A/The Disambiguating based on Recursive Neural Network

Yang Gao

October 15, 2017

1 Problem Description

The give challenge task is asked to process a plain text file from, which is a novel written by famous English writer Charles Dickens, i.e., *"A Tale of Two Cities"*. The articles (a/the) in this novel are somehow obfuscated. Thus a problem raises, i.e., how to recover the original correct usage of articles, which becomes the main focus of this challenge task.

Two plain text files are given to validate the final model and results, which are the original novel and its obfuscated version. In the following, I will go through some ideas and results achieved during these two days. A roughly tuned model will reach an accuracy of 73%. Due to the limited time and high computation consumption, I could not train on a large scale corpus and thus propose some possibilities and ideas as discussion.

2 Input Data Preprocessing and Basic Analysis

Since the original data is not accessible until the evaluation (otherwise the data will be memorized by the model, which is kind of cheating), we will train on another corpus. Here I chose another novel of Charles Dickens, which is *"Oliver Twist"* based on the following reasons:

- The number of words is lager than the given file, the basic information of this two files is summarized in Table 1. A larger corpus is suitable for training and should be able to generalize.

- The writing style is similar to the given file, since both books are written by a same person and thus share some basic structures and styles which could be learned and transferred to the testing file.
- The number of vocabularies are close (slightly larger than the given one), thus the trained model will not suffer from the problem of two many unknown vocabularies.

Novel	Number of Words	Number of Vocabularies	Number of Sentences
"A Tale of Two Cities"	123096	9745	5809
"Oliver Twist"	200780	12251	9129

Table 1 Basic information regarding to the two files.

All the data files are preprocessed and split into sentences, and save as "*.p". The vocabulary dictionary and the reverse vocabulary dictionary are established from the training corpus, which will be dumped and applied to the testing file after the model is well trained.

3 RNN-based Disambiguating

In this section, I will go through some milestones where we generate some results. The first milestone is achieved by using vanilla-RNN network, where the accuracy reached 65% (the accuracy is defined as the fraction of correct a/the pairs to the whole a/the pairs, if no method is applied, this accuracy should be around 0.5, namely by guessing).

3.1 Vanilla-RNN Model

The first idea I came up with is to construct a RNN network, this network takes batches of sentences with fixed length and send the Softmax predictions as output, as structured in Figure 1.

The inputs are batches of sentences with fixed length, they are tokenized and represented by IDs, and then is fed into a matrix for word embedding. The word embedding will go through the RNN cells (we use LSTM cells in our implementation). The output tensors will handled by a Softmax layer to give a prediction of corresponding word (as probability), which will be compared with the sentences in original text file during the training phase (The detailed implementation and explanation could be found in the notebook file "vanilla_rnn.ipynb").

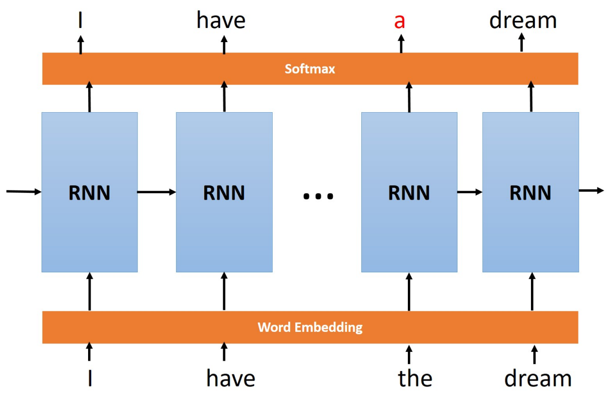

Fig. 1 The structure of basic RNN model.

I tuned the parameters, such as epochs, size of rnn hidden layer, size of word vector, length of sequence, size of stacked RNN etc. However, the accuracies will be slightly improved w.r.t. 65%. Some parameters and the corresponding accuracies are summarized in the Table 2.

3.2 Bidirectional RNN

As we know, the articles are sometimes depending on the word after them, so another possible solution is to construct a bidirectional RNN, we applied one layer bidirectional RNN with LSTM cells and the accuracy reaches 73%, which is also the best results I could get before the given deadline. The structure is shown in Figure 2.

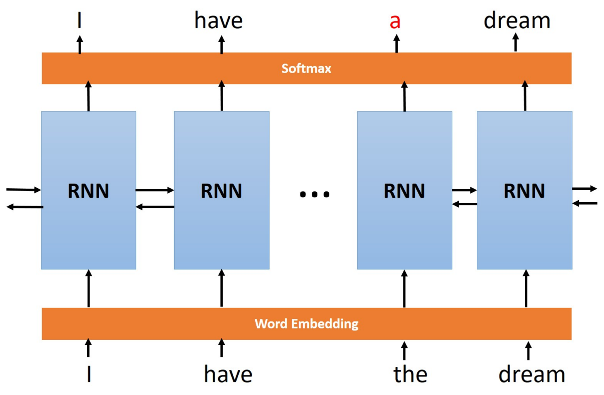

Fig. 2 The structure of bidirectional RNN model.

