UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: ΙΩΑΝΝΑ ΠΟΥΛΟΥ
sdi: <sdi2100161>

Course: *Artificial Intelligence II (M138, M226, M262, M325)*
Semester: *Fall Semester 2023*

Contents

1. Abstract

<Briefly describe what's the task and how you will tackle it.>

The goal of the task is to create a sentiment classifier using deep neural networks for a twitter dataset about the Greek general elections. In the first assignment, we approach the same subject using logistic regression and now we will examine whether deep neural networks will lead to better and more efficient results.

2. Data processing and analysis

2.1. Pre-processing

<In this step, you should describe and comment on the methods that you used for data cleaning and pre-processing. In ML and AI applications, this is the initial and really important step.

For example some data cleaning techniques are: Dropping small sentences; Remove links; Remove list symbols and other uni-codes.>

The first step i did for this task was to clean the data in order to increase the quality of them.Specifically,I removed from my data stop words,urls,emojis,spaces,mentions, panctuation and other special characters.Moreover, I dropped small and very large words and i removed the verbs and the accents of the words in order not to let them affect my data.However,after experimentation i realized that preprocessing doesn't provide substantial improvements in the results and this is probably due to the better vectorization we use now in contrast to the first assignment. For example, now the words "ζωγραφιά" and "ζωγραφίζω" are not two completely different words and are close to each other in the vector space so lemmatization methods might not be as important since the vectorization techniques decrease the distance between different forms of a word.Although, data cleanning is always a good technique in natural processing language so i didnt't remove it from my project.

2.2. Analysis

<In this step, you should also try to visualize the data and their statistics (e.g., word clouds, tokens frequency, etc). So the processing should be done in parallel with an analysis. >

It is worth noting that before and after Pre-processing,I visualized the data using word clouds and tokens frequency diagrams.In this way ,i could understand how efficient the data clean was and it helped me to make Pre-processing better.For example,thanks to word clouds,i get rid of the panctuation mark ':',which for some reason was the most common word.The word clouds before and after the data cleaning are listed below, while you can create tokens frequency diagrams by running the related code in my notebook.

Figure 1: Word Cloud before Preprocessing

Figure 2: Word Cloud after Preprocessing

2.3. Data partitioning for train, test and validation

<Describe how you partitioned the dataset and why you selected these ratios>

I let train and validation datasets as they was given to us.The ratio 80:20 is the optimal since we want variety of training data,but we also don't want our model overfits them.It is worth noting that,among others, I drop from Xtrain the column "Party", because the cost of using the Parties was greater than their contribution to model's performance.

2.4. Vectorization

<Explain the technique used for vectorization>
For converting text data into vectors i use two different approaches of the Word2Vec model.The first one works as follow: For all the tweets, each word is converted to a vector and then the mean of the word embeddings is calculated in order to represent the tweet.On the other hand, the second method attempts to simulate an approach similar to CBOW. For each word in the tweet, it considers a context window of words around it and calculates the mean of the word vectors of these surrounding words, excluding the target word and this resulting vector is the vector of the target word.Then as in the first approach, the mean of the word embeddings is the vector that represents the tweet.As we expected both methods create similar results.That's because,despite the CBOW approach tends to generate better embeddings for certain words by taking into consideration the context window of them, the calculation of the mean of words eliminates this difference.

3. Algorithms and Experiments

3.1. Experiments

<Describe how you faced this problem. For example, you can start by describing a first brute-force run and afterwords showcase techniques that you experimented with. **Caution:** we want/need to see your experiments here either they increased or decreased scores. At the same time you should comment and try to explain why an experiment failed or succeeded. You can also provide plots (e.g., ROC curves, Learning-curves, Confusion matrices, etc) showing the results of your experiment. Some techniques you can try for experiments are cross-validation, data regularization, dimension reduction, batch/partition size configuration, data pre-processing from 2.1, gradient descent>

In this section,i will represent you the results of some techniques i used (some methods were implemented in order to help me understand better how the network works and others to make the model better and increase the scores).It is worth noting that for speed reasons many of the experiments are not included in the notebook.

3.1.1. Dimensionality Reduction. To begin with, after I represented the tweets as vectors, I used the t-SNE method, which is a technique used to visualize high-dimensional data in a lower-dimensional space. T-SNE was really useful because it helped me comprehend my data better and distinguish the relationships between the tweets. Here are the first 50 training sentences of my dataset, visualized using the t-SNE technique, because visualizing the entire dataset would be meaningless and time-consuming.

