Sentiment Classification using Convolutional Neural Network (CNN) & Long Short-term Memory Networks (LSTM)

Homework 2 - CSE582 Natural Language Processing

Rishu Verma
Department - EECS
Pennsylvania State University
State College, PA, USA
rfv5129@psu.edu

PROBLEM STATEMENT

Sentiment classification is a widely studied problem in the field of natural language processing (NLP). Deep learning models, such as Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM), have shown great promise in improving the accuracy of sentiment classification tasks. However, the effectiveness of these models depends on several factors, including the choice of hyperparameters, the size and quality of the training data, and the type of text preprocessing used. Additionally, there is still a lack of understanding of how CNN and LSTM models compare in terms of performance, training time, and the number of parameters required for effective sentiment classification. Therefore, the objective of this homework is to compare the performance of CNN and LSTM models for sentiment classification, by implementing both models and evaluating their accuracy and training time for Yelp academic review sentiment analysis dataset. In addition, we will explore the impact of different activation functions on the performance of these models. Furthermore, we will analyze the number of parameters required for effective sentiment classification using both models. This assignment will contribute to the understanding of the strengths and weaknesses of CNN and LSTM models for sentiment classification, and provide insights for future research in this field with a great learning experience.

1 INTRODUCTION

Sentiment Classification is the ability to accurately classify the sentiment of a text. These classifications, for example: can be emotions (angry, happy, sad, disturbed, etc.) or can be a general classification of an act like bullying, violence, negative, positive, etc. It has many practical applications, such as social media monitoring, customer feedback analysis, etc.

CNN is a type of neural network architecture that uses convolution layers to extract features from input data. Well, technically if we think about it in terms of mathematics it's more a correlation operation than a convolution. These convolution layers in the CNNs apply filters on the input data in order to identify patterns and features in the data and further reduce the data to the desired classes. In order to reduce the data to the desired classes these filters are applied multiple times on or after multiple layer operations thereby leading to the predicted output.

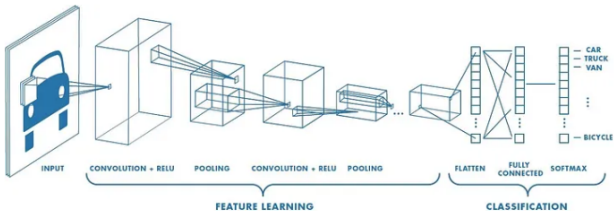


Figure 1: Basic CNN Architecture[10]

LSTM is another type of neural network architecture that is particularly suitable for sequential data. As the name suggests "Long-Short Term Memory", LSTM used memory cells to store short-term past data in order to learn and capture the long-term dependencies in the data to do better predictions. This is one reason LSTMs and RNNs are popular in the context of NLP where we need the context from the past text in order to make sense of the current or future text.

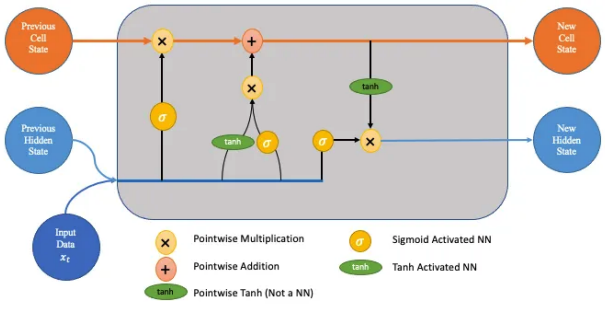
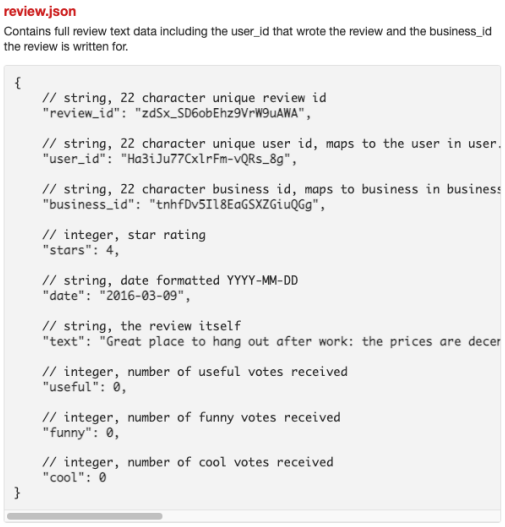


