**University of Athens**
**Dept. of Informatics & Telecommunications**

# Artificial Intelligence II (Deep Learning for Natural Language Processing)

Rontogiannis Dimitrios – `sdi1900165`

Fall 2022 - Homework 3: Sentiment Classifier with Bidirectional stacked RNNs and LSTM/GRU

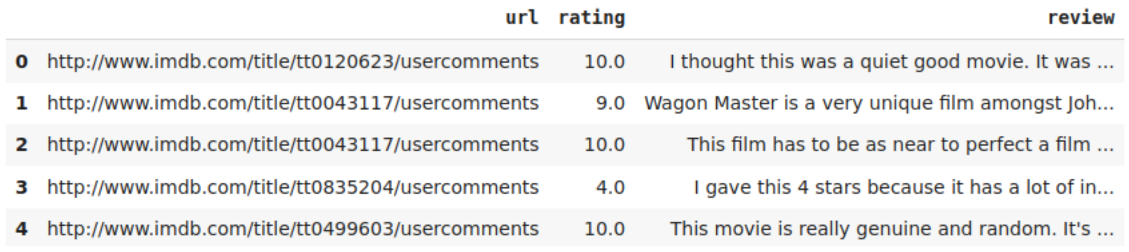
## 1 Introduction

We implement a sentiment classifier for the imdb-reviews dataset. We use bidirectional stacked Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) cells. We experiment with different architectures, including the number of stacked RNNs, use of attention, the number of hidden layers, the type of cells used, the dropout probability, etc. We use the Adam optimizer and the cross-entropy loss function. As input to our model, we use the GloVe word embeddings, for representing words in a numerical format. We will present the results and discuss the different architectures that we experimented with. We will analyze the impact of the different hyperparameters on the performance of our models, and finally, we will choose the best architecture for this problem.

## 2 Preprocessing - Cleaning data

The section below is **similar** with the one from the previous homework as preprocessing the data is an important part of the process. Text data contains noise in various forms like punctuation, text in a different case, symbols with no meaning etc. We perform **seven** different steps to make sure that our data will be ready to be fed to our model.

Initial head of data frame:



| | url | rating | review |
|---|---|---|---|
| **0** | http://www.imdb.com/title/tt0120623/usercomments | 10.0 | I thought this was a quiet good movie. It was ... |
| **1** | http://www.imdb.com/title/tt0043117/usercomments | 9.0 | Wagon Master is a very unique film amongst Joh... |
| **2** | http://www.imdb.com/title/tt0043117/usercomments | 10.0 | This film has to be as near to perfect a film ... |
| **3** | http://www.imdb.com/title/tt0835204/usercomments | 4.0 | I gave this 4 stars because it has a lot of in... |
| **4** | http://www.imdb.com/title/tt0499603/usercomments | 10.0 | This movie is really genuine and random. It's ... |

## 2.1 HTML

These reviews are taken straight from the IMDB site, so there are some HTML tags that do not affect the sentiment of the sentence. We use regular expressions to filter the tags.

## 2.2 Number

Numbers might indicate the number of stars a user rated the movie, but also they could indicate the time they watched it, it's price when it was released in the cinemas, etc. Therefore, we use reguler expressions to filter every numeric character.

## 2.3 Punctuation

Punctuation could show both positive and negative emotion. For example the exclamation mark in the sentence "I wish I could watch this movie again!" has a positive sentiment, but in the sentence "I turned off the tv after 5 minuntes!" has a negative sentiment. We use regular expressions to replace any non-word and non-sentence.

## 2.4 Uppercase

We lowercase every letter because our model would treat upper case and lower case. We would not want 'A' and 'a' to be seen as different characters.

## 2.5 Tokenization - Bonus step

Before we continue with the next preprocessing techniques we need to tokenize our text into separate tokens which are essentially individual words.

## 2.6 Stopwords

We remove these words, which have low-level information, from our text, in order to give more focus to the important information.

