
Chatbot based on Recurrent Neural Networks *

Yufei Wang

Department of ECE
Stevens Institute of Technology
Hoboken, NJ, 07030
@stevens.edu

Jiangchuan Li

Department of ECE
Stevens Institute of Technology
Hoboken, NJ, 07030
jli173@stevens.edu

Abstract

We chose to design a Chatbot as our project of EE628 because of our interest in natural language processing. A Chatbot, or a Neural Conversation Model is an important part in natural language processing. It can reply to some questions automatically after trained on large amount of dialog data. Although Deep Neural Networks (DNNs) can achieve great results in almost every difficult learning task whenever large labeled training sets are available, they cannot be used to map sequences to sequences which we will need in our model. In this paper, we will discuss how we used Recurrent Neural Networks (RNNs) to design a Sequence to Sequence Model, how we trained the model and how good is it by showing the results. Our model uses a multilayer Gated Recurrent Unit (GRU) to map the input sequence to a vector, and then another deep GRU to decode the target sequence from the vector. This RNN Encoder–Decoder system can be trained end-to-end and thus requires much fewer hand-crafted rules. We find that we can generate simple conversations given a large conversational training dataset despite the dataset is noisy or specific.

1 Introduction

Deep neural networks are extremely powerful tools for data scientists when it comes to applications such as objection recognition or speech recognition. Their ability to perform arbitrary parallel computation for a modest number of steps is the reason of its powerful performance. And what's more, if we train large DNNs with supervised backpropagation whenever the labeled training set has enough information to specify the network's parameters, then if there exists a parameter setting of a large DNN that achieves good results, supervised backpropagation will find these parameters and solve the problem.

But with all that in mind, DNNs have a weakness—it can only be used after we fixed the dimension of both inputs and outputs. When it comes to question answering, the length of answer sequences are unknown, which makes the weakness a significant limitation. It's necessary to introduce a domain-independent method that learns to map sequences to sequences.

Therefore, Recurrent Neural Networks or RNNs are introduced in our model. RNNs will infer the meaning of symbols by looking at the structure of the text and relative positions of symbols instead of knowing the meanings of symbols. We used two complex version of RNNs, Long Short Term Memory (LSTM) as an Encoder and a decoder. The encoder maps a variable-length source sequence to a fixed-length vector, and the decoder maps the vector representation back to a variable-length target sequence.

We experiment the model with twitter chat log, a large and noisy dataset and find the model is capable of generating simple answers from some questions.

*Github page—<https://github.com/babyshambles/Chatbot>

2 Model

2.1 Recurrent Neural Networks

Normally A recurrent neural network (RNN) is consisted by a hidden state \mathbf{h} , a sequence of inputs (x_1, \dots, x_T) and a sequence of outputs (y_1, \dots, y_T) . At each time step t , we have

$$h_t = \text{sigm}(W^{hx}x_t + W^{hh}h_{t-1})$$

$$y_t = W^{yh}h_t$$

A simple RNN model can map the input sequence to a fixed size vector then map the vector to the target sequence in theory, but the results are not ideal when long term dependencies were introduced. The Long Short-Term Memory (LSTM) is explicitly designed to avoid the long-term dependency problem.

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A sigmoid

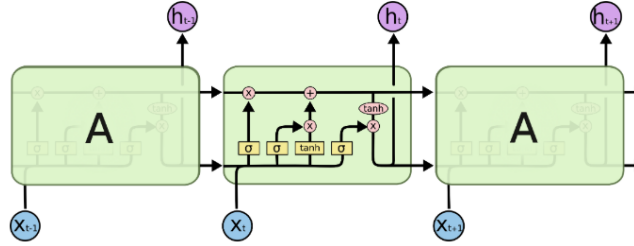


Figure 1: A standard repeating LSTM model with four layers

layer decide what information we are going to keep by iterating the following equation:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

The sigmoid layer outputs a number between zero and one. 1 represents “completely keep this” while 0 represents “completely get rid of this.”

