Homework 2 report

Sentiment classification on Yelp review using CNN and LSTM

Kevin Zhu

Dataset

The dataset was downloaded from the Yelp company directly. The dataset contains about 7 million reviews of businesses stored as a file formatted in JSON. Because of limited time and computing resources, I only selected 100,000 reviews and stored it in a Pandas DataFrame.

Preprocessing

The original reviews contain columns of review ID, user ID, business ID, stars, useful, funny, cool, text, and date. However, only text and stars columns are useful in this project. The first step was to map stars to sentiment. There are 5 stars ranging from 1 to 5 on Yelp. Reviews with 1 to 2 stars were mapped to negative, reviews with 3 stars were mapped to neutral, and the rest of the reviews are mapped to positive. I encoded those sentiments as -1, 0, and 1 accordingly and created a new column named sentiment, which was used as the label. Note that those mapping are changed to 0, 1, and 2 during the training stage of models to ensure positive loss.

The distribution of the sentiments is not uniform as shown in the following figure 1. There are more

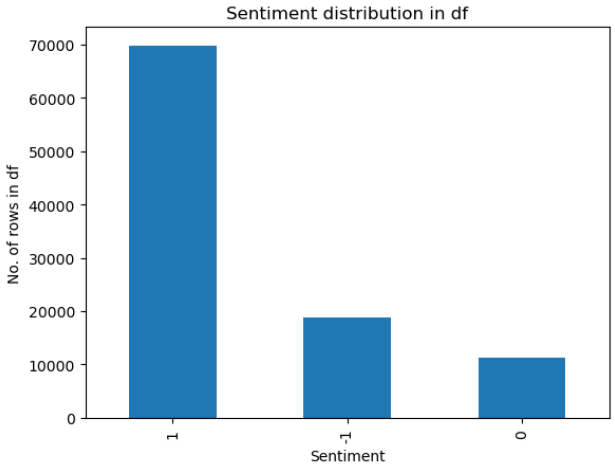

Figure 1. Sentiment distribution

positive reviews than neutral and negative ones. To make the model less biased towards positive reviews, the DataFrame will cut into a smaller one that contains equal number of 3 sentiments, with 10,000 samples for each sentiment.

Tokenization was applied to the text column. This is a required preprocessing step for most of the NLP tasks because we will need some sorts of embeddings for each of the word. Tokenizing the sentences is done by using Gensim's simple_preprocess function. Simple_preprocess will convert each sentence into a list of lowercase tokens, ignoring tokens that are too short or too long. Ignoring long or short tokens is helpful because rare words are hard to deal with and they usually do not convey any sentiments. The resulting list of tokenized sentences are store in another column.

Then, stemming was applied to tokenized sentences to map words to their root form. It is useful in sentiment classification. For example, a word can have many variants with prefix or suffix. Mapping words to their root form creates smaller vocabulary while retaining their meanings.

Finally, dataset was split into training set and test set with 70% and 30% ratios. Each set contains equal number of sentiments so the model would not be trained on biased data. Column stemmed tokens was used as input to ML models and sentiment column was used as label.

Model

Word2Vec

A word2vec model was trained on stemmed tokens in the entire dataset to get embeddings for each token. I used Gensim's untrained word2vec model and trained it using skip-gram algorithm. There was one extra token called pad added to the vocabulary for training. Because we need to pad the sentences into the same length for the models to work properly. The model will produce 500-sized embeddings. One thing to note that I think publicly available might not be appropriate for this project because embeddings for stemmed tokens from reviews might be hard to find.

Convolutional Neural Network

The model was implemented using PyTorch. It was defined with 1 embedding layer, 4 convolution layers, and finally followed by fully connected layers. The embedding layer applies pretrained word2vec embedding to each token in a sample. There are four sizes of kernels for convolution layers, 1, 2, 3 and 5. For each convolution layer, 10 kernels are applied to 2-D array of tokens with size (length of review) * (embedding size). Padding was also applied to each convolution layer to make the sliding kernels work fine without decreasing the length. Non-linear activation functions and max pooling are used after each convolutional layers to add non-linearity and reduce the size of the output. A final SoftMax layer is used to get the probabilities.

