**Report Assignment 2**

**Student**

Maria Alejandra Zapata Montano

**Instructor**

Dr. Ankur Mali

**TA**

Theophilus Amaefuna

**Class**

Natural Language Processing

Contents

[**Introduction** 3](#_Toc190177491)

[ **Objectives** 3](#_Toc190177492)

[ **Overview of the work** 3](#_Toc190177493)

[**Methodology** 4](#_Toc190177494)

[**Experiments and Results** 6](#_Toc190177495)

[ Comparison between character-level and word-level tokenization. 7](#_Toc190177496)

[ Tables/graphs for hyper-parameter experiments. 7](#_Toc190177497)

[ Final model performance with mean accuracy and standard error. 9](#_Toc190177498)

[**Discussion** 11](#_Toc190177499)

[**Conclusion** 12](#_Toc190177500)

## **Introduction**

### **Objectives**

The main goal of this assignment is to analyze the impact of different tokenization methods on model performance. Specifically, the task involves modifying the given code by switching from character level tokenization to word level tokenization. By comparing both approaches, we want to evaluate differences in vocabulary size, sequence length, and overall model effectiveness. Additionally, this assignment focuses on hyperparameter optimization by experimenting with various settings, including learning rate, number of hidden layers, hidden sizes, batch sizes, optimizers, and activation functions. The objective is to identify the best combination of parameters that improve model performance. To ensure robust results, each experiment will be conducted multiple times with different random seeds, allowing for the computation of mean accuracy and standard error. Finally, all findings will be documented and analyzed to provide insights into the effectiveness of tokenization strategies and the best hyper parameters.

### **Overview of the work**

In this assignment, we will compare character level and word level tokenization while also optimizing the model’s performance. First, we will preprocess the dataset using both tokenization methods and analyze how they affect data representation. Then, we will work with adjustable hyper parameters and test different configurations to find the best setup. To ensure good results, we will run multiple trials and report key metrics like mean accuracy and standard error. we will also perform robustness checks by running the best model with different random seeds to see how consistent the performance is. Finally, we will document all the results, including comparisons, visualizations, and discussions. This will help us better understand how different tokenization methods impact model performance and why hyperparameter tuning is important in machine learning and NLP.

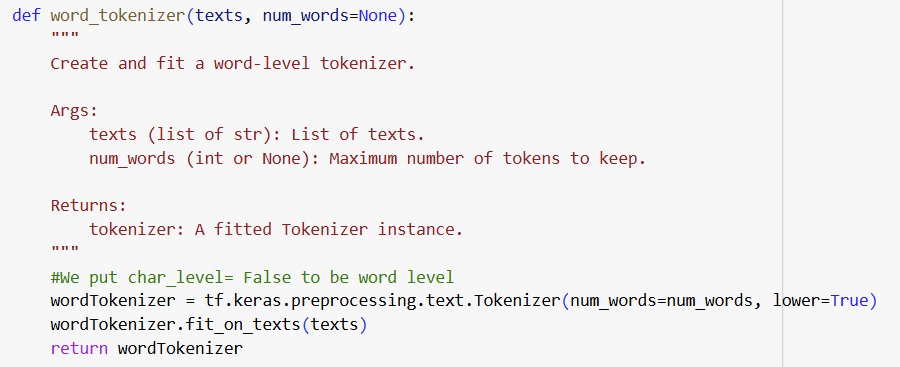
## **Methodology**

## Tokenization changes

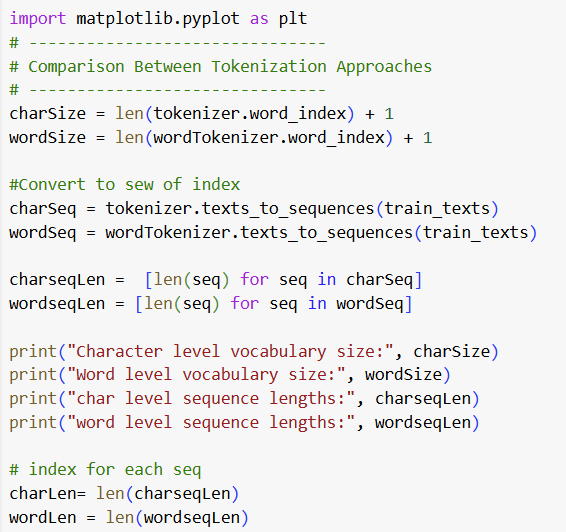
For the tokenization process, we had two main tasks: first, to replicate character level tokenization while saving and logging the processed data, and second, to modify the code to perform word level tokenization, also ensuring the data was saved and logged properly.

