CMPS 242 Homework 5 Text Classification Report

Ran Xu, Sha Tong, Wen Cui, Lixue Zhang

1 Baseline - Logistic Regression

We reused the logistic regression model from homework 3, which implements BOW features together with Tf-idf to get the feature matrix. And then apply gradient descent with l2 regularization to optimize the cross entropy loss. As for model selection, we utilize 10 fold cross validation and select our best model to predict the final test set. The best model has loss of 0.20667 on the Kaggle competition test set.

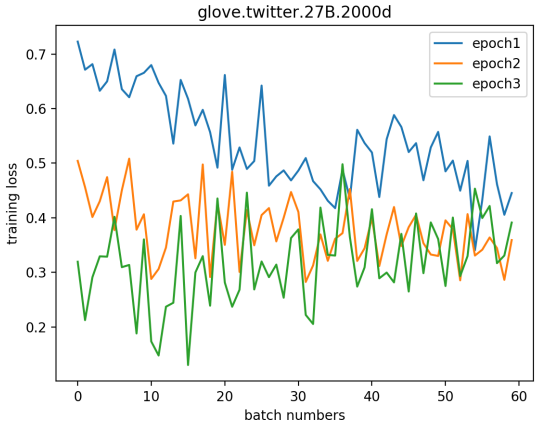
2 RNN Model

Then we implement our RNN model with the following structure in tensor flow. We feed our BOW feature matrix into a network consisting of an embedding layer, a LSTM layer. And then we take the last output to feed into a fully connected neuron network and then apply softmax to get our predictions. For the training, we do a batch update and split the data into 80% as training and 20% as evaluation.

2.1 Word Embedding Layer

- **Training Embedding layer** We implement a trainable embedding matrix with dimension of embedding dimension $\times$ vocabulary size.The vocabulary of our training set is 11293 whereas 1193514 in pre-trained model. We simply set the embedding matrix as a variable in tensor flow and during the training process, it will get updated.
- **Pre-trained language model** We also utilize the GloVe pre-trained language model based on twitter dataset.
- **Performance comparison** We compare the two embedding methods with 200 embedding dimension. (Other parameter set as BATCH_SIZE = 64, LEARNING_RATE = 0.001, DECAY_STEPS = 1200,DECAY_RATE = 0.9, DROPOUT_KEEP_PROB = 0.8). In Figure 1 we can see the training loss are smaller and stable in Figure 2a with trainable embedding.

(a) Training Loss on Trainable Embedding



(b) Training Loss on GloVe Pretrained Model

Figure 1: Performance on embedding methods

2.2 LSTM

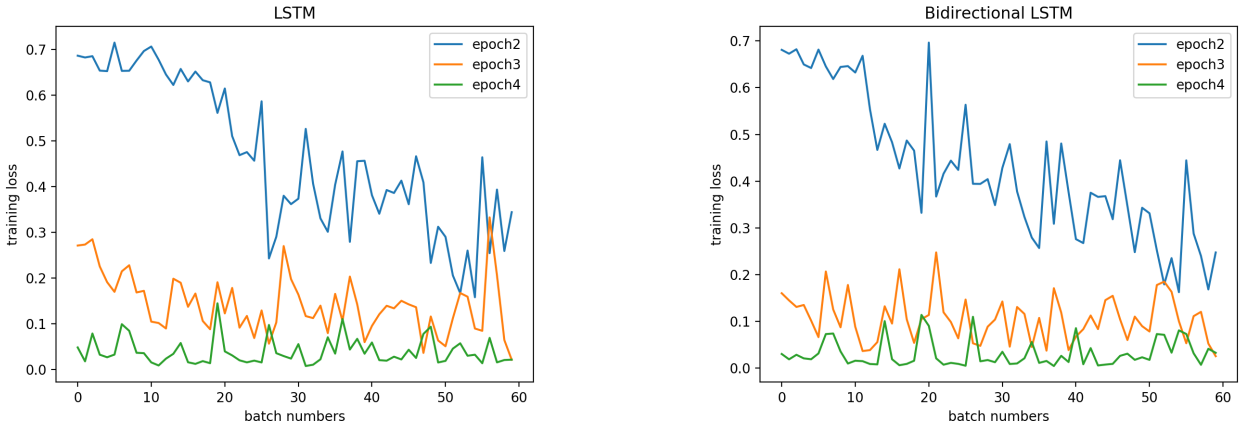
We used the static RNN cell in tensor flow and we also tried the dynamic RNN cell. But there is no significant difference except the syntax of building the RNN inputs.

2.3 Bidirectional LSTM

We also tried out the bidirectional LSTM with two basic RNN cell mentioned above and feed our input sequence in normal and reversed order. So that we utilize the most of information from inputs.

