Natural Language Understanding with Distributed Representation

Assignment 2

Submission by : Sean D Rosario

1. Pre-processing and Cleaning the data

The data is cleaned by the file "data_cleaning_and_preprocessing.py". Punctuation is spaced out so that when the text is tokenized, the punctuation acts as individual words. I built a dictionary out of these words, by iterating over all the reviews and counting the word frequencies. Using this dictionary, I set the vocabulary to be the top 10k most frequent words, and replace words not in the vocab with <oov>. I save these reviews as a list of cleaned and preprocessed reviews using the pickle library

I count the length of each review and plot the histogram (see Fig 1). This gives me and idea of what the ideal max_document_length should be. Having is as the maximum would result in a very large and sparse matrix which would be computationally inefficient. A length of 313 was calculated to be the 75th percentile of the distribution, hence I chose 300 as the max_document_length.

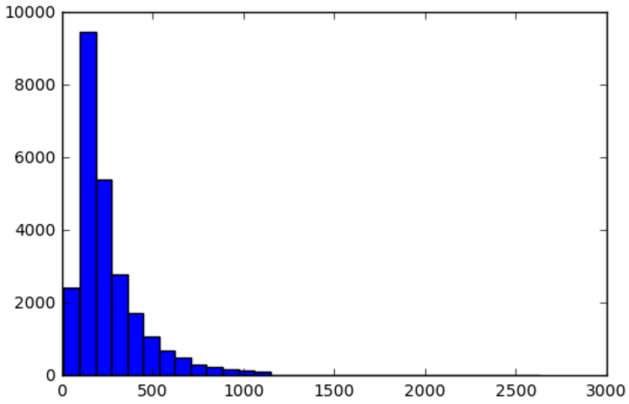

Fig 1. Histogram of review lengths

2. Training the CBOW model

The input matrix for the CBOW unigram model consists of the words in the review, padded with zeroes till the max_document_length which I have set as 300. 20000 reviews are used for the training set and 5000 reviews are used for the dev set. An embedding matrix is randomly initialized and then added as another dimension to the input matrix. I then take the average across the embeddings for all documents. This results in a 2 dimensional tensor where each

document (row) has an average embedding vector. This is fed into a single-layer MLP on which a softmax is applied.

I trained the CBOW model for unigrams and used the following parameters:

BATCH_SIZE=32
CHECKPOINT_EVERY=200
EMBEDDING_DIM=64
EVALUATE_EVERY=200
L2_REG_LAMBDA=0.0
NUM_EPOCHS=50

Result: Accuracy is 87.96 %

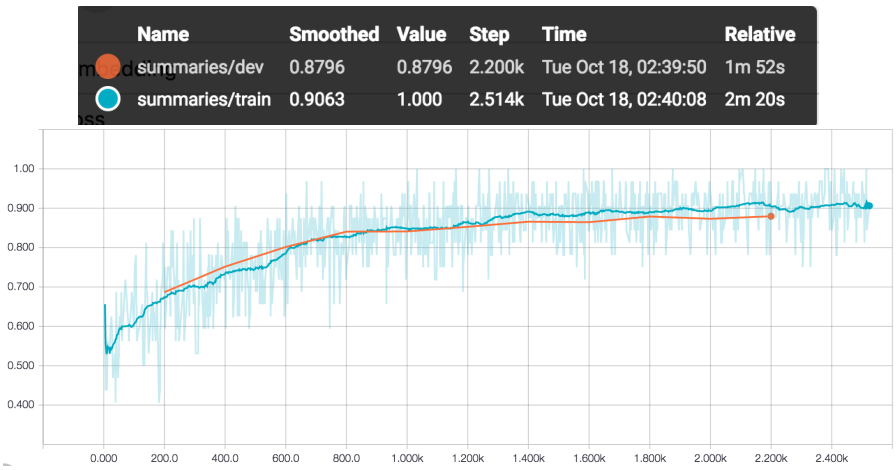

Fig 2. Accuracy across the training steps for unigram CBOW

3. Training the CBOW bi-gram model

Similar to the unigram dictionary mentioned in part 1, I build a vocabulary out of the top 10k most frequent bigrams. Bi-grams not in the vocabulary were set to <oov>. I appended the bigrams to the unigram text and fed that as the input to the CBOW model, with total vocab size of 20k

I trained the CBOW model for bigrams using the following parameters:

BATCH_SIZE=32
CHECKPOINT_EVERY=200
EMBEDDING_DIM=64
EVALUATE_EVERY=200
L2_REG_LAMBDA=0.0
NUM_EPOCHS=50

Result: Accuracy is 88.14 %

The accuracy increase by <0.5%, hence adding bigrams only marginally increases the accuracy.

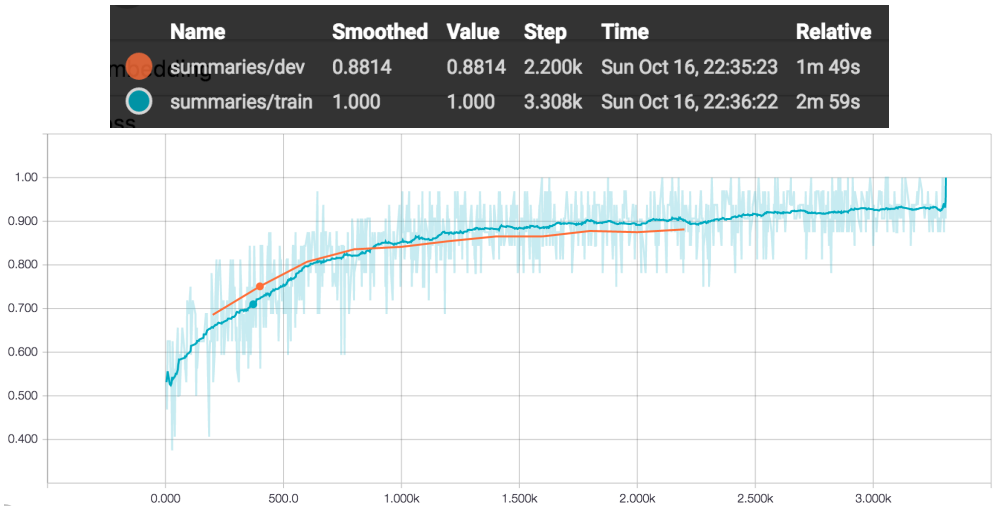

Fig 2. Accuracy across the training steps for bigram CBOW

3. Benchmarking against Facebook's fastText

I re-format the input so that it can be fed as an input to the Facebook's fastText model, which for all intents and purposes, I treat as a black-box model. Similar to the previous sections, I use 20000:5000 as the train:dev ratio

I create one large string which contains the label: either `__label__POS` or `__label__NEG`, followed by the actual review text, where each review ends with '\n'

I then train the model on the training set from the terminal

```
$ ./fasttext supervised -input train.txt -output model -dim 64 -lr 0.1 -wordNgrams 2
-minCount 1 -epoch 50
```

and evaluate the performance on the test set

```
$ ./fasttext test model.bin test.txt
```

Result:

```
P@1: 0.887
R@1: 0.887
```

Increasing the Embedding dimension for Either the CBOW model of fastText does not significantly increase accuracy. Hence, computationally, it would make more sense to stick with a smaller embedding size, so that there are less parameters for the model to learn.