To compare both approaches, we looked at key metrics such as vocabulary size, sequence lengths, and a graph that visualizes how the data is distributed.

The main change was made in this function, specifically in the commented line. If the char\_level parameter is not explicitly set, it defaults to False, which means the tokenizer will operate at the word level by default.



The other functions, such as `texts\_to\_bow` and `one\_hot\_encode`, can remain the same because, in the end, what we need to pass as parameters to the models are the encoded vectors. After this, we calculated the vocabulary size and sequence length and printed a graph.



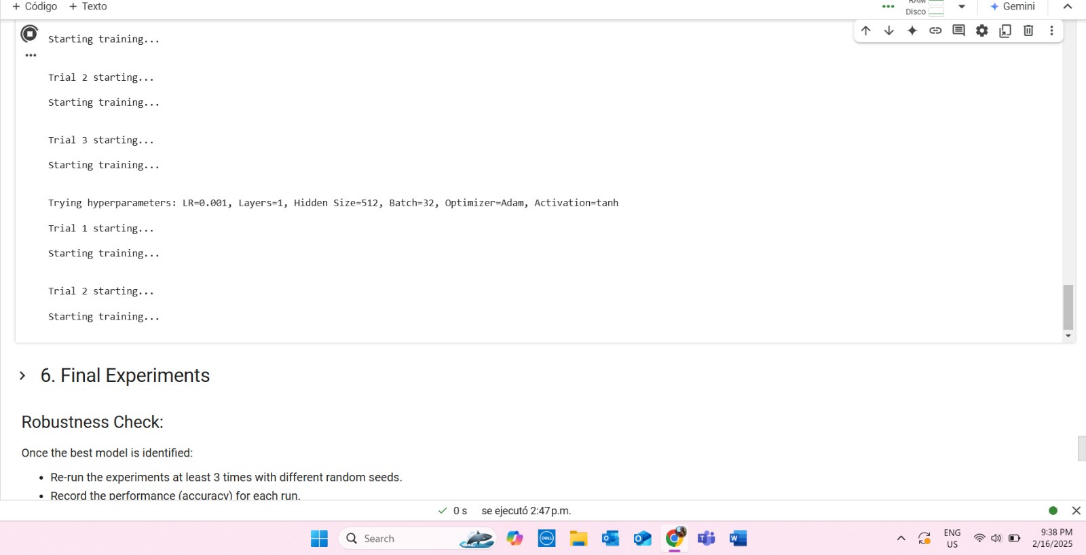
We are going to compare the results in the next section. It is important to note that the MLP model provided to us on GitHub works with both tokenization methods. The only change we needed to make was the input size parameter, which is based on a different X\_train.

## hyper-parameter optimization strategy

The initial strategy was to generate all possible combinations of hyperparameter configurations. However, this would result in 729 combinations, which is impractical to run. Instead, we decided to test a smaller subset of configurations by selecting them randomly. We used the random command to achieve this. We run the code few times, the first time was with a small amount of data and the second time was with a bigger subset. We are going to compare both results too.

You can see the code here: 

We chose to do it with this amount of data because, with just this amount, the total runtime of the program was literally more than 8 hours. And I ran it a few times.

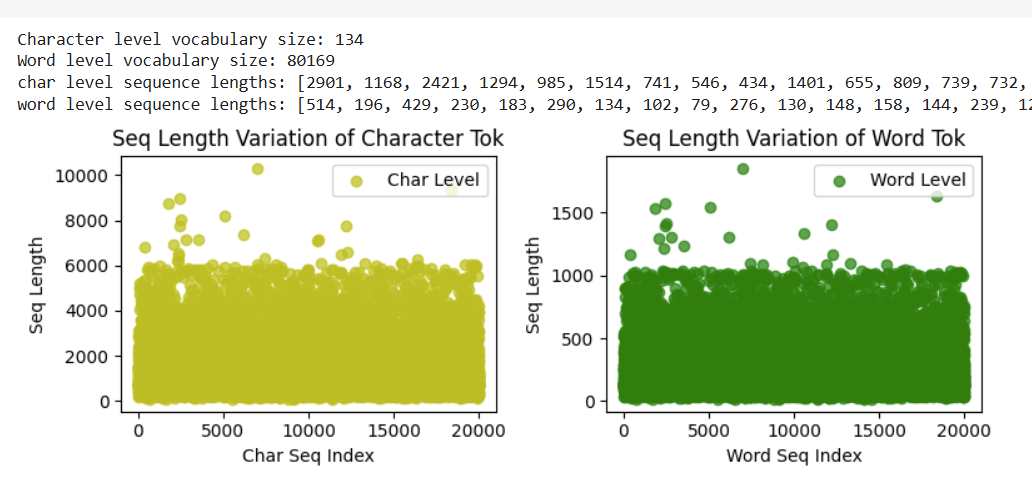


In this picture, you can see that the starting time was 2:47, and the program was still running at 9:30 PM.