Figure 2: Basic LSTM Architecture[11]

With this excitement and motivation around CNN and LSTM, the implementation of these models is done on Yelp academic dataset. Further details regarding the implementation, dataset, and results are organized in the upcoming sections of the report: Section 2 provides describes the dataset and the preprocessing involved, Section 3 describes the model implementations, Section 4 presents the results and observations made after changing the activation functions and other hyperparameters, Section 5 discusses the challenges faced, Section 6 briefs about the learning outcomes, Section 7 concludes the report with a brief summary.

2 DATASET

This section provides a comprehensive overview of the dataset used namely: the Yelp Academic dataset[1,2]. The dataset is downloaded in the form of a '.json' file with a size of approx 5GB (GigaBytes). The dataset consists of 9 datapoints namely: "*review_id, user_id, business_id, stars, date, text, useful, funny, cool*". The data type of each of these components is shown in Figure 3.



[2] Figure 3: Structure of the dataset.

As the given data was in the '.json' format therefore in order to feed this data to the models it required some pre-processing. The preprocessing involved - .json to .csv conversion, data encoding, cleaning the outliers, tokenization, and stemming. Further, as the dataset was huge and due to resource limitations it was not possible to train and test the models on this huge dataset therefore, it was crucial to create a subset of the dataset which is small and balanced.

2.1 JSON TO CSV

In order to effectively perform this step, I first referred to the official Python converter provided by Yelp for .json to .csv conversion [3]. Although it's a well-written script it is outdated with the latest libraries and highly depends on *mrjob* (Map Reduce Library). Hence the output data was in a binary format which further required cleaning. Therefore, I wrote my own script using pandas [4]. In this script, the dataset is read in chunks, then normalized before appending it to the .csv file.

2.2 DATA INTERPRETATION

Our task is to classify the sentiments and the given dataset doesn't have labels for sentiments. Therefore, I have used the data from the "stars" column to interpret the sentiment - positive (1), negative (-1), or neutral(0). The encoding is done in a way that if a particular review has stars equal to 3 that means it's a neutral review, for stars less than or equal to 2 it's a negative review and the rest all are termed as positive reviews. This encoding gives us the label "sentiment" with three classifications. Further out of all the given columns, the most relevant column for sentiment analysis was the "text" column which has the reviews submitted by the user. Therefore, X (input) to the model is taken as the "text" and y (labels) is taken as the "sentiment".

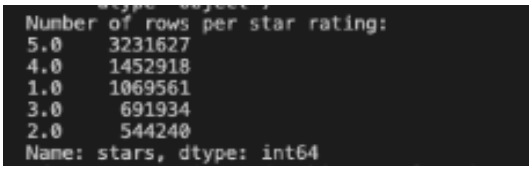


Figure 4: Distribution of the "stars" in the dataset

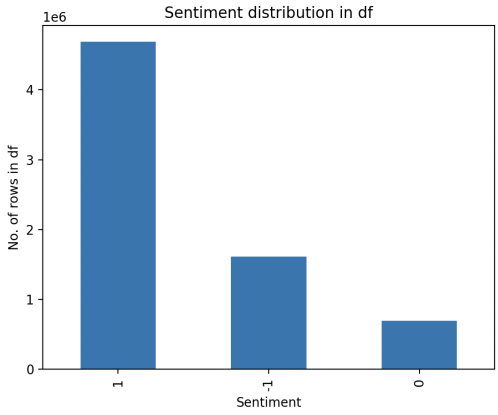


Figure 5: Data interpretation.

