SENTIMENT CLASSIFICATION USING CNN & LSTM

❖ *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

❖ *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 we 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.

- *Approach for Data Preprocessing –*

1. Used a mapping function as shown below to convert/map stars to the respective sentiments

```
# mapping stars to sentiment function
def map_sentiment(stars_received):
    if stars_received <= 2:
        return -1
    elif stars_received == 3:
        return 0
    else:
        return 1
```

Fig 1: Mapper Function

2. Upon feature analysis we select the **top 10k rows per sentiment** is selected to overcome the issue of the class imbalance as shown below

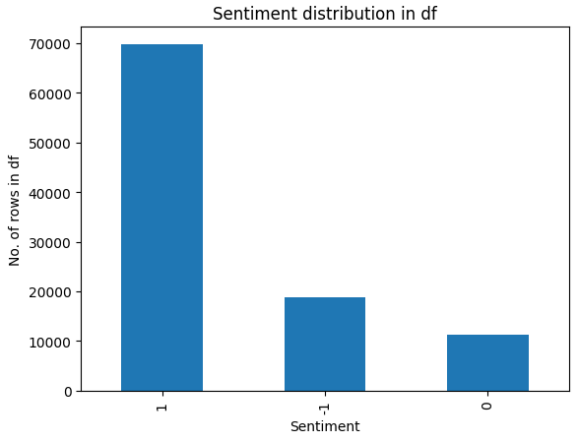

Fig 2: Class Imbalance post mapping per sentiment

3. Didn't remove the stop words to the newly selected dataset as it would remove the necessary words that would be needed to get the sentiments.
4. Achieved tokenization using Gensim simple_preprocess function
5. Performed Stemming using Porter library from Gensim
6. Trained a custom **word-2-vec model** on this new data frame with
   * Emebedding Size: 500
   * Padding token is used to fill extra remaining words, needed for sentences that are shorter than the longest sentence in the corpus. This is done to keep the length of sentences the same
7. The **Train Validation Test** Split is **80:10:10**

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

❖ ***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.
CNNs are well-suited for sentiment analysis because they can learn to identify patterns in text. They do this by applying a series of convolutions to the input text. A convolution is a mathematical operation that takes two inputs, a kernel, and a matrix, and produces an output matrix. The kernel is a small matrix of numbers, and the matrix is the input text. The convolution operation is applied to each word in the input text, and the results are then combined to produce a feature vector.

The benefits of involving CNNs in sentiment analysis are as follows:
- They can learn local patterns in the text, which can help identify sentiment.
- They are very efficient at processing large amounts of data.

The drawbacks of involving CNNs in sentiment analysis:
- They can be computationally expensive to train.
- They can be sensitive to the choice of hyperparameters.

- Model Specifics:

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: CrossEntropyLoss*
- *Learning Rate: 0.001*

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. The CNN

model consists of four different 2d layers with the final layer as a fully connected layer that outputs 2 features. These 2 features are nothing but the probabilities of the different sentiments.

```
CnnTextClassifier(
  (embedding): Embedding(26780, 500, padding_idx=1113)
  (convs): ModuleList(
    (0): Conv2d(1, 10, kernel_size=(1, 500), stride=(1, 1))
    (1): Conv2d(1, 10, kernel_size=(2, 500), stride=(1, 1), padding=(1, 0))
    (2): Conv2d(1, 10, kernel_size=(3, 500), stride=(1, 1), padding=(2, 0))
    (3): Conv2d(1, 10, kernel_size=(5, 500), stride=(1, 1), padding=(4, 0))
  )
  (fc): Linear(in_features=40, out_features=3, bias=True)
)
Total parameters: 13445163
Trainable parameters: 55163
```

Fig 3: 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.

	precision	recall	f1-score	support
0	0.71	0.82	0.76	1011
1	0.66	0.53	0.59	990
2	0.77	0.80	0.79	999
accuracy			0.72	3000
macro avg	0.71	0.72	0.71	3000
weighted avg	0.71	0.72	0.71	3000

	precision	recall	f1-score	support
0	0.75	0.80	0.77	1011
1	0.67	0.57	0.62	990
2	0.77	0.83	0.80	999
accuracy			0.73	3000
macro avg	0.73	0.73	0.73	3000
weighted avg	0.73	0.73	0.73	3000

