**Assignment 2:**

**Designing and Comparing MLP Models with Word and Character Tokenization**

Maxim Ryabinov (U02204083)

CAP4641: Natural Language Processing

Instructor: Dr. Ankur Mali

University of South Florida

**Introduction**

Multilayer Perceptrons, or MLPs, are feedforward neural networks and a great tool in the area of machine learning. They have many applications in the field, one of which will be the focus of this report. The main objective of the assignment is to modify the text preprocessing from character-level tokenization to word-level tokenization and then create an MLP model that used the word-level preprocessed IMDB dataset to train itself. Additionally, hyper-parameter optimization will be performed to experiment and find a set of best settings for the model created, followed by further testing and a comparison between a random MLP model to further show robustness.

**Methodology**

To do so however, the first step in the process was to switch the current method of tokenization to word tokenization and then train an MLP model based off the preprocessed text to compare the two methods and determine which one results in a more accurate model. This was done by simply modifying the “char\_level” parameter for the tokenizer to False, so that it would use word level tokenization on the IMDB dataset instead. Once this was done, a bag of characters and words, as well as one-hot encodings using the dataset labels were generated (one pair for training, validation, and testing). It should also be mentioned that the word-tokenization was limited to the top 10,000 most common words in the dataset as the full vocabulary was over 80,000 words.

Now that the preprocessing of the text was done, two MLP models were instanced for each of the tokenization methods used. They were each then trained individually using the same set of hyper-parameters for consistency. After training each model over three iterations, it was found that the word model performed marginally better than the character model (see experiment and results section for more details). This was mainly due to the vastly increased amount of contextualization the word model had access to, given its large vocabulary.

So now that the best model was identified, in order to attempt to gain an even greater increase in performance, hyper-parameter optimization was implemented next. The following hyper-parameters were used: learning rate, hidden layers, hidden layer sizes, batch size, optimizer, activation function. Moving forward, the process of implementing hyper-parameter optimization was relatively straight-forward. First, a range of possible values were assigned to each parameter, such as a model being able to have one, two, or three hidden layers or using Adam, SGD, or RMSprop optimizer during backward propagation. Then, a sequence of 50 runs were carried out, each run choosing a unique subset combination of hyper-parameters randomly to ensure parameters changed with each iteration. During each iteration, the model would be trained at least three times, and then finally tested against the test dataset to obtain some performance metrics such as accuracy and loss. Lastly, once all 50 runs were complete, the set of hyper-parameters that resulted in the highest accuracy was chosen as the best overall (see experiment and results section). The reason why accuracy was chosen as the performance metric is because for well-balanced datasets such as the IMDB dataset being used, accuracy is a good metric for judging the performance of a model trained using it.

The last experiments conducted were to prove the robustness of the model using the best hyper-parameters found. This was done by running a comparison between the current MLP model against the random MLP provided, to see which one would have better accuracy overall and lower standard error. Each model was given the best hyper-parameters, and then ran through five unique runs, involving three iterations of training each. Once the accuracies were collected across all five tests for each model, the mean accuracy and standard error were calculated.

**Experiment and Results**

This section displays all the results collected throughout the experiments performed during the assignment. Additional data on performance metrics during training and such can be found in the Jupyter Notebook provided with the report. The first major experiement performed was the comparison between character and word tokenization when it came to preprocessing the IMDB dataset. The results below show that word tokenization did perform better as it has a greater level of accuracy while also having a loss closer to zero.

|  |  |  |
| --- | --- | --- |
| **Character Vs. Word Tokenization** | | |
|  | **Character Tokenization** | **Word Tokenization** |
| **Test Accuracy** | 0.6004 | 0.8670 |
| **Test Loss** | 0.6656 | 0.3572 |

