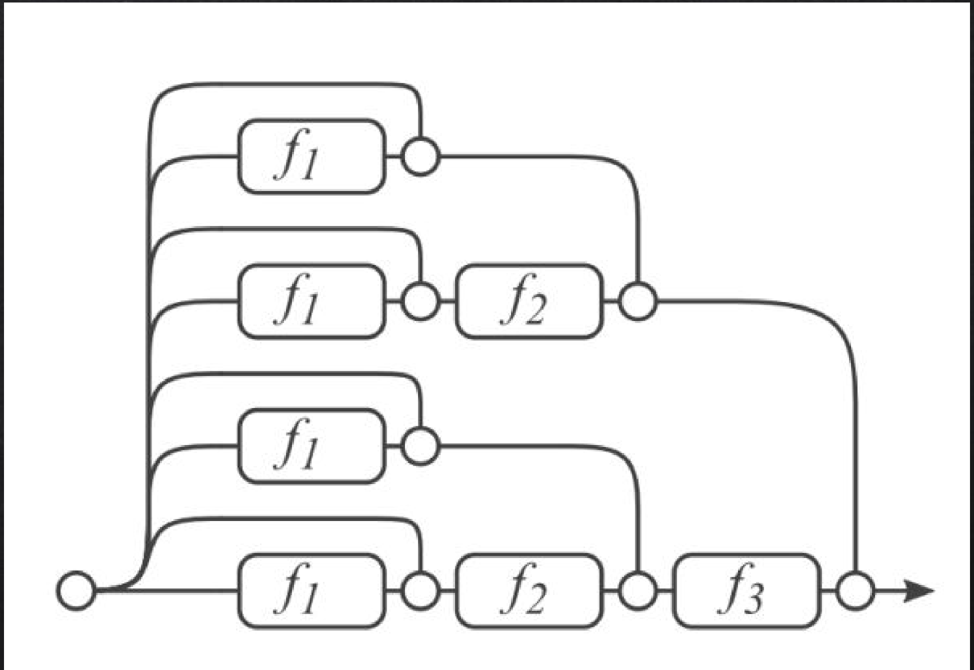
$$y_3 = f_3(y_2) + y_2 \ldots(3)$$

$$y_3 = f_3(f_2(y_1) + y_1) + f_2(y_1) + y_1 \ldots(4)$$

$$y_3 = f_3(f_2(f_1(y_0) + y_0) + f_1(y_0) + y_0) + f_2(f_1(y_0) + y_0) + f_1(y_0) + y_0 \ldots(5)$$

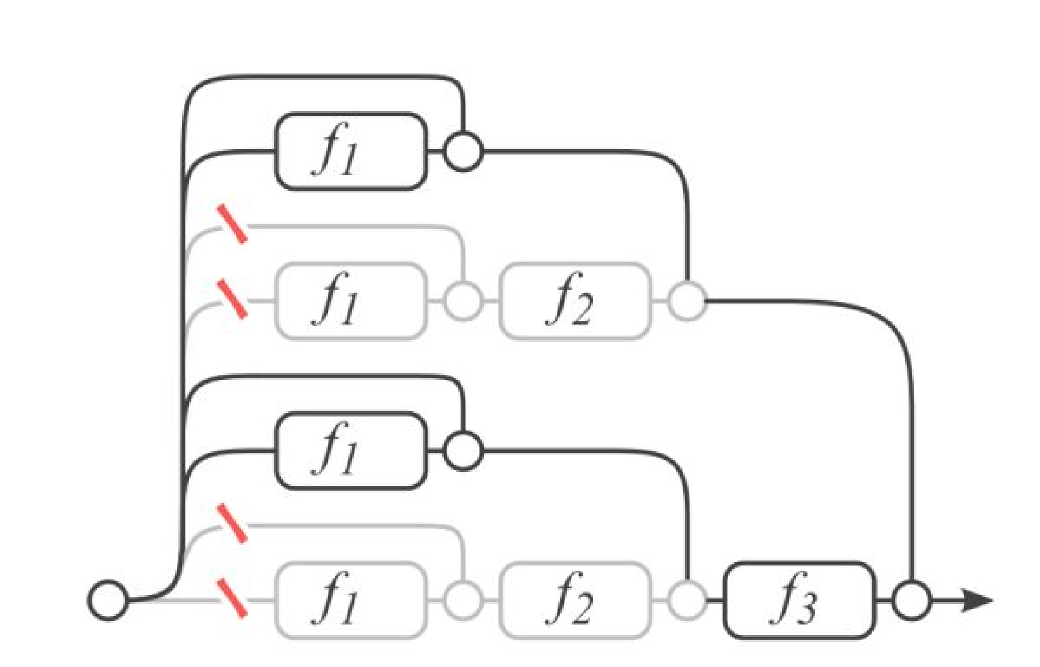
"Residual Networks Behave Like Ensembles of Relatively Shallow Networks" (2016)

Residual Connection as ensemble



"Residual Networks Behave Like Ensembles of Relatively Shallow Networks" (2016)

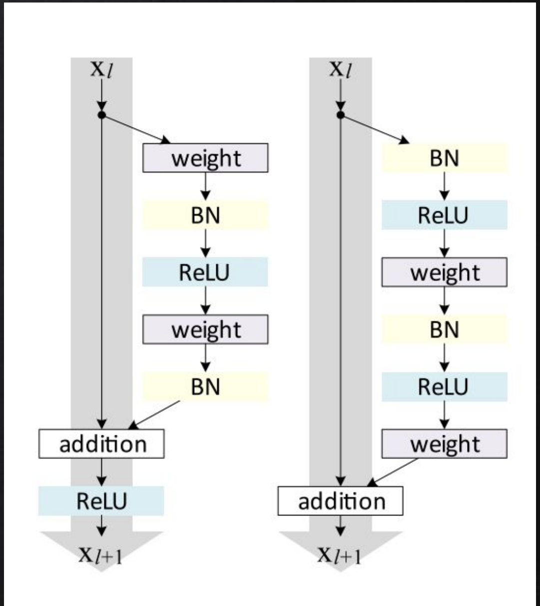
Residual Connection as ensemble



“Residual Networks Behave Like Ensembles of Relatively Shallow Networks” (2016)

Advanced Residual Connection

- What is the most effective combination of submodules?
  - Residual connection, RELU, Batch Norm...
- "Pre-activation"



(a) original (b) proposed

"Identity Mappings in Deep Residual Networks" (2016)

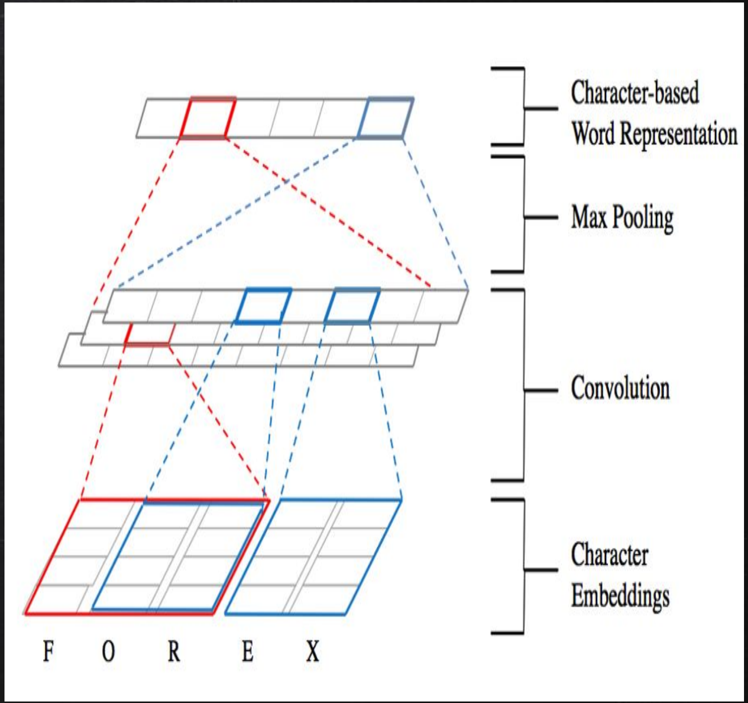
Char-CNN for text classification

- Word-level 모델링의 단점
  - Out-of-Vocabulary => 모두 <unk>으로 뭉뚱그려 처리

- 문자/자소 단위로 모델링하면!
  - Training set에서 보지 못했던 단어가 출현하더라도
  - 문자/자소 벡터의 조합으로 단어 벡터를 생성