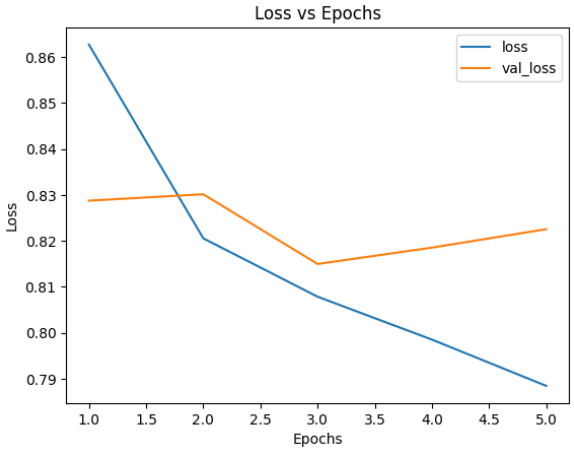

Fig 4: Tanh as Activation Function

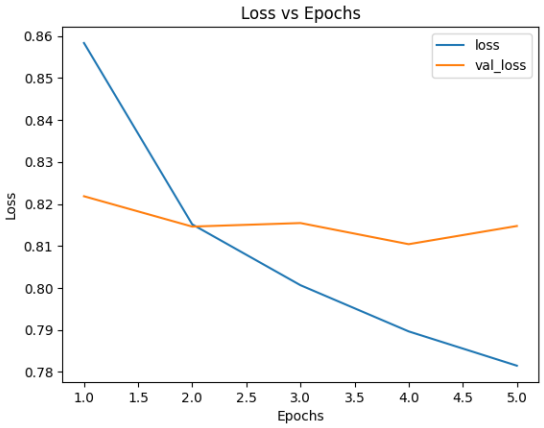

Fig 5: Relu as Activation Function

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 ***72%*** and ***73%*** with **Tanh** and **Relu** as activation functions respectively.

❖ ***LONG SHORT-TERM MEMORY (LSTM):***

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. LSTMs are well-suited for sentiment analysis because they can learn to remember information from previous inputs. This is important for sentiment analysis because the sentiment of a piece of text can often be determined by the context of the text. This makes them a good choice for sentiment analysis, which is the task of identifying the sentiment 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:

Hyperparameters

- *Embedding Size: (26780,500) for custom word-2-vec and (1001,64) for torch embeddings*
- *No of LSTM cells: 2*
- *Hidden Dimension: 512*
- *Activation Function: Tanh/Relu with Sigmoid*
- *No of Epochs: 5*
- *Optimizer: Adam*
- *Lerarning Rate: 0.001*
- *Loss Function: BinaryCrossEntropyLoss*
- *Batch Size: 50*
- *Learning Rate: 0.001*

The model was trained for 5 epochs only. Before going ahead with the final given LSTM architecture there were a couple of iterations I performed by changing the LSTM units and the activation functions. What I observed over there is that the hidden dimensions and LSTM units play a major role in the network to learn.

```
SentimentLSTM(
  (embedding): Embedding(26780, 500, padding_idx=1113)
  (lstm): LSTM(500, 512, num_layers=2, batch_first=True, dropout=0.5)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=512, out_features=1, bias=True)
  (tanh): Tanh()
  (sigmoid): Sigmoid()
)
Total parameters: 17568433
Trainable parameters: 4178433
```

Fig 6: 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.