2.2 Word Embedding

The first layer of our model is an embedding layer that maps a word (index to word in vocabulary) from vocabulary to a dense vector of given size. Word embedding have been exceptionally successful in many NLP tasks. Vector space models have been used in distributional semantics since 1990s. During this time, many models for estimating continuous representations of words have been developed. But the utility of pre-trained word embedding was not showed until 2008, when Collobert and Weston published Their landmark paper *A unified architecture for natural language processing*. They not only establishes word embeddings as a useful tool for downstream tasks, but also introduces a neural network architecture that forms the foundation for many current approaches. However, the word embeddings eventually become popular because of Mikolov et al who created *word2vec*, a toolkit that allows the seamless training and use of pre-trained embeddings. Word embeddings are one of the few currently successful applications of unsupervised learning. Their main benefit arguably is that they don't require expensive annotation, but can be derived from large unannotated corpora that are readily available.

The embedding for a given title is close in the embedding vector space to the embedding of a similar title, even if the titles' wordings are different. For example, "The squad is ready to win the football match" and "The team is prepared to achieve victory in the soccer game" have the same meaning but share almost no vocabulary. But in this embedding representation, they should be close to one another in the embedding space, because their semantic encoding is very similar. But do notice that word embedding won't be helpful in tasks that do not rely on these kind of relationships. (Figure 3)

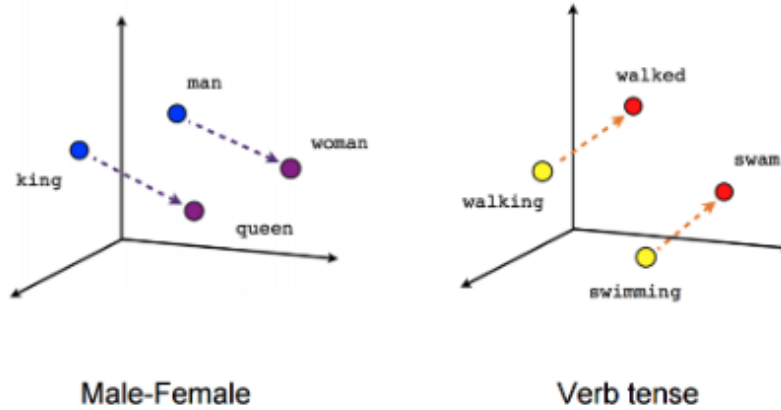


Figure 2: Some semantic relationships between words

We use Google News Word2Word as our embedding layer.

2.3 GRU of Encoder and Decoder

Gated Recurrent Unit (GRU) (Figure 3) is very similar to LSTM, with forget gate but has fewer parameters than LSTM. This is the structure of Gated Recurrent Unit. We choose to use GRU instead of LSTM because we find that GRU has shorter training time than LSTM. GRU's has fewer tensor operations, and the performance is similar to LSTM, sometimes even better in some small data sample cases.

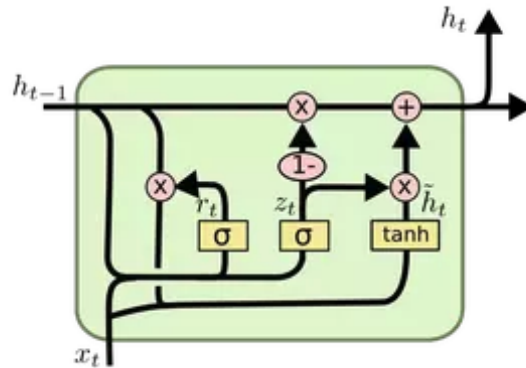


Figure 3: Structure of GRU

In our model, we use two GRUs: one for the input sequence as encoder and one for the output sequence as decoder (Figure 4). The benefits of using two GRUs are first it's important to reverse the input sentence before mapping because doing so can introduce many short term dependencies in the

data that make the optimization problem much easier, second a deeper GRU performs much better than a shallow one.