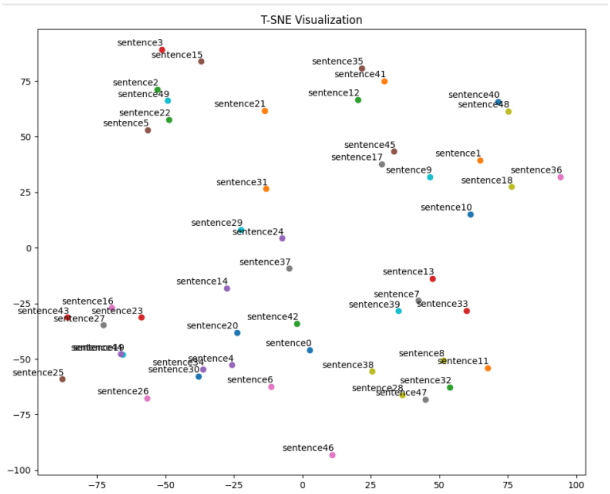

Figure 3: T-SNE Visualization

3.1.2. FIRST BRUTE-FORCE RUN. In this scenario, I created a model featuring a singular hidden layer comprising 64 nodes. For optimization, I opted for the Mean Squared Error loss function, coupled with the Stochastic Gradient Descent optimizer.

This network is the starting point of my attempt to find the best model. Upon scrutinizing the results, I noticed that occurs underfitting ,because f1 training and validation scores are are steady low around 0.33 and that SGD Optimizer requires a more extended duration to converge towards the minimum. For this reason,further investigation is necessary in order to enhance the efficiency and improve the overall F1 scores.

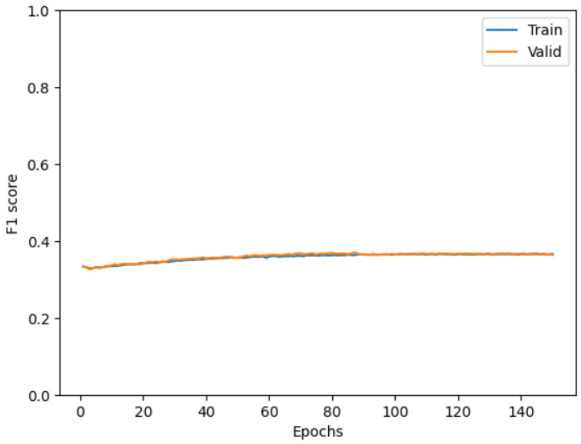

Figure 4: Brute Force: F1 Learning Curves

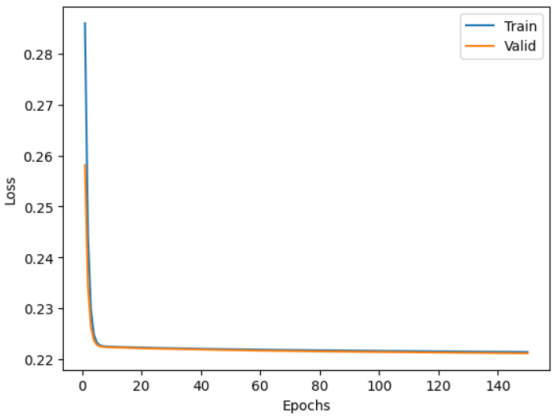

Figure 5: Brute Force: Loss Learning Curves

3.1.3. A SECOND ATTEMPT. After the initial brute-force approach, I examined the interactions between various activation functions, optimizers, and loss functions to understand the difference in their final performance.Obviously, I present you below some of the experiments I conducted and not all them.

- **Sigmoid-BCEWithLogitsLoss-AdamW**: This combination creates unsatisfactory results and the F1 validation and training scores are about 0.33. That's probably because the BCEWithLogitsLoss loss function and the Sigmoid activation are designed for binary classification scenarios, not multiclass. Note: Sigmoid function was applied only in the last layer.