During training time, index of each word in the pretrained word2vec model are retrieved and fed into the model. Then the embedding layer will convert index to their corresponding embedding and create tensors. Then for each row in the training set, a 1-D list of embeddings is converted to 2-D tensor as if the embeddings formed an image. Each convolutional layer will have {[(window size) * (embedding size) + 1] * 10} parameters to be trained for. Finally, outputs of the convolution layers are fed into the fully connected layers and SoftMax layer to produce probabilities. The loss function used was cross entropy loss. It was trained using Adam with 0.001 learning rate for 30 epochs. The performance of the model was also tested properly.

Long Short-Term Memory

The architecture of LSTM follows a similar pattern with CNN but replaced convolutional layers with LSTM layer. The training and testing stage of LSTM follows the same method for LSTM. Total number of trainable parameters of an LSTM layer is 4 *[(embedding size) * (hidden size) + (hidden size)^2 + (hidden size)]

Results

CNN

The training time of CNN is around 39 minutes for both Tanh and ReLu activation. The accuracies of them are very similar with tanh slightly better. The following figures 2 and 3 show the loss of both activation function during training.

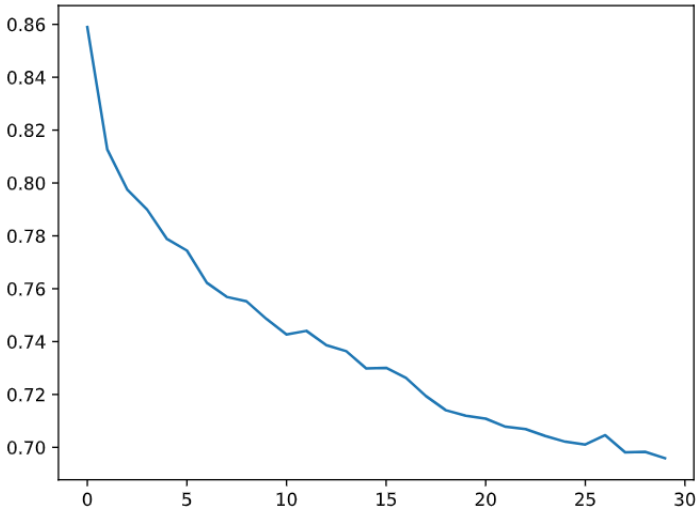

Figure 2. Loss of Tanh activation.

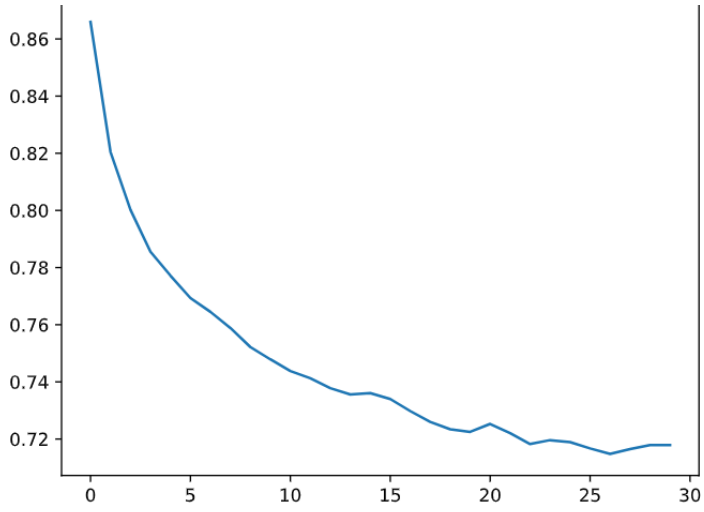

Figure 3. Loss of ReLu activation.