2.4 Performance Comparison

The intuition tells us bidirectional LSTM should outperform feed forward LSTM. Which is shown in Figure 2a. The training loss is slightly less and stable using bidirectional LSTM in Figure 2b. (Other parameter set as EMBEDDING = 150, BATCH_SIZE = 64, EPOCH = 3, LEARNING_RATE = 0.001, DECAY_STEPS = 1200,DECAY_RATE = 0.9, DROPOUT_KEEP_PROB = 0.8.)



(a) Feed-forward LSTM

(b) Bidirectional LSTM

Figure 2: Performance on different LSTM

3 Tuning Parameters

After we have our basic model set up, we have lots of parameters need to tune. Here we compare the performance and select our model based on the best parameters. Note that some parameters are highly related to each other. Like the learning rate and epochs. Which makes the tuning process challenging.

3.1 Optimizer

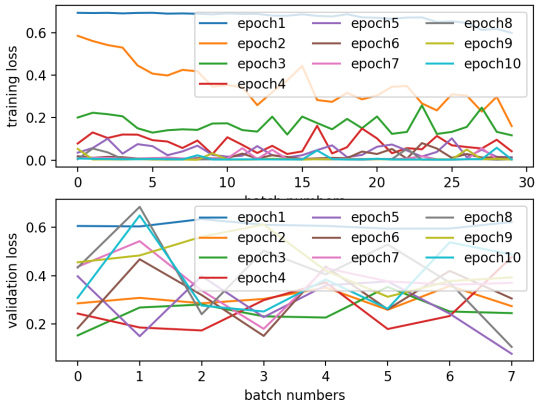
We tried different optimization methods, like Adam, SGD, Momentum. And Adam usually converges quickly in about 2 epochs it can reach 0.9 accuracy in training set.

3.2 Epochs

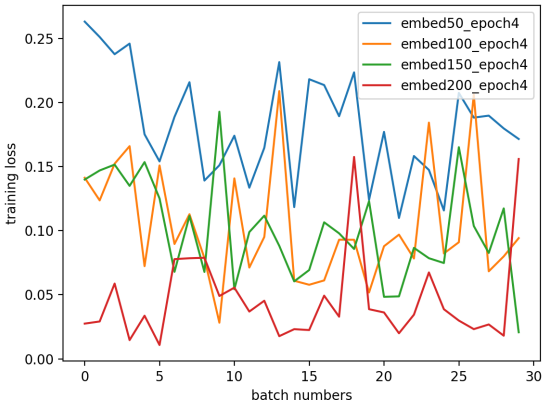
Since we have a relative smaller dataset for RNN training and the model over fits quickly. So early stop comes into place where we need to find the appropriate epochs for training. In Figure 3a shows the over-fitting happens when epoch is greater than 4 since the validation loss starts increasing. (Other settings are BATCH_SIZE = 128,DROPOUT_KEEP_PROB = 0.8,LEARNING_RATE = 0.001,DECAY_STEPS = 1200 ,DECAY_RATE = 0.9,EMBEDDING_DIM = 50.)

3.3 Embedding Dimension

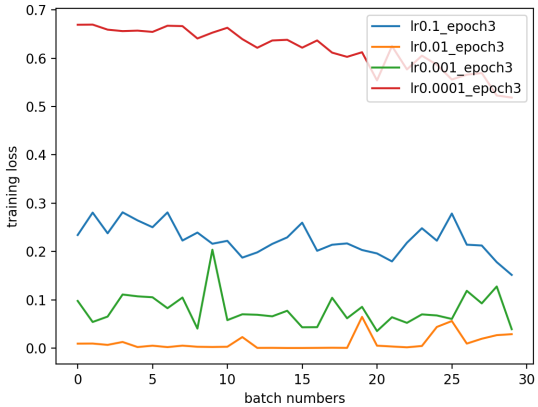
Again with relative small dataset, we need to find the appropriate embedding dimension. In Figure 3b we compare the training loss of embedding dimension of 50,100,150,200 on epoch 4. It turns out embedding dimension of 200 gives better performance.(Other settings are NUM_EPOCHS = 3,BATCH_SIZE = 128,DROPOUT_KEEP_PROB = 0.8,LEARNING_RATE = 0.001,DECAY_STEPS = 1200 ,DECAY_RATE = 0.9.)