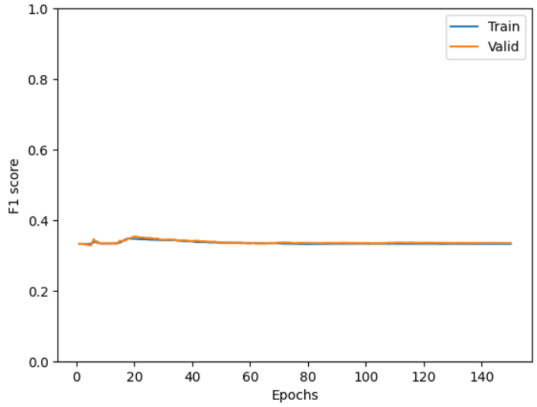

Figure 6: F1 Learning Curves for batch size 2048

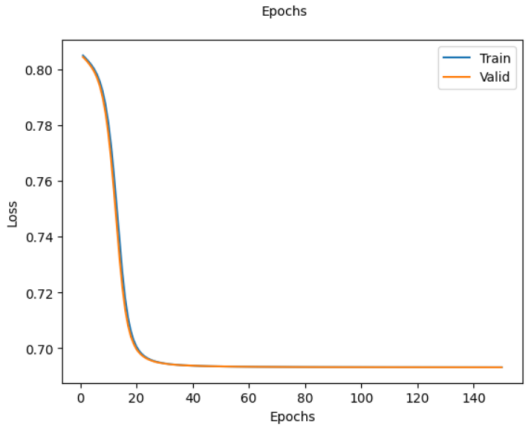

Figure 7: Loss Learning Curves for batch size 2048

- **SELU-MSE-RMSprop-Softmax** : The particular combination is better than the previous one, because now F1 training and validation scores are continuously maintained at 0.39. However ,as we can notice, the training loss is quite a bit more than the validation loss, which means that our model overfits.

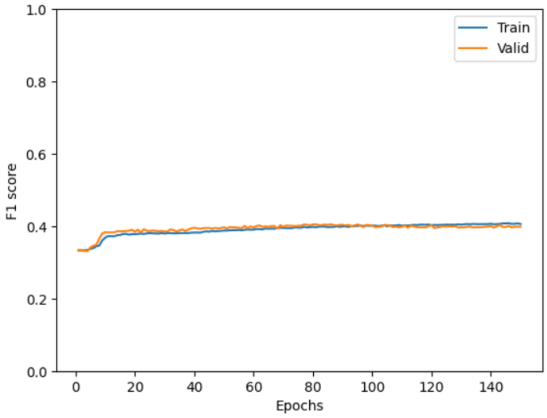

Figure 8: F1 Learning Curves for batch size 2048

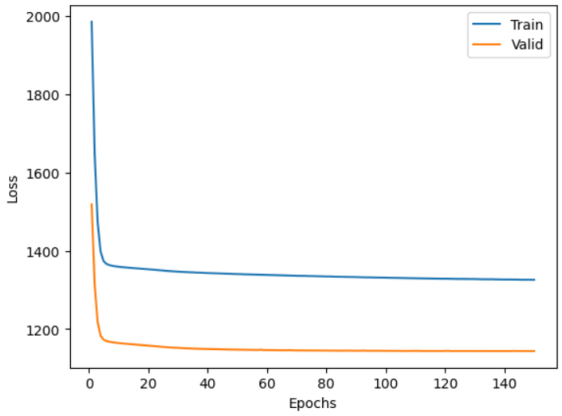

Figure 9: Loss Learning Curves for batch size 2048

- **ReLU-CrossEntropy-Adam**: This is the best combo so far, because of the smooth convergence of both training and validation F1 scores, which reach a ceiling around 0.40. Moreover, the loss is decreasing, reaching a 1.08 floor for validation and 1.074 floor for training. The result is quite expected, because the CrossEntropy() is one of the most appropriate loss functions for a multiclass classification tasks with one-hot encoded targets. Henceforth, I am going to apply various methods to this model in order to improve the results.

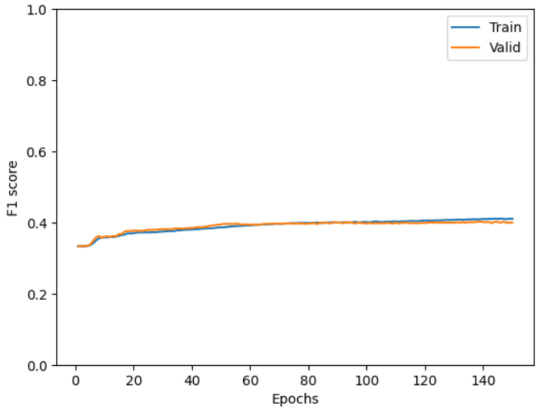

Figure 10: F1 Learning Curves for batch size 2048

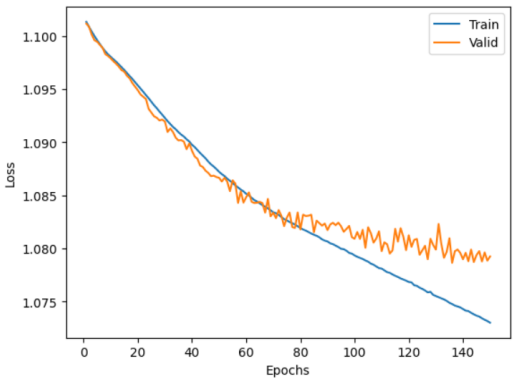

Figure 11: Loss Learning Curves for batch size 2048

3.1.4. BATCH SIZE CONFIGURATION. In the context of machine learning, the term "batch size" refers to the number of training examples into which the dataset is divided in one iteration.During experimentation, I tried the following three batch size options:

- **Stochastic mode**: This is the case where batch size is equal to one and therefore the neural network parameters are updated after each sample.Despite the fact that stochastic mode is advantageous when we want to escape local minima, this approach comes with certain disadvantages.More specifically, due to frequent updates, the training is very slow and the result can be noisy,because it is affected by the randomness factor(creates overfit as we can see below).

