Word-Level LSTM Based Sequence to Sequence Chatbot

Lindi Chen
Electrical and Computer Engineering
lindic@andrew.cmu.edu

Ran Chen
Electrical and Computer Engineering
rchen2@andrew.cmu.edu

Xinyan Wu
Electrical and Computer Engineering
xinyanw@andrew.cmu.edu

Abstract

This paper presents a chatbot system with LSTM based sequence to sequence (seq2seq) algorithm word-level model. This paper illustrates the pattern matching techniques by using an unrolled recurrent neural network. The research on chatbot based on LSTM algorithm is implemented in two steps: 1) Completed the character based LSTM model for chatbot 2) Introduced the word based LSTM model for chatbot. The models are built using Keras library in Python and the training was done on AWS GPU instances. We use dataset from Cornell movie dialog corpus [1] prepossessed by [2].We conclude that with enough training data, the word-level model is capable of making meaningful and well structured replies in a casual conversation.

1. Introduction

Chatbots are interactive software system which use the human natural language to chat with a human user. Chatbots are powered by machine learning and deep learning algorithms. They are a kind of virtual assistants, which are often integrated into the dialog systems, and widely used in customer service and information acquisition. This services can be used as an instant message platform, such as deliver important life-saving health messages, check the forecast or help user purchase a new pair of shoes or anything else.

Artificial neural networks are computing systems which can response to a task or a question by learning the given training data. Long short-term memory(LSTM) is a recurrent neural network. A word based chatbot with sequence to sequence learning is introduced in our project, and it is best at responding to casual conversations.

1.1. Motivation

We get motivated to study chatbots for several reasons. Firstly, chatbots play important role in commerce. Chatbots improve the customer service as it response to the users query instantly. If a customer have simple questions about a product but our customer can not find useful information to figure them out, instead of hiring a support staff to the customer, a simple built-in chatbot can solve the problem quickly. Therefore, chatbot can contribute to the commerce by saving the time and money for both companies and customers. Secondly, chatbots is promising to help with the education. If the dataset of the chatbot is the knowledge from the text book, It will help with the students by giving the answer to the question that our students asked. Thirdly, chatbot is the combination of deep learning and human nature language processing, the topic of which is attractive for each of our group members.

1.2. Problem Definition

This project is aimed to develop an instant artificial intelligent chatbot based on the LSTM sequence to sequence model. The tasks to develop a chatbot model can be divided into several parts: choosing the proper dataset, cleaning and manipulating data, choosing the proper deep learning model for the system, improving the model to achieve high accuracy, testing, developing the front-end part.

For the conversation dataset we use Cornell movie dialog corpus [1] and prepossessed by [2]. Further modification to the dataset, such as eliminating punctuation and removing uncommon words, was done to suit our purpose. For the deep learning model, we build on top of the character-level seq2seq model presented in [3] to create a word-level seq2seq model. We train our model on AWS GPU instances installed with a proper machine learning image to provide support for GPU usage. Finally, we modified the web front-end component from [4] and loaded with our output to create a user interface.

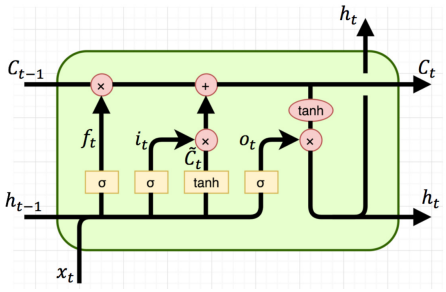

Figure 1. Long Short-Term Memory Model [6]

1.3. Contribution

The contribution we made in project includes:

- Built a word based seq2seq chatbot using Keras.
- Successfully using AWS community image to train models on GPU instances.
- Explored the trade-offs between sequence size, maximum number of words.
- Deployed a chatbot website on AWS to allow users to interact with the chatbot.[1]
- The code for building a sequence to sequence LSTM based chatbot using Keras is available online [2].

2. Background

2.1. Long Short-Term Memory (LSTM)

Long Short Term Memory networks - usually just called LSTMs - are a special kind of RNN, capable of learning long-term dependencies. LSTMs are explicitly designed to avoid the long-term dependency problem. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. LSTMs also have this chain like structure, but the repeating module has a different structure. Figure 1 shows the structure of LSTM model. Instead of having a single neural network layer, there are four, interacting in a very special way [5].