## 2.7 Lemmatization

Lemmatization helps us to achieve the root forms of words. We aim to remove inflectional endings only and to return the base or dictionary form of a word. This way, words like "give", "gave", "giving" will be treated equally.

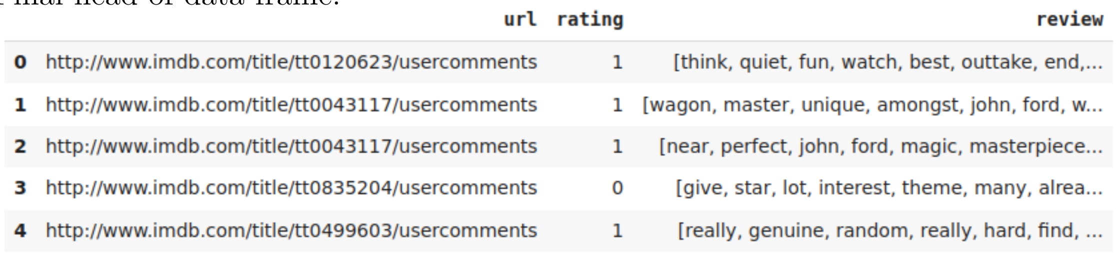
## 2.8 Rating - Bonus step

We know that a rating in the range $[0, 4.0]$ is categorized as negative and a rating in the range $[7.0, 10.0]$ is categorized as positive. As we will use Logistic Regression, we transform every negative rating to 0 and every positive rating to 1.

## 2.9 Rare/Frequent words

Rare and frequent words almost always offer no information for the sentiment analysis. For example words like ”this”, ”and”, ”so” etc. are not useful for our model training.

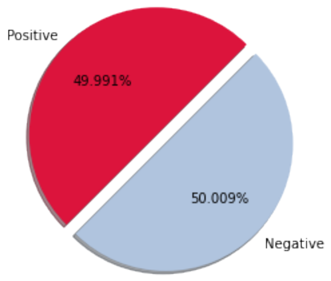
Final head of data frame:



| | url | rating | review |
|---|---|---|---|
| 0 | http://www.imdb.com/title/tt0120623/usercomments | 1 | [think, quiet, fun, watch, best, outtake, end,... |
| 1 | http://www.imdb.com/title/tt0043117/usercomments | 1 | [wagon, master, unique, amongst, john, ford, w... |
| 2 | http://www.imdb.com/title/tt0043117/usercomments | 1 | [near, perfect, john, ford, magic, masterpiece... |
| 3 | http://www.imdb.com/title/tt0835204/usercomments | 0 | [give, star, lot, interest, theme, many, alrea... |
| 4 | http://www.imdb.com/title/tt0499603/usercomments | 1 | [really, genuine, random, really, hard, find, ... |

## 3 Training

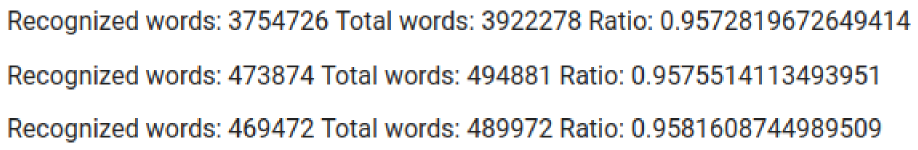
We train our model with data with almost the same number of positive and negative reviews:



## 3.1 GloVe

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. For our dataset we will use the pre-trained vectors included in the file **glove.6B.zip** take from stanford’s website. We will split our dataset in training, validation, and testing datasets and for each set we will vectorize our sentences by taking the mean vector of the vectors presented in the sentence.

The file **glove.6B.zip** contains vectors of 400.000 words but some of the words presented in our dataset may not be recognized. However, actually only around **0.06 percent** of the dataset is not recognized, which is very acceptable. Below you can the percentage of recognized words in the datasets for training, validating, and testing.



```
Recognized words: 3754726 Total words: 3922278 Ratio: 0.9572819672649414

Recognized words: 473874 Total words: 494881 Ratio: 0.9575514113493951

Recognized words: 469472 Total words: 489972 Ratio: 0.9581608744989509
```

Due to limitations in CUDA memory, we were only able to experiment with a maximum of 50 features. Attempting to utilize more resulted in the failure of the model.