- 구현 시 Word-CNN에서 네트워크 형태는 바뀔 필요 없음
  - Tokenization / Vocabulary 만 수정하면 됨
    - 단어 단위 => 문자/자소 단위

- 한글은 가능한 문자의 조합이 너무 다양함
  - 감 => 'ㄱ+ㅏ+ㅁ' 처럼 자소로 분리



Char-CNN for text classification

- ✘ W2v: pretrained word2vec
- ✘ LK: 훈련시킨 단어 임베딩
- ✘ Full: 대소문자 구분
- ✘ Th: Thesaurus 사용

- ✘ 작은 데이터셋에는 n-gram TFIDF가 강세
- ✘ Word-CNN / Char-CNN 간에는 큰 차이 없음
  - Parameter가 훨씬 적어서 효율적

Test Error (낮을수록 좋음)

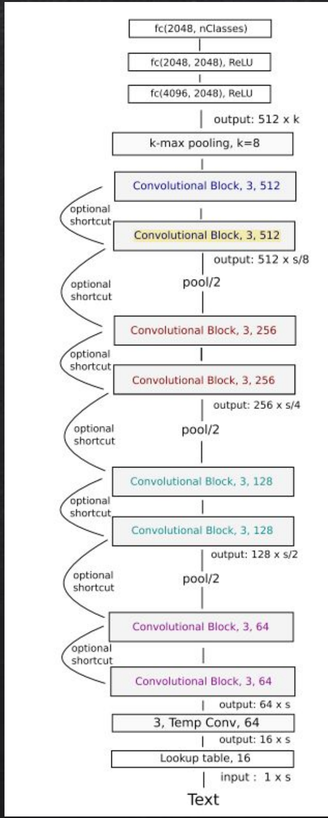
12만 문장 ← 데이터 크기 → 360만 문장

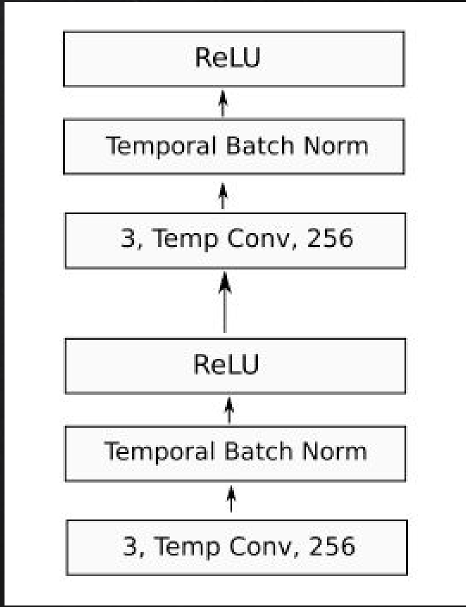
Model	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
BoW	11.19	7.15	3.39	7.76	42.01	31.11	45.36	9.60
BoW TFIDF	10.36	6.55	2.63	6.34	40.14	28.96	44.74	9.00
ngrams	7.96	2.92	1.37	**4.36**	43.74	31.53	45.73	7.98
ngrams TFIDF	**7.64**	**2.81**	**1.31**	4.56	45.20	31.49	47.56	8.46
Bag-of-means	**16.91**	**10.79**	**9.55**	**12.67**	**47.46**	**39.45**	**55.87**	**18.39**
LSTM	13.94	4.82	1.45	5.26	41.83	29.16	40.57	6.10
Lg. w2v Conv.	9.92	4.39	1.42	4.60	40.16	31.97	44.40	5.88
Sm. w2v Conv.	11.35	4.54	1.71	5.56	42.13	31.50	42.59	6.00
Lg. w2v Conv. Th.	9.91	-	1.37	4.63	39.58	31.23	43.75	5.80
Sm. w2v Conv. Th.	10.88	-	1.53	5.36	41.09	29.86	42.50	5.63
Lg. Lk. Conv.	8.55	4.95	1.72	4.89	40.52	29.06	45.95	5.84
Sm. Lk. Conv.	10.87	4.93	1.85	5.54	41.41	30.02	43.66	5.85
Lg. Lk. Conv. Th.	8.93	-	1.58	5.03	40.52	28.84	42.39	5.52
Sm. Lk. Conv. Th.	9.12	-	1.77	5.37	41.17	28.92	43.19	5.51
Lg. Full Conv.	9.85	8.80	1.66	5.25	38.40	29.90	40.89	5.78
Sm. Full Conv.	11.59	8.95	1.89	5.67	38.82	30.01	40.88	5.78
Lg. Full Conv. Th.	9.51	-	1.55	4.88	38.04	29.58	40.54	5.51
Sm. Full Conv. Th.	10.89	-	1.69	5.42	**37.95**	29.90	40.53	5.66
Lg. Conv.	12.82	4.88	1.73	5.89	39.62	29.55	41.31	5.51
Sm. Conv.	15.65	8.65	1.98	6.53	40.84	29.84	40.53	5.50
Lg. Conv. Th.	13.39	-	1.60	5.82	39.30	**28.80**	40.45	**4.93**
Sm. Conv. Th.	14.80	-	1.85	6.49	40.16	29.84	**40.43**	5.67

"Character-level Convolutional Networks for Text Classification" (2015)

Very Deep CNN (VDCNN)

- ✘ 이전 CNN 텍스트 모델은 최대 6레이어
- ✘ 최초로 'Deep' NN for Text 학습 성공
  - Residual connection 덕분
- ✘ 깊은 모델 성능 향상
  - 29층까지는 깊게 쌓을수록 결과가 좋았다.
  - 49층 모델은 트레이닝 실패
- ✘ SOTA (n-gram TF-IDF) 는 넘지 못함





"Very Deep Convolutional Networks for Natural Language Processing" (2016)

Very Deep CNN (VDCNN)

- ✘ 이전 CNN 텍스트 모델은 최대 6레이어
- ✘ 최초로 'Deep' NN for Text 학습 성공
  - Residual connection 덕분
- ✘ 깊은 모델 성능 향상
  - 29층까지는 깊게 쌓을수록 결과가 좋았다.
  - 49층 모델은 트레이닝 실패
- ✘ SOTA (n-gram TF-IDF) 는 넘지 못함

Table 3: Best published results from previous work. Zhang et al. [20] best results use a Thesaurus data augmentation technique (marked with an *). Xiao and Cho [19] propose a combination of convolution and recurrent layers. n-TFIDF corresponds to the ngrams version of TF-IDF.