The next set of results shown below are the 50 tests carried out of different sets of hyper-parameters. Each row represents an individual run, with the hyper-parameters that were used and the resulting accuracy obtained after using the trained MLP model against the test dataset.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Run** | **Learning Rate** | **Hidden Layers** | **Hidden Size** | **Batch Size** | **Optimizer** | **Activation Function** | **Test Accuracy** |
| **1** | 0.0005 | 3 | 256 | 32 | Adam | ReLU | **0.8595** |
| **2** | 0.0001 | 1 | 128 | 128 | SGD | ReLU | **0.5290** |
| **3** | 0.001 | 3 | 512 | 64 | Adam | Leaky ReLU | **0.8495** |
| **4** | 0.0001 | 1 | 512 | 128 | Adam | Tanh | **0.8640** |
| **5** | 0.0001 | 1 | 512 | 64 | SGD | Tanh | **0.5216** |
| **6** | 0.0005 | 1 | 512 | 32 | SGD | Leaky ReLU | **0.6331** |
| **7** | 0.0001 | 1 | 512 | 32 | Adam | Tanh | **0.8756** |
| **8** | 0.001 | 3 | 256 | 128 | SGD | ReLU | **0.5806** |
| **9** | 0.001 | 1 | 512 | 64 | Adam | ReLU | **0.8830** |
| **10** | 0.0001 | 1 | 128 | 32 | SGD | ReLU | **0.5116** |
| **11** | 0.0001 | 1 | 512 | 64 | SGD | ReLU | **0.5031** |
| **12** | 0.0005 | 2 | 512 | 32 | Adam | Tanh | **0.8670** |
| **13** | 0.0010 | 2 | 128 | 64 | Adam | Leaky ReLU | **0.8826** |
| **14** | 0.0005 | 2 | 256 | 64 | RMSprop | Tanh | **0.8750** |
| **15** | 0.0005 | 1 | 256 | 128 | Adam | Tanh | **0.8852** |
| **16** | 0.0001 | 3 | 256 | 64 | SGD | ReLU | **0.5190** |
| **17** | 0.001 | 2 | 256 | 32 | RMSprop | ReLU | **0.8577** |
| **18** | 0.0005 | 2 | 512 | 64 | RMSprop | ReLU | **0.8596** |
| **19** | 0.0005 | 1 | 512 | 128 | Adam | Leaky ReLU | **0.8809** |
| **20** | 0.0005 | 1 | 512 | 64 | SGD | Leaky ReLU | **0.5610** |
| **21** | 0.001 | 1 | 256 | 32 | Adam | Tanh | **0.8856** |
| **22** | 0.001 | 1 | 512 | 32 | RMSprop | Tanh | **0.8674** |
| **23** | 0.0005 | 2 | 128 | 128 | SGD | ReLU | **0.5446** |
| **24** | 0.0001 | 2 | 128 | 32 | SGD | ReLU | **0.5612** |
| **25** | 0.0001 | 2 | 512 | 32 | SGD | ReLU | **0.5736** |
| **26** | 0.0001 | 3 | 256 | 32 | RMSprop | ReLU | **0.8546** |
| **27** | 0.0005 | 1 | 256 | 128 | Adam | Tanh | **0.8833** |
| **28** | 0.0001 | 2 | 256 | 32 | Adam | ReLU | **0.8547** |
| **29** | 0.0001 | 1 | 128 | 128 | Adam | Leaky ReLU | **0.8262** |
| **30** | 0.0005 | 1 | 128 | 128 | RMSprop | ReLU | **0.8722** |
| **31** | 0.0001 | 1 | 128 | 32 | Adam | Tanh | **0.8645** |
| **32** | 0.0005 | 3 | 256 | 32 | RMSprop | Tanh | **0.8734** |
| **33** | 0.0005 | 3 | 512 | 64 | SGD | Tanh | **0.5909** |
| **34** | 0.0001 | 3 | 128 | 128 | Adam | Tanh | **0.8377** |
| **35** | 0.0005 | 2 | 256 | 128 | SGD | Leaky ReLU | **0.5394** |
| **36** | 0.0010 | 3 | 256 | 64 | Adam | Tanh | **0.8787** |
| **37** | 0.0001 | 3 | 256 | 32 | SGD | Tanh | **0.5255** |
| **38** | 0.001 | 2 | 128 | 64 | SGD | ReLU | **0.5392** |
| **39** | 0.001 | 2 | 256 | 64 | SGD | Leaky ReLU | **0.6144** |
| **40** | 0.0005 | 1 | 256 | 128 | SGD | ReLU | **0.5099** |
| **41** | 0.0005 | 3 | 256 | 128 | RMSprop | Tanh | **0.8717** |
| **42** | 0.001 | 1 | 512 | 128 | SGD | Leaky ReLU | **0.5606** |
| **43** | 0.0005 | 3 | 128 | 128 | SGD | Leaky ReLU | **0.4878** |
| **44** | 0.001 | 1 | 256 | 64 | Adam | ReLU | **0.8780** |
| **45** | 0.0001 | 2 | 256 | 64 | SGD | Leaky ReLU | **0.5106** |
| **46** | 0.0001 | 3 | 256 | 128 | SGD | Leaky ReLU | **0.5192** |
| **47** | 0.001 | 3 | 512 | 64 | Adam | Leaky ReLU | **0.8603** |
| **48** | 0.0001 | 3 | 128 | 64 | RMSprop | Tanh | **0.8642** |
| **49** | 0.0005 | 2 | 256 | 32 | RMSprop | Leaky ReLU | **0.8770** |
| **50** | 0.0001 | 3 | 128 | 32 | SGD | Tanh | **0.4906** |

