NLP ASSIGNMENT 2

NAME: SRUJAN ALUMULA  
UID: U42726372

* **Introduction:** **Objectives and overview of the work.**

Analysing and contrasting the effectiveness of character-level and word-level tokenization techniques on a text classification task using the IMDB dataset is the main goal of this work.  
Experimenting with different hyperparameters, such as activation functions, learning rates, batch sizes, and optimizers, is also intended to maximize the performance of the model.

Tokenization plays a crucial role in the text preprocessing significantly impacting the model performance. This study compares the two tokenization approaches to understand their influence on the model’s ability to classify the text data.

Hyperparameter optimization is used systematically to determine the best combination of parameters for achieving the better high accuracy.

* **Methodology:** **Detailed explanation of tokenization changes and hyper-parameter optimization strategy**.

Tokenization Changes :Character Level vs Word Level   
The main focus was on two distinct types that are character level tokenization and word level tokenization. The main objective was to compare their impact on the model performance, training efficiency and the final evaluation metrics.   
Character Level Tokenization  
Character level tokenization involves breaking down text into the individual characters, rather than the whole words. This approach is especially useful for handling rare words, misspellings, or language with complex word structures. In this project the following steps were followed for character level tokenization.

Text Preprocessing : The text data first preprocessed to convert all the characters to lowercase ensuring the consistency across inputs.  
Tokenizer Setup: A character level tokenizer was built using the tensorflow tokenizer api with the charlevel = true parameter. This tokenizer was then fitted on the training data to learn the vocabulary of characters.  
Bag of Characters – After Tokeinzation the text was transformed into the binary bag of characters representation using the matrix. This matrix represented the presence or absence of each character in the vocabulary.

**Hyperparameters Optimized:**

**Activation Function** : activation function is the how the weighted sum of inputs is transformed into an output. Serval activation functions were tested   
Leaky relu – A variant of relu that allows small nonzero gradients when the input is negative potentially preventing dead neurons.

Relu : A widely used activation function that outputs zero for negative inputs and the same value for the positive inputs.

Tanh : A squashed activation function with outputs between -1 and 1 often used the data distribution is centered around 0.

**Optimizer** : The optimizer is responsible for the adjusting the weights during training . The following optimizers were I tested.

Adam: An adaptive optimizer that computes adaptive learning rated for each parameter.

SGD: Stochastic Gradient Descent, a simple optimizer often used with momentum for faster convergence.

RMSprop: An optimizer that adjusts the learning rate for each parameter based on the recent gradient information.

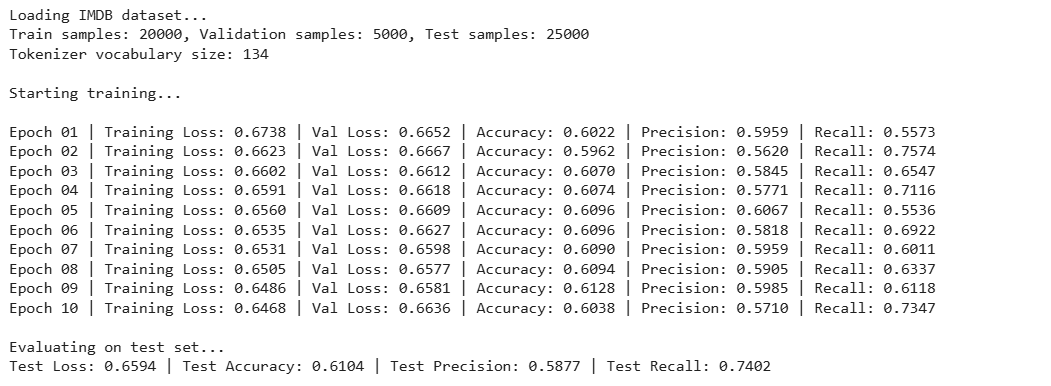
* And Also the remaining all I tested with these   
  **Learning Rate:** [0.001, 0.0005, 0.0001]
* **Hidden Layers:** [1, 2, 3]
* **Hidden Sizes:** [128, 256, 512]
* **Batch Sizes:** [32, 64, 128]
* **Optimizers:** [Adam, SGD, RMSProp]
* **Activation Functions:** [ReLU, Tanh, LeakyReLU]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SEED** | **Activation Function** | **Optimizer** | **Batch Size** | **Learning Rate** | **Testing Accuracy** | **Test Loss** |
| 655 | Relu | RMSProp | 512 | 0.0001 | 60.37 | 66.35 |
| 655 | Relu | RMSProp | 128 | 0.0001 | 60.78 | 66.06 |
| 655 | Relu | SGD | 128 | 0.0001 | 53.39 | 69.07 |
| 655 | Tanh | Adam | 512 | 0.0001 | 60.44 | 66.12 |
| 655 | LeakyRelu | Adam | 512 | 0.0001 | 60.44 | 66.25 |
| 655 | Relu | Adam | 512 | 0.0001 | 60.64 | 66.28 |
| 655 | Relu | Adam | 256 | 0.001 | 60.97 | 65.77 |
| 655 | Relu | Adam | 256 | 0.0005 | 60.65 | 66.11 |
| 655 | Relu | Adam | 256 | 0.0001 | 60.75 | 66.06 |
| 655 | Relu | Adam | 256 | 0.001 | 60.66 | 66.14 |
| 655 | Relu | Adam | 256 | 0.001 | 60.64 | 66.08 |
| 655 | Relu | Adam | 256 | 0.0001 | 60.51 | 66.15 |
| 655 | Relu | Adam | 256 | 0.0005 | 60.88 | 65.90 |
| 655 | Tanh | Adam | 128 | 0.0001 | 60.44 | 66.12 |
| 655 | Relu | Adam | 128 | 0.001 | 60.64 | 66.14 |
| 655 | Relu | SGD | 512 | 0.0001 | 60.22 | 70.54 |
| 655 | Tanh | Adam | 512 | 0.001 | 60.68 | 65.95 |
| 655 | LeakyRelu | Adam | 512 | 0.001 | 61.01 | 65.79 |
| 655 | LeakyRelu | SGD | 512 | 0.01 | 53.90 | 68.91 |
| 655 | LeakyRelu | RMSProp | 512 | 0.01 | 56.90 | 68.41 |
| 655 | LeakyRelu | Adam | 256 | 0.01 | 61.04 | 65.94 |
| 655 | LeakyRelu | Adam | 128 | 0.01 | 60.64 | 66.11 |
| 1567 | LeakyRelu | Adam | 256 | 0.01 | 58.74 | 60.81 |
| 45 | LeakyRelu | Adam | 256 | 0.01 | 61.14 | 65.83 |
| 655  (RANDOM)  model | LeakyRelu | Adam | 256 | 0.01 | 58.52 | 67.29 |
| 45 | LeakyRelu | Adam | 256 | 0.01 | 59.10 | 67.07 |