2.3 DATA PREPROCESSING

The X (input) is tokenized using simple_preprocess by gensim[5]:: this helped convert the input text to lowercase and remove all the punctuations (as can be seen in Figure 5). Also, as there can be multiple synonyms of the same word, and if we reduce those synonyms to their root words we can further reduce the processing load and better predictions. This is why stemming is applied to this

tokenized data. To stem, PorterStammer by Gensim is used because it is more efficient and faster as compared to another classic stammer in the NLTK or Gensim[6]. Also, for LSTM the stopwords were removed from the dataset to further reduce the load, this was done using the in-built function in NLTK.

```
1     [ve, taken, lot, of, spin, classes, over, the,...
3     [wow, yummy, different, delicious, our, favori...
4     [cute, interior, and, owner, gave, us, tour, o...
6     [loved, this, tour, grabbed, groupon, and, the...
7     [amazingly, amazing, wings, and, homemade, ble...
10    [my, experience, with, shalimar, was, nothing,...
11    [locals, recommended, milktooth, and, it, an, ...
12    [love, going, here, for, happy, hour, or, dinn...
13    [good, food, loved, the, gnocchi, with, marina...
14    [the, bun, makes, the, sonoran, dog, it, like,...
Name: tokenized_text, dtype: object
```

Figure 6: Sample tokenized text generated.

2.4 DATA SPLITTING

Before, splitting the data for training, validation, and testing a subset of data was picked from the whole dataset approx. 2-2.5 GB with the number of records equal to approx. 170000 (which was also the maximum number of records with a neutral sentiment. The selection was made in a way that the data from all three sentiment classes is picked in an equal ratio. After this, the subset was split into training, validation, and test sets in a ratio of 60:20:20.

3 IMPLEMENTATION[6,8,9]

In this section we will discuss the key implementations done to perform the analysis on the models. The input to both the models namely Convolutional Neural Networks (CNN) and the Long-Short Term Memory model (LSTM) is in the form of Embeddings[8] as the neural networks work on the numerical data. Word2Vec is used to convert tokens to embeddings. The embedding size is taken as 500 for both models, however, for the experiment and analysis purposes the embedding size was changed to 65, 200, and 400. While implementing the CNN model word2vec model was trained for 500 embedding sizes and this pre-trained model was reused for LSTM.

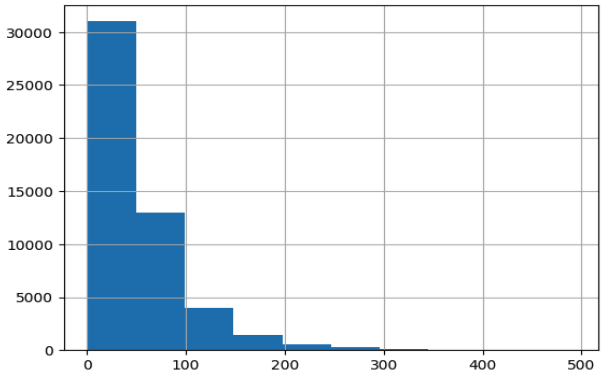


Figure 7: Number of sentences vs Length of the sentences

3.1 Convolutional Neural Networks (CNN)

The main component of CNN is the convolution layers and multiple convolutions and pooling operations are performed in these layers to get the required output from the network. In our case, the input to this network in the review from the user and the predicted output should be one of the three classes namely, positive (1), negative (-1), and neutral (0).
The parameters used in the baseline model are as defined:

- Number of classes (NUM_CLASSES) = 3.
- Embedding size for Word2Vec Embeddings = 500.
- Window_size is a list [1,2,3,5]
- Padding size = 400 (after analyzing the max length from Figure 7)
- Learning Rate = 0.001
- Number of filters = 10
- Activation Function = Tanh
- Loss Function = Cross Entropy Loss
- Optimizer = Adam
- Number of Epochs = 30