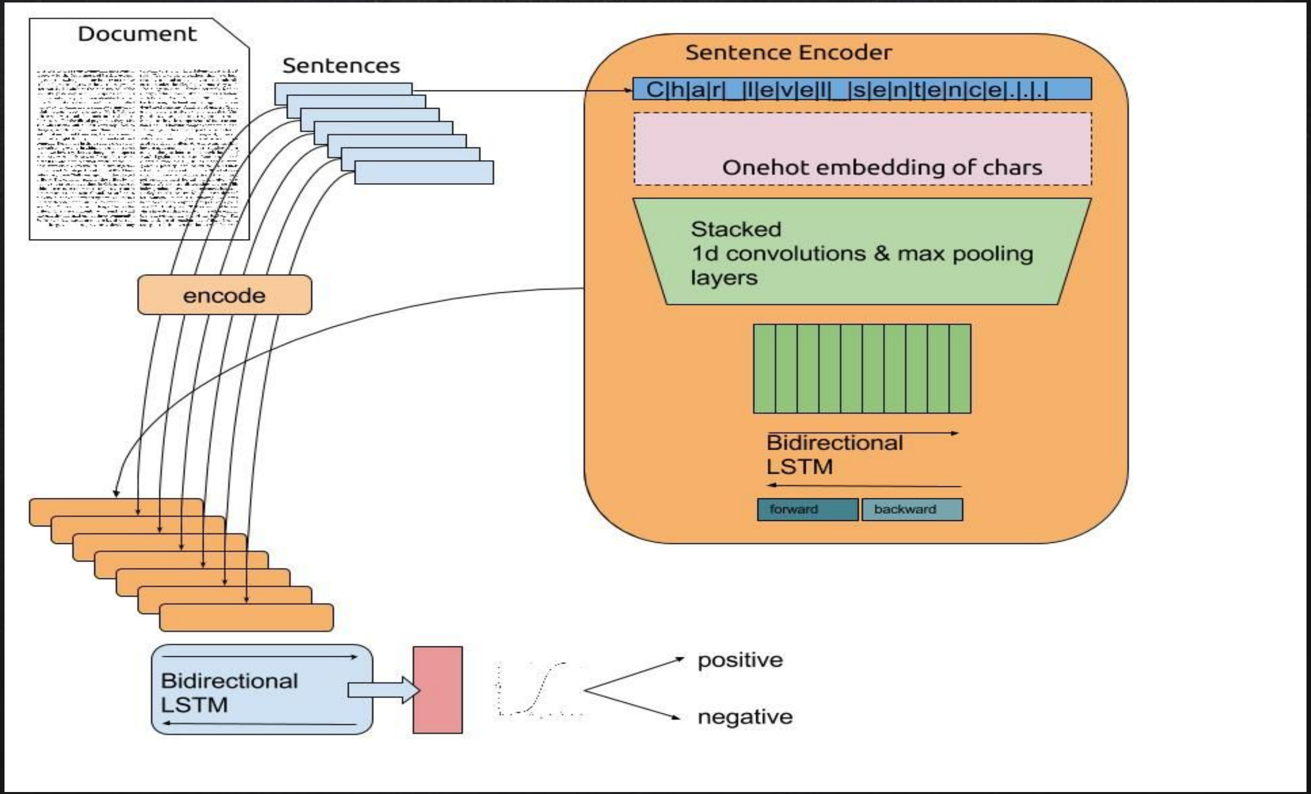
Corpus:	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
Method	n-TFIDF	n-TFIDF	n-TFIDF	ngrams	Conv	Conv+RNN	Conv	Conv
Author	[Zhang]	[Zhang]	[Zhang]	[Zhang]	[Zhang]	[Xiao]	[Zhang]	[Zhang]
Error	7.64	2.81	1.31	4.36	37.95*	28.26	40.43*	4.93*

Table 4: Testing error of our models on the 8 data sets. The deeper the networks the lower the error for all pooling types. No data preprocessing or augmentation is used.

Depth	Pooling	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
9	Convolution	10.17	4.22	1.64	5.01	37.63	28.10	38.52	4.94
9	KMaxPooling	9.83	3.58	1.56	5.27	38.04	28.24	39.19	5.69
9	MaxPooling	9.17	3.70	1.35	4.88	36.73	27.60	37.95	4.70
17	Convolution	9.29	3.94	1.42	4.96	36.10	27.35	37.50	4.53
17	KMaxPooling	9.39	3.51	1.61	5.05	37.41	28.25	38.81	5.43
17	MaxPooling	8.88	3.54	1.40	4.50	36.07	27.51	37.39	4.41
29	Convolution	9.36	3.61	1.36	4.35	**35.28**	27.17	37.58	**4.28**
29	KMaxPooling	**8.67**	**3.18**	1.41	4.63	37.00	27.16	38.39	4.94
29	MaxPooling	8.73	3.36	**1.29**	**4.28**	35.74	**26.57**	**37.00**	4.31

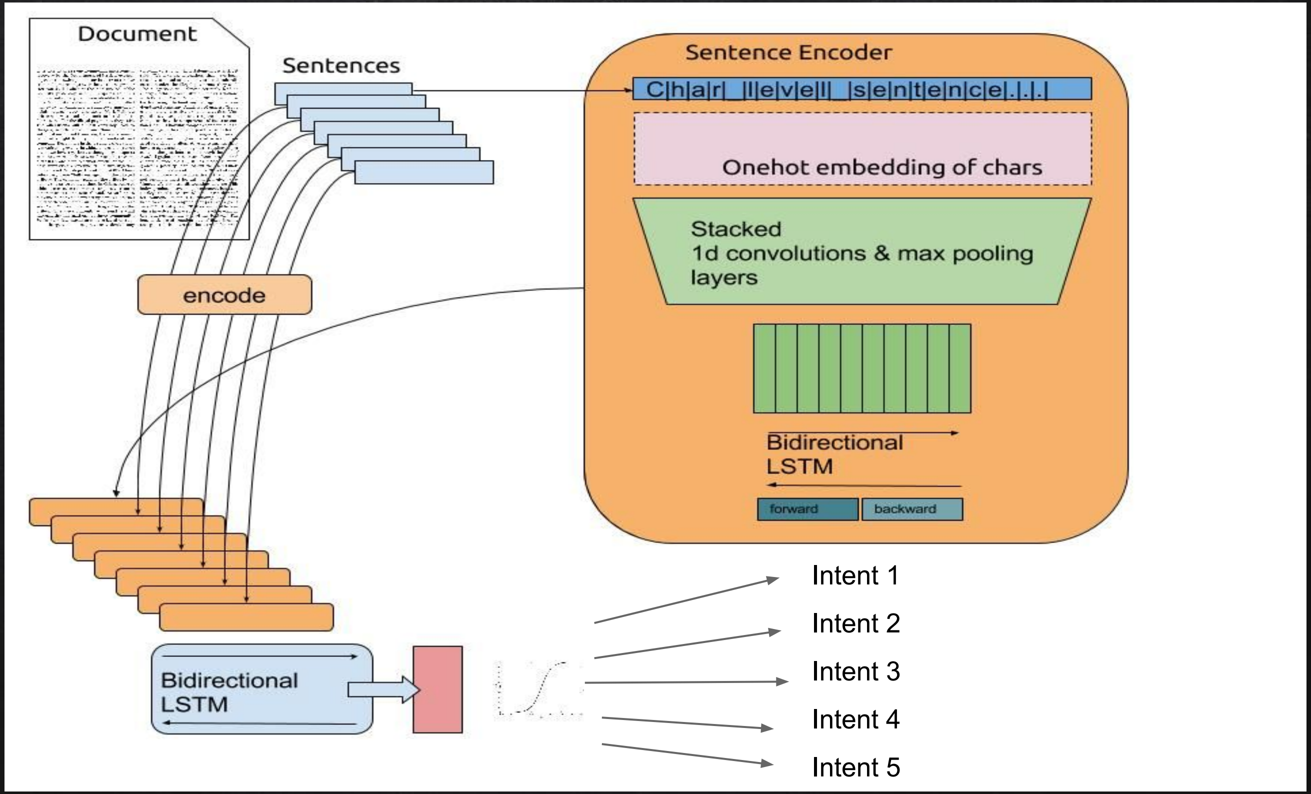
"Very Deep Convolutional Networks for Natural Language Processing" (2016)

Suggestion



"Text Understanding from Scratch" (2015)

Suggestion



“Text Understanding from Scratch” (2015)

