Automatic Response Generation to Conversational Stimuli

Vishal Raj Dutta
vishal15115@iiitd.ac.in

Sanidhya Singal
sanidhya15085@iiitd.ac.in

Abstract

We present the interim project report for Group 36. In this project, we compare and contrast a few machine learning models that have been recently proposed and used in the automatic response generation to conversational stimuli. The project is based on generative models rather classification or clustering. Hence, sufficient effort has been involved in understanding and learning about them.

1 Introduction

1.1 Problem Statement

The problem we deal is how to model a machine that a comprehend conversations and respond coherently to them, using deep learning. We aim to create a model that can generate responses to a conversational stimulus and at the same time, sound more human.

We′ve always been fascinated by how chatbots work and what makes them so realistic. In fact, response generation is a big challenge from the perspectives of machine learning and NLP, with new innovations always on the brink of unfolding. It can be used in several areas like e-mail, customer service and of course, chatbots.

1.2 Motivation

Natural language (for example, English language) has always been a hard domain to model, due to its complex structure and grammar. Until recently, any conversational model that was present, relied on an elaborate set of rules for conversing with any user (Eliza chatbot is an example). This obviously couldn′t scale well, since a lot of rules would be required to account for the unpredictability of english sentence structure. However, with the onset of Recurrent Neural Networks (RNNs), Sequence-to-Sequence (**seq2seq**) models [5] are being used in a lot of NLP tasks such as machine translation, text generation, etc.

For our project on Response Generation, we followed the same backbone of thought as Machine Translation. In machine translation, given a sentence S and a set of plausible translations T for S, the model tries to generate a translation $t \in T$ or is very close to the elements in T. This closeness between the two translations is evaluated by the BLEU score [6].

We treated conversing as akin to translation [4], where given a sentence S, its response r is a translation of S, and belongs to a set of valid responses R. Hence, our model aims to generate a best response $r \in R$. In translation, the model learns the translation space T, because during training, both input sequences and their correct translations are provided. Similarly, our model learns the response space R, because in the training phase, the sentence and its response is provided.

2 Related Work

There has been a significant work being done on the given problem for the past few years. Following are some of the relevant ones:

1. In [10], the neural conversational model – a two-layered LSTM using AdaGrad with gradient clipping – was trained on the large albeit noisy open subtitles dataset with a vocabulary of the most frequent 100K words. The dataset contained 62M sentences (923M tokens) as training examples, and the validation set had 26M sentences (395M tokens). At convergence the perplexity of the recurrent model on the validation set was 17. Some snippets are:
   Human: What is the purpose of life?
   Machine: to serve the greater good.
   Human: What is the purpose of living?
   Machine: to live forever.
   Human: What is the purpose of existence?
   Machine: to find out what happens when we get to the planet earth.

2. In [7], the neural translation model – a 4 layer deep LSTM – was trained on a subset of 12M sentences consisting of 348M French words and 304M English words, with a vocabulary of 80K words a side. At convergence, perplexity was 5.8 on the validation set.

3. More specific to our dataset, i.e., the Cornell Movie Dialog Corpus, people have posted their results on chatbots online at a popular data science blogging website, *Medium* [2]. A recent blog states a perplexity of 2.74 in a dataset which comprises of the Cornell Corpus, among few others. This model is comparable to ours, as the corpus chosen is same, as well as the size of the data used. We use their model as the state-of-the-art model for our project.

3 Data set and Evaluation

3.1 Data set

The dataset comprises of conversations extracted from the **Cornell Movie Dialog Corpus**, which is a collection of dialogues by movie characters, consisting of more than $6,00,000$ lines of

dialogue. We assumed consecutive sentences as action-response pairs, since they were uttered by different characters. Overall, 20K conversation pairs were extracted, and 2K pairs were used for the validation set, with a vocabulary of 1000 words. Rest of the words were flagged as unknown with a special tag <vsunk>.