The vocabulary (number of unique root words in the corpus) was taken as top 1000. Conv2d function (from pytorch) is used to define the convolution layers as the input to the model is 2D word embeddings. The convolution filters are of the size 1*1, 2*2, 3*3, and 5*5.
In forward propagation, the model takes the input and converts it to word embedding. Then the embedded tensor is passed to the convolution layers which is followed by a 1D max_pool layer (which takes the maximum value across the sequence length and passes it to the next layer). Pooling techniques are used to reduce the output of the convolutional layer to a single value per filter. This is done for each window size with the output of each convolution layer passed via the activation function (in our case tanh).
After the input is processed through all the convolutional filters then the concatenated tensor is flattened and passed through the linear layer (also known as the fully connected layer) the output of which is passed through the softmax function to obtain the probability of the input to a particular classification.
Given the model involves multiple layers and each layer adds to the number of parameters. In the convolution layer, we have (number of filters) 10 * (window size) [1, 2, 3, 5] * 500 (embedding size) as the number of parameters. Also as the input, the linear layer is flattened hence the number of parameters in that layer is (number of filters) 10 * (window size) [1, 2, 3, 5] * (number of classes) 3.
The baseline CNN model was implemented with *tanh* as the activation function. Further, analysis was made by changing the activation function to *ReLu*, results of which can be found in section 4.
**(The trained baseline model can be found here saved here and the model implementation can be found here).

3.2 LSTM

In the case of LSTM the model takes the input as the sequence of words (X) and outputs the predicted classification (y_predit) along with the hidden state (h). As implemented in the case of CNN, likewise the input is passed through an embedded layer to get the word embeddings. This embedded tensor is then passed through the LSTM layer (for the current implementation, LSTM from torch.nn module is used). The output from this 2-layer LSTM mode is passed through the dropout layer - which helps prevent overfitting and makes the model more robust of the model by randomly dropping out the given fraction of the output samples from the previous layer. Then the output from this layer is fed to the fully connect layer and then to the activation function (Sigmoid in our case).

The parameters used in the baseline model are as defined:

- Batch Size = 50
- Padding Length = 400
- Embedding Dimension = 500
- Number of hidden nodes in LSTM = 256
- Number of LSTM Layers = 2
- Learning Rate = 0.001
- Gradient Clip Maximum Threshold= 5
- Number of Epochs = 4
- Activation Function = Sigmoid

The number of parameters[11] in each layer will be as LSTM layer inputs embeddings, therefore, ((embedding_size) 500* (hidden nodes) 256 * + (2 layers) 256 * 256) * (2 layers) 2 * (gates in the LSTM cells) 4. In the linear layer, the number of parameters will be (hidden nodes) 256 * (output classifications) 3. If we consider word embeddings as one layer then the number of parameters in that layer will be (vocabulary size) 33029 * (embedding_size) 500.

**(the model implementation can be found here).

4 RESULTS & OBSERVATIONS

- As we reduce the size of the word embedding a significant decline in accuracy is seen. The models showed the best accuracy at the embedding size of 400 and 500 (with the embedding sizes of 400 and 500 showing almost equivalent accuracy).
- Training time per epoch highly depends on the environment in which the training is happening Figure 6 shows the training time of the CNN on the local machine (CPU) taking approx. 6 mins/epoch and Figure 7 shows the training time taken on Google Colab (GPU) taking approx. 1 min/epoch. This reduced the computing time by almost 79% and I was able to train the CNN for 30 epochs.

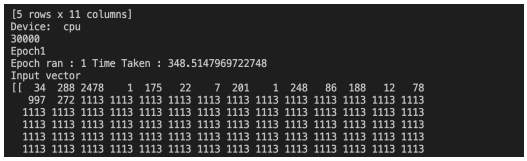


Figure 8: CNN runtime on the CPU

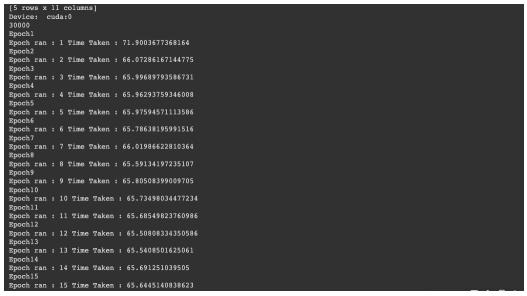


Figure 9: CNN runtime on the GPU

- Training time per epoch for baseline-CNN was on average around 68 seconds/epoch and for baseline-LSTM was observed as 141 seconds/epoch. Thereby showing an increase in the training time by about 51% from CNN to LSTM.
- The encoding of the classification has an impact on the predictions of the model. Upon passing -1 as the encoding for the negative sentiment to the LSTM model an interesting observation was found showing a terrible performance by the model with negative training and validation loss and an accuracy of just 45%. That is when I changed the encoding from positive(1), neutral (0), and negative (-1) to 0, 1, and 2 respectively.