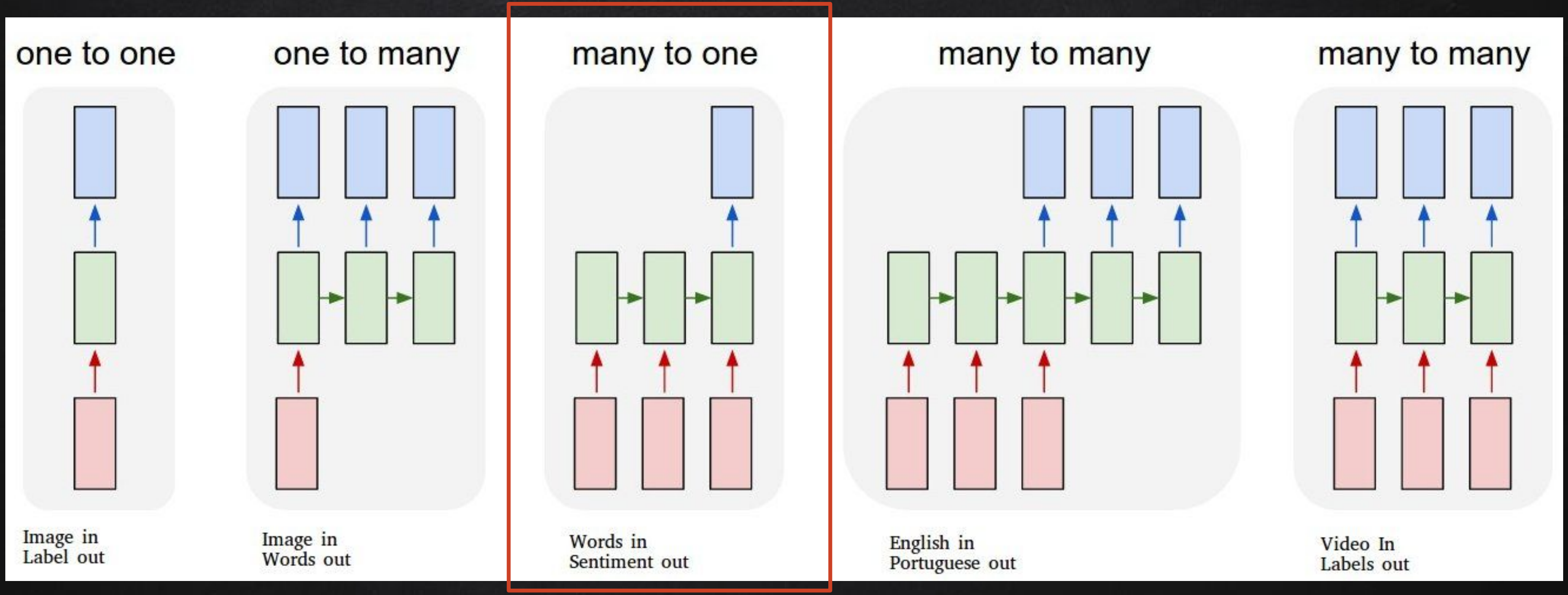
2.

RNN for text Classification

Bidirectional RNN
Recursive Neural Networks
Tree-LSTM
Dual-Encoder LSTM

RNN for text classification

- Many-to-One 모델
- RNN 을 단어/문자 단위로 입력



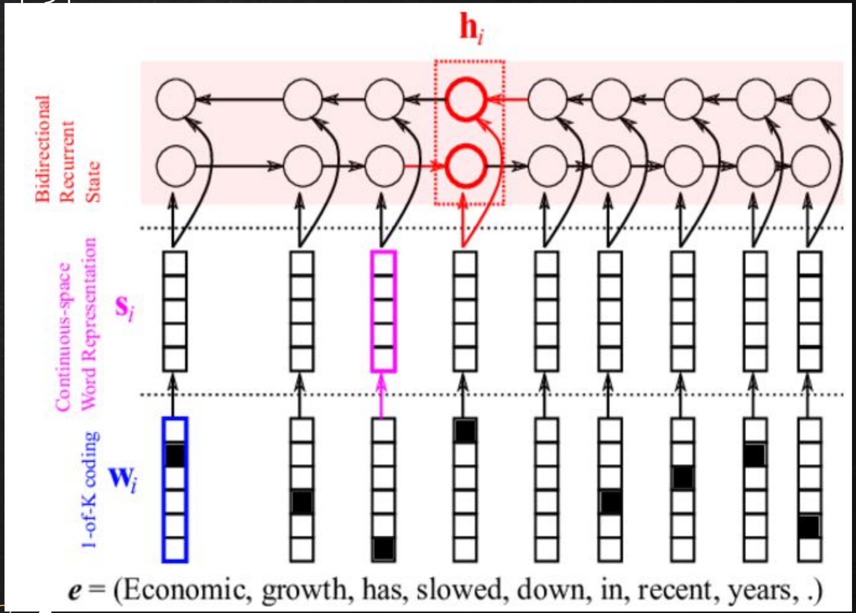
"CS231n: Convolutional Neural Networks for Visual Recognition" (2016)

Bidirectional RNN

- 마지막 state에는 앞부분 단어 정보가 희미해짐
- 문장의 끝에서부터 반대 순서로 한번 더 읽은 정보 추가
- 총 RNN 2개 (Parameter 2배, 출력 차원 2배)

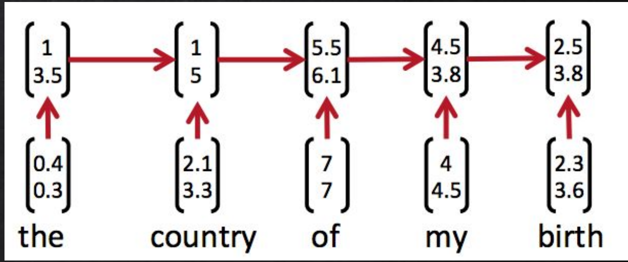
- Ex) 오늘 밥이 맛있다
  - Forward RNN
    - 입력: 오늘 밥이 맛있다
    - 출력: $h^f_{\text{오늘}}$ $h^f_{\text{밥이}}$ $h^f_{\text{맛있다}}$
  - Backward RNN
    - 입력: 맛있다 밥이 오늘
    - 출력: $h^b_{\text{맛있다}}$ $h^b_{\text{밥이}}$ $h^b_{\text{오늘}}$
  - Concatenate
    - $[h^f_{\text{오늘}};h^b_{\text{맛있다}}]$, $[h^f_{\text{밥이}}; h^b_{\text{밥이}}]$, $[h^f_{\text{맛있다}}; h^b_{\text{오늘}}]$

Concatenate



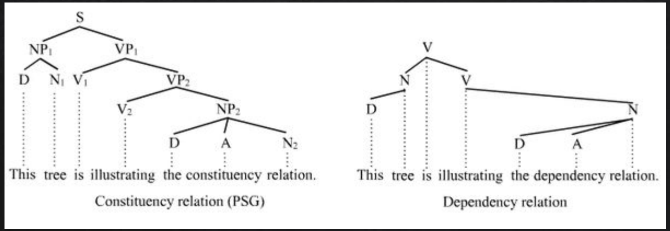
Recursive Neural Networks

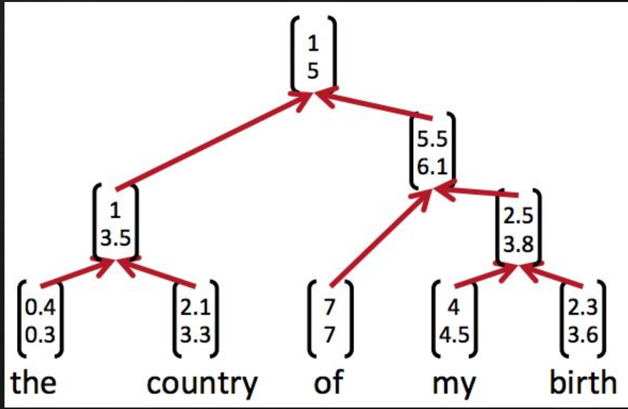
✘ 문장의 단어들은 의미적/문법적 구조를 지님
- ○ 단순히 '왼쪽 => 오른쪽'으로 정보가 전달되지 않음
- ○ 기존 Recurrent NN은 마지막 단어가 강조됨



"Learning to Compose Words into Sentences with Reinforcement Learning"
=> Train parser simultaneously!