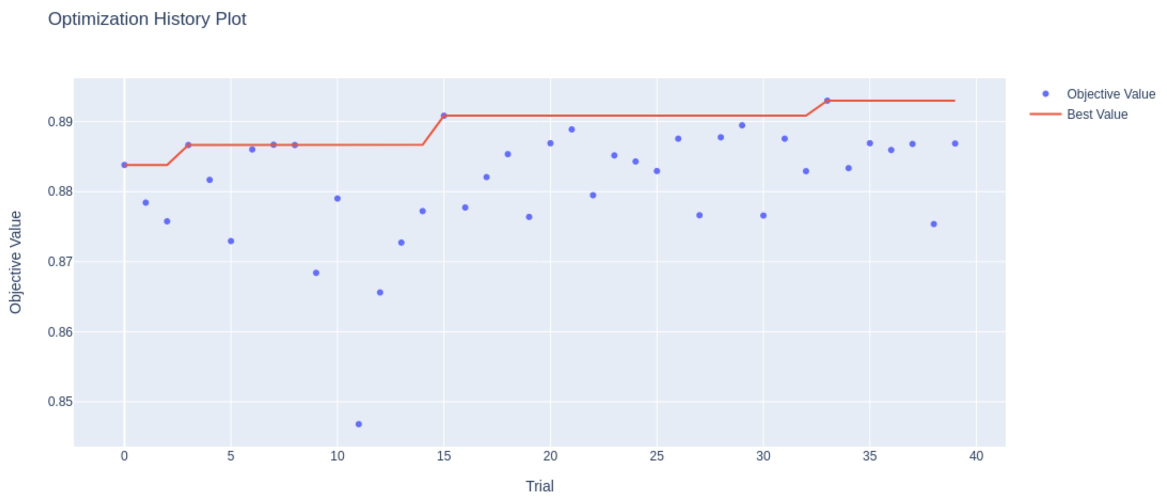
## 3.2 BiRNN with LSTM/GRU - Optuna

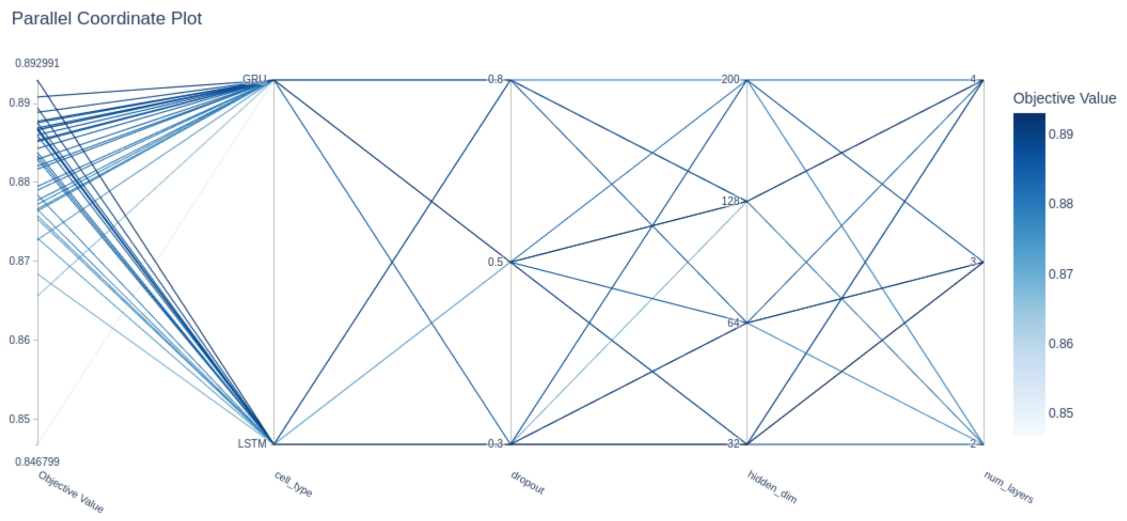
We build a **Bidirectional Recurrent Neural Network (RNN)** module that takes a review as input and output a value representing the sentiment of the review. The user can choose to use **vanilla RNN**, **Long Short-Term Memory (LSTM)**, or G**ated Recurrent Unit (GRU)**. It is initialized with the size of the vocabulary, embedding dimension, hidden dimension, output dimension, number of layers, dropout rate, padding index, and number of layers to be skipped. It also has an embedding layer, a dropout layer, and a linear layer, which connects the output of the RNN to the final output that is passed through a sigmoid function to make it in the range of 0 to 1. In the forward function, the review is passed through the embedding layer, the chosen type of RNN, and then the final hidden state of the RNN is concatenated and passed through the linear layer, dropout layer, and sigmoid function to produce the final output.