2.2. Word Embedding

Word Embedding is the method of representing words in a series of real numbers. In the context of word level representation of input text, the one-hot encoding method is no longer suitable as it will generate sparse matrix. Imagine that a word need to be represented as a vector of 1999 0s and one 1 with a vocabulary of 2000 words. Adding a word embedding layer will provide the neural networks with dense representation of words. Popular algorithms to obtain the word embedding includes GloVe and Word2Vec. In our project, we use pretrained GloVe word embedding.

2.3. Related Work

We have explored various models in building a Chatbot. The problem can be dissolved into the following 2 parts. Extracting conversations from the dataset and train the model of choice with the dataset. This would be very similar to performing machine translations, where the input in one language is mapped into an output in other language. Building a ChatBot would be outputting the reply of the input instead of the translation. Seq2seq algorithm is the predominant method in implementing this task. Nicolas-ivanov have implemented a sequence to sequence based learning algorithm [7] using the Cornell Movie dataset. Codedak have implemented a Chatbot project [2] using Cornell Movie Dialogue Corpus dataset. And saurabhmathur96 have implemented a recurrent-neural-network based neural-chatbot in [4] based on the Cornell Movie dataset as well. Keras provided an example in [3] using LSTM to perform translation from English to French using Tab-delimited Bilingual Sentence Pairs dataset in [8] provided by Tatoeba Project. The Keras model have shown good result in translating from English from French, and we have done more experiments and improvement based on this model.

3. Methodologies

This section will be focused on the methodology for building a word-level LSTM based seq2seq chatbot model. For building a character-level model please refer to [3]. With a conversation dataset, [3] can be easily used as a chatbot. This paper build on top of that to create a word-level model with the word embedding layer and modification in the process of generating the output word.

Sequence-to-Sequence (seq2seq) method is used in this paper. The idea is similar to that of machine translation where input in one language is mapped to output in another language. In our setup, conversation query is mapped to reply. The methodologies in this paper can be divided into two parts: train time methodology and run time methodology. Train time methodology pertains to how the model is defined and trained. And Run time methodology pertains to how a reply is generated given an unseen sequence of input text.

3.1. Train Time Methodology

3.1.1 Input Prepossessing

The input to the model consists of up to 100 thousand lines of conversations each contains a input query and an output reply separated by tab character. The input text and output

[1] The website was closed after the poster session. The code provided in the GitHub repository can be used to deploy the website again.

[2] https://github.com/cld2005/LSTM_seq2seq_chatbot

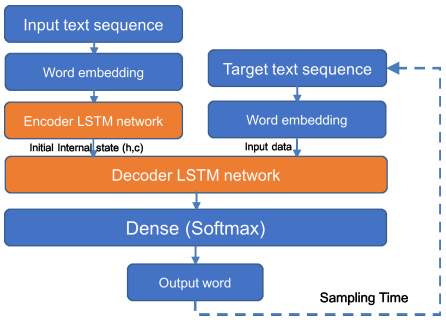

Figure 2. Architecture of LSTM Chatbot Model

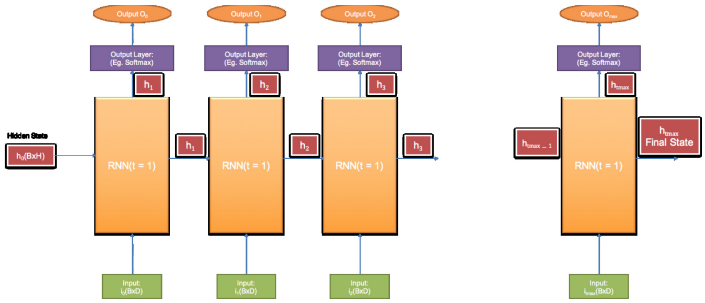

Figure 3. Architecture of Teacher Forcing Process [11]

text are tokenized separately, and therefore the integer index may not be representing the same word in the tokenized sequence. Once tokenziation is done, the embedding matrix for both input sequence and output sequence can be built from the pretrained GloVe word embedding. Detailed steps for preparing the word embedding matrix are presented in [9].

3.1.2 Model Architecture

The model used in the chatbot consist of two LSTM cells, one encoder for encoding input text, and one decoder for generating the output reply. The architecture of LSTM chatbot model is shown in figure 2. Both of the encoder and decoder takes input from a word embedding layer. The internal state of the encoder cell is used as the initial state of the decoder cell, and the encoder output was discarded. The decoder output is fed into a Dense layer with Softmax activation function to generate the word prediction which is represented by one hot encoding. We cannot avoid sparse matrix in the output, and therefore a large vocabuary size will slow down training significantly.

