747 Assignment 1 Report

Zijing Gu

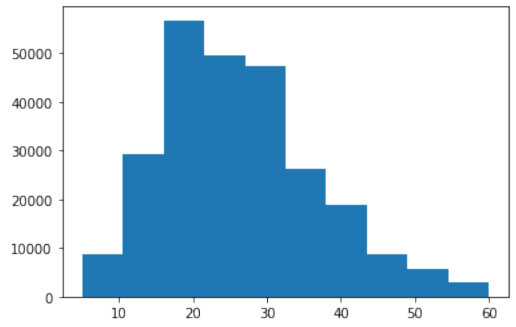
February 11, 2020

1 Introduction

The goal of this project is to train a text classifier from scratch and try to reach the test accuracy of 81.99%. Direct read and usage of any pre-trained word embedding is allowed, but not the contextual ones unless we re-implement by ourselves. In this project, I chose Fasttext as the pre-trained word embedding. Then I referenced and modified code from [2] and [3] to build two slightly different LSTM networks (pre-padding versus no pre-padding) using PyTorch. Then I trained the models and evaluate the validation accuracy.

2 Dataset

The dataset contains roughly 253909 training sentences, 643 validation ones and 697 test ones. Both training and validation sets are provided with labels while the test set is not. Each sentence is consisted of five to around seventy words. The below graph shows the distribution of the lengths of all sentences in training set. For each sentence, the label is a short phrase that indicates the topic of this sentence (ex., "Warfare", "History") and there are a total of 16 different labels. Our goal is to predict the topic of a given sentence.

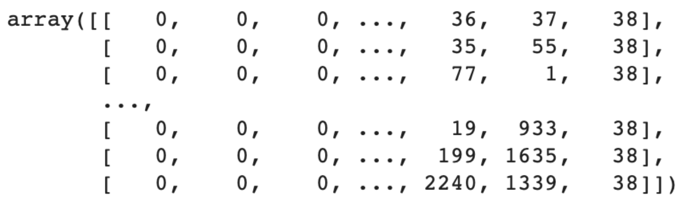


2.1 Pre-Processing

2.1.1 Words

After loading the datasets, I removed weird characters that does not appear in ASCII chart. For the training set, I split each sentence by space and constructed a dictionary that contains all vocabularies of the dataset. Then I mapped each word to a unique integer. Among these, `<pad>` is mapped to 0 and `<unk>` to 1. All the words that appears less than 3 times in the whole dataset are treated as `<unk>`. The indexed vocabulary size is 61867. Then I vectorized each sentence by replacing each word with its unique integer representation. Validation and test sets are encoded according to the training set word dictionary.

Pre-Padding Experiment Since the maximum sentence length is 76, I padded each sentence to 76 with all the zeros in the front. See below graph for details.



```
array([[   0,    0,    0, ...,   36,   37,   38],
       [   0,    0,    0, ...,   35,   55,   38],
       [   0,    0,    0, ...,   77,    1,   38],
       ...,
       [   0,    0,    0, ...,   19,  933,   38],
       [   0,    0,    0, ...,  199, 1635,   38],
       [   0,    0,    0, ..., 2240, 1339,   38]])
```

No Pre-Padding For simplicity of further exploration, I used TorchText to read data the tokenize each word. The feature array simply contains lists of sentences representations with different lengths. There are no paddings to make them equal length. We will use `pack_padded_sequence` later to ensure equal length.

2.1.2 Labels

I constructed another dictionary that mapped each of the 16 labels to a unique integer, and encoded the label vector by mapping the actual labels to their representing integers. Validation set label is encoded according to the training set label dictionary.

2.2 Embedding Matrix

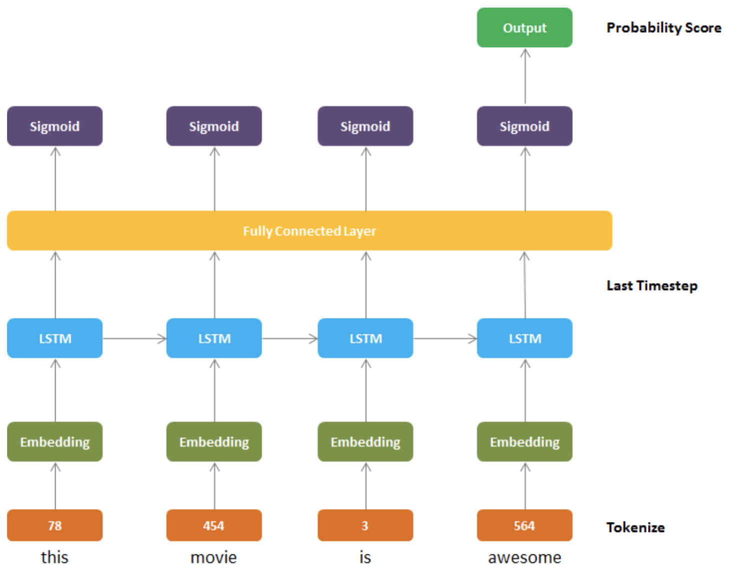
Next step is to construct the embedding matrix. I downloaded the Fasttext English embedding with 300 dimensions. The file is too huge to be loaded into memory, so I looped each row of the file, and if certain word is contained in my training set word dictionary, I extracted its embedding and put it to the matrix row whose number corresponds to the representing integer of that word. For other words that are not included in the Fasttext embedding, I randomly assigned values between -0.25 and 0.25 to their corresponding rows in the matrix.

