THE FIRST ATTEMPT OF BUILDING CHATBOT WITH TENSORFLOW

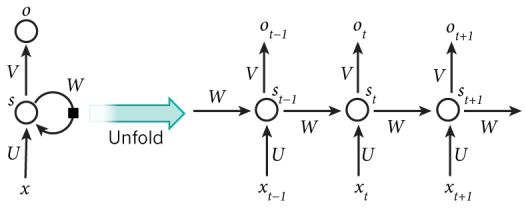
Zhe Huang, Daoyang Shan and Yulin Shen
DSGA-1007 Final Report, Fall 2017

Abstract

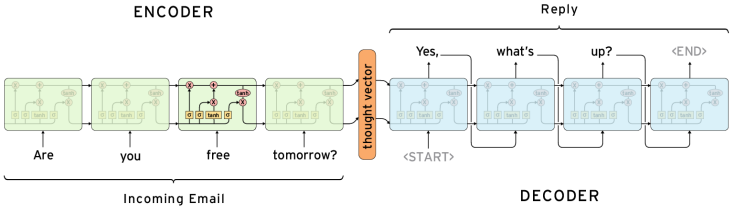
In the era of booming artificial intelligence, chatbot is one of the import applications that people developed based on the increasingly sophisticated deep learning theories and largely enhanced computation capacity. Widely used in different areas like customer service or portable device user assistance, chatbot appears more and more frequently in our daily life. In our project, we attempted to build our first DNN chatbot with TensorFlow–an open source deep learning tool developed by Google, and trained our chatbot on the famous Cornell Movie Dialogue Corpus. Although we didn't get a satisfactory result for our model, we're still optimistic in our current achievement and confident to work further on this topic.

1 Introduction

Alexa and Siri, you might have heard of these two names. In modern world various chatbots have been developed for different kind of needs. In general, the theoretical base of most of the chatbots is Recurrent Neural Network (RNN). Compared to normal neural network, which only feeds data forward through its layers, RNN has a unique design in which the output data of a a hidden layer will be fed back to this layer again as in a recursion. This design is significantly important for chatbot since it helps chatbot to 'remember' knowledge that we feed into the model before–just as a human baby learns how to talk. In practice, indeed we can't maintain an infinite recursion, and we normally use the 'unfold' version which maintains a long enough chain that simulates the infinite recursion.



(a) RNN and its unfolded version



(b) seq2seq model basic structure

One of the currently popular model for chatbot is the Sequence-to-Sequence (seq2seq) model, in which we have two RNNs: encoder and decoder. Encoder receives the question word by word and converts it to a single vector that represents the 'meaning', and decoder receives this vector and transfer it to the corresponding answer. The intuition behind such mechanism is that, with sufficient training, seq2seq model can fully explore and remember the relation between words and phrases, just as human being understand natural language.

There're three ideas that further enhance the seq2seq model performance:

1. **Long-Short Term Memory (LSTM):** To explain this in a simple way, LSTM is a special form of neural network cell that helps model to learn pattern with long-term dependencies. For

example, in a long paragraph with some information about a city at the beginning and the city name at the end, RNN with LSTM cell can capture the relationship between city and the associated information, while normal cell can hardly do it. Indeed even LSTM has its 'memory limit', but still it performs much better on such task.

2. **Attention:** Our default encoder design always compressed the 'meaning' of a sentence into a fixed length vector. However, if the input sentence is extremely long, can this fixed sized vector always capture all 'meaning'? Attention is a newly developed mechanism to overcome this limit. Instead of building a single fixed length vector after receiving all inputs, attention model encodes the input into a sequence of vectors and chooses a subset of them for each word in output. This brings an obvious computation burden but produces a better model.

3. **Dropout:** Similar to other machine learning models, neural network suffers from overfitting as well, and dropout is one of the strategies to reduce the risk of overfitting. The idea is, we ignore a certain percentage of cells in each layer so we randomly get a simpler model each time. This reduces the training time for each epoch but requires more epochs in total.

2 Methodology

1. **Data preparation:** We first process our data into integer format. Basically, we need the data to be organized in two lists, one for questions and one for answers, where each question and answer is represented by a list of integers, each integer represent a unique word in the vocabulary. Notice that the language is not already standardized in our dataset, for example 'I'm' and 'I am'. After we standardize those phrases, two more tasks are required: first, we abandon those sentence with length larger than 20 words; second, we identify those rare words and replace them by **UNK**. Finally we add three more special markers to our vocabulary-integer conversion dictionary: **PAD** (fill the empty spots after short sentences since all input sentences in the same batch should have the same length), **EOS** (marks the end of sentence, ignore the words after it in encoder) and **GO** (marks the start of a sentence in decoder).

2. **Modeling:** We have to admit that building a seq2seq model from scratch is far beyond our scope of knowledge. Instead, we used the skeleton from this tutorial and also the whole seq2seq model from the tutorial on botsfloor.com (see appendix for tutorial link). We used TensorFlow 1.01 and Python 3.5 to build our model (TensorFlow 1.0 doesn't support Python 3.6, see ReadMe for the detail to install the correct version of TensorFlow in order to run our code.)