3.1.3 Training procedure

The training consists of the following steps. First, the input text and target output text will be loaded from the data file. Punctuation and words that are not present in the pretrained word embedding are removed. This is due to the fact that although there are 400 thousand words in the GloVe embedding, thereare still about 10% of the words in the input text that are not found in the pretrained word embedding, those words are not important to us. After the tokenization described in 3.1.1 is done, the input sequence will be padded and truncated to the same length. This padded and truncated sequence will be used as the input to the word embedding layer for the encoder LSTM. In reality, input will be fed to the embedding layer one by one, and the corresponding word vector will be fed to the LSTM encoder whose internal states will change accordingly. Latter, the internal states of the LSTM encoder is used as the initial state of the LSTM decoder. This method will ensure that both the words and theirs sequence in the input text will affect the state of the decoder, therefore this is an improvement from the bag words feature where the sequence does not matter. The decoder takes the encoder state as the initial state and the start word as the first input to generate a output which will latter be fed to the Dense layer to predict the output simply using argmax. Here the categorical_crossentropy is used as the loss function and the error is propagated back to the layers before. So now we have the first word of the output text generated, however during training, this the output from the previous time stamp is not used as the input to the decoder, instead, the correct output of the previous word is used and being fed to the word embedding layer. As shown in Figure 3.This process is called Teacher Forcing(Figure 3) [10].

3.2. Run Time Methodology

Once we have our model trained, we can use it to generate replies to unseen input text sequences, which is the most essential feature of a chatbot.

Everything up to generate the internal state for the encoder LSTM is the same as what is described in 3.1.3. However, the decoding phase is different. First, we do not have the correct target output text for the unseen input text in the run time, therefore we cannot use teacher enforcing here. As a result, the prediction output from the previous time stamp is used as the input to the decoder LSTM. Second, if all the outputs are decided by taking argmax from the Dense layer, the same input query text will generate the same reply text every time and therefore make the chatbot less interesting. Here, we propose an alternative method to make the reply more interesting. While predicting the first word of the reply, we always take a random draw from the dictionary. The probability of a word being selected is equal to the Dense layer output which uses a softmax activation function and therefore will make sure that sum of the probabilities of all the words in the dictionary equals to 1.

The reset of the words are generated based on the internal state of the decoder and the output of the previous time sample as input to the decoder LSTM until the maximum

sentence length is reached or a stop character is generated. This process can be best explained by 4.

4. Experiments and Result Analysis

This section will be focused on experimental setup for word-level based model. We will explain the trade-offs we made and present the result.

4.1. Experiment Setup

4.1.1 Input Data

In our word-level model we have to set some limitations on the maximum length of the input and output text and maximum number of words in order to achieve a reasonable training speed. Those limitations are not necessary in the character-level model. Padding is not necessary in the character-level model, the reason can be traced back to the detailed implementation of Keras, therefore was not explained here. So far, to pass the inputs in batches in a word-level model, the inputs must have the same length, and therefore requires padding and truncating. The Maximum sequence length is required, otherwise we would have to pad all the sentences to the maximum length. Most sentences will be padded with excessive padding as the longest sentence is most likely to be an outlier. In this case training speed would suffer.

The number of words allowed would also affect the training speed significantly. In the Dense layer, the Softmax value must calculated on every output (word), and the number of parameters in the Dense layer is proportional to the number of outputs. Character-level model does not have this problem as for a given language there is a fixed alphabet and punctuation character. The size of number of characters is in the order of 10s, usually less than 100 including punctuation, and therefore could hardly affect the training speed.

We have set the maximum sequence length to 20 words as the average sentence length is 10.60. Also we have set the maximum number of words to 2200, as there are only 2142 words that appears more than 30 times in 100k input query sentences, and 2038 words that appears more than 30 times in 100k output target reply sentences.

4.1.2 Number of Samples and Epochs

The models are trained with 10k samples for the development and 100k samples for the final product. 20% of the training samples are used for validation. However we believe this could be eliminated as a chatbot would hardly generate the "correct" reply measured by the categorical_crossentropy, as you can imagine that the chatbot is trained with thousands of movie conversations it would at best generate a reply that makes sense, but it would be impossible to generate the exact same reply as the target output for an unseen data. Therefore, the validation split is not necessary, the end result would be based purely on how good we think the chatbot is as described in 4.1.3 instead of the validation loss.