The classification reports of 2 models are shown in the following tables.

Tanh	Precision	Recall	F1-socre	Support
0	0.79	0.71	0.75	2992
1	0.64	0.60	0.62	3044
2	0.72	0.85	0.78	2964
Accuracy			0.72	9000
Macro average	0.72	0.72	0.72	9000
Weighted average	0.72	0.72	0.72	9000

ReLu	Precision	Recall	F1-socre	Support
0	0.79	0.74	0.77	2992
1	0.66	0.51	0.58	3044
2	0.68	0.88	0.77	2964
Accuracy			0.71	9000
Macro average	0.71	0.71	0.70	9000
Weighted average	0.71	0.71	0.70	9000

LSTM

The first model failed because LSTM did not work well with padding appended to the end of sentence. The length distribution of reviews is shown in the following figure.

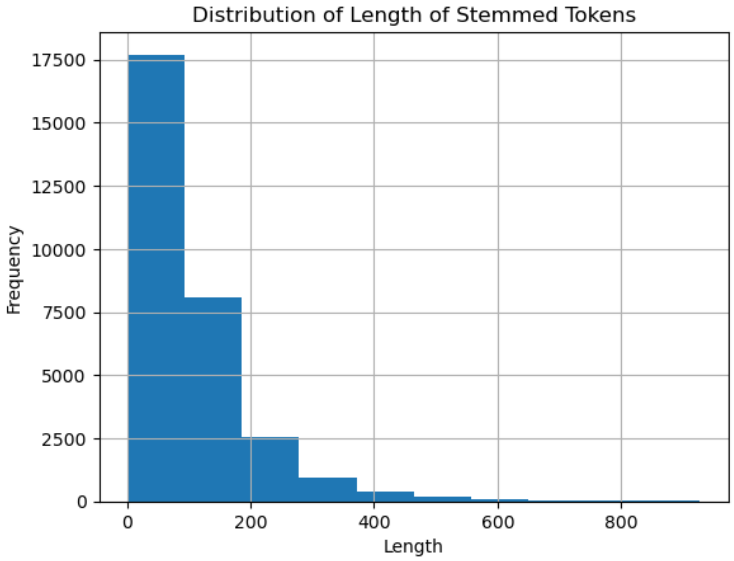

Figure 2. Distribution of length of reviews

The majority of the reviews lie within the range between 0 and 300 while the longest review was 927. The padding procedure was to make all reviews align with the longest review. Due to most reviews are padded with hundreds of meaningless token “pad”, the LSTM will not produce good results. Because LSTM processes a sequence from left to right and then produce an output. Instead, the model will make the same prediction over all samples, resulting in 33% accuracy because the sentiments are distributed uniformly for selected data. The issue can be solved by adding token “pad” at the beginning of sentences to keep all meaningful tokens near the end of sentences. The training time of 1 LSTM layer and 50 hidden layer size over 30 epochs is about 54 minutes.

The loss graph is shown in the following figure. The following table shows the classification report.

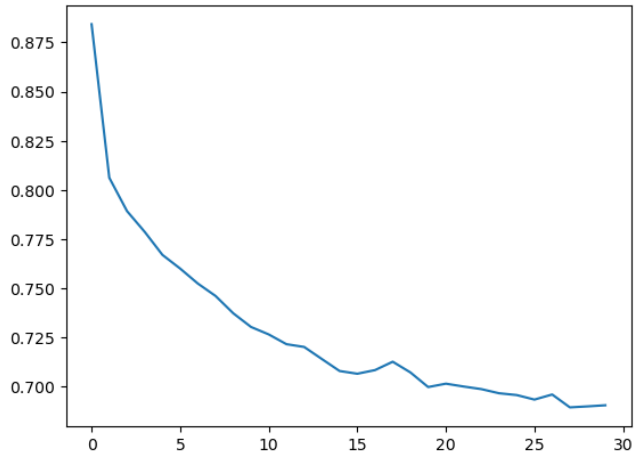

Figure 3. Loss of LSTM

LSTM	Precision	Recall	F1-socre	Support
0	0.79	0.76	0.77	2992
1	0.66	0.61	0.63	3044
2	0.76	0.84	0.80	2964
Accuracy			0.74	9000
Macro average	0.73	0.74	0.74	9000
Weighted average	0.73	0.74	0.73	9000

