**ASSIGNMENT 2: SENTIMENT CLASSIFICATION**

* *Aim:*

Use Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) for sentiment classification.

* *Github Implementation:*

https://github.com/Vedantsahai18/CSE582-NLP-Assignment2-986195304

* *Observations:*

1. Dataset Overview –

The dataset used for the same is the Yelp Restaurant Review Dataset which consisted of millions of rows. However, for simplicity purposes, we selected the *top 100k rows*. During data analysis, we encountered data imbalance across classes, and thus to balance the classes I choose the top *10000 entries per sentiment for training-testing purposes*. Post the data selection, basic pre-processing techniques like tokenization, stop words removal, lemmatization, and stemming were applied to the final dataset which was then used to create the *VOCAB* for the model. Moreover, since the length of every sentence varies, for model training we had to make sure that every sentence has the same length. Upon further feature analysis, I came up with a *sentence length max to be 500 words* where sentences having a length greater than this are clipped, and less than this are padded with *pad* tokens. A *custom word–2–.vec model* was trained, which was used to generate the word embeddings for all the words.

*NOTE: The data and the embeddings used across both models remains the same for comparative purposes*

1. Convolution Neural Network (CNN) –

CNNs are a kind of deep-learning model that is regularly utilized for image classification. Nonetheless, they can likewise be utilized for text classification tasks, like sentiment analysis. In sentiment analysis, the goal is to classify a piece of text as positive or negative.

The benefits of involving CNNs in sentiment analysis are as follows:

* They can learn local patterns in the text, which can help identify sentiment.

The drawbacks of involving CNNs r sentiment analysis:

* They can be computationally expensive to train.
* They can be sensitive to the choice of hyperparameters.
* Model Specifics:

1. Hyperparameters

* Embedding Size: 500
* No of filters: 10
* Activation Function: Tanh/Relu
* No of Epochs: 5
* Optimizer: Adam
* Lerarning Rate: 0.001
* Batch Size: 32
* Loss Function: BinaryEntropyLoss

The total parameters for the below-given model are around *13445163*. However, out of these, the trainable parameter is around *55163*. The model was trained for 5 epochs only.

Text

Description automatically generated

Fig 1: CNN Architecture

The table below shows the comparative analysis of the performance of the CNN model with TanH and Relu as activation functions for the same set of hyperparameters and input.

A screenshot of a computer

Description automatically generated with low confidence A screenshot of a computer

Description automatically generated with low confidence

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Fig 2: Tanh as Activation Function Fig 3: Relu as Activation Function

From the above-given testing results, I observe that the if *Relu* is used as an activation function it performs slightly better than the *TanH*. This is because it does not suffer from the *vanishing gradient problem*, thereby increasing the performance of the model.

The graphs show the training and validation loss across the 5 epochs. However, after a certain epoch, the model is kind of overfitting and I believe with the proper set of hyperparameters we can overcome this overfitting and increase the robustness. Also, the classification report shows the metrics based on the testing set. On an overview level, the model can achieve an accuracy of *86%* and *87%* with Tanh and Relu as activation functions respectively.

Some improvement suggestions are as follows:

* Running a hyperparameter search to optimize your configurations.
* Using pre-trained word embeddings like Glove word embeddings
* Increasing the model complexity like adding more layers and dropouts

1. Long Short-Term Memory (LSTMs) –

Long short-term memory (LSTM) networks are a type of recurrent neural network (RNN) that are well-suited for tasks that require the model to remember information from previous inputs. This makes them a good choice for sentiment analysis, which is the task of identifying the sentiment (positive, negative, or neutral) of a piece of text.

Advantages of using LSTMs for sentiment analysis:

* They can learn long-term dependencies in the text, which is important for sentiment analysis.
* They can handle variable-length inputs, which is important for sentiment analysis.
* They can generalize well to new data.
* Model Specifics:

1. Hyperparameters

* Embedding Size: 500
* No of LSTM cells: 2
* Hidden Dimension: 512
* Activation Function: Tanh/Relu
* No of Epochs: 5
* Optimizer: Adam
* Lerarning Rate: 0.001
* Loss Function: BinaryEntropyLoss
* Batch Size: 50

The total parameters for the below-given model are around *17568433*. However, out of these, the trainable parameter is around *4178433*. The model was trained for 5 epochs only.

*NOTE: when compared to the CNN model, the LSTM model has way more trainable parameters and thus will eventually take more time to train per epoch. The below table shows an approximate per-epoch time taken by both models.*

|  |  |
| --- | --- |
| *CNN PER EPOCH TIME(s)* | *LSTM PER EPOCH TIME(s)* |
| *21* | *108* |

Text

Description automatically generated

Fig 4: LSTM Architecture

The table below shows the comparative analysis of the performance of the LSTM model with TanH and Relu as activation functions for the same set of hyperparameters and input.

Chart, line chart

Description automatically generated

A picture containing text, orange

Description automatically generated

Fig 5: Tanh as Activation Function

Chart, line chart, scatter chart

Description automatically generated

Text

Description automatically generated

Fig 6: Relu as Activation Function

From the above-given testing results, I observe that if *TanH* is used as an activation function it performs better than *ReLU*. This is because it has a bounded output range, which can help to prevent exploding gradients. Additionally, TanH is a smooth function, which can make it easier for the LSTM to learn.

The graphs show the training and validation loss across the 5 epochs. However, after a certain epoch, the model is kind of overfitting and I believe with the proper set of hyperparameters we can overcome this overfitting and increase the robustness. Also, the classification report shows the metrics based on the testing set. On an overview level, the model can achieve an accuracy of *86%* and *86%* with TanH and ReLU as activation functions respectively.

Some improvement suggestions are as follows:

* Running a hyperparameter search to optimize your configurations.
* Using pre-trained word embeddings like Glove word embeddings
* Increasing the model complexity like adding more layers/ using bidirectional LSTMs