NLP HW1: word2vec

Zhang Naifu 2018280351
znf18@mails.tsinghua.edu.cn
funaizhang@github

Tsinghua University
April 9, 2019

About

The directory structure of this folder is as follows.

- word2vec (Part 2)
  - train: contains .py script to train
  - test: contains .py to evaluate and output of cosine similarity in .csv file
- fasttext (Part 3)
  - train: contains .py script to train
  - test: contains .py to evaluate (same script as above) and output of cosine similarity in .csv file

1 Part 1

$$\mathcal{L} = -log\, p(o|c)$$

$$\mathcal{L} = -log \frac{exp\left(u_o^T v_c\right)}{\sum\limits_w exp\left(u_w^T v_c\right)}$$

$$\frac{\partial \mathcal{L}}{\partial v_c} = \sum_w \frac{\partial \mathcal{L}}{\partial\left(u_w^T v_c\right)} \frac{\partial\left(u_w^T v_c\right)}{\partial v_c}$$

$$\frac{\partial \mathcal{L}}{\partial v_c} = \sum_w \left(y_w - 1\{w = o\}\right) u_w$$

2 Part 2

Under the word2vec folder, you can find the script to train word embedding with Wikipedia corpus and evaluate with WordSim353, built on Tensorflow.

This is adapted from tensorflow word2vec_basic.py tutorial.

In all the runs, we use the following hyperparameters. Out of the default hyperparameters provided in word2vec_basic.py tutorial, the most notable change is a smaller negative sample size - this appears to do better in practice. Total number of steps has been increased to allow better convergence.

Hyperparameters	
Batch size	128
Skip window	1
Neg sample	6
Steps	200k
Vocab size	50k

Results are reported in the table below.

Embedding no.	100	200	300
Final loss	3.81	3.22	3.41
Spearman r	0.032	0.039	0.108

In line with theory, bigger embedding numbers require more training data, but can lead to more accurate models. The best model turns out to be 300 embedding number.

A graph of the loss of the three models is produced below.

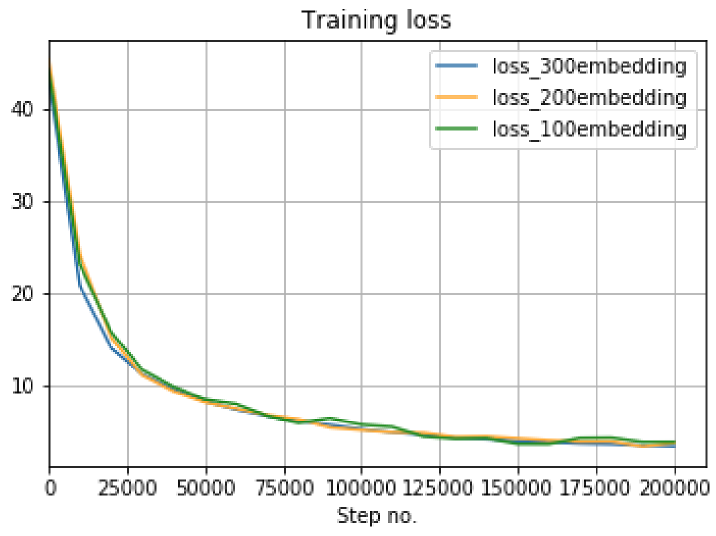

Figure 1: Training loss

3 Part 3

While running the basic model, I noticed that some of the words in WordSim353 are rare words and not given a proper embedding.

Therefore, I try to improve on the result with fasttext, an extension of word2vec. One advantage of fasttext is speed. Another is better representation of rare words. fasttext breaks the words up into n-grams and for which the embeddings are trained. Since it is more likely these n-grams also appear in other words, we hopefully get better representation of words.

The output for 300 embedding size is compared against word2vec. Unexpectedly, it performs worse.

Model	word2vec	fasttext
Spearman r	0.108	0.051

A possible reason is that the same set of hyperparameters are used for fasttext. Better results might be achieved with parameter tuning. Another possible reason is implementation error.