1. *3 sentiments with Torch Embeddings:*

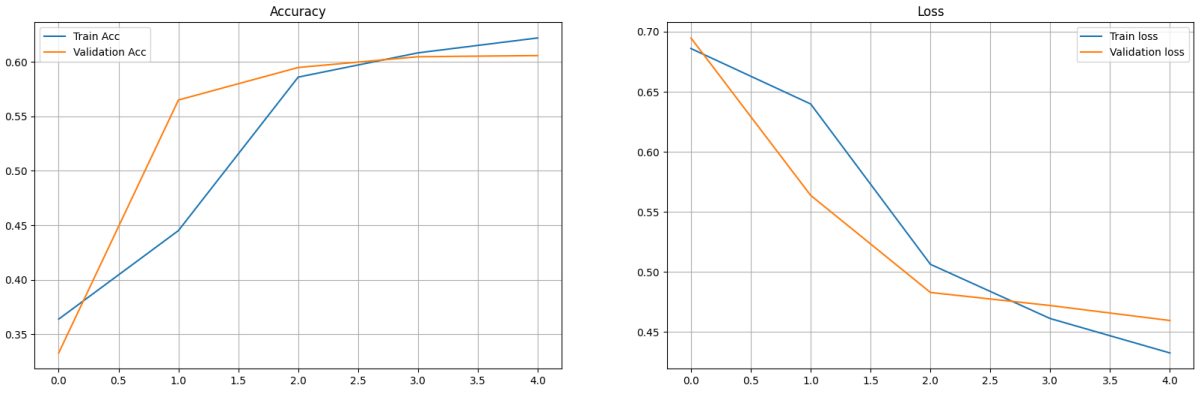

Fig 7: Tanh as Activation Function Results

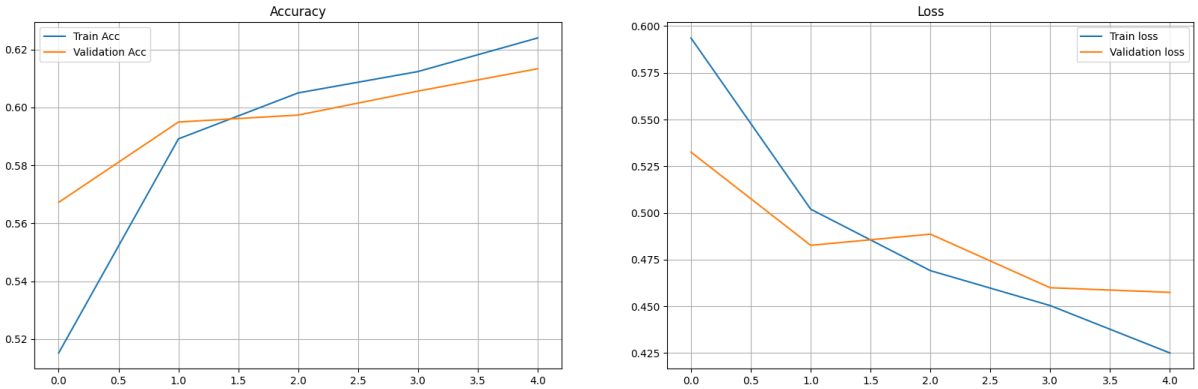

Fig 8: Relu as Activation Function Results

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.

On an overview level, the model can achieve an accuracy of ***68%*** and ***64%*** with TanH and ReLU as activation functions respectively on the testing dataset.

However, if we look at the test loss ***0.5124*** for the ReLU activation function is comparatively more than the TanH activation function which is ***0.4234***.

2. *2 sentiments with Custom Word-2Vec Embeddings:*

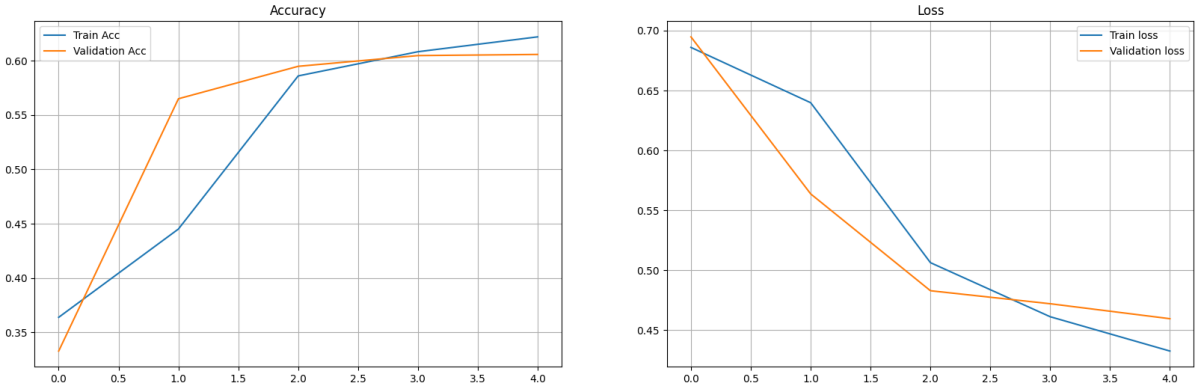

Fig 9: Tanh as Activation Function Results

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.

On an overview level, the model can achieve an accuracy of ***82%*** with **TanH** as the activation function and Sigmoid as the output activation function on the testing dataset.

However, if we look at the test loss of ***0.3729*** for the **TanH** activation function

❖ ***COMPARATIVE STUDY:***

- *Model Configuration vs Activation Function vs Training Time Per Epoch*

MODEL	ACTIVATION	TIME (s)	NO. OF SENTIMENTS AS OUTPUT	TRAINABLE PARAMETERS	EMBEDDINGS
CNN	*ReLU*	*85.34*	*3*	*55163*	*Custom Word-2-Vec*
CNN	*TanH*	*84.54*	*3*	*55163*	*Custom Word-2-Vec*
LSTM	*Sigmoid + ReLU*	*63.72*	*3*	*3349569*	*Torch Embeddings*
LSTM	*Sigmoid + Tanh*	*63.98*	*3*	*3349569*	*Torch Embeddings*
LSTM	*Sigmoid + Tanh*	*124.92 seconds*	*2*	*4178433*	*Custom Word-2-Vec*

Inference:

- Using ReLU as an activation function generally results in more training time
- LSTM has more trainable parameters than the CNNs irrespective of the output layers
  Reason:
  LSTM processes data one step at a time, in order. This can be slower than processing data in parallel, as CNNs do. Moreover, LSTMs are often used for tasks that require long-term memory. This means that they need to be able to remember information from previous steps in the sequence
- Embedding plays a huge role in deciding the overall working of the model and its efficiency. Using robust word embedding vectors can have a very huge impact on model learning.
  In this case, the Custom Word 2 Vec embedding performs better than the normal torch embeddings.
- The learning rate impacts the training time for each model a lot.
- Embeddings also impact the training time irrespective of the total trainable parameters.