**Observation** :  
Among the activation function tested the LeakyRelu is performed consistently well achieving the highest accuracy of 61.16% with a test loss of 65.83 at 256 batch size and learning rate of 0.01.

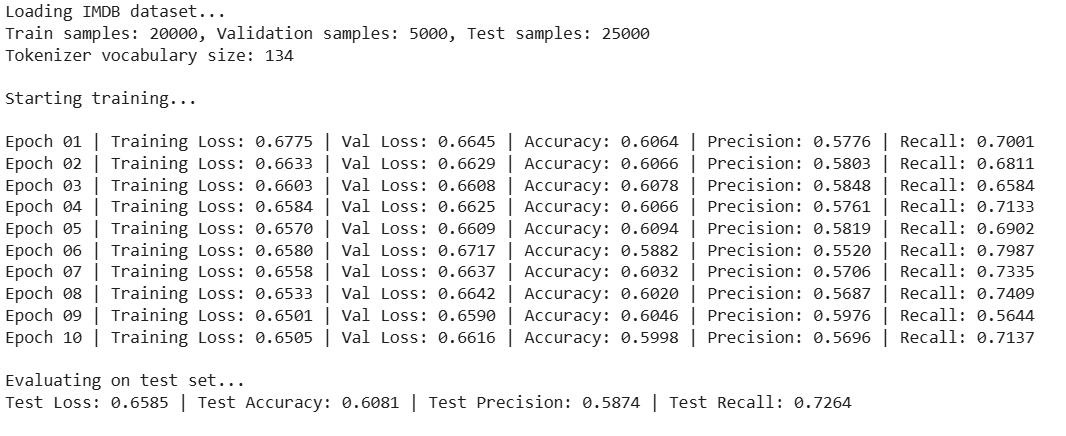
Optimizer : Adam appeared to be the most reliable optimizer across various configurations producing higher accuracy results compared to RMSProp or SGD.  
Batch Size : Larger batches generally resulted in lower accuracy and higher loss compared to medium batches sizes with the leakyrelu and adam optimizer.

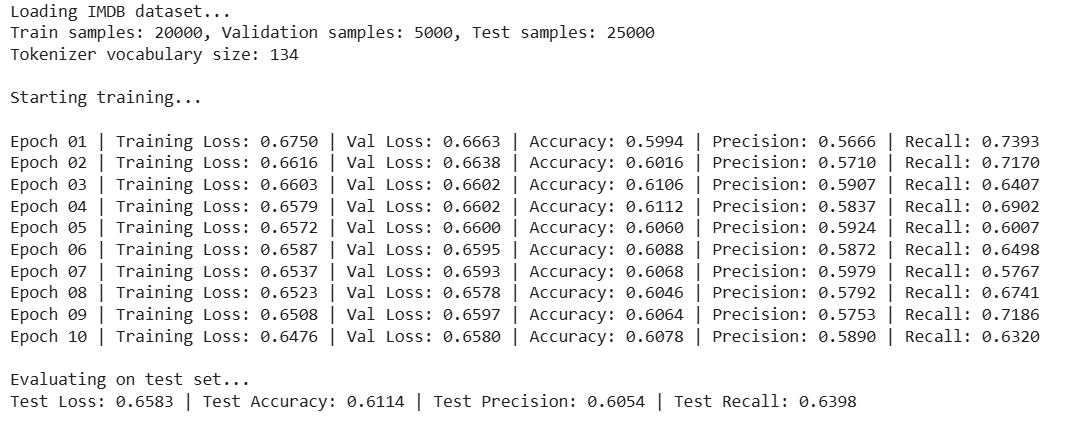
Learning Rate: Learning rate around 0.01 consistenly showed better results than smaller rates with the higher learning rated tending to achieve slightly better performance.  
  