A bi-directional LSTM was also implemented to see if it can improve the accuracy. It took 72 minutes to train but resulted in different losses (shown in the following figure) but the same classification report with

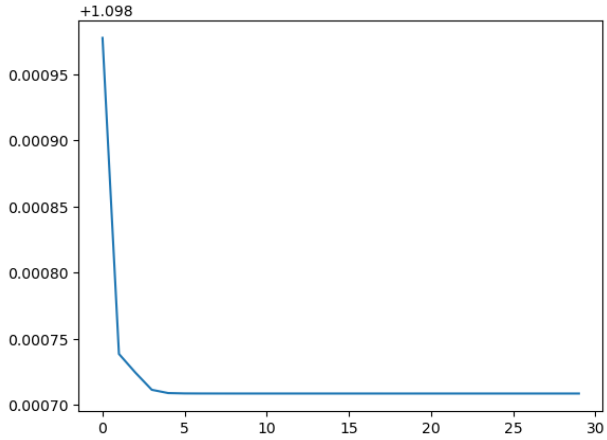

Figure 4. Loss of bi-directional LSTM

1-directional LSTM. The reason behind this is that there is one direction that does not generate good results at all, so the output was dominated by the "good" direction.

PyTorch also supports stacked LSTM, with the second LSTM taking in outputs of the first LSTM and computing the final results. It was trained for 30 epochs for 71 minutes and had similar accuracy with single LSTM layer model. Below are the loss graph and classification report.

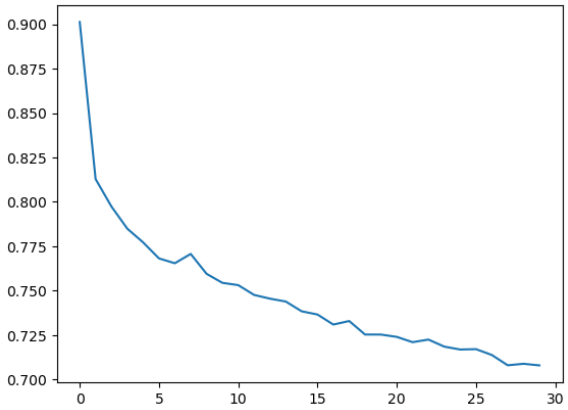

Figure 5. Loss of stacked LSTM

Stacked LSTM	Precision	Recall	F1-socre	Support
0	0.74	0.80	0.77	2992
1	0.65	0.58	0.62	3044
2	0.79	0.82	0.80	2964
Accuracy			0.73	9000
Macro average	0.73	0.73	0.73	9000
Weighted average	0.73	0.73	0.73	9000

Comparison

Time

training time of LSTM is longer than CNN because there are more trainable parameters. There are 80040 parameters for convolutional layers but 110200 for LSTM layer according to the equations from the previous sections. The embedding layer are the same and fully connected layers only differ by 10 in terms of number of neurons in the input layer. So, training LSTM should take longer as expected.

Accuracy

The LSTM model had slightly better accuracy over CNN by 2%. The result of CNN and LSTM should not differ much because they both take the context in consideration. CNN does it by using convolutions while LSTM does it by using recurrence. Tuning the hyperparameters will probably give a better result, but it is too time consuming and out of the scope of this project. Various sources pointed out that use convolution before LSTM layer can greatly increase the accuracy, but this can also be a part of the future work.