3.2 Feature Extraction: Word Embeddings

To extract features from the corpus, we used two common feature extraction techniques based on word embeddings, namely, One Hot Vectorization and Word2Vec. These techniques are explained in detail below. We created two separate models based on the two techniques. After evaluating the performances of the two models, we decided to move with the One Hot Vectorization technique, since it provided better results. Please refer to section 4 for more details.

3.2.1 One Hot Vectorization

As chats tend to be shorter than normal conversations, we select only those sentences from the corpus which have no more than 20 words. Each sentence is tokenized using Keras′s inbuilt tokenizer, and an integer ID is assigned to each token. Every word in the vocabulary is too assigned a unique ID, with the words which occur rarely in the corpus (frequency of occurrence < 10), being replaced by the <vsunk> specifier. The sentences are, then, padded with a special <vspad> token so that every sentence has the same length. Now, each sentence is converted into a sequence of integers, which leads to a one-hot representation of each word in the sentence.

3.2.2 Word2Vec

For our word-embeddings based model, we use gensim′s inbuilt Word2Vec model, which assigns a 300-dimensional word vector to each word in the vocabulary. Word2Vec, thus, converts a word to a fixed length vector with real number values. 'Similar' words, as seen in the corpus, are placed closer in the vector space.

3.3 Evaluation

We considered two metrics for evaluation: Perplexity and Human Evaluation.

Perplexity [10] is a measure of confidence of predicting the next word in the response given. A perplexity of value k means that there are around k possible candidates while predicting the next word. Obviously a better trained model will have lesser value of perplexity as it can better filter appropriate candidates and hence, is more confident about its prediction. The perplexity on a set of N test samples is computed using the following formula:

$$P_r = exp(-\frac{1}{W}\sum_{i=1}^{N}\ln(\hat{P}(r_1^i, ..., r_m^i|o_1^i, ..., o_n^i)))$$

where W is the total number of words in all N samples, $\hat{P}$ is the learned distribution and r^i and o^i are the i-th response and original message respectively. Note that in the equation above only response terms are factored into P_r.

Human Evaluation is the other evaluation metric and we found this reasonable since, ultimately, humans will be the end users and can judge the quality of a chatbot to a good extent based on responses.

4 Methodology

4.1 Data Pre-processing

We use the Cornell Movie Dialog Corpus to train our data. The movie lines and conversations are extracted as per the instructions provided along with the corpus. Every conversation between any two actors is treated as a question answer pair. The text is cleansed using Keras's tokenizer. Upon doing some analysis, we see that 85% of the sentences are of length 19 words or less. Thus, we discard any sentence with length more than 20 words. This looks realistic as chat messages are, in general, short in length. Now, we create a vocabulary based on the question answer pairs. We restrict the size of our vocabulary to 1000 words, by marking any words with frequencies less than 10 as unknown. For this, we use a special tag: <vsunk>. About 4% of the total words are marked unknown. Each word in a sentence is assigned a unique ID and two dictionaries are maintained for the same. Since sentences are of varying lengths, we pad each sentence with a special tag: <vspad>. Each question starts with a tag <vsgo> and each answer ends with a tag <vseos>. These special tags too are provided with unique IDs.

4.2 Training of Data

We use two separate models to train our data as discussed below.

Feature Extraction: One Hot Vectorization vs. Word2Vec

Word embeddings assign each word in the training corpus a word vector of fixed dimensions. Initially, we used the Word2Vec model of embeddings to generate a 300-dimensional (akin to what was done for a Wikipedia dump) embeddings for words in our corpus, keeping a vocabulary of 10K words. We used a straightforward seq2seq model with 4 LSTM layers and sigmoid activation in each. Also, to convert output embedding vectors to words in a sentence, a nearest neighbour strategy was used. However, our results were not great even after 100 epochs, possibly because of the simple seq2seq setup (stacking one layer over another) that was used. The chat results were also poor, even after increasing the dimensions of the LSTMs and the word embeddings to 1000.