Training on 10k samples is much faster than training on 100k samples, and is large enough for us to debug our implementations.

For the number of epochs, we train our models to up to 270 epochs, and saves the model in every 30 epochs. Our objective to the select the model with the least number of epochs that gives the most meaningful replies in the unseen test set.

4.1.3 Model Testing

A test with 100 unseen input query sentences will be performed in every 30 epochs. Whether the model gives meaningful replies is highly subjective (the same as how good is the chatbot). Here, we look at the outputs to see first, whether the replies makes a readable English sentence; second, whether a output makes sense as the reply of the input; third, how well a model gives different answers to different inputs, as a model that does not generalize well tend to give the same reply to the same query. Effectively, we are performing some kind of Turing test.

4.2. Result and Analysis

The dialogues that word-level Keras LSTM seq2seq model produces does make sense. And the output is not hard coded. So for different input query, the model will generate the output in real-time. Some results of the model trained with 10k and 100k samples with 60 epochs are shown in Table 1. As can be seen from the results, 60 epochs results in reasonable replies from the chatbot, so we will be be using the 60-epoch result in this paper. Notice that those replies for the 10k sample model are exactly the same as the training data except for the punctuation[3]. This shows that our model is able to over fit the 10k training sample data. However we will not be able to achieve the same level of fitting in the 100k sample model. In a normal machine learning task, under fitting may not be good, however, in our case, it can to some extend help to give out more interesting replies.

The test results for testing dataset which has not been seen during training time are show in Table 2. As can be

[3]training dataset can be found in https://github.com/cld2005/LSTM_seq2seq_chatbot/blob/master/conv/codedak_conv_full.txt

[4]Full result can be found in https://github.com/cld2005/LSTM_seq2seq_chatbot/blob/master/results/result_10k_training_data

[5]Full result can be found in https://github.com/cld2005/LSTM_seq2seq_chatbot/blob/master/results/result_100k_training_data

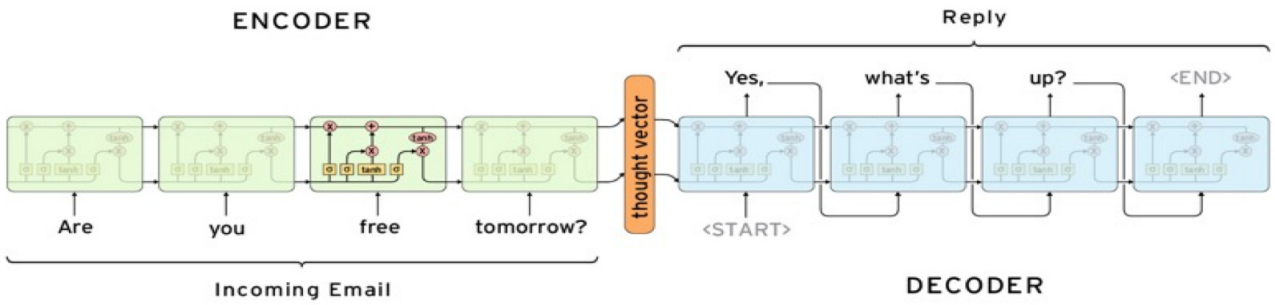

Figure 4. Example of sequence to sequence text generation [12]

Query	Reply: 10k samples [4]	Reply: 100k samples [5]
wow	lets go	now you know that
you looked beautiful last night you know	so did you	pictures of course
he was like a total babe	but you hate joey	if you mean of course

Table 1. Training Data Fitting with 10k and 100k samples

Query	Reply: 10k samples [6]	Reply: 100k samples [7]
you know chastity	no of .	i dont know
get out	we we have	no
youre completely demented	i am	im not to you
hey there tired of breathing	whatever is that	i dont know what else to do
you hate me dont you	im gonna be that got on him at at im	what

Table 2. Training data test. Comparison between models trained with 10k samples and 100k samples in 60 epochs

seen from the results, models trained with 100k samples gives more meaningful replies. This result is in no way comprehensive, but it shows that in terms of readability and sentence structure, the 100k sample models performs better. Also we can see from the results that it does not give any pad word in the answer. This shows that the mask padding option is taking effect in the embedding layer.