We use the **Optuna** library to optimize the model. We set some initial hyperparameters, including the input dimension, embedding dimension, and output dimension for the model. We use the trial object to suggest different values for the hidden dimension, cell type, number of layers, dropout rate, and number of skip layers for the model. Moreover, we use the trial object on number of epochs and the clip value. The validation accuracy is returned at the end of the training and the objective is to maximize it.

We performed multiple runs of the Optuna, with different values each time. We keep the hyperparameters of the model with the best validation accuracy.

Below we can see optimization history plot and the parallel coordinate plot.

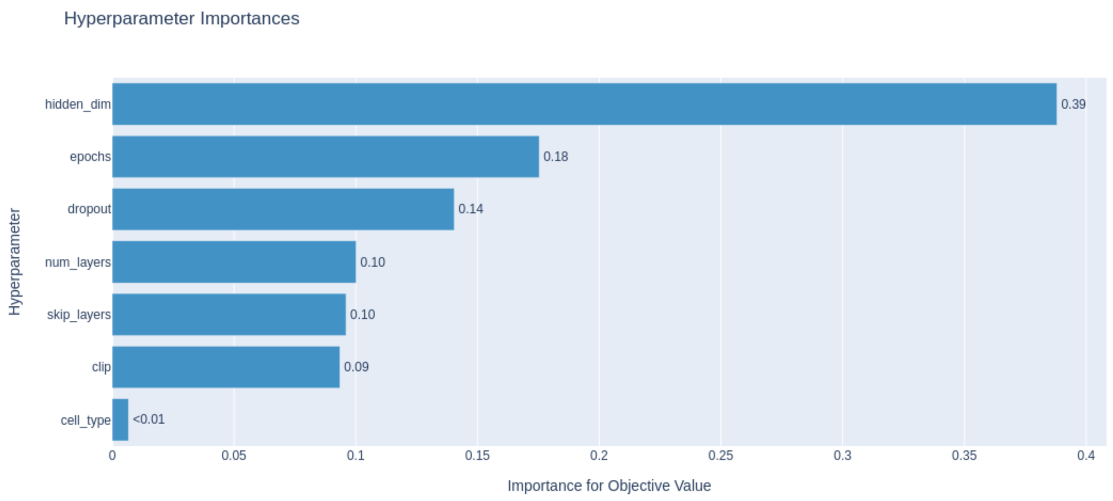




The last run of optuna found a model with accuracy of **89.8** in the validation dataset. It appears to be that the best model hyperparameters are:

- **hidden dimension**: 64
- **cell type**: 'LSTM'
- **number of layers**: 4
- **dropout**: 0.3
- **embedding dimension**: 1
- **output dimension**: 1
- **clip**: 5
- **skip layers**: 0

From Optuna's plot for the importance of the parameters we get the following plot:



We can see that the variable that affects most our model is probably the size of the **hidden dimension**. A larger hidden dimension means that our model will have more neurons in the hidden state. Having multiple neurons lead to learning more complex representations of the data and as a consequence improves performance (accuracy) of the model. However, more neurons in the hidden state means that the model will have more parameters to learn, which can increase the risk of overfitting, as the model starts to memorize the training data. On the contrary, smaller hidden dimensions limit the capacity of the model to learn patterns and this results in a naive model that does not overfit but also does not perform well, resulting in a lower accuracy.