3 Experiment

For universal hyperparameters, the vocabulary size is the length of the training set word dictionary. The output size is the number of classes which is 16. The embedding dimension is the the Fasttext embedding dimension which is 300.

3.1 Pre-Padding Experiment

The model structure has an embedding layer that converts word tokens into the corresponding word embeddings, a bidirectional LSTM layer, a dropout layer to prevent overfitting, a fully connected layer to map outputs to 16 classes, and a sigmoid layer to map output values to values between 0 and 1. During each epoch, initialize a pair of zero tensors for hidden state and cell state of LSTM. Pass input into embedding layer. Pass the resulting embeds and initialized hidden state to LSTM. Take the last node (should contain all information of the entire sequence) of each sentence within a batch and put into dropout layer. Put the resulting output to sigmoid function and return the output as well as the hidden layer for next round of training.



For the hidden dimension, I picked 300. And for number of layers I picked 2. For learning rate I picked 0.0005 and train for 15 epochs.

I constructed two dataloaders for training (shuffled) and validation (not shuffled) sets respectively with batch size 32.

3.2 No Pre-Padding

The model structure has an embedding layer just like the above model. But the main different between this model and the above one is that, right after embedding, we use `pack_padded_sequence` to take the padded input but ignores the padded part returns the hidden state of the non padded element, so that padding does not really affect the training. The results are then passed into LSTM and get back packed output and the pair of hidden state and cell state of LSTM. Since bidirectional, we concat the final forward and backward hidden state. The results are passed into fully connected layer and then sigmoid layer.

For the hidden dimension, I picked 128. And for number of layers I picked 2. For learning rate I picked 0.001 and train for 15 epochs.

Instead of using dataloader like the above experiment, I used BucketIterator to construct data iterators for training and validation sets. Since `pack_padded_sequence` requires sentences within each batch are sorted by length in descending order, the iterator will automatically to the sorting and padding within a sequence. The benefit is that, sentences of similar sequences are batched together and the paddings are reduced to minimum.

4 Results

4.1 Pre-Padding Experiment

I first tried the no-lowercase version. With a learning rate of 0.001, hidden dimension of 100, and 2 layers the validation accuracy can achieve over 78.1%. I tuned the hyperparameters a little with a learning rate of 0.0005, hidden dimension of 256 the validation accuracy can achieve over 80.6%.

Then I tried the lowercased version with learning rate of 0.001, hidden dimension of 128, and the validation accuracy is only around 77.8%. But with higher hidden dimension such as 150 or 200, the model was having a very hard time to learn.

4.2 No Pre-Padding

It surprising to see that training dataset with all sentences lower cased has over 2% better validation accuracy than original sentences (over 81% for lowercased dataset and around 78% for original dataset).

For the simplicity of exploration, I used the in-built GloVe 300 dimension as the embedding, with a learning rate of 0.01, batch size of 32, 128 hidden nodes, 2 layers and dropout rate of 0.3, and lower case the sequences. The validation was able to reach 84%! Maybe GloVe is more suited for this problem than Fasttext. For the original data (no lower sequence) and same settings, the accuracy is around 81.7, still better than Fasttext embedding. I tuned batch size to 64, the result did not improve (around 81%). I increased the hidden dimension and decreased learning rate, there is no improvement either.

Since the highest validation accuracy came from this set of experiments, I will output my develop results and test results from this model.

5 Future

I can try to further fine-tune the hyper-parameters such as learning rate decrease at later epochs, hidden dimensions, number of layers and batch sizes, even though these may not have huge affects on the current results. I can also initialize the word embedding matrix in different ways. For example, all zeros for `<unk>` embedding or Fasttext `<unk>` embedding rather than random initialize. More importantly, I can try the multichannel CNN introduced in [1] and make comparison with the LSTM experiment in this project.

References

[1] Yoon Kim. *Convolutional Neural Networks for Sentence Classification.* New York University, 2014.

[2] Sentiment Analysis using LSTM (Step-by-Step Tutorial) `https://towardsdatascience.com/sentiment-analysis-using-lstmstep-by-step-50d074f09948`

[3] Build Your First Text Classification model using PyTorch `https://www.analyticsvidhya.com/blog/2020/01/first-text-classification-in-pytorch/`