NLP Homework 2

Sentiment Classification using CNN and LSTM

Chitrang Goyani cbg5586@psu.edu

Introduction

The task assigned is to perform sentiment classification on yelp review dataset. We will be using both CNN and LSTM to perform this classification and compare the results of the same.

Dataset

The yelp review dataset consists of 1 million reviews with 9 columns. However, we will only be dealing with the 'text' and the 'stars' columns. Due to resource constraints, we will be loading only 500k reviews for training.

Training split: 80%

Validation split: 20%

Data-Preprocessing

The dataset provided is in .json format and needs to be converted to .csv format so that it can be loaded into a pandas dataframe. Since the number of reviews (rows) is 1 million, the data is read in chunks. Following is the list of pre-processing actions applied:

1. Read the data from .json in chunks
2. Filter each line to remove \n, \\n and \n\n
3. Convert to .csv
4. Load top 500k rows into a pandas dataframe
5. Generate labels according to the constraint that reviews having the number of stars > 3 will be assigned as a positive (1) review and <= 3 will be assigned as a negative (0) review.
6. Clean the data
    a. Remove punctuation
    b. Convert to lowercase and split them
    c. Remove stop words
    d. Perform stemming

Embeddings

- For each of the below methods, the cleaned data is tokenized with a vocabulary size of 500k.
- Max length of each review is capped to 100 tokens.
- For reviews that have < 100 tokens, padding is applied.
- The model will be supplied with either of the 3 types of embeddings:

1. Untrained embeddings randomly initialized by keras Embedding layer
2. Word2Vec pretrained embeddings on wiki news 1M with 300 features
    a. The wikinews.vec file is read using KeyedVectors from genism and converted to an embedding matrix of dimension

3. Glove pretrained embeddings on Glove 6B with 300 features.
    a. The glove6B.txt file is read using Python file library and converted to an embedding matrix of dimension

Performance of the model for each type of embedding technique is reported and compared.

Models

CNN

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 100, 300)          150000000

 conv1d_3 (Conv1D)           (None, 98, 10)            9010

 max_pooling1d_2 (MaxPooling  (None, 49, 10)           0
 1D)

 dropout_2 (Dropout)         (None, 49, 10)            0

 conv1d_4 (Conv1D)           (None, 47, 10)            310

 max_pooling1d_3 (MaxPooling  (None, 23, 10)           0
 1D)

 dropout_3 (Dropout)         (None, 23, 10)            0

 conv1d_5 (Conv1D)           (None, 21, 10)            310

 global_max_pooling1d_1 (Glo  (None, 10)               0
 balMaxPooling1D)

 dense_7 (Dense)             (None, 1)                 11

=================================================================
Total params: 150,009,641
Trainable params: 150,009,641
Non-trainable params: 0
```

LSTM

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 100, 300)          150000000

 lstm_2 (LSTM)               (None, 100)               160400

 dense_4 (Dense)             (None, 24)                2424

 dense_5 (Dense)             (None, 1)                 25

=================================================================
Total params: 150,162,849
Trainable params: 150,162,849
Non-trainable params: 0
_________________________________________________________________
```

Comparison of Neural Network Architectures

- Vocabulary size: 500k
- Dimension of feature vectors: 300
- Activation in CNN hidden layers: relu
- Activation in LSTM hidden layers: sigmoid

		Time to first epoch (s)	Validation Loss	Accuracy (%)
CNN	Untrained Random (Keras)	61	0.3248	90
	Glove	**85**	**0.2381**	**90.66**
	Word2Vec	63	0.2418	90.43
LSTM	Untrained Random (Keras)	249	0.2076	91.82
	Glove	**244**	**0.2025**	**92.19**
	Word2Vec	274	0.2026	91.87

Studies

- Change in activation functions in hidden layers:

		Loss	Accuracy
CNN (3 conv layers)	Tanh	0.2431	90.28
	Relu	**0.2381**	**90.66**
	Sigmoid	0.2519	89.97
LSTM (1 fc layer)	Tanh	**0.1805**	**93.11**
	Relu	0.2153	91.09
	Sigmoid	0.2025	92.19

Note: Since the final (output) layer in both the models is a binary classification layer, it does not make sense to use softmax (when we have only one neuron in the last layer) or relu (as we want output between 0 and 1), therefore sigmoid is a standard output layer activation function. Tanh on the other hand is possible to be used, however the data labels need to be [-1,1] instead of [0,1] and since in our case we have considered [0,1] labels therefore we stick to sigmoid.