✘ Recursive Neural Networks
- ○ 문법적 구조 정보를 추가적으로 이용 (Tree Parser 가 있어야 사용할 수 있음)
- ○ Child node의 벡터로 parent node의 벡터 계산
- ○ Root node의 representation으로 문장 분류
- ○ Constituency parsing / Dependency parsing



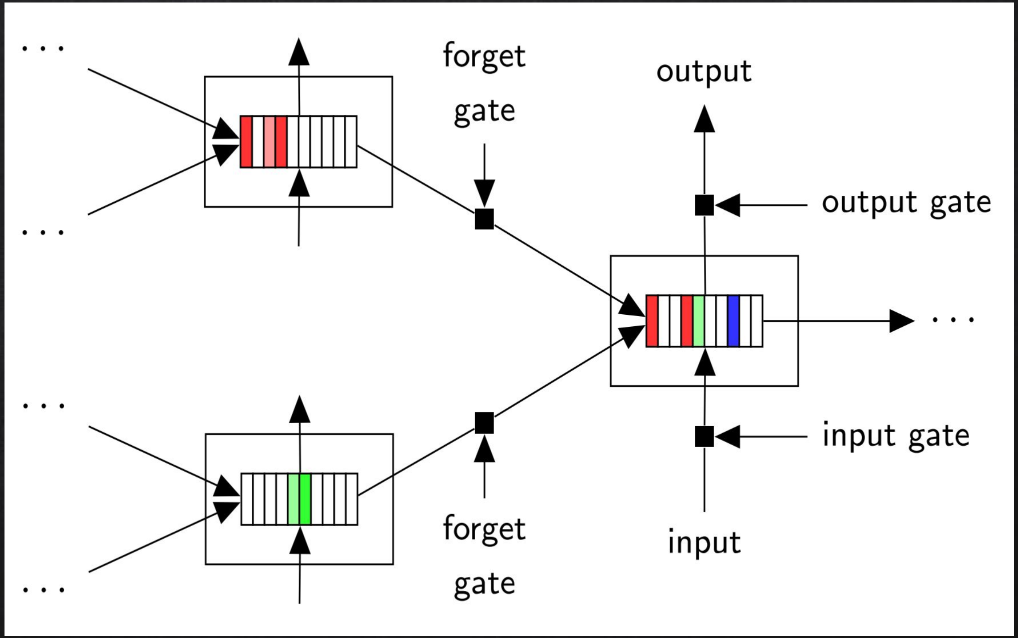


"CS224n: Natural Language Processing with Deep Learning" (2017)

Tree-LSTM

- Tree-LSTM 이전에도 SU-RNN, MV-RNN, RNTN, 등의 Recursive NN 구조가 있었습니다

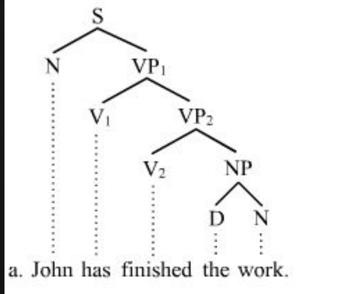
✘ Tree 에서 중간 node의 representation을 LSTM 으로 계산하자!



"Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks" (2015)

Tree-LSTM

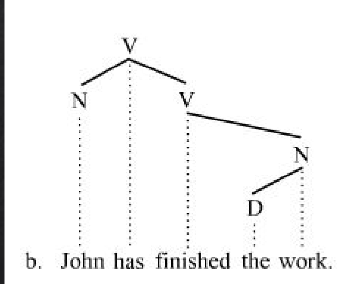
- Constituency parsing
  - Child node 개수 일정
  - 각 node 별로 별개의 parameter 설정



Constituency structure

$$i_j = \sigma\left(W^{(i)}x_j + \sum_{\ell=1}^{N} U_\ell^{(i)} h_{j\ell} + b^{(i)}\right),$$
$$f_{jk} = \sigma\left(W^{(f)}x_j + \sum_{\ell=1}^{N} U_{k\ell}^{(f)} h_{j\ell} + b^{(f)}\right),$$
$$o_j = \sigma\left(W^{(o)}x_j + \sum_{\ell=1}^{N} U_\ell^{(o)} h_{j\ell} + b^{(o)}\right),$$
$$u_j = \tanh\left(W^{(u)}x_j + \sum_{\ell=1}^{N} U_\ell^{(u)} h_{j\ell} + b^{(u)}\right),$$
$$c_j = i_j \odot u_j + \sum_{\ell=1}^{N} f_{j\ell} \odot c_{j\ell},$$
$$h_j = o_j \odot \tanh(c_j),$$

- Dependency parsing
  - Child node 개수 다름
  - 먼저 다 더하고 일반 LSTM처럼 계산



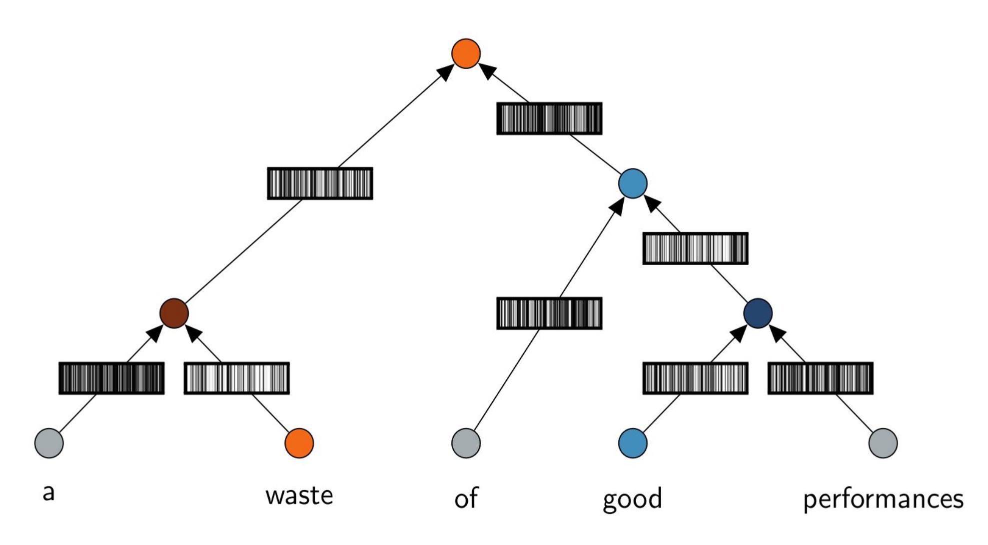
Dependency structure

$$\tilde{h}_j = \sum_{k\in C(j)} h_k,$$
$$i_j = \sigma\left(W^{(i)}x_j + U^{(i)}\tilde{h}_j + b^{(i)}\right),$$
$$f_{jk} = \sigma\left(W^{(f)}x_j + U^{(f)}h_k + b^{(f)}\right),$$
$$o_j = \sigma\left(W^{(o)}x_j + U^{(o)}\tilde{h}_j + b^{(o)}\right),$$
$$u_j = \tanh\left(W^{(u)}x_j + U^{(u)}\tilde{h}_j + b^{(u)}\right),$$
$$c_j = i_j \odot u_j + \sum_{k\in C(j)} f_{jk} \odot c_k,$$
$$h_j = o_j \odot \tanh(c_j),$$

"Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks" (2015)

Tree-LSTM

- Forget Gate Activation
  - 어떤 단어가 감정 분석 시 덜 중요한가?
    - 어떤 단어를 잊어버릴까?
  - 'a' vs 'waste'
  - 'a waste' vs 'of good performance'



"Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks" (2015)

Tree-LSTM

- 영화 리뷰 감정분석 (Stanford Sentiment Treebank)
  - 5-class: 1~5점 중 점수 예측
  - Binary: 긍/부정 예측 (1, 2점 vs 4, 5점)

Method	5-class	Binary
RNTN (Socher et al., 2013)	45.7	85.4
Paragraph-Vec (Le & Mikolov, 2014)	48.7	87.8
Convolutional NN (Kim 2014)	47.4	**88.1**
Epic (Hall et al., 2014)	49.6	–
DRNN (Irsoy & Cardie, 2014)	49.8	86.6
LSTM	46.4	84.9
Bidirectional LSTM	49.1	87.5
Constituency Tree-LSTM	**51.0**	88.0

- Tree-LSTM이 기존 모델들보다 복잡한 문장 성능이 좋음
  - Ex) 이중 부정

The longer the movie goes , the **worse** it gets , **but** it 's actually **pretty good** in the first few minutes .

