Project 2
Word Embeddings For Adjectives

z5135897
Mingkai Ma

Abstract:
For this project, I managed to implement word embeddings for adjective words. I used word2vec models to produce my own word embeddings. After got these vectors, I used genism in Python to get top k similar words with the target input word. To get the word embeddings, I used the skip-gra model.

Introduction:
There are three separate steps.
Firstly, Preprocess the raw data. I read the large text corpora from BBC_Data. During this step, I use the spaCy module in python to parse the data and preprocess.
Secondly, Use the preprocess data to train the model, therefore I get the wanted adjective_embeddings, this process takes a long time, several hours. During this step, I use the tensorflow module to train the model.
Thirdly, Then according to the input_adjective, the compute_topk function will get the top k words which are similar with the input adjective in meanings. During this step, I use the genism to load the final embedding result, e.g. adjective_embedding.txt, then use the most_similar function to get the top k result.

Methodology:

I will introduce the methodology I use in the project from three steps: preprocess data, training the data and get the top k result.

1.Preprocess data.

First read files one by one. During reading the file, firstly I get every line of the data then use spacy to process the natural language, tokenization, then I use spacy is_alpha function to select only words consisting of alphabet. No numbers, no punctuation. Then I use the spaCy to parse and extract linguistic features, part-of-speech tags. I perform lemmatization if the word is noun or verb or adjective then store it into lexicon list. And, according to the token.pos_, if it is 'PRON' or 'PROPN' or 'DET' or 'ADP' or 'ADP' or 'CCONJ', I discard the word. Otherwise, I store the lowercase word into lexicon list. Hence I got the vocabulary whose size is 13341. And the total tokens in the data is 492109.

Then I walkthrough the whole original data to index every distinct word into an integer ids, and store the word, id into dictionary. And for look up purpose, I create the reverse dictionary. This is the process of preprocess.

2.Training data.

The model I use is word2vec – skip gram model. I will introduce this model as follow.

We are given a specific word in a sentece, and look at the words nearby and pick one at random. The skip gram model network can tell us the probability for every word in our vocabulary of being the neighbours of the word we chose. Then window size is needed for us. If the window size is 2, meaning 2 words behind and 2 words ahead(four in total). This is the meaning of parameter: skip_window. We will train the whole network by feeding it word pairs found in our data. I implement this by the function generate_batch. The network will learn the statistics from the number of times each pairing shows up. Thus we represent a word to the network as a one-hot vector, this vector will have #vocabulary size components, and the vector has only '1' corresponding to the word we are representing, other places are all 0. The output of the network is a single vector which contains the probability that a randomly selected nearby word is that vocabulary word. When we evaluate the trained network with an input

word, the output vector will be a probability distribution(a bunch of floating numbers.)
If two distinct words have very similar contexts, then the training model will produce very similar results for the two words, that is the two words' vectors are very similar. Therefore if two words have similar contexts, the training model will output simialr word vectors for these two words. We know that is two words have similar contexts, it means there two words are like to be synonyms.
Above all are the brief introduction to skip gram model from my understanding. I implement and train the model by using tensorflow module in python. As required, the loss function is sampled softmax loss, and the optimizer is Adam Optimizer with a low learning rate= 0.0001. The average loss is around 5.

3.Get top k result.

From step 2, we can get the trained word embeddings in a txt file. Then I load the file by genism module in python, this genism module has a very convenient function,
`most_similar`(*positive=None*, *negative=None*, *topn=10*, *restrict_vocab=None*, *indexer=None*)¶
It will calculate the cosine distance between two vectors, represeting two words(one is input word) and output the most similar words with ragard to the input word.
The format of the load file is fixed.
number of words vector size
word1 x00 x01 … x0N
word2 x00 x01 … x0N
Each line starts with the word and follows the vector.

Results And Discussion:

In skip gram model, there are several parameters which we need to tune to improve the performace. For example the batch_size, which defines the number of times each step the model will use to update the embeddings. The skip window, num_samples , vocabulary_size , learning rate , number of negative samples. Different values for these parameters will result different performaces for the top k results.
I have ever set the skip window as 10 words, the num_samples is 20 times, I found the final performace is not as good as a smaller skip window and num_samples. And for learning rate, there was a time that I set a large

beginning rate, say 4. And unfortunately, I got very bad loss, these losses were very high. Then I realized that I need a smaller learning rate, I set it 0.0001, and the loss was very small and the optimizer would converged to a minimum.

Conclusion:

This project is very practical. (I realized that I need to get a better computer for courses like this☺.)
We are required to iterate the training process 100,001 times, and the optimizer is AdamOptimizer, which performs better as suggested, but remarkably slow. It takes several hours to finish the training everytime. But I have really learned a lot and have a better understanding of the word vector methodology. As Professor Wang said, the model is unsupervised, and the dataset is not large enough, so the performace is not very good, I only achieved . I have tried different method to preprocess the data, and different parameters' values, but still cannot improve the performace to a better level. I still need to learning a lot.