This Is the Best I verified from the above all   
with seed 655 , activation function Leaky Relu , Optimizer Adam , batch Size =256 Learning Rate 0.01



These Model is verified with Seed Numbers

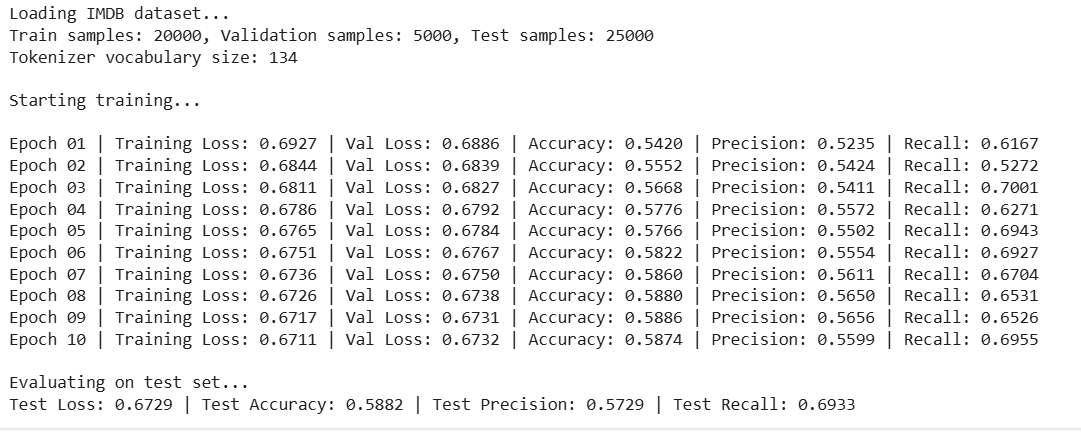
Test 2 with - Seed Number 1567



Test 3 With seed Number – 45  
  
  
Now These Model is compared with the Random model

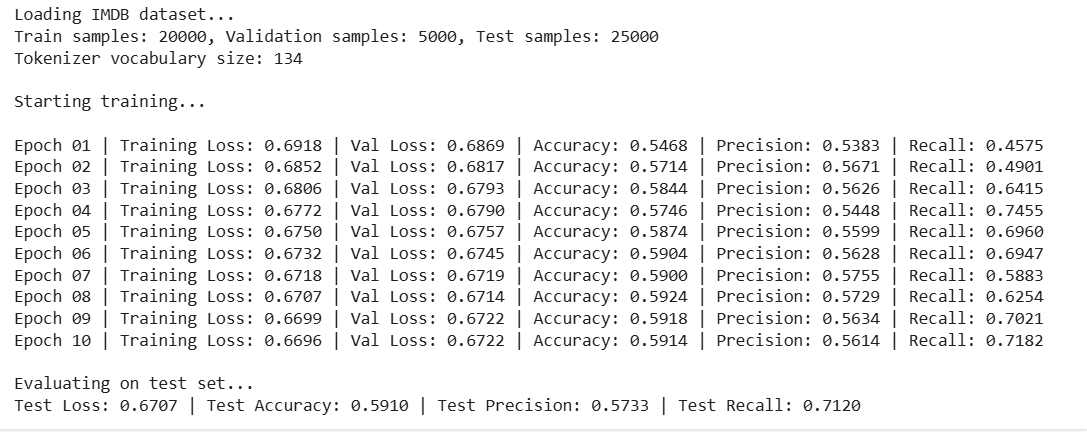
Random MLP on IMDB Dataset with seed number  of 655

Learning Rate of 0.001 and actiation function leakyrelu with an optimizer ADAM , batach size=256



Random MLP on IMDB Dataset with seed number  of 45

Learning Rate of 0.001 and actiation function leakyrelu with an optimizer ADAM , batach size=256



Conclusion:

Through the series of experiments I observed that certain configuration consistently yielded better performance. Specifically, LeakyRelu combined with the Adam optimizer provided the most stable and highest performace across various batch size and learning rated. Among the batch sizes with the learning rate consistently resulted in better accuracy and lower loss. The performance of the model was evaluated using the character level tokenization with the final model performance indicating that word level tokenization provided a more balanced.

The hyperparameter optimization strategy by adjusting activation functions optimizers and learning rate highlighted that Adam was the most effective optimizer, outerforming both RMSProp and SGD.   
In conclusion the experiments demonstrated the importance of selecting the right combination of hyperparameters as well as the choice between the tokenization in improving the model performance.