Hence, we switched over to an encoder-decoder based seq2seq approach, which uses a one-hot based vectorization for representing words in a vocabulary instead of Word2Vec. This model is described below:

Model 1: seq2seq Based Model

The proposed model is an encoder-decoder based sequence to sequence model, which given a sequence of words as input, predicts a sequence of words as output. The encoder processes the input sentence term by term and creates a fixed length representation of the sentence in a vector space. The decoder then uses this context vector and generates a response, term by term, using the previous term and the context vector.

To internalise the dependance of the current state on the previous states, we use RNNs rather than feedforward neural nets. However, since natural language contains long range dependencies, we need to capture that too. For example: "A girl came here

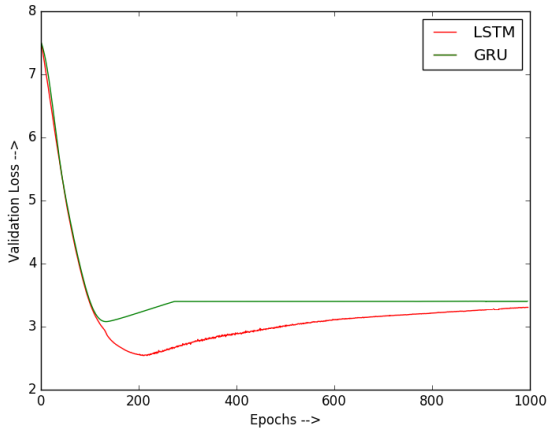

Figure 1. Comparitive Study of LSTM and GRU based on Perplexity values (Batch Size = 64, Latent Dimensions = 512)