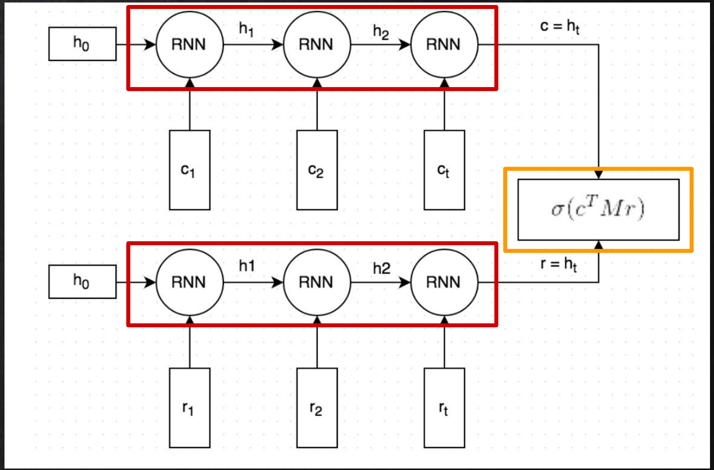
LSTM	**Tree-LSTM**	**Gold**
+	−	−

- 도입 시 고려사항
  - 충분히 빠르고 정확한 **tree parser**가 있는지?
  - 데이터에서 복잡한 문장이 많은지?

“Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks” (2015)

Dual-LSTM

- ✘ 현재 주어진 문장 (context) 에 답하기
  - ○ 미리 대답 문장들을 만들어 놓았음
  - ○ 그 중 score가 가장 높은 문장 출력
- ✘ Score 계산 시 Cosine similarity 대신
  - ○ Bilinear + Sigmoid
- ✘ Siamese Network
  - ○ Context 문장과 Response 문장에 같은 Parameter 공유
- ✘ Evaluation: Recall@k
  - ○ 정답 문장과 무작위로 선택한 문장들을 score 순위로 정렬
  - ○ 정답 문장의 score가 k위 안에 드는가?



Method	TF-IDF	RNN	LSTM
1 in 2 R@1	65.9%	76.8%	**87.8%**
1 in 10 R@1	41.0%	40.3%	**60.4%**
1 in 10 R@2	54.5%	54.7%	**74.5%**
1 in 10 R@5	70.8%	81.9%	**92.6%**

"The Ubuntu Dialogue Corpus: A Large Dataset for Research in Unstructured Multi-Turn Dialogue Systems" (2015)

3.

Advanced CNN/RNN Architecture

Char-CNN for Language Modeling
Character and Word embedding
CNN+RNN: QRNN
Fast RNN: SRU
Dilated Causal Convolution: ByteNet
Depthwise Separable Convolution: SliceNet
Sequential Tagging: Bi-directional LSTM-CNNs-CRF

Char-CNN for Language Modeling

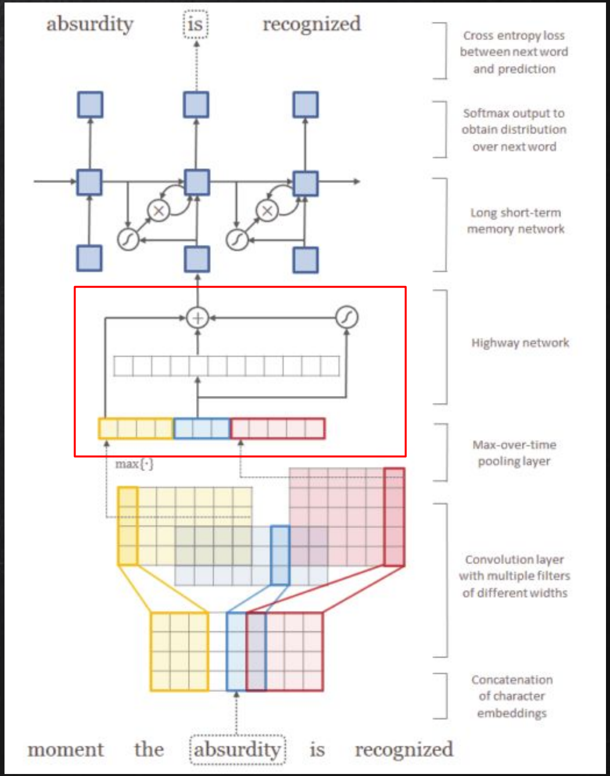
- Classification 대신 Language Modeling
- Highway Network
  - Residual Connection 대신 사용
  - 후속 레이어와 이전 레이어를 gating

$$\mathbf{z} = g(\mathbf{W}\mathbf{y} + \mathbf{b})$$

$$\mathbf{t} = \sigma(\mathbf{W}_T\mathbf{y} + \mathbf{b}_T)$$

$$\mathbf{z} = \mathbf{t} \odot g(\mathbf{W}_H\mathbf{y} + \mathbf{b}_H) + (\mathbf{1} - \mathbf{t}) \odot \mathbf{y}$$

Perplexity => 낮을수록 좋음	LSTM-Char	
	Small	Large
No Highway Layers	100.3	84.6
One Highway Layer	92.3	79.7
Two Highway Layers	90.1	78.9
One MLP Layer	111.2	92.6



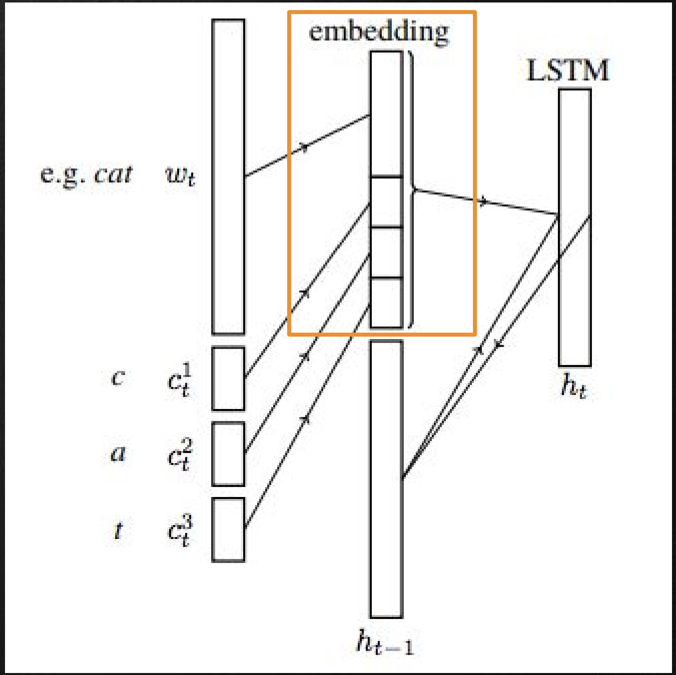
"Character-aware neural language models" (2016)

Character and Word embedding

- ✘ 단어 벡터 / 문자 벡터 중 선택할 것이 아니라 둘 다 사용하자!

