Natural Language Understanding with Distributed Representations - Assignment 2

Urjitkumar Patel
up276@nyu.edu

In this assignment we are replacing the convolution layer which we implemented in the last assignment with CBOW (continuous bag of words) Fasttext implementation, and then want to compare with Facebook's Fasttext results. I will first discuss uni-gram and fasttext models within the context of CBOW and then I will present the comparison between the our implementation of CBOW fasttext and Facebook's Fasttext model (with the same parameters tuning).

1 CBOW for Text Classification and Evaluation

Basically using continuous bag of words, we represent a text sentence into a single vector, which we calculate by taking average over all word vectors for that sentence. Once we have the sentence vector representation matrix, we just try to find linear solution which produces two score values for two sentiment classes. At the end we just apply argmax to get our final prediction for that sentence.

(Please find attached image with email for CBOW architecture. Due to rotating issue I could not attach in this file.)

We can try different methods here withing the context of CBOW model. I tried two models as below.

1. Uni-gram CBOW model (less complex , less learning)

2. Fasttext (n-gram) CBOW Model (More complex , More learning)

For the training of this model, I used 25K samples from available in train set and divided it into 80:20 ratio for training and validation.

1.1 Uni-gram CBOW model

For uni-gram CBOW model, we just learn embedding for all unique words and we just consider each word independently to create the sentence vector representation. This model works well as the baseline. Only issue is with uni-gram we do not take internal sentence structure into consideration because we are just taking average over all word vectors. This mat lead to a problem as stated below,

If we take an example of two sentence,

- Movie is good, movie is not bad

- Movie is bad, movie is not good

According to uni-gram vector representation, both sentences would end up with same word embedding which is misleading.

1.2 Fasttext n-gram CBOW model

I implemented Bi-gram model to see how does it affect the improvement. In Bi-gram model, on top of all unique words we also consider pair of adjacent words to create the sentence representation. hence in our previous example, <is good> <not bad> <is bad> <not good> all these bi-grams would have different vector representation which will make the two sentences different from each other.

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

1.3 How did I improve the Performance?

- **Baseline with just 10K words**: For the baseline, I just replaced the convolution layers in last assignments code with CBOW implementation. Also I realized that, If I consider all sentences with all unique words, computation cost will be too high for dictionary creation and vector creation. As very required step, I used 10K most frequent words rather than using all unique words. I used only these words to create vectors for the sentences, all other words I mapped to <oov> (out of vocabulary). This was really crucial step in terms of computing cost.
- **Limit on sentence length**: On top of calculating frequent words, I set the limit on the maximum length of the sentence. the reason for doing this is to reduce the use of unwanted space for the short sentences in corpus representation matrix. I did experiments with few values and came up with 300 sentence length as optimal one.
- **Bigram Implementation**: For Bi-gram implementation, instead of creating all possible bi-grams and then using the top k frequent bi-grams, I used another approach, where I used top 5000 frequent words to create bi-grams for particular sentence. Doing this would take care of size of the new sentence with bi-grams and also we will ignore less important bi-grams. With this implementation, I observed a big change in performance. I was able to get very good and steady accuracy on dev data set very quickly.
- **Hyper Parameters tuning**: I used variable scalling initializer over xavier initializer to initialize the weights. Also I played with sequence length, embedding dimension, learning rate etc.

Accuracy Improvement with Fasttext:

As you can see in figure 2 and figure 3, with fasttext bi-gram implementation, we are getting the good accuracy at very early stage also we can see the notable difference in the accuracy achieved with bi-gram model (0.87) compare to uni-gram model (0.71).

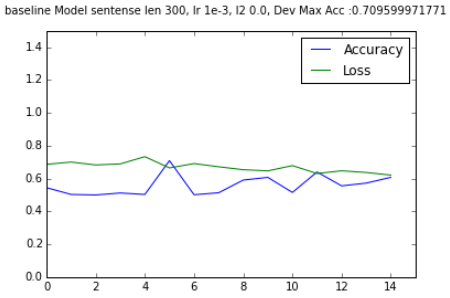

Figure 1: Baseline Uni-gram

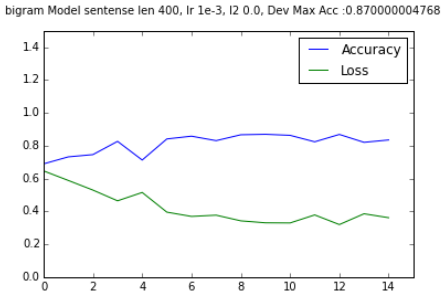

Figure 2: Fasttext bi-gram

Here Below is the parameters which worked best for me for two setups:

parameter	Baseline Uni-gram	Fasttext Bi-gram
Batch Size	50	50
Embedding dimensions	64	64
No. Frequent words	10K	10K
Bi-gram	-	from 5K frequent words
sequence length	300	400
optimizer	AdamOptimizer	AdamOptimizer
weight initializer linear layer	xavier	variance scaling
L2 Reg. LAMBDA	0	0
learning rate	0.001	0.001
no.of epochs	50	50
Drop Out Keep Prob	-	-

2 Fasttext Implementation

For the implementation of Facebook's fasttext code, I installed the library from github and proceed as per the instructions. I had to process the data and make it in consumable format for the algorithm to work (please find attached ipython notebook with email which I used to prepare the data).

I was actually surprised to see the speed of Facebook's fasttext. It hardly took few seconds for training and testing with almost the same result as we achieved with our time taking bi-gram implementation. I achieved 0.875 precision and recall for with the default settings. Although, when I tried same parameters as we had in our Fasttext implementation (lr 0.001, dim 64, epoch 50 etc.), It gave me only 0.55 precision and recall.

2.1 Comparison between our CBOW implementation and Fasttext

Here below is the best results I achieved with two setups I described,

Network	**Accuracy**
Baseline uni-gram	aprox $\tilde{0}.71$
Fasttext Bi-gram	aprox **0.87**
Facebook Fasttext	aprox $\tilde{0}.875$

3 Conclusion

In conclusion, Fasttext implementation with n-grams outperforms the simple uni-gram model in terms of speed and accuracy. Although, our implementation takes time for training, I noticed Facebook's implementation must be using some really good indexing/hashing techniques which makes the implementation really fast. I also noticed Facebook's fasttext implementation works well with default parameters.

4 References:

1. Fasttext Github Repository (https://github.com/facebookresearch/fastText)