- *Model Configuration vs Activation Function vs Accuracy after 5 epochs on a 30k rows*

MODEL	ACTIVATION	NO. OF SENTIMENTS AS OUTPUT	ACCURACY	EMBEDDINGS
CNN	*ReLU*	*3*	*73%*	*Custom Word-2-Vec*
CNN	*TanH*	*3*	*72%*	*Custom Word-2-Vec*
LSTM	*Sigmoid + ReLU*	*3*	*64%*	*Torch Embeddings*
LSTM	*Sigmoid + TanH*	*3*	*68%*	*Torch Embeddings*
LSTM	*Sigmoid + TanH*	*2*	*82%*	*Custom Word-2-Vec*

Inference:

- LSTM is better than CNN models in terms of accuracy as they can learn way better than CNNs due to local contextuality.
- On a training time basis, CNN is faster than LSTMs
- ReLU as an activation function between layers gives slightly better accuracy in CNNs but the trend reverses in LSTMs.
  Reason:
  1. In CNNs ReLU, helps to mitigate the vanishing gradient problem
  2. In LSTMs ReLU, it leads to the problem of "dying ReLUs"
- The model trained on the Custome Word-2-Vec model performs way better than the Torch embedding function.

- *Suggestions for Improvements are as follows*:

  - Running a hyperparameter tuning to optimize your configurations.
  - Using pre-trained word embeddings like Glove word embeddings

- Increasing the model complexity like adding more layers, filters, and dropouts in case of CNNs
- For LSTMs add depth to the layers, increase the number of units and maybe use Bidirectional LSTMs
- Increase the size of the dataset. Use the whole dataset instead of just 30k rows

❖ ***LEARNINGS:***

1. NLP concepts learned:
    - Tokenization: This is the process of dividing the text into individual words or tokens.
    - Word embeddings: This is a technique for representing words as vectors of numbers.
    - Sequence modeling: This is a technique for processing sequences of data, such as text.
2. Refreshed data preprocessing knowledge such as:
    - Cleaning the text: This is the process of removing errors and noise from the text.
    - Transforming the text into a format that the models can work with: This is the process of converting the text into a format that can be used by the models, such as a vector or a sequence.
3. Learned more about deep learning architectures such as CNN and LSTM concerning the NLP aspect
    - CNN: A CNN is a type of neural network that is commonly used for image recognition. However, it can also be used for natural language processing tasks, such as sentiment analysis.
    - LSTM: An LSTM is a type of recurrent neural network that is commonly used for tasks that require long-term memory, such as sentiment analysis.
4. Hyperparameter tuning for performance optimization. Tuned the following hyperparameters for performance optimization:
    - Embedding: This is a hyperparameter that controls how efficiently the model learns.
    - Activation function: This is a hyperparameter that controls how the model outputs its predictions.
    - Learning Rate: decides how fast the model should train.
5. Model evaluation
    - I evaluated the models to understand their strengths and weaknesses. The CNN model performed better on the training data, while the LSTM model performed better on the test data. This suggests that the CNN model is better at learning from the training data, while the LSTM model is better at generalizing to new data.

❖ ***CONCLUSION:***

In sentiment classification, the goal is to classify a piece of text as positive, negative, or neutral. CNNs can be used for sentiment classification by extracting local features from the text, such as the presence of positive or negative words. LSTMs can be used for sentiment classification by processing the text in sequence, taking into account the order of the words. Both CNNs and RNNs are effective for sentiment classification. However, CNNs tend to be faster and more efficient than LSTMs. LSTMs, on the other hand, are better at capturing long-range dependencies in the text.

In general, CNNs are a good choice for sentiment classification tasks when the text is short and the features are local. LSTMs are a good choice when the text is long and the features are long-range.

❖ ***REFERENCES:***

1 https://www.kaggle.com/code/arunmohan003/sentiment-analysis-using-lstm-pytorch
2 2https://towardsdatascience.com/sentiment-classification-using-cnn-in-pytorch-fba3c6840430
3 https://github.com/lukysummer/Movie-Review-Sentiment-Analysis-LSTM-Pytorch/blob/master/sentiment_analysis_LSTM.py
4 https://www.kaggle.com/code/pawan2905/imbd-sentiment-analysis-using-pytorch-lstm