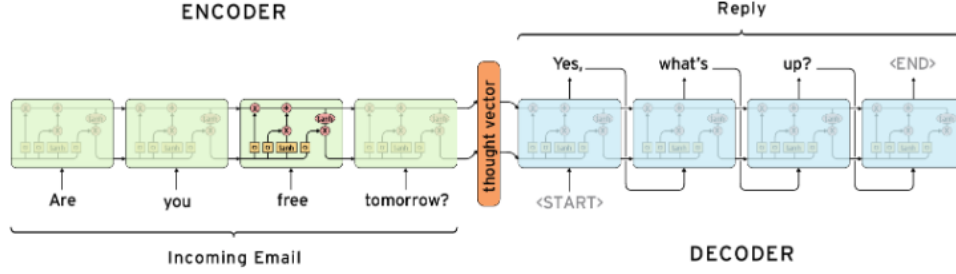


Figure 4: A simple demonstration of a Seq2Seq model

The encode reads each symbol of an input sequence x sequentially and emits a context. The hidden state is updated by

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, x_t)$$

After reading the end of the sequence, the hidden state of the RNN is a summary c of the whole input sequence.

The decoder generate output by predicting y_t given the hidden state h_t . The hidden state of the decoder at time t is computed by

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, y_{t-1}, \mathbf{c})$$

The conditional distribution of the next symbol is

$$P(y_t | y_{t-1}, y_{t-2}, \dots, y_1, \mathbf{c}) = g(\mathbf{h}_{(t)}, y_{t-1}, \mathbf{c})$$

Finally we train the model to maximize the conditional log-likelihood

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^N \log p_{\theta}(\mathbf{y}_n | \mathbf{x}_n)$$

θ is the model parameters and (x_n, y_n) are pairs of inputs and outputs in training sets. After we trained the model, we can use it to generate an output sequence by a given input sequence.

3 Experiments

3.1 Dataset

In this project we decide to use the Twitter corpus. This corpus includes plenty of conversations and comments provided by twitter and provides us enough corpus information for our training model.

3.2 Data pre-processing

The first to prepare the training data is to transform the raw data into the right format to feed our model.

3.2.1 Data Cleaning

This raw data twitter comments contains so many tags and symbols which are useless for the training process. We delete all the tags and symbols and only keep the words information. At the same time we delete the too long and too short sentences to make sure the data as a orderly one.

3.2.2 Vocabulary

Here we get the real lists of words and we call it the whole vocabulary, this vocabulary includes the frequency of every word. We also create the index of word, words to index dictionary.

3.2.3 Tokenization

As we know Deep Learning models can not read text directly, so our job is to convert this word-based text to integer index. First we split the data into question set and answer set, then add zero padding to these both sets. Here we should note that if the word is not in the vocabulary, we should replace the word with 'unk' and get the final results indices of words. This are the final training data that would go through our training model.

3.3 Training

As we get starting with the training process, we set the batch size 32, and the training epoch 50. We send the training set into the model. Since the data set corpus is such a large set and the structure of the DL model is a little complicated, it takes nearly one hour for one epoch to train. So the total training time is about 50 hours. So we take AWS into consideration to make this training process accelerated. Here we import the tqdm, which could let us create a simple progress bars with just a few lines of codes. This make a grate help that allow us to know how much work this model has done and how many more time it will need.

3.4 Amazon Web Service

Here we create a EC2 instance, choose the ubuntu deep learning base Amazon Machine Image and set the GPU instance p2.xlarge. When this instance is done, we use Pycharm to build the remote connection to run code on the server. Since this training process takes more than two days to go, we use ubuntu screen to create a remote separate session on terminal, so that we can run the code on the server and without keeping the computer working all the time.

3.5 Result

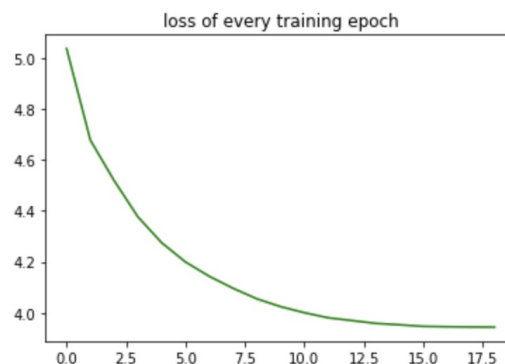



Figure 5: This plot shows the convergence of loss in every training epoch

After 19 training epoch and more than 24 hours of training time, we finally get the chance to see how good is our Chatbot communication skills. When the training process finished, we begin to chat with this model. Below Figure 6 is a simple dialogue chatting with this model.