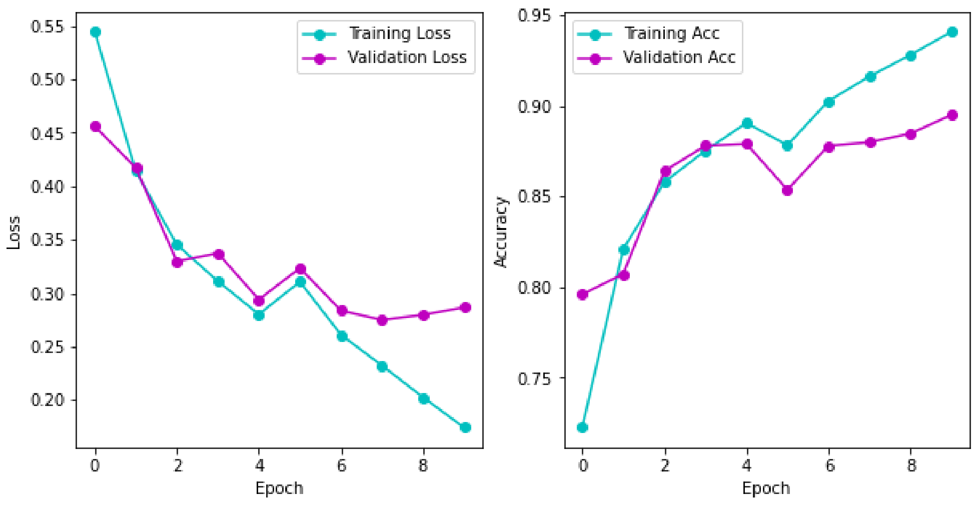
We can see that the number of **epochs** come second to the importance of affecting the accuracy of the model. It is true that the larger the number of epochs is, the more times the model will review the training data, learn from it and update the model's weights based on errors of previous passes. On the one hand, if the number of epochs is too low then the model will not have the opportunity to learn from the training data and it will result in lower accuracy. On the other hand, if the number of epochs is too high, the model will again start to memorize the training data, the performance will increase, but the model will probably overfit.

The third hyperparameter on the scale of importance for the model's performace is **dropout**, which is a regularization technique to prevent overfitting. It randomly drops out a certain percentage of activations during the training process. This improves the model's ability to generalize unseen data. However, in this case too a dropout rate that is too high may result in underfitting, because in this case the model does not have enough information to learn the patterns of the data. In contrary, a dropout rate which is too low may result in overfitting, where the model will start to memorize the data.

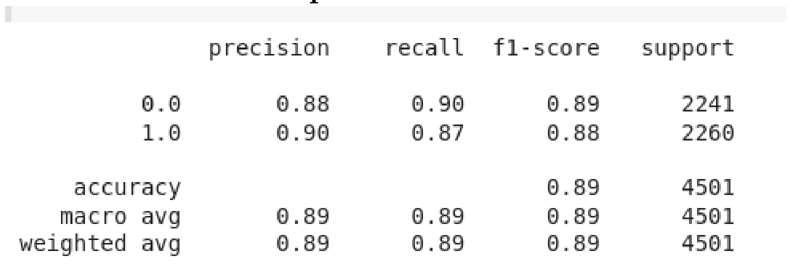
Concluding the discussion about the hyperparameters, one can observe that the decision of the best hyperparameters for our model is not an easy task. There are many factors affecting the problem and optuna does not always produce a model with good accuracy, or produces models that overfit.