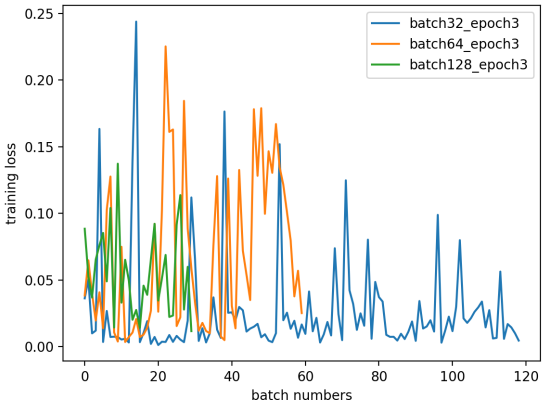
(a) Each epoch of training and validation loss



(b) Embedding dimension on only epoch 4



(c) Learning rate on only epoch 3



(d) Different batch size on only epoch 3

Figure 3: Parameters effect on training loss

3.4 Learning Rate

As mentioned earlier, just tuning the learning rate is not sufficient. Here we just show how the learning rate affect the training loss in Figure 3c with 0.1, 0.01, 0.001, 0.0001.(Other settings are EMBEDDING_DIM = 150, NUM_EPOCH = 3, BATCH_SIZE = 128,DROPOUT_KEEP_PROB = 0.8,DECAY_STEPS = 1200,DECAY_RATE = 0.9.)

3.5 Batch Size

We also explore whether batch size will affect our model. So we choose 32, 64, 128 batch size and compare the training loss in Figure 3d. It shows no significant effect since it may only affect the training time.

3.6 Dropout

We did try different dropout values (0.5, 0.8) where we didn't find significant improvement of performance.

Table 1: Test set loss comparison

Model	Test Loss
Logistic Regression	0.20667
Our Model	0.17469

4 Final Model and Results

Then from our experiments above, we get a better selection of parameters and show its performance in Figure 4. The final parameters are bidirectional LSTM with BATCH_SIZE = 128,DROPOUT_KEEP_PROB = 0.8, NUM_EPOCHS = 3,LEARNING_RATE = 0.001,DECAY_STEPS = 1200,DECAY_RATE = 0.9, EMBEDDING_DIM = 200. We compare the loss on Kaggle test set in Table 1, which shows our model has 15% improvement over the baseline.

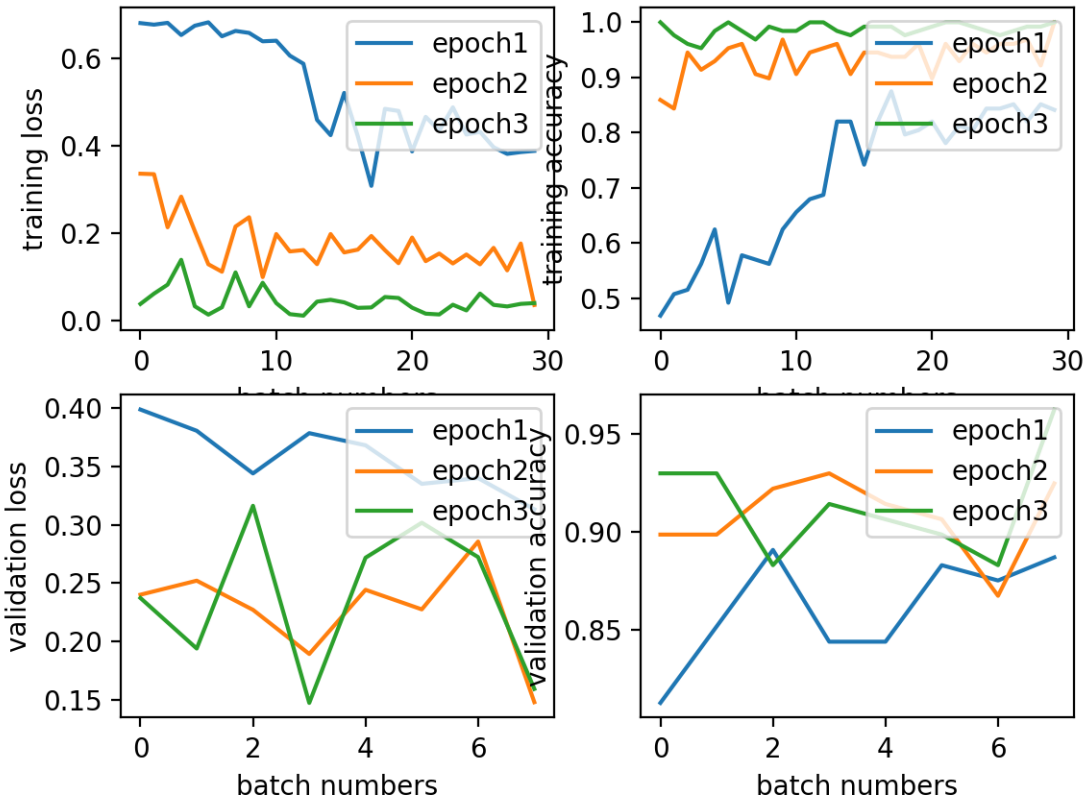


Figure 4: Best model performance