4 Future works

1. More dialog corpus. Because of the time limitation, we only use Twitter dialog to train this model. Our model didn't perform good enough when it answering questions because this corpus contains mostly comments, not conversations. And the model only generate meaningful responses in certain



```

You: I love your t-shirt it is so cute
Query > I love your t-shirt it is so cute
> thanks babe
> thanks unk
You: Yowza, that's a big file. Try again with a file smaller than 25MB.
Query > Yowza, that's a big file. Try again with a file smaller than 25MB.
> thanks for the update
> thanks for the update
You: happy birthday have a nice day
Query > happy birthday have a nice day
> thank you
> thank you so unk unk unk unk
You: donald trump won last nights presidential debate according to snap online polls
Query > donald trump won last nights presidential debate according to snap online polls
> i know you have to go to the debate and unk the debates
> i hope you are right
You: yeah i'm preparing myself to drop a lot on this man, but definitely need something reliable
Query > yeah i'm preparing myself to drop a lot on this man, but definitely need something reliable
> i know right
> i know right but its not a good place
You: i'm okay with that. it had nothing to do with a bright orange astros jersey i was wearing
Query > i'm okay with that. it had nothing to do with a bright orange astros jersey i was wearing
> i thought i saw this
> i thought i was unk to unk i was thinking about that

```

Figure 6: Dialogue with the final model

context. In the future, we can use dialogs like Cornell Movie Dialogs corpus and Supreme Court Conversation Data to improve our performance.

2. More training time. Due to the tight schedule and computing power limitation, we did a lot of works to reduce training time. We limited the length of dialogs, chose simpler model with less training time and reduced epoch times. We believe we can achieve better results with a longer training process.

3. An user friendly interface. Now we chat with our Chatbot in command windows, this is more like a test of our model rather than chatting. If we want users to use our product, we need to design an user friendly interface. It's possible to design a beautiful website connected to our server by using Django and Redis.

Acknowledgments

We would like to thank our professor Sergul Aydore for the wonderful lectures. We thank our teaching assistant Tianhao Zhu for helping us with any problems kindly. We thank Amazon Web Service team and Google Brain team for the help with this project.

References

- [1] Mikolov, T. Statistical Language Models based on Neural Networks. PhD thesis, Brno University of Technology, 2012.
- [2] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In NIPS, 2012.
- [3] K. Papineni, S. Roukos, T. Ward, and W. J. Zhu. BLEU: a method for automatic evaluation of machine translation. In ACL, 2002.
- [4] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001. [16] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural Computation, 1997.
- [5] T. Mikolov, M. Karafi'at, L. Burget, J. Cernock'y, and S. Khudanpur. Recurrent neural network based language model. In INTERSPEECH, pages 1045–1048, 2010.
- [6] S. Hochreiter and J. Schmidhuber. LSTM can solve hard long time lag problems. 1997.
- [7] N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In EMNLP, 2013.

- [8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998.
- [9] T. Mikolov. Statistical Language Models based on Neural Networks. PhD thesis, Brno University of Technology, 2012.
- [10] Q.V. Le, M.A. Ranzato, R. Monga, M. Devin, K. Chen, G.S. Corrado, J. Dean, and A.Y. Ng. Building high-level features using large scale unsupervised learning. In ICML, 2012.
- [11] Le Hai Son, Alexandre Allauzen, and François Yvon. 2012. Continuous space translation models with neural networks. In Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT '12, pages 39–48, Stroudsburg, PA, USA, 2012.
- [12] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality, 2013.
- [13] Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1, NAACL '03, 2012.
- [14] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. 2013. Advances in optimizing recurrent networks. In Proceedings of the 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2013), May.
- [15] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. Université de Montréal, 2014.
- [16] Ilya Sutskever, Oriol Vinyals, Quoc V. Le. Sequence to Sequence Learning with Neural Networks. Google doc article, Google, 2014.