The parameters and accuracy for bidirectional RNN model is also listed in Table 2. The training loss and validation loss are shown in Figure 3, from the figure we can conclude that the training loss will decrease along with the epoch, however, the validation loss will decrease in the beginning and keep unchanged. The generalization noise is unavoidable, however, one can enhance the generalization capabilities of model by use larger corpus.

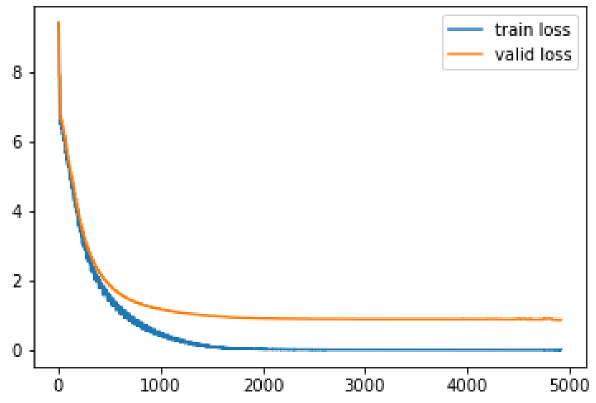

Fig. 3 The training performance of bidirectional RNN model.

epochs	hidden Size	wordvec size	sequence length	layer size	model type	accuracy
100	100	100	20	1	LSTM	65%
200	300	300	50	2	LSTM	67.76%
200	200	200	30	1	LSTM with dropout	66.6%
200	200	200	30	1	Bidirectional LSTM	73%

Table 2 Parameters and accuracies.

4 Other Potential Ideas

It is very costly to train the data, I thus performed all the experiments on AWS with GPU, however, due to the time limitation, I could not implement all the ideas and tune the parameters in a systematical way. A little bit pity! In the following, I will initialize some other ideas which are partially implemented or even only draft ideas.

4.1 Sequence to Sequence Model

Another kind of model which is fit for this problem is sequence to sequence model, the basic structure is shown in Figure 4. The model contains two part-

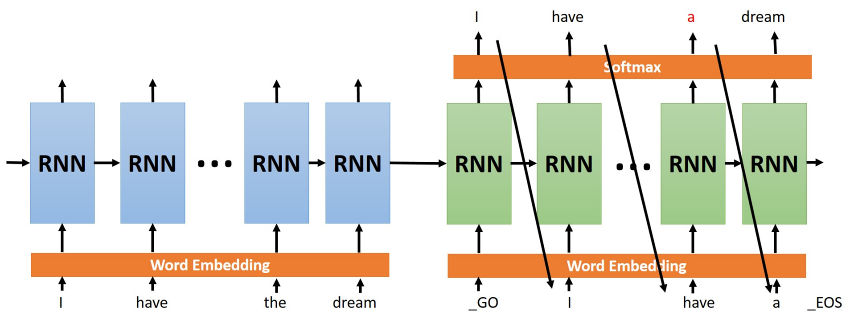

Fig. 4 The structure of seq2seq RNN model.

s, i.e., encoder and decoder. The encoder takes the obfuscated sentences as input, after reading the input sentences, the encoded hidden state is fed into the decoder. The decoder takes the original sentences as input (during training) and the output of decoder should be correctly predicted sentences. This idea is only partially implemented and trained (Refer to the notebook file "seq2seq.ipynb", the inference part is not fully implemented). The training process are partially shown in the notebook, even though I did not compute the accuracy, at least some clues could be seen from the printed training loss and accuracy.

4.2 Other Ideas

Here are some other ideas I did not get time to implement, but worth trying (my opinion).

- If we have a paragraph of text like this:

  Although I am not disposed to maintain that the being born in a workhouse, is in itself the most fortunate and enviable circumstance that can possibly befall a human being, I do mean to say that in this particular instance, it was the best thing for Oliver Twist that could by possibility have occurred.

  and we process the data into the following form: *[Although I am not disposed to maintain that]* ***the***

  [being born in,] ***a***

  [workhouse, is in itself,] ***the***

  [most fortunate and enviable circumstance that can possibly befall,] ***a***

  [human being, I do mean to say that in this particular instance, it was,] ***the***

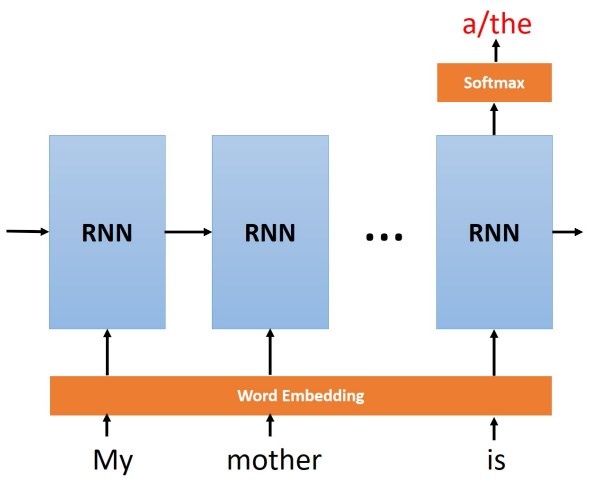

Fig. 5 The structure of idea.