   Generally speaking, the model we applied is written with TensorFlow low level APIs (compared to high level APIs such as Keras) and it implements a RNN with LSTM cell. The model also implements attention and dropout mechanisms. We divided our pre-cleaned data into almost equally sized batches, feed the question of one batch into our model, and check the loss between the output and target (answer). TensorFlow has appropriate APIs for each of the mechanisms mentioned before. We didn't modify the model besides adjusting some hyper-parameters, and we will skip discussing the model step by step. Interested readers can get more details in the original tutorial.

3. **Train:** We trained the model several times with different set of hyper-parameters. Since we do not have adequate experiences in chatbot, we tried hyper-parameters from a large range of possible values, and combine them in the reasonable way that we thought. See Result section for detail of the different set of hyper-parameters we tried.

We record the loss along the training, and validate the test set twice in a single epoch to observe the validate loss. Whenever we update the model (in particular, update the weight between edges), we cap the gradient to stay between 5 and -5 because the gradient of RNN may easily go enormous without gradient capping, which is dangerous for our model. Also, in some training attempts we let the learning rate decay after each validation step until it reaches a pre-defined minimum level, since learning rate decay can help model to avoid overfitting and divergence in many cases.

Finally, we also used TenforFlow saver API to save the model along the training process. In some cases we save only when we reach a new minimum validate loss, and in other cases we save regardless of current validate loss. Later we can simply load a pre-trained model with saver to avoid training again (indeed this is necessary for every chatbot due to the daunting training time needed!)

3 Result and Discussion

As we discussed in our proposal, training a chatbot can sometimes be disappointing for amateurs since it's not guaranteed to get a desirable result. Unfortunately, we didn't get a very convincing result ourselves. The following chart shows the major attempts and the results we got.

Table 1: Results of some attempts

BatchSize	RNNSize	NoL	LR	Epochs	Result
32	128	2	0.005-0.001(decay)	40	Only returns several meaningless words
32	128	3	0.1-0.01(decay)	40	Only returns one word each time, only 2 words are returned in total (last logit) Some random phrases, meaningless (logit with best validation loss)
32	256	3	0.5-0.01(decay)	10	Return some meaningless words like "arctic blue blue we but you" etc
64	256	3	0.5(no decay)	50	Only returns one word, meaningless
64	256	3	0.5-0.1 (decay)	50	Only returns one meaningless phrase (last logit) Returns some meaning less phrases, only few phrases in total (logit with best validation loss)
64	512	3	0.1-0.01(decay)	50	Returns some meaningless phrase or word
128	512	3	0.005-0.001(decay)	10	Returns only several sentences like 'I don't' or 'I don't know'
128	512	3	0.5(no decay)	50(20)	Error diverges, stopped around 20 epochs (last logit) Some random phrases, meaningless (logit with best validation loss)
128	512	3	0.1-0.01(decay)	20	Return some meaningless words

LR:learning rate; NoL: number of layers in RNN

We can observe a strange pattern that in most of the attempts our chatbot only returns a few words

or phrases in total. Sometimes the result is slightly more flexible when the loss reaches minimum, but ultimately it converges to one or two words at the end. If we check the answer logit instead of the actual testing answer, in a few cases our model returns exactly the same logit regardless of the input word, although in most of the cases we ca observe different answer logit despite the same answer.

At present, fully identifying and solving the problem is a huge challenge for us. We admit that without the deep understanding of RNN and experience in TensorFlow, we can hardly get a satisfactory bot in this semester. Still, we proposed several causes for the dilemma we're facing right now.

1. **Data Processing:** Right now we're process raw data in a really naive way, and it's possible that we need some advanced techniques of data processing. However we checked some other tutorials and it seems that other authors process data in a similar manner, which may indicate that data processing is not the underlying reason.
2. **Modeling/Wrong Model:** It's possible that we're applying an erroneous model even if the model runs in a normal and smooth way, and we've double checked the model implementation before using it. Both we and the author may ignore some mistakes within the model which caused this problem.
3. **Hyper-parameter:** We've tried different sets of hyper-parameters, and most of them are directly from either tutorials or papers. However we can't guarantee that we have used the reasonable set and trained with appropriate time.
4. **Test Step/Chatting Test:** If everything above are actually correct (as the training record may suggest), it's possible that we handle the chatting test in a wrong way.

One interesting thing is, the dilemma we described above may not be a unique pattern for our model. At the same time we ran this model, we used another chatbot from Stanford tutorial (see appendix for link) and rewrote it in TensorFlow 1.2 for comparison test. Although we didn't fully train that Stanford bot, we observed a similar pattern (bot only returns a few words/phrases) as well. Indeed it's possible that we made similar mistakes in both models, but such possibility is not high since two bots have totally different structures and implementations.

4 Conclusion

In project we attempted to build a RNN based chatbot with TensorFlow and Cornell Movie Dialogue Corpus. Although we failed to produce a satisfactory chatbot, we still gained precious experience in both deep learning and TensorFlow. We plan to extend this project in future course work and hope to solve the problems that we're not able to handle right now.

5 Appendix

`Botsfloor Tutorial:`
`https://tutorials.botsfloor.com/how-to-build-your-first-chatbot-c84495d4622d`
`Stanford Tutorial:`
`https://github.com/chiphuyen/stanford-tensorflow-tutorials/tree/master/assignments/chatbot`

Data Set:
https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html
RNN and LSTM basics:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Attention:
https://machinelearningmastery.com/how-does-attention-work-in-encoder-decoder-recurrent-neural-networks/