## 4 Learning Curves

So now we have come to the end of our experiments with the best model. Below we can see all the necessary curves for our model. For the loss plot, **losses** for both training and validation start high and gradually decrease over each epoch, and have similar behaviour, resulting in alike lines. The same we can observe for the training and validation **accuracy**:



The **classification report** on the test dataset:



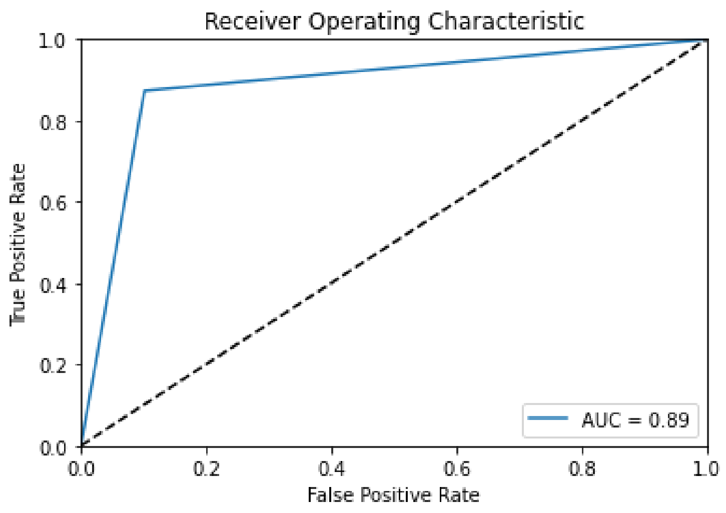
```
              precision    recall  f1-score   support

         0.0       0.88      0.90      0.89      2241
         1.0       0.90      0.87      0.88      2260

    accuracy                           0.89      4501
   macro avg       0.89      0.89      0.89      4501
weighted avg       0.89      0.89      0.89      4501
```

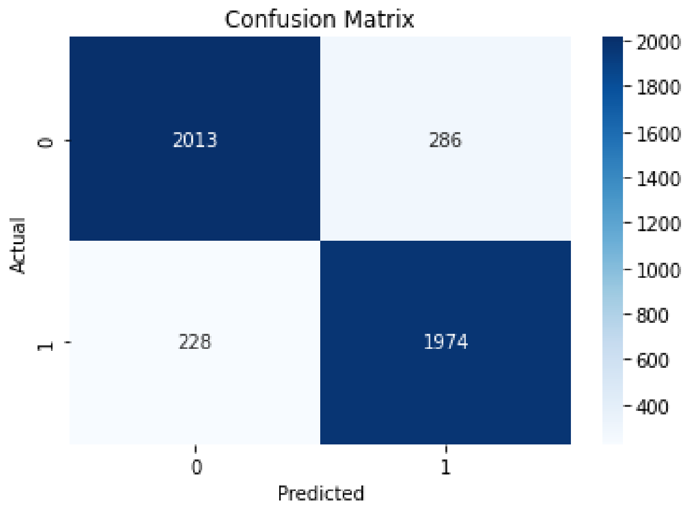
This is performance evaluation metric for our classification model. Precision is the number of true positives divided by the number of true positives plus false positives. This measures the model's ability to identify positive instances. Recall is the number of true positives divided by the number of true positives plus false negatives, and this measures the model's ability to find all the positive instances in the data. F1-score is the harmonic mean of precision and recall. The higher the F1-score the better performance the model has. We can see that the model has and **F1-score** of **0.89** which is quite good.

The **ROC curve**:



ROC curve is the graphical representation of the performance of a binary classifier. It is a plot of the true positive rate (TPR - true positive predictions divided by the number of true positive plus false negative predictions) against the false positive rate (FPR - number of false positive predictions divided by the number of false positive plus true negative predictions) at various threshold settings. One common way to compare models is to measure the area under the ROC curve (AUC) which ranges from 0 to 1. Our models has an **area of 0.89** which is quite good.