Then this is problem of sentence classification, i.e., given sentence, predict the next possible “a” or “the”, as shown in Figure 5.

- Another idea is a small trick for the vanilla model (not sure it will work well), the structure is depicted in Figure 6. The input data and the RNN structure keep unchange, what makes it difference is the output labeling during training. Instead of map the output to all the words, we map the output to three classes: 0 (other words), 1 (indefinite article “a”) and 2 (definite article “the”). I hope the model can learn when to use definite articles and when indefinite from the input corpus. The training should be faster since the output matrix gets smaller (the output dimension of the last fully connected layer is 3).

5 The File Structure

The materials submitted contain some related repositories and files, which are structured as following:

- ChallengeProblemDescription.txt: original problem description and files from Nuance.

- createdata: some utility functions to preprocess the text file (just back-up, will not be used in the final implementation.)

- model: training related data files and model implementation.

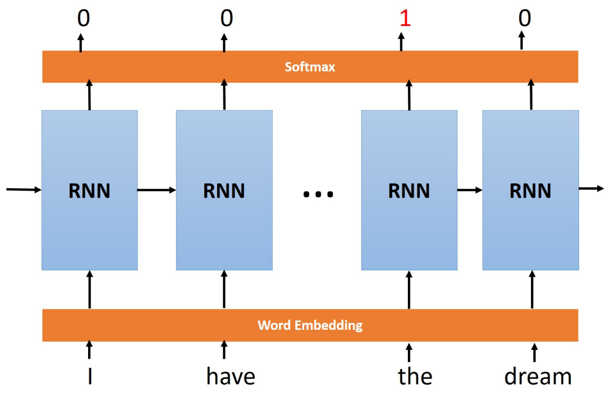

Fig. 6 The structure of idea.

  – novel: original text file (Oliver Twist).
  – processeddata: preprocessed data and saved data.
    — original.p: sentences of training text (Oliver Twist).
    — params.p: trained parameters.
    — preprocess.p: dumped variables, i.e., vocabulary dictionary of training data etc.
    — twotaledumped.p: dumped variables, i.e., sentences tokens etc.
  – results: the output of the model.
    — results.tsv: Three column TSV.
    — a_tale_of_two_city_original.txt: original text file (processed).
    — a_tale_of_two_city_disambiguated.txt: disambiguated text file (processed).
  – savedmodel: trained model.

- vanilla_rnn.ipynb: vanilla RNN implementation.

- vanilla_rnn.html: a running instance of vanilla_rnn.ipynb.

- seq2seq.ipynb: sequence to sequence model (trainable, but not fully finished).

- seq2seq.html: a running instance of seq2seq.ipynb.

6 Time Allocation

The challenged task was running from 9:00 13.Oct.2017 to 9:00 15.Oct.2017 (Germany time), within the 48 hours, I am required to implement the ideas, train the model and write the report. The time is allocated as shown in Table 3.

Time Period	What I have done
13.Oct.2017	
9:00 - 12:00	Try to understand the problem, prepare the data and analyze the basic structure of the data.
13:00 - 18:00	Find whether there is state of the art work, initialize the idea and start to implement the basic RNN model.
19:00 - 22:00	Continue the implementation, try to train the model on a small artificial corpus in order to check whether the implementation works well.
22:00 - 1:30	Debug and train on the real training data.
14.Oct.2017	
8:00 - 12:00	Debug, continue to train the basic RNN model, test the trained model on the original data and compute the accuracies.
15:00	First milestone: 0.65 accuracy.
15:00 - 18:00	Tune the parameter, start writing report during training and try to apply bidirectional RNN.
19:00	Second milestone: 0.73 accuracy.
19:00 - 0:00	Focus on the report, clean the implementation and finalize the files

Table 3 Time Allocation

7 Conclusion

This is a very interesting challenge task, through these two days I gained a lot of experience and knowledge regarding to the neural network, natural language processing and deep learning framework like Tensorflow. The results are not that perfect, i.e., only 0.73 accuracy before the deadline. Before turning to other better models, I would like to say that the training data I took is relevantly small, only 200K words. By training on larger corpus, it is promising to achieve some better results.

8 Acknowledgement

I was inspired a lot by the tutorials written by Goku Mohanda [1], especially the RNN part, very helpful. I was also benefit from Udacity [2] deep learning nano courses, and a lot of ideas about implementation are from there. Besides, Tensorflow [3] official website is always a good place to get new ideas.

[1]https://theneuralperspective.com/tag/tutorials/
[2]https://www.udacity.com/nanodegree
[3]https://www.tensorflow.org/