DS-GA 3001: Assignment 2

Jiali Zhou, jz2312

October 16, 2016

1 Data Preprocessing and Word Embedding

1.1 Data Preprocessing

First for data preparation, I collected all the positive and negative movie reviews from training data and tokenized all the words. Then I fed them into a vocabulary and selected the top 10k frequent words, discarded the rest and labeled them as $< oov >$. Then for each of the sentence, I scanned it and indexed each of the word according to the vocabulary. The vocabulary and matrixes of index for each of the sentence would be fed into embedding layer for next step.
For splitting, the training set has 24000 samples, and validation set has 1000 samples. For testing, I used separate 25000 samples from testing data.

1.2 Continuous Bag-of-Word Classification(Word Embedding)

For CBOW model, I implemented word embedding for the raw input with embedding size = 64, maximum sentence length = 300. For each of the sentence, the input size will be (300, 64). Then I used the average of each of the columns in the matrix for every sentence. The output size of the embedding layer of the corpus is a matrix with size=(25000, 64).
After embedding layer, I implemented a fully connected layer with multiple output nodes. After adding bias, I used sigmoid for activation and added one dropout layer. This comes with the final layer with output size = 2 classes.

2 Experiments and Results

2.1 Experiments

For learning rate, although the MLP learning process doesn't overfit as severely as CNN, I added a decaying learning rate with decaying to 0.96 after every 3000 steps.
For dropout, I added it after the fully connected layer with $dropout\_keep\_prob = 0.8$.
For optimization method, I tried Adam, Adadelta, Gradient Descent and listed the result as below in Table 1, where MSL stands for maximum sentence length. I used n-gram features for word embedding, here n=2,3,4,5. And I found it works best when n=5. Here I listed the results of accuracy of testing set as well as the tuning parameters.
From Table 1, we can see that Adam Optimizer works best within 12 to 15 epochs, while Gradient Descent and Adadelta converges more slowly.

Table 1: List of Parameters and Results(MSL=maximum sentence length) without mask

n-gram(n)	MSL	epochs	optimization	eval loss	eval acc	test acc
1	300	15	GD	0.693	0.485	0.499
1	300	15	Adadelta	0.687	0.535	0.576
1	300	12	Adam	0.552	0.89	0.873
2	300	12	Adam	0.528	0.886	0.8722
2	500	12	Adam	0.522	0.898	0.8744
3	500	12	Adam	0.517	0.898	0.882
5	500	12	Adam	0.523	0.885	0.877
5	550	20	Adam	0.519	0.889	0.882

From the above, we can find that even though I tried the n-grams, the test accuracy doesn't improve much. This is because the maximum sentence length restricts the model from selecting n-gram features as they are added in the end of the feeding matrix of the embedding layer. One solution is to increase the maximum sentence length, the other way is to put the n-gram features along with the unigram. I've tried the both ways. Increasing the maximum sentence length to 550 can improve the accuracy by 1%. However after insert the n-gram features within unigram features, the accuracy in fact decreases to about 0.82. I guess it is because one thing the training process doens't converge, and second the actual features decrease, so I tried to increase the epochs and increase the maximum sentence length. After this, the accuracy comes to 0.83. From this, I infer this is because I didn't mask those padding sentences.
So the next step is to mask the padding sentence. The code is listed as below:

Listing 1: Mask Codes

```
# Embedding layer
with tf.device('/cpu:0'), tf.name_scope("embedding"):
    used = tf.sign(self.input_x)
    length = tf.reduce_sum(used, reduction_indices=1)
    length = tf.cast(length, tf.float32)
    mul_length = tf.tile(length,[embedding_size])
    y = tf.reshape(mul_length, [embedding_size,-1])
    self.mask_length = tf.transpose(y)
    # mat = tf.placeholder(1.0, [embedding_size,])
    # self.mask_length = tf.transpose(tf.mul(mat, length))
    embedd_W = tf.Variable(
        tf.random_uniform([vocab_size, embedding_size], -1.0, 1.0),
        name="W")
    self.embedded_chars = tf.nn.embedding_lookup(embedd_W, self.input_x)

    self.embedded_sum = tf.reduce_sum(self.embedded_chars,1)
    self.embedded_reduced = tf.div(self.embedded_sum, self.mask_length)
```

And I also listed the result as below. The results showed that after masking the padding sentence, the model worked better for longer features. However, still, the n-gram model doesn't work better than unigram model. I infer that this is due to the sparsity of n-gram features. The next step would be to implement Glove word representation for improvement.

Table 2: List of Parameters and Results(MSL=maximum sentence length) with mask

n-gram(n)	MSL	epochs	eval loss	eval acc	test acc
1	500	30	0.522	0.886	0.875
2	500	30	0.536	0.864	0.849
5	500	30	0.517	0.893	0.869
5	650	30	0.546	0.898	0.878
5	750	30	0.596	0.886	0.877

2.2 Evaluation and Test

Here comes the figure to show accuracy and loss of different parameters in dev set. From the figures, we can clearly see that accuracy starts to converge after 8k steps. Models start to overfit after 12k steps. So there should be a compromise between accuracy and loss.

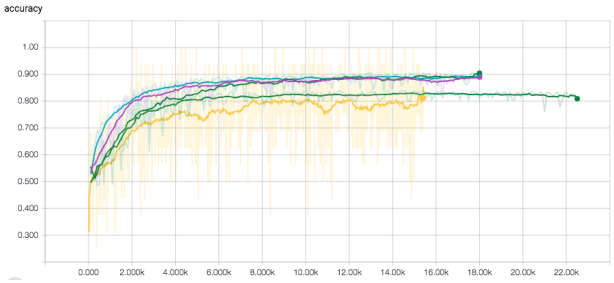

Figure 1: Accuracy in different settings of features in dev set.

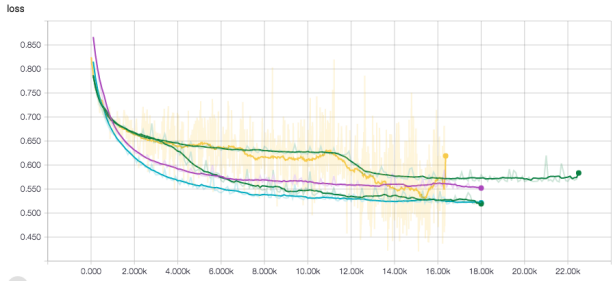

Figure 2: Loss in different settings of features in dev set.

2.3 Compared with FastText from Facebook

The basic setting from FastText tool from Facebook gives a result of accuracy= 0.845, which is worse than the model, but much faster. Setting the parameters to 5-gram and 30 epochs gives an accuracy = 0.85, which is still worse than the model.

2.4 Further improvement

The most effective changes are maximum sentence length and n-gram(n=5). For further improvement, I would suggest add more data for training and also try recurrent neural network as well as implement Glove for word representation to add information for the word order.