- ✘ 문자 단위 representation을 만들 때
  - ○ 단어 벡터 & 단어를 구성하는 문자 벡터 모두 사용
    - ■ Concatenation
    - ■ Sum / Average
    - ■ Highway Network
    - ■ More complex NN

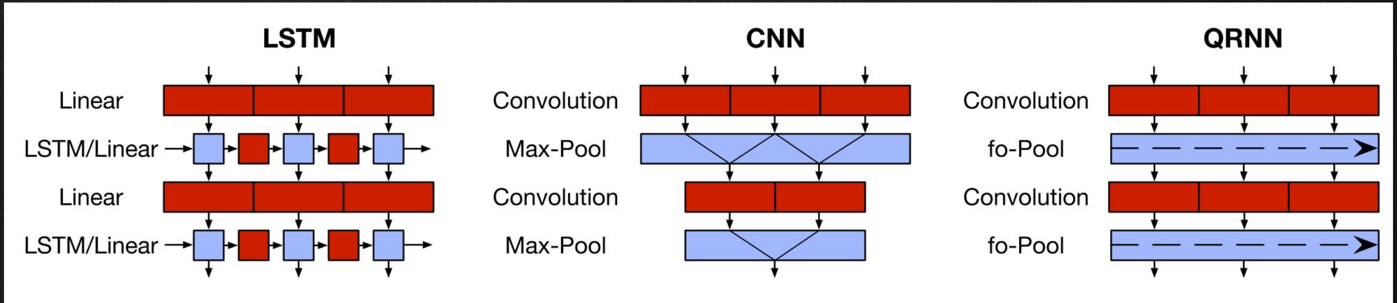
- ✘ Rule of Thumb
  - ○ 단어 벡터
    - ■ pretrained (Word2Vec/GloVe)
    - ■ 데이터 크롤링 => FastText / SpaceText
  - ○ 문자 벡터
    - ■ Char-CNN 학습



"Character-Word LSTM Language Models" (2017)

Quasi-RNN (QRNN)

- 병렬화가 가능한 CNN + 순서 정보를 인코딩하는 RNN 의 장점 결합
- 같은 크기의 LSTM에 비해 적은 Parameter & 빠른 학습



$$\mathbf{z}_t = \tanh(\mathbf{W}_z^1 \mathbf{x}_{t-1} + \mathbf{W}_z^2 \mathbf{x}_t)$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_f^1 \mathbf{x}_{t-1} + \mathbf{W}_f^2 \mathbf{x}_t)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o^1 \mathbf{x}_{t-1} + \mathbf{W}_o^2 \mathbf{x}_t).$$

Matrix multiplication
=> Element-wise product

$$\mathbf{Z} = \tanh(\mathbf{W}_z * \mathbf{X})$$
$$\mathbf{F} = \sigma(\mathbf{W}_f * \mathbf{X})$$
$$\mathbf{O} = \sigma(\mathbf{W}_o * \mathbf{X}),$$

f-pooling $\mathbf{h}_t = \mathbf{f}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t$

fo-pooling
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t.$$

ifo-pooling
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{z}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t.$$

"Quasi-Recurrent Neural Networks" (2016)

Quasi-RNN (QRNN)

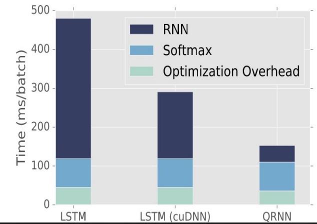
- 감정 분석
  - IMDB 긍/부정 구분
  - D.C.: DenseNet

Model	Time / Epoch (s)	Test Acc (%)
BSVM-bi (Wang & Manning, 2012)	–	91.2
2 layer sequential BoW CNN (Johnson & Zhang, 2014)	–	92.3
Ensemble of RNNs and NB-SVM (Mesnil et al., 2014)	–	92.6
2-layer LSTM (Longpre et al., 2016)	–	87.6
Residual 2-layer bi-LSTM (Longpre et al., 2016)	–	90.1
Our models		
Deeply connected 4-layer LSTM (cuDNN optimized)	480	90.9
Deeply connected 4-layer QRNN	150	91.4
D.C. 4-layer QRNN with $k = 4$	160	91.1

- 언어 모델링
  - Penn Treebank (PTB)

Model	Parameters	Validation	Test
LSTM (medium) (Zaremba et al., 2014)	20M	86.2	82.7
Variational LSTM (medium) (Gal & Ghahramani, 2016)	20M	81.9	79.7
LSTM with CharCNN embeddings (Kim et al., 2016)	19M	–	78.9
Zoneout + Variational LSTM (medium) (Merity et al., 2016)	20M	84.4	80.6
Our models			
LSTM (medium)	20M	85.7	82.0
QRNN (medium)	18M	82.9	79.9
QRNN + zoneout ($p = 0.1$) (medium)	18M	82.1	78.3

Training speed advantage



Batch size	Sequence length 32	64	128	256	512
8	5.5x	8.8x	11.0x	12.4x	16.9x
16	5.5x	6.7x	7.8x	8.3x	10.8x
32	4.2x	4.5x	4.9x	4.9x	6.4x
64	3.0x	3.0x	3.0x	3.0x	3.7x
128	2.1x	1.9x	2.0x	2.0x	2.4x
256	1.4x	1.4x	1.3x	1.3x	1.3x

- Char-level 번역
  - IWSLT German-English

Model	Train Time	BLEU (TED.tst2014)
Word-level LSTM w/attn (Ranzato et al., 2016)	–	20.2
Word-level CNN w/attn, input feeding (Wiseman & Rush, 2016)	–	24.0
Our models		
Char-level 4-layer LSTM	4.2 hrs/epoch	16.53
Char-level 4-layer QRNN with $k = 6$	1.0 hrs/epoch	19.41

"Quasi-Recurrent Neural Networks" (2016)

Simple Recurrent Unit (SRU)

- RNN 연산과정 중 h_{t-1}에 대한 dependency 제거

GRU

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{R}_f \mathbf{h}_{t-1} + \mathbf{b}_f)$$

$$\begin{aligned} \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{x}_t \\ &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \tilde{\mathbf{x}}_t \end{aligned}$$

$$\begin{aligned} \tilde{\mathbf{x}}_t &= \mathbf{W}\mathbf{x}_t \\ \mathbf{f}_t &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{b}_f) \\ \mathbf{r}_t &= \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{b}_r) \\ \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \tilde{\mathbf{x}}_t \\ \mathbf{h}_t &= \mathbf{r}_t \odot g(\mathbf{c}_t) + (1 - \mathbf{r}_t) \odot \mathbf{x}_t \end{aligned}$$

Bottleneck (matrix multiplication)

Highway

$$\begin{aligned} \mathbf{h}'_t &= \mathbf{r}_t \odot \mathbf{h}_t + (1 - \mathbf{r}_t) \odot \mathbf{x}_t \\ &= \mathbf{r}_t \odot g(\mathbf{c}_t) + (1 - \mathbf{r}_t) \odot \mathbf{x}_t \end{aligned}$$