[6]Full result can be found in https://github.com/cld2005/LSTM_seq2seq_chatbot/blob/master/results/result_10k_testing

[7]Full result can be found in https://github.com/cld2005/LSTM_seq2seq_chatbot/blob/master/results/result_100k_testing

5. Conclusion

There are many challenges with word-level model: one is that the word-level results sparse matrix representation with one hot encoding; another one is that sentence length varies, padding results flooding of pad words in reply; besides, training takes long time, and model may over fit with more epochs.

We tried many ways and found that there are some methods to effectively solve such problems: one way is to use pretrained word embedding to better represent words; another way is to use padding and truncating, which needs to mask padding in the mode; using AWS GPU instance and community image with drivers configured is helpful; we can also limit input sentence length and vocabulary size; and we can save models regularly for every 30 epochs intervals.

As can be seen from the result, limiting the input sequence length, and the size of vocabulary will still allow the chatbot to generate meaningful replies that resembles casual human conversation. There are times when it is obvious that some words are missing, this may be the result of the limited vocabulary size. Some replies does not make sense at a glance. However, when we peek into the training dataset, the conversations are hardly logical in certain training samples and therefore we could not require the chatbot to reply something perfectly when generating a reply to an unseen input sentence.

The size of the training data does matter as using 100k training samples, the replies are much more vivid and diversified. Therefore, we believe that aside from the model selection, the training data in fact plays a crucial role in the training of a chatbot.

6. Future Work

Several works can be done in the future to improve our project:

1. Improve the measurements metrics to evaluate a chatbot system.
   Due to the limit of time we use the turning test to check whether we can receive the meaningful reply. Many other measurement methods can be taken to evaluate the system, such as dialogue efficiency metric and dialogue quality metric.

2. Provide data analysis framework interface to make the chatbot extendable
   In this project our dataset are limited to the film conversation, while if we modify the program as an interface then it can support different dataset plugins that from different area. This can be done in the future to ensure the extendability of the program.

3. Explore other ML algorithms and alternative Deep Neural Network structures.
   There are many other machine learning algorithms that can be applied to the chatbot such as Clustering Algorithms, Association Rule Learning Algorithms. We can combine these algorithms with LSTM to improve our model in the future. Also, deeper neutral network structures can be explored. For example, Convolution Neural Networks (CNNs) are good at extracting features and reduce the number of parameters trainable.

References

[1] "Cornell movie-dialogs corpus," https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html, (Accessed on 12/10/2017).

[2] "codedak/chatbot: Victor- a generative chatbot based on sequential neural network and deep learning which can be trained on any desired dataset for specific purposes. instead of ordinary chatbots which are based on hard-coded responses, it can understand context and respond accordingly." https://github.com/codedak/chatbot, (Accessed on 12/10/2017).

[3] "A ten-minute introduction to sequence-to-sequence learning in keras," https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html, (Accessed on 12/10/2017).

[4] "saurabhmathur96/neural-chatbot: A neural network based chatbot," https://github.com/saurabhmathur96/Neural-Chatbot, (Accessed on 12/10/2017).

[5] "Understanding lstm networks," http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 8 2015, (Accessed on 12/10/2017).

[6] "Lstm and gru – formula summary — isaac changhau," https://isaacchanghau.github.io/2017/07/22/LSTM-and-GRU-Formula-Summary/, (Accessed on 12/11/2017).

[7] nicolas ivanov, "Make seq2seq for keras work," https://github.com/nicolas-ivanov/debug_seq2seq, (Accessed on 12/10/2017).

[8] http://www.manythings.org/anki/fra-eng.zip, (Accessed on 12/10/2017).

[9] "How to use word embedding layers for deep learning with keras - machine learning mastery," https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/, (Accessed on 12/10/2017).

[10] "What is teacher forcing for recurrent neural networks? - machine learning mastery," https://machinelearningmastery.com/teacher-forcing-for-recurrent-neural-networks/, (Accessed on 12/10/2017).

[11] "(1) what is the teacher forcing in rnn?" https://www.quora.com/What-is-the-teacher-forcing-in-RNN, (Accessed on 12/11/2017).

[12] "Practical seq2seq," http://suriyadeepan.github.io/2016-12-31-practical-seq2seq/, (Accessed on 12/11/2017).