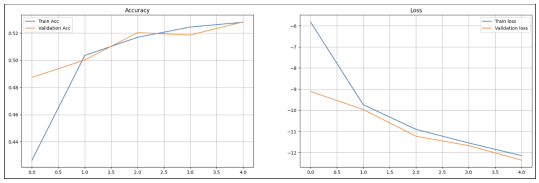


Figure 10: Performance decline in LSTM due to (-1) encoding

Also, another anticipated reason for this could be the activation functions because in the baseline-LSTM sigmoid is used which removed all the negative values. Using *tanh* in the LSTM indeed help solve the problem but due to time constraints, I could only run that model for one epoch.

- Validation accuracy of baseline-CNN when trained for one epoch = 71% and when trained on 30 epochs = 83%. Figure 11 shows the implementation loss vs epoch plot for 30 epochs showing a decrease in the validation loss with an increase in the number of epochs thereby indicating further training would improve the model's accuracy.

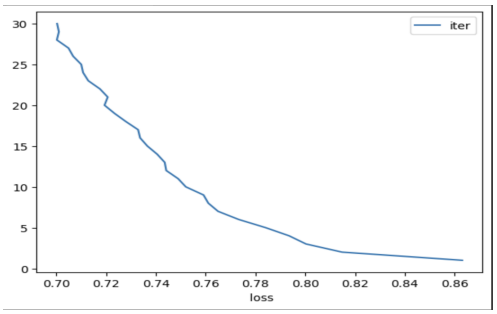


Figure 11: Validation Loss decreases as epochs increase (CNN)

- Validation accuracy of baseline-LSTM was 86%. It was observed that as the optimizer was Adam and the dropout value was 0.3 therefore the validation loss was overshooting but when the optimizer was changed to SGDM (Stochastic Gradient Descent with momentum) then the validation accuracy was 91% after 5 epochs. With average test accuracy of 95%
- Upon changing the activation function for CNN from Tanh to ReLu a very slight increase in the training time per epoch was observed (reference Figures: 8 and 12). However, the performance in terms of accuracy was almost the same. No major difference was observed.

```
Device:  cpu
30000
Epoch1
Epoch ran : 1 Time Taken : 398.0029389858246
Epoch2
Epoch ran : 2 Time Taken : 459.94134402275085
```

Figure 12: CNN (ReLu) runtime on the CPU

- After referring to the [7], I believe that Bi-direction LSTM has the potential to perform even better for sentiment classification tasks.

5 CHALLENGES

Lack of adequate computational resources: Besides training and testing the model (which was not much emphasized in the current assignment) data processing and cleaning require fast computing (given the dataset size was significant).

6 LEARNING OUTCOMES

Overall this assignment was a great learning experience giving us the chance to deep dive (hands-on) into CNNs, LSTMs, activation functions, and hyperparameter tuning. Not only this but this assignment also added to my software development skills as I got the chance to implement a project end-to-end single-handedly, learn Pytorch, and read more about the work done in the community. Thereby giving more insights into the wider research opportunities in this field.

7 SUMMARY

On concluding lines from all the implementations, experiments around the hyperparameters, activations functions, etc. It can be observed that training a CNN was quick and easy as compared to LSTM irrespective of any activation function. Although LSTM showed better performance in terms of accuracy as compared to CNN. The reason could be the architecture of the LSTM and the length of the reviews in our dataset. Overall this assignment was indeed a great learning experience giving us the chance to deep dive (hands-on) into CNNs and LSTMs.

REFERENCES

[1] Download Yelp Dataset
[2] Yelp Dataset Documentation
[3] Yelp - JSON to CSV
[4] Pandas Documentation
[5] Github: Gensim
[6] Data Preprocessing
[7] Detecting bullying on Tweets
[8] Word Embeddings for Sentiment Classification
[9] LSTM for Sentiment Classification on IMDB Movie data
[10] Basic CNN Implementation
[11] LSTM - In detail
[12] Course notes and slides