**Experiments and Results**

### Comparison between character-level and word-level tokenization.

When running the experiments and getting the results we get this:



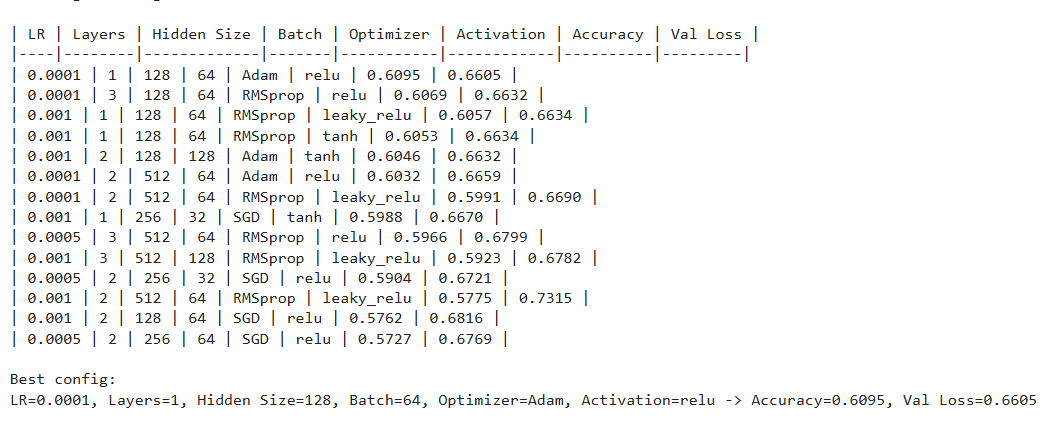
Analyzing the results, we can see that they are logical. The vocabulary size for word level tokenization is naturally larger than for character level tokenization. Additionally, the graph confirms that the number of characters in each text is greater than the number of words, as words are composed of multiple characters.

Another interesting thing to note is that both graphs have a similar overall structure or distribution, meaning the data follows a similar pattern regardless of the tokenization method.

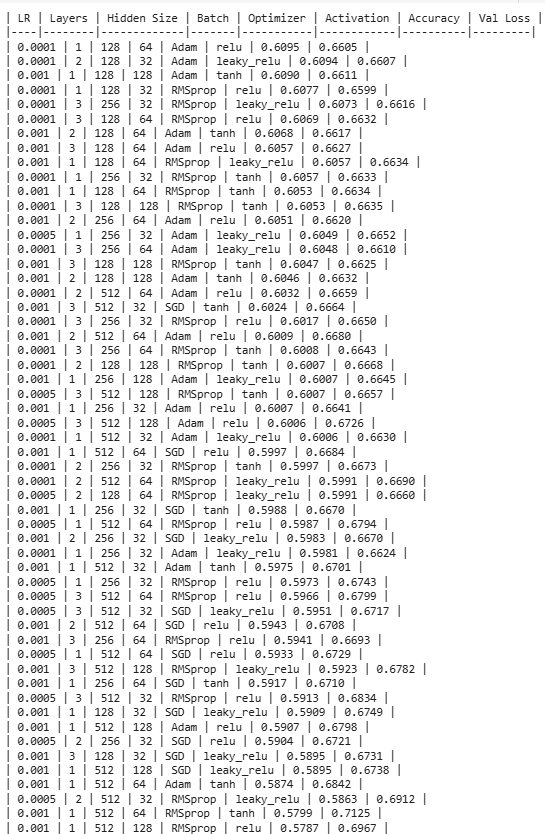
### Tables/graphs for hyper-parameter experiments.

