Z5004195

Zihao Ye

COMP6714 Project 2 Report

Abstraction

Skip-gram is an model that utilise the idea of neural network model to learn the embeddings of the words. We are going to train the neural network to be able tell the probability a word that would be in the context of the input word.

This report would briefly talk about the major steps and ideas of the project such as data pre-processing, model training, parameter tuning and data post-processing, as well as the reason behind them.

Introduction

Skip-gram is an alternative way to get dense vectors instead of method like SVD. This method is inspired by neural net language models which learn dense embeddings for the words in the training corpus. The main idea of skip-gram model is to maximise the probability(using softmax) the neural net work can give a correct prediction of a context word regarding to the input word. The network contains of two matrices of embeddings, namely target embeddings and context embeddings. The learning process would be staring with some random initial embeddings and iteratively improve the embeddings for a word.

Since this project is specialised for predicting synonyms for a given input adjective, the data pre-processing and post-processing method would also be specialised for this purpose.

Methodology

Data pre-processing:

At the very beginning of the project, I only applied some very basic data pre-processing methods, namely preserve only the alpha words, remove all the punctuations and change all the characters to lower case. However, such basic data cleansing turned out to be not good enough, because the accuracy increase was not very significant comparing to not doing anything at all.

After trying the most basic methods, I tried something else which seems to be more appropriate for this project which is preserve only the noun, verb, adjective and adverb. Thanks to the help of Spacy, the program can always tell the correct pos tag of a given word. The result turned out to be significantly better than before but still not good enough.

Later on, I decided to take an bold step which is instead of using the original context of the corpus I completely changed my strategy of data pre-processing. The method is we only accept adjectives and it's ancestors from the corpus. Suppose we have cleansed the data properly which means the whole corpus have transform into a sequence of sentences. Then we can take advantage of the dependency parser in Spacy, which can find out the relations between every words in a sentence. Therefore, we can build a synthetic context which are merely the ancestors the adjective. After applying this method, the accuracy had a big jump comparing to before.

The reasonings behind this method are as follow. Firstly, it greatly helped to filtered out the non-adjective words which can be very noisy when predicting. Secondly, the syn-

thetic context might be better in some sense since the dependency parser can help to find out the words that really have relation with the adjective, rather than just some less meaningful neighbours.

At the last minute, I also added the lemmatisation method to the data, since similar adjectives can be applied on variants of similar words.

Model Training:

Most of the steps and code are same as the word2vec demo. Due to the time constraint, I am not able to come up with a smarter way to generate batches. So I am not going to talk much about this part.

Parameter tuning:

This part is arguably most interesting and time consuming part of this project. At the very beginning of this project, at the time I only adapted the most basic data preprocess method, I had my computer to run for days to try on different combinations of parameters. Vocabulary size range from 5000 to 100,000, window size range from 1 to 16, number of negative sampling range from 8 to 64, learning rate range from 0.001 to 2, batch size range from 16 to 128. This kind of brute force method is indeed not a wise one but it really helped me to get some senses in tuning the parameters. The best accuracy i was able to achieve at this stage is around 0.06 and the parameters are vocabulary size as 16,000, skip window size as 1, number of negative sample as 8, batch size as 16 and learning rate as 1.

After applying the better data pre-processing method, I found out that the accuracy can be gradually improved by decreasing the vocabulary size(down to less than 3000). So i once again used this brute force method to find out the best set of parameters. The result would be shown in detail in the next section.

Data post-processing:

The work done in this part is very straight-forward. Since we are only required to deal with adjectives, we can once again take advantage of Spacy to find out the fine grain pos tag of each word. At last, we only output a list words that has the same tag of the input adjectives.

Results

In this section, I would only put on the results and the parameters which can give accuracy higher than 0.085 for the dev test set.

```
vocab size: 2000, batch_size: 24, skip_window: 1, p_sample: 2, n_sample: 32, l_rate: 1
0.08522727272727272
vocab size: 2000, batch_size: 32, skip_window: 1, p_sample: 2, n_sample: 16, l_rate: 1
0.08607954545454545
vocab size: 2000, batch_size: 24, skip_window: 1, p_sample: 2, n_sample: 24, l_rate: 1
0.08707386363636363
```

What's more, after I added the lemmatisation in data preprocessing and restrict the number of context word to 2 in the last minute, the average hit achieve higher than 9.

```
Average Hits on Dev Set is = 9.325000
Average Hits on Dev Set is = 9.625000
Average Hits on Dev Set is = 9.425000
```

Conclusion

The accuracy of the skip gram model can be improved significantly if the data preprocessing and post processing is properly done. But in terms of how into tune the parameters more scientifically, we might need to study the math and neural model underneath in greater depth.