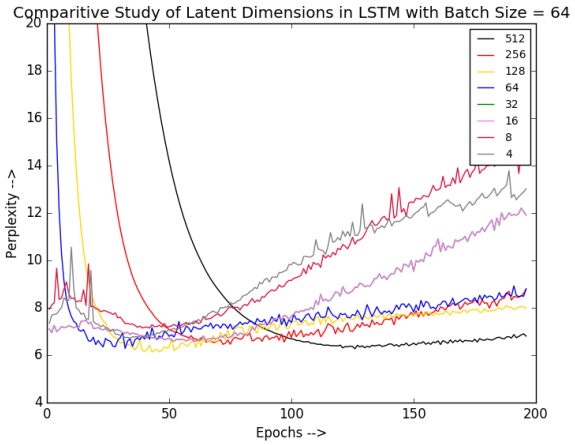

Figure 3. Comparitive Study of Latent Dimensions in LSTM with Batch Size = 64

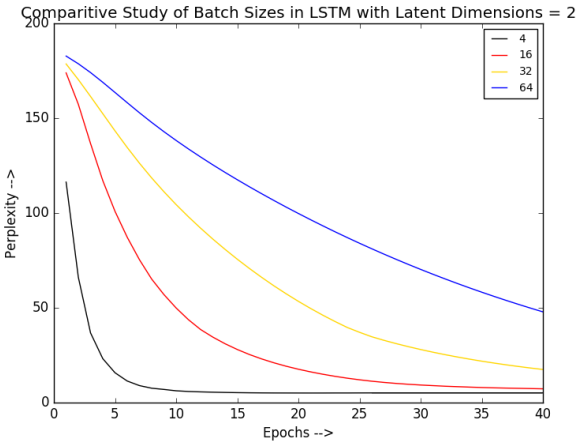

Figure 2. Comparitive Study of Batch Sizes in LSTM with Latent Dimensions = 2

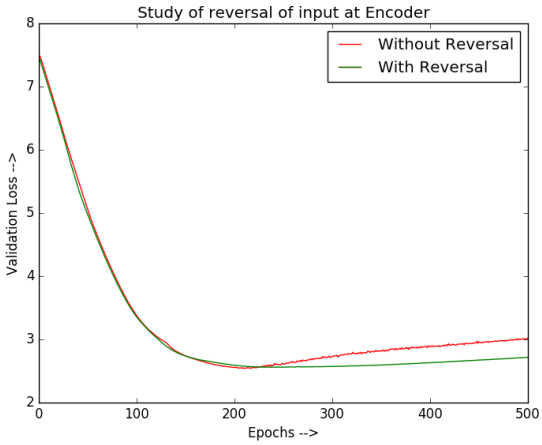

Figure 4. Study of Reversal of Input at the Encoder

a while ago. Did you see her?". The model must remember the context well enough to associate 'girl' and 'her'. But since vanilla RNN cells suffer from vanishing gradients, we instead use LSTMs and GRUs [3]. The models are created out using Tensorflow Backend and Keras API [5].

Figure 1 summarises the performance of LSTM vs GRU wrt perplexity on the validation set.

We, then, experimented with various dimensions of the LSTM layers, so as to better generate responses. We did so by fixing all other parameters (cell type, batch size, epochs, etc.) and only varying the LSTM cell dimensions. We note that for LSTMs with dimension equal to 512, our validation perplexity is least, though it takes a greater number of epochs to reach the same. This may be because greater number of LSTM cells lend it more power to form meaningful sentences. Note that we ran the model for 200 epochs since after that perplexity fluctuates very less.

We, then, also experimented with the batch size for training in each epoch, as learning in small batches might prove better than entire batches on our huge dataset. Batch size was varied keeping other parameters constant. The results were found as given in figures 2 to 4. We note that for batch size 4, the perplexity decreased faster than for others. This is probably because the variability of sentence length tends to increase over bigger batches, but the model prefers to learn same length sentences at a time.

Thus, after this series of evaluations the best model achieved was an LSTM with 512 cells in both the encoder and decoder, with softmax activation function (training with batch size of 4). The perplexity score achieved at convergence was 8.04.

The backbone of all seq2seq modelling techniques was [7].

Model 2: HMM Based SMT Model

We also tested our HMM based Statistical Machine Translation (SMT) model, which gave a validation perplexity value of 26.24. This is well below the state-of-the-art, but in regard to our SMT model, this is still a huge leap ahead.

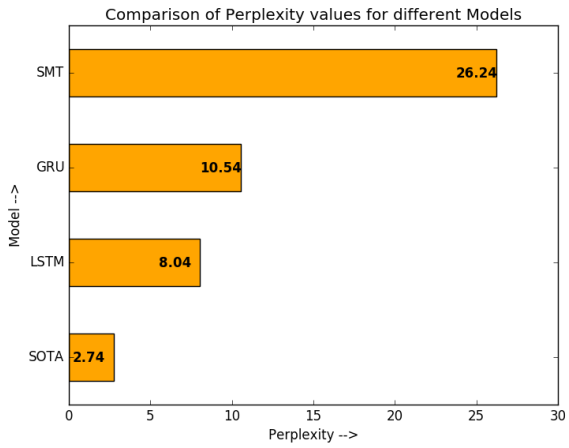

Figure 5. Comparative Study of Different Models

5 Results & Analysis

To evaluate our model qualitatively, we took a survey of 17 candidates and asked them to rate 10 outputs each generated by the statistical model, the baseline and our best model respectively on the grounds of human-ness (i.e., how human it sounds) of the responses in each case. The score varied from 0 to 10, with 0 being the least comprehensible and 10 being the most comprehensible. The human-ness of the model was then calculated by taking the mean of all the $(10 * 17 = 170)$ scores per model. The 'human-ness' scores achieved were: 2.982 for the HMM-based SMT model, 4.671 for seq2seq based model and 5.135 for the state-of-the-art model. We note the closeness of our model in comparison to the-state-of-the-art model. Please refer to figure 6.

To evaluate our model quantitatively, we used validation perplexity as a metric. We again compare the perplexities of our statistical, baseline and our best model.