- c_t는 그대로 유지 => 여전히 Autoregressive
- CNN만큼 빠른 RNN 연산

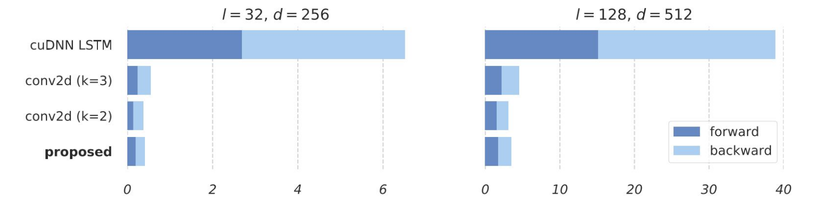
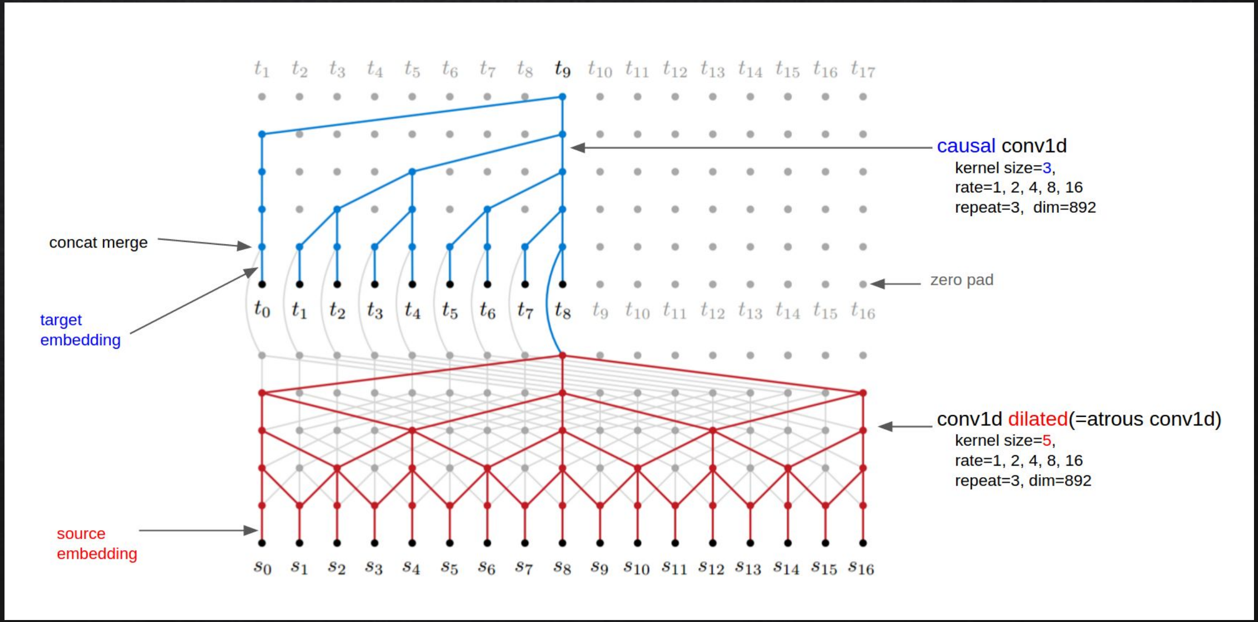


Figure 1: Average processing time (in milliseconds) of a batch of 32 samples using `cuDNN LSTM`, word-level convolution `conv2d`, and the proposed RNN implementation. l: number of tokens per sequence, d: feature dimension and k: feature width. Numbers reported are based on PyTorch with an Nvidia GeForce GTX 1070 GPU and Intel Core i7-7700K Processor.

"Training RNN as Fast as CNNs" (2017)

ByteNet

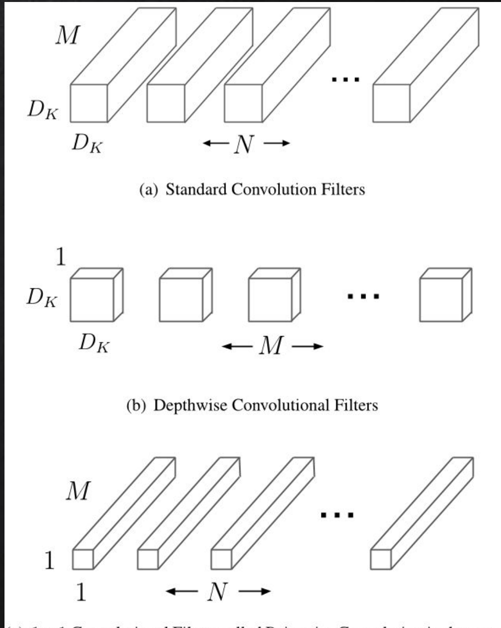
- 문자 단위 번역 모델
- Encoder, Decoder에 Dilated Causal CNN
  - 음성 합성/인식 모델인 WaveNet 에서 제시



"Neural Machine Translation in Linear Time" (2016)

SliceNet

- ✘ 문자 단위 번역 모델
- ✘ Depthwise Separable Convolution
  - ○ Xception (Extreme Inception) 에서 제시
    - ■ Spatial 정보 / Channel 간 정보 모델링 역할 filter 분리
    - ■ 적은 Parameter & 성능 향상
- ✘ Encoder-Decoder에 multi-layer Depthwise Separable Convolution
- ✘ Attention 에도 Convolution 적용



(a) Standard Convolution Filters

(b) Depthwise Convolutional Filters

(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Translation Model	Training time	BLEU (difference from baseline)
Transformer (T2T)	3 days on 8 GPU	28.4 (+7.8)
SliceNet (T2T)	6 days on 32 GPUs	26.1 (+5.5)
GNMT + Mixture of Experts	1 day on 64 GPUs	26.0 (+5.4)
ConvS2S	18 days on 1 GPU	25.1 (+4.5)
GNMT	1 day on 96 GPUs	24.6 (+4.0)
ByteNet	8 days on 32 GPUs	23.8 (+3.2)
MOSES (phrase-based baseline)	N/A	20.6 (+0.0)

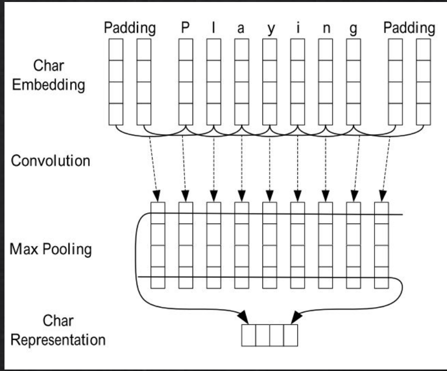
구글 번역기

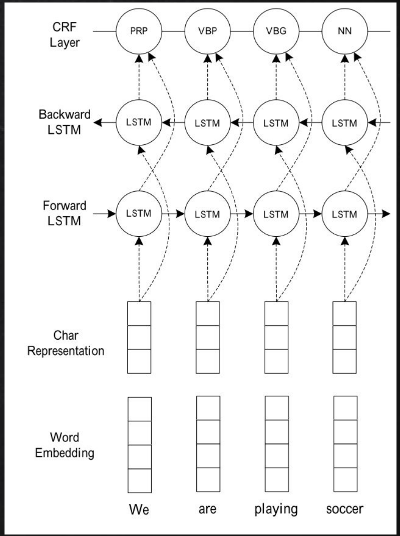
BLEU scores (higher is better) on the standard WMT English-German translation task.

"Depthwise Separable Convolutions for Neural Machine Translation" (2017)

Bi-directional LSTM-CNNs-CRF

- ✘ Base model: Bi-LSTM + CRF (2015)
  - 품사 (POS) 태깅 / 개체명 인식 (NER) SOTA
  - End-to-End
  - 90% 가 넘는 accuracy
- ✘ Char-CNN 으로 문자 벡터, GloVe로 단어 벡터 생성
  - 모르는 단어도 태깅 가능
- ✘ 추가적으로 사전 이용 가능
- ✘ CRF => 성능 향상 / 속도 저하
  - Viterbi algorithm의 연산량이 많음





Model	POS Dev Acc.	POS Test Acc.	NER Dev Prec.	NER Dev Recall	NER Dev F1	NER Test Prec.	NER Test Recall	NER Test F1
BRNN	96.56	96.76	92.04	89.13	90.56	87.05	83.88	85.44
BLSTM	96.88	96.93	92.31	90.85	91.57	87.77	86.23	87.00
BLSTM-CNN	97.34	97.33	92.52	93.64	93.07	88.53	90.21	89.36
BRNN-CNN-CRF	97.46	97.55	94.85	94.63	94.74	91.35	91.06	91.21

"End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF" (2016)

Review

- CNN for text classification
  - Word-CNN
  - Dynamic-CNN
  - Char-CNN
  - Very Deep CNN
- RNN for text classification
  - Bidirectional RNN
  - Recursive Neural Networks
  - Tree-LSTM
  - Dual-Encoder LSTM
- Advanced CNN/RNN Architecture
  - Char-CNN for Language Modeling
  - Character and Word embedding
  - CNN+RNN: QRNN
  - Fast RNN: SRU
  - Dilated Causal Convolution: ByteNet
  - Depthwise Separable Convolution: SliceNet
  - Sequential Tagging: Bi-directional LSTM-CNNs-CRF

THANKS!

Any questions?