Based on the runs carried out above, the set of hyper-parameters shown resulted in the best performance. As mentioned previously, this is determined by accuracy, since that is a good metric to use given that the IMDB dataset is being used.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Best Model Found** | | | | | | | |
| **Run** | **Learning Rate** | **Hidden Layers** | **Hidden Size** | **Batch Size** | **Optimizer** | **Activation Function** | **Test Accuracy** |
| **21** | 0.001 | 1 | 256 | 32 | Adam | Tanh | **0.8856** |

After identifying the best set of hyper-parameters, a test for additional robustness of the model was performed. The current MLP model was put through more iterations of training and testing. Then, a random MLP model with the same hyper-parameters was used and put through the same iterations of training. The following table below is a summary of the overall average accuracy and standard error for the two models based on the best hyper-parameters found:

|  |  |  |
| --- | --- | --- |
| **MLP Model Vs. Random MLP Model Comparison** | | |
|  | **Mean Test Accuracy** | **Standard Error** |
| **MLP Model** | 0.8849 | 0.0008 |
| **Random MLP Model** | 0.7201 | 0.0008 |

Once the final robustness experiment was done, this concluded the experimental portion of the assignment as results could now be drawn.

**Discussion**

Based on the results collected, word tokenization performs marginally better than character tokenization. The most obvious reason as to why the accuracy is higher when the model uses word-based tokenized text is because there are more unique combinations of patterns. It can be very difficult to draw patterns from comparing individual characters. On the other hand, parsing text by words uncovers more context and semantic meaning, allowing for more complex patterns to be recognized.

Moving forward, looking at the hyper-parameter optimization experiments, it seems the best gradient optimization algorithm was Adam, as it tended to result in better performance when compared to SGD and RMSprop. This makes sense as Adam is usually the preferred type when training neural networks. Additionally, both ReLU and Tanh activation functions provide the highest consistent accuracy. Tanh also tends to converge fairly quickly compared to other activation functions, while also overall usually leading to better accuracy results. Now as for the hidden layers, having less layers but a higher number of neurons per layer tended to provide good results. The best parameters found ended up actually being a single hidden layer with 256 neurons. Having additional layers of this size though might lead to overfitting though, so there is a balance there. All in all, the best hyper-parameters found from the optimization experiment make sense.

When comparing the random MLP model to the standard MLP model, it was expected that the standard MLP would perform better. Even when using random seeds, the MLP model was able to consistenly perform with around 0.8850 accuracy. However it was interesting to observe how the random MLP was able to perform quite well too, given that only the output layer is being updated during backpropagation to learn. The reason I believe this might be the case is because of one of the main weaknesses MLP has. Rather than understanding the meaning of text, MLP identifies and understands patterns, which is then used for classification. And so, seeing that the random MLP model did somewhat well, it could be interesting to carry out further hyper-parameter optimization to see how performance improves.

The biggest challenge encountered during this assignment was the hyper-optimization process. Specifically, the great amount of time it took to train the MLP model multiple times, across a multitude of parameters. The whole process ended up taking a few hours. Another major challenge that posed was understanding all the information that was being calculated and logged during training, as well as the overall flow of data within an MLP. Lots of time was spent learning specific metrics meant, or how different hyper-parameters affected performance of the model. For instance, I was conflicted in whether I should use the loss or accuracy to determine overall performance. After doing research online, I found that for balanced datasets like the IMDB dataset I was using, accuracy tends to be a good indicator for performance. On the other hand, using loss to determine performance is better for imbalanced datasets, and can give a more nuanced perspective.

Overall, this assignment gave significant insight into how MLPs are constructed and operate. It allowed me to experiment with performance and make comparisons between different types of MLP models, as well as come to conclusions about what ceratin results meant.

**Conclusion**

All in all, this assignment dealt with comparing word-level and character-level tokenization and comparing how different combinations of hyper-parameters affected the performance of MLP models, which were trained on the IMDB dataset. Results showed that models trained with Adam optimizer and used Tanh for their activation function tended to lead to better results in performance, specifically with regards to accuracy. The final best model was able to achieve an accuracy of 0.8856. Finally, the robustness of the model and hyper-parameters chosen were confirmed through a comparison between the standard model and the random model provided, resulting in the standard model performing over 15% better than the random model.