**The confusion matrix**:



The confusion matrix describes the performance of a classification model, where the rows are the true class labels and the columns are the predicted class labels. There are 4 cells. True positive (review where the classification was positive and the model also predicted positive). False Positive (reviews where the classification was negative but the model predicted positive). True Negative (reviews where the classification was negative and the model also predicted negative). False Negative (reviews where the classification was positive but the model predicted negative). We can see the the the cells for **True Positive** and **True Negative** are darker, meaning that most reviews were classified correctly.

## 5 Attention

After running all the experiments and built the necessary learning plots, we decided to experiment with attention also (a bit late). The basic idea behind attention is to **assign a weight** to each element of the input sequence. The bigger the weight, the more attention should the model pay to the element. After assigning these weights, they are used to create a weighted sum of the input sequence, which is used as inputs for the next layer. To summarize, attention mechanisms allow the model to select in which parts of the input sequence to **focus** which can improve the performance of the model on this kind of tasks.

In our implementation, attention was added as an option the BidirRNN class, and was implemented in the forward method. First of all, the output and hidden states are produced by passing the input review through the RNN layer and the embedding layer. Then, the hidden states are concatenated, and the skip connection mechanism is implemented. After that, if the attention flag is set to True, a linear layer is used to calculate the attention weights by passing the hidden states through the linear layer, and then applying the **softmax** function to the output. Then, the weights are used to weight the hidden states, by element-wise multiplication, creating a weighted sum of the hidden states. Finally, the output of the attention mechanism is passed through a sigmoid activation function and a linear output layer to produce the final output.

The results seem to be very **promising**. Most trials have a very good accuracy, with the best being near the best without attention. However, now the **importance** of **parameters** seem to **change**, with the epochs being first, hidden dimension size being second, and number of layers to be skipped being third.

## 6 Testing

The reviewer is welcome to change the values of dataset in the beginning of the implementation and testset path which is located near the end of the file. The reviewer is able to comment out the section of the code where optuna is used, in order to test the model faster. Moreover, the reviewer is advised to run the implementation on the GPU. Finally, some tools of torch are used that are in previous versions. In the beginning of the implementation, where the proper modules are imported, the reviewer should uncomment the first two lines with pip, then run, comment, then restart the runtime and run again.

## 7 Conclusion - Comparing

In the first assignment we had used **Logistic Regression with TFIDF vectorizer** for the sentiment classification of the imdb reviews. This had resulted in a **86 percent** accuracy. On the second assignment we had used **Feed Forward Neural Networks with GloVe** word embedding with **300** features and had gotten a model with **84 percent** accuracy. In this assigment, we designed a model with 89.8 percent accuracy with Bidirectionl Stacked Recurrent Neural Networs with LSTM.

Bidirectional Stacked RNNs with LSTM perform better than Logistic Regression as expected. Long Short Term Memory is able to capture long-term dependencies in the data, which is very important for classifying sentiment, because the meaning of a word can change depending on the context. On the other hand, Logistic Regression, which is a linear model, is not able to capture these long-term dependencies. Moreover, stacking multiple layers of RNNs allows the model to learn more complex patterns. In the contrary, Logistic Regression, is a simpler model that sometimes is not able to learn these complex representations, resulting in worse performance.

The gap in performance of Bidirectional Stacked RNNs with LSTM against the Feed Forward Neural Networks is also expected. With RNNs information of previous steps is maintained, which makes it a very good option for handling sequential data. In addition, with bidirectional connections the input reviews are processed from both directions allowing the model to consider the context from both past and future steps. On top of that, with Long Short Term Memory cells the models capture long-term dependencies in the data. Furthermore, utilizing stacked RNNs allows the model to learn more complex patterns, resulting in improved performance. Finally, with the option of Skip Connections, gradients are allowed to flow more freely which also may improve performance. On the other hand, Feed Forward Neural Networks process the review inputs in a single pass and make predictions based on the current input.