We note that our best model's perplexity is higher than the state of the art by 5.30. However, with respect to the SMT baseline, which has a perplexity of 26.24, we are clearly better off.

Shortcomings may be due to less layers in the encoder-decoder framework. We were constrained by resources and time to create a model that has more layers, since one-hot encodings take up a lot of memory. The dataset may also be expanded to cover more conversational aspects.

6 Contributions

6.1 Deliverables

1. Finished various analyses based on different models, hyper-parameters, word reversal, word embedding analyses, building statistical machine model, etc.

2. Did not finish character by character model, and need to further better performance of the best model.

6.2 References & Citations

Online Blogs and Tutorials: [1], [2], [3], [5], [8], [9]
Science Articles: [4], [6], [7], [10]
Please refer to the end of the paper for more details.

6.3 Individual Contributions

1. Vishal: Created word-embeddings based model, Created encoder-decoder model, Created HMM model, Analysis on the word-embedding based model.

2. Sanidhya: Pre-processed the data, Created encoder-decoder model, Various analyses on the encoder-decoder model, Did the Survey.

6.3.1 List of Files

1. Vishal: `*Seq2SeqModel.py`, `MLProject_Model_RNN+GRU.py`, `CreateHMMTrainingSet.py`, `createEmbeddingsOnConvPairs.py`, `test_handler.py`, `hmm-Viterbi.py`, `dataPreprocessor.py`

2. Sanidhya: `MLProject_Model_RNN+LSTM.py`, `MLProject_Model_RNN+LSTM_v2.py`, `convertToConvPairs.py`, `train_handler.py`, `CleanDataAndPlot*.py`

References

[1] G. Corrado. Computer, respond to this email. https://research.googleblog.com/2015/11/computer-respond-to-this-email.html.

[2] D. Currie. Bot tutorials, medium: How to build your first chat bot. https://tutorials.botsfloor.com/how-to-build-your-first-chatbot-c84495d4622d.

[3] Deeplearning4j. A beginners guide to recurrent networks and lstms. https://deeplearning4j.org/lstm.html.

[4] A. Kannan, K. Kurach, S. Ravi, T. Kaufman, B. Miklos, G. Corrado, A. Tomkins, L. Lukacs, M. Ganea, P. Young, and V. Ramavajjala. Smart reply: Automated response suggestion for email, 2016. https://research.google.com/pubs/pub45189.html.

[5] Keras. A ten-minute introduction to seq2seq learning in keras. https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html.

[6] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: A method for automatic evaluation of machine translation. http://www.aclweb.org/anthology/P02-1040.pdf.

[7] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. http://arxiv.org/abs/1409.3215.

[8] T. Tran. Creating a language translation model using sequence to sequence learning approach. https://chunml.github.io/ChunML.github.io/project/Sequence-To-Sequence/.

[9] T. Tran. Creating a text generator using recurrent neural network. https://chunml.github.io/ChunML.github.io/project/Creating-Text-Generator-Using-Recurrent-Neural-Network/.

[10] O. Vinyals and Q. V. Le. A neural conversational model, 2015. https://arxiv.org/pdf/1506.05869.pdf.

Input Sentence	*seq2seq Model*	*State-of-the-art Model*	*SMT Model*
are you a computer?	what me that	Certainly, Doctor	you re you re
get lost!	no	You got it!	you re
are you a human?	what me that	No, not real	you re you re
who is the president?	no one is it?	Nice,me	you . i m
you are not making sense	i don’t know	yeah, I know	. you . i m not
are you drunk?	what me that now please would	I ‘ m him .	you re not
Hi!	really	Hi!	s
am i a doctor?	was in vsunk work i'm at work u	sure but a can always be a.	you . i m
when will the world end?	wouldn't me	You mean last night would you?	you . i m not
can you teach me something?	this back	what do I do to install ?	you re not . i m not

Figure 6. Table showing the comparison of results obtained on the three different models