Let’s show 2 experiments:

1 experiment:



2 experiment: (For the second experiment the table is larger, you can see the rest of the results in the Colab)



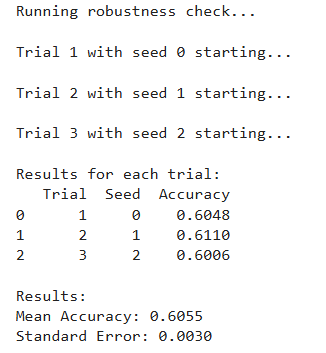


For the first experiment highlights, the best-performing model used a single hidden layer with 128 neurons, the Adam optimizer, and ReLU activation. The model achieved an accuracy of 0.6095 and a validation loss of 0.6605. But, also for the second experiment where we trained with more combinations of hyperparameters we got the same results. This is really interesting because the first though I had, as a learning student of NLP, was that with more hidden layers, we will get better results.

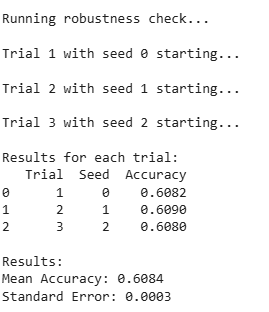
### Final model performance with mean accuracy and standard error.

Rerunning the best configurations with different seeds, we got:

For the first experiment:



For the second experiment:



So, let’s focus on the second experiment. The robustness check shows how the model performs with different random seeds. In each trial, the accuracy was a bit different:

* Trial 1 (seed 0) gave an accuracy of 0.6082.
* Trial 2 (seed 1) had 0.6090.
* Trial 3 (seed 2) resulted in 0.6080.

The mean accuracy across all three trials is 0.6084, which means the model is performing pretty consistently. The standard error is 0.0003, showing there isn’t much fluctuation in the results, so the model seems pretty stable across different runs.

Overall, the model’s performance doesn’t change a lot with different seeds, and it seems fairly reliable with a solid average accuracy around 0.6084.

We can observe that with more amount of data that we used in the training, we got better results in this check.

In addition, something that can be improved in future experiments is that instead of allowing the model to try all possible combinations of hyperparameters, we can first create a subset of parameters that make sense and are likely to yield better results. Then, we can test these combinations. This is a better approach that can save time and is more intelligent.

* **Comparing with Random MLP**

Now, let's do something else. Let's compare the results of our model with the results of the random model.

If we at the end evaluate in the test set for the standard MLP model: 

And these are the results of the Random MLP model:



The difference in results comes from the fact that the standard MLP is trained, while the Random MLP is not. During training, the standard MLP adjusts its weights using backpropagation, allowing it to learn important patterns from the data. On the other hand, the Random MLP keeps its weights at their initial random values, so it doesn’t actually learn anything. Because of this, its predictions are mostly random, leading to higher loss, lower accuracy, and worse recall. Then. this shows why training is so important for the models to work well.

## **Discussion**

The results from both the hyperparameter optimization and the robustness check indicate that the model is fairly consistent and stable. In the optimization, the best configuration achieved an accuracy of 0.6095 and a validation loss of 0.6605, with a single hidden layer, 128 neurons, Adam optimizer, and ReLU activation proving most effective.

The robustness check, conducted with three different random seeds, showed that accuracy values were consistent, with the mean accuracy being 0.6084 (61%) and a low standard error of 0.0003, suggesting that the model’s performance isn't heavily affected by the choice of random seed. These results suggest the model is stable and reliable. Also, it is interesting to note, that the more data we used to train, the better results we get.

The most challenging aspect of this experiment was the waiting time for training the models, which could take more than 8 hours to train the model and find the best configurations of hyperparameters. For future experiments, one way to improve is by narrowing down the hyperparameter search. Instead of letting the model try all possible combinations, we can first select a smaller set of parameters that make sense and are more likely to give better results. This way, we save time and make the process more efficient.

## **Conclusion**

In this assignment, the main objective was to analyze the impact of different tokenization methods: character level versus word level, on the performance of a machine learning model. We also focused on optimizing hyperparameters to find the best configuration that would improve the model’s effectiveness. The process involved switching between character and word level tokenization, examining their effects on vocabulary size and sequence length, and testing various hyperparameter settings such as learning rate, hidden layers, batch size, optimizers, and activation functions. The experiments were conducted multiple times with different random seeds to ensure robust results, calculating mean accuracy and standard error to evaluate consistency. The results showed that the best configuration for hyperparameters involved a single hidden layer, 128 neurons, Adam optimizer, and ReLU activation, with a mean accuracy of 0.6084.

In conclusion, this assignment provided valuable knowledge into how tokenization methods and hyperparameter tuning affect model performance, emphasizing the importance of the configurations to achieve optimal results.