- Change in input length (LSTM):

Input Length	Loss	Accuracy
50	0.2794	88.14
100	**0.1805**	**93.11**

- Change in embedding dimensions (LSTM):

Embedding dim	Time to first epoch (s)	Loss	Accuracy
300 (GloVe 6B 300d)	**270**	**0.1805**	**93.11**
100 (GloVe 6B 100d)	253	0.2170	91.26

- Change in optimizer (LSTM):

Optimizer	Loss	Accuracy
Adam	**0.1805**	**93.11**
rmsprop	0.2251	90.11

- Change in number of hidden layers:
    - CNN: In CNN the good fit number of hidden layers are 8 (3 conv1d, 3 maxpool, 2 dropouts)
    - LSTM: In LSTM the good fit number of hidden layers are 2 (1 LSTM, 1 Fully Connect)
- Change in number of LSTM units:

When changing from 50 to 100 LSTM units, performance improved by approximately 10% whereas going beyond 100 LSTM units was very resource intensive and hence 100 LSTM units were finalized.

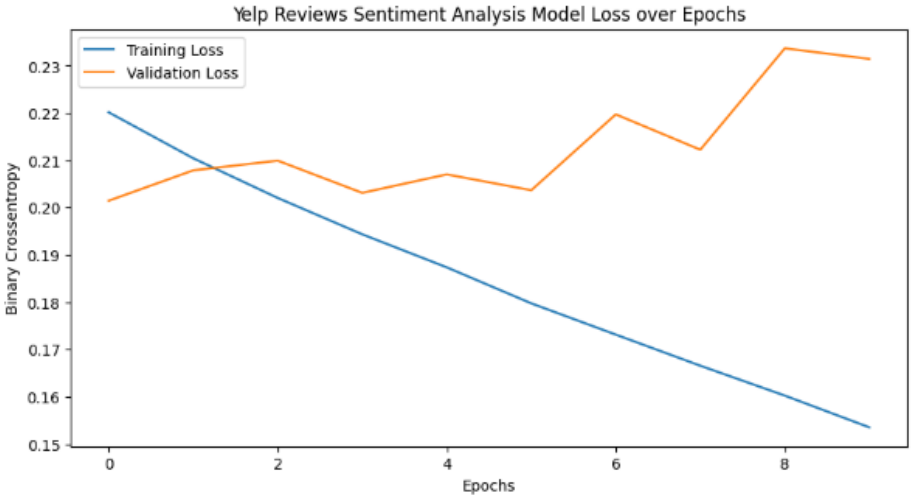
Results

LSTM Hyperparameters:

Epochs	5
Batch Size	1024
Optimizer	adam
Loss function	Binary crossentropy
Layers	Input embedding + 2 (LSTM + FC) + Output
Embedding Type	GloVe
Embedding Dimensions	300
Activation Layer	tanh
Learning Rate	(Default 0.001)

Validation loss: 0.1805

Accuracy: 93.11%



Test Cases

```
review = ['This is the worst  product I have ever used']
test(review)
```

```
1/1 - 0s - 59ms/epoch - 59ms/step
[0.0318245]
Negative
```

```
review = ['This is the best product I have ever used']
test(review)
```

```
1/1 - 0s - 66ms/epoch - 66ms/step
[0.9306341]
Positive
```

- We see the impact of worst and best as extreme words affecting the sentiment quite heavily.

```
review = ['This is a good restaurant to eat, however the service was slow!']
test(review)
```

```
1/1 - 0s - 45ms/epoch - 45ms/step
[0.35082385]
Negative
```

```
review = ['This restaurant had a bad vibe but the food was tasty!']
test(review)
```

```
1/1 - 0s - 43ms/epoch - 43ms/step
[0.47295862]
Negative
```

- We see an interesting impact of having a balance of positive and negative words on the sentiments.
- “Good restaurant” and “service was slow” results in a sentiment value which is closer towards the negative.
- However, “bad vibe” and “food was tasty” results in an almost negative sentiment.

```
review = ['This restaurant had an okayish vibe but the food was tasty!']
test(review)

1/1 - 0s - 63ms/epoch - 63ms/step
[0.7238563]
Positive
```

- Here we see how changing the vibe from “bad” to “okayish” makes the sentiment towards the positive side, showing that the model is sensitive to every token in the sentence.

Learnings

- Loading a large-scale dataset into pandas data frame using chunking.
- Cleaning the dataset including stop words filtering and stemming.
- Loading GloVe and Word2Vec embeddings.
- CNN implementation for sentiment classification and training.
- LSTM implementation for sentiment classification and training.
- Sigmoid and tanh are more suitable activation functions for binary classification tasks as usually we output only one neuron with value between [0 and1] suitable for softmax or between [-1 and 1] suitable for tanh.
- In the tests, one can see that each positive and a negative word can have a huge impact on the sentiment value.

References

https://towardsdatascience.com/sentiment-classification-using-cnn-in-pytorch-fba3c6840430

https://www.embedded-robotics.com/sentiment-analysis-using-lstm/

https://www.kaggle.com/code/lystdo/lstm-with-word2vec-embeddings/script

https://keras.io/