- **Batch mode**: Batch mode refers to the case where batch size is equal to the total dataset. The advantage of this method is that it leads to faster training times as the updates to the model parameters are less frequent. Although, this approach requires more memory and may converge to a suboptimal minimum.

- **Mini-batch mode**: Mini-batch mode means that batch size is greater than one and less than the size of the dataset.In this way,this method combines some of the advantages of both methods like process parallelism and memory efficiency (we do not have to store the whole dataset in memory).Therefore, due to the balanced benefits this approach provides, my model employs the mini-batch gradient descent method for this assigment.

Below are the F1 scores learning curves of the three methods using the same hyperparameters,loss functions and optimizers, in order to notice the effect of the different batch sizes.
Note 1: The number of epochs we choose for this experiment is 30, because stochastic mode acquires a lot of time to run.
Note 2: For the batch mode case I set the learning rate to 1e-3 (instead of 1e-4), in order to converge faster.

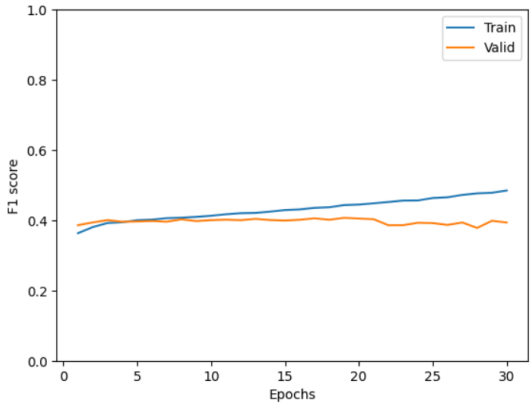

Figure 12: Stochastic mode: F1 Learning Curves for batch size 1

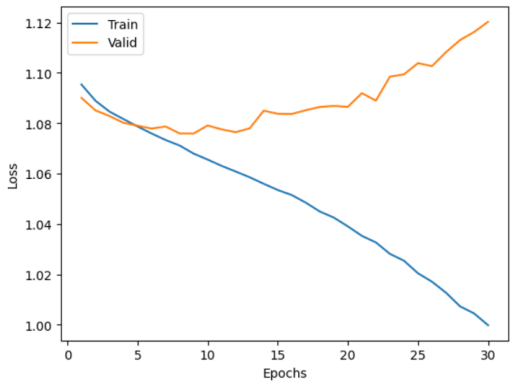

Figure 13: Stochastic mode: Loss Learning Curves for batch size 1

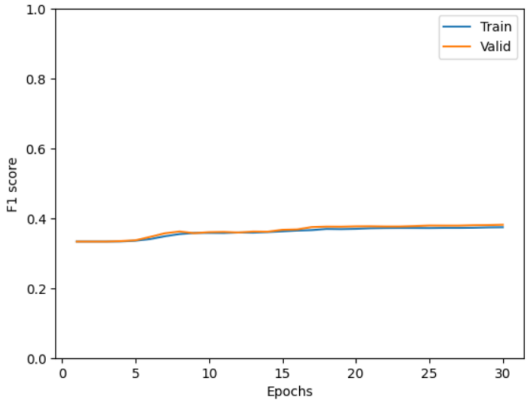

Figure 14: Mini-batch mode: F1 Learning Curves for batch size 2048

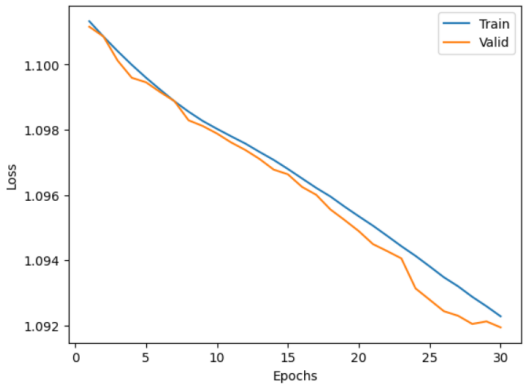

Figure 15: Mini-batch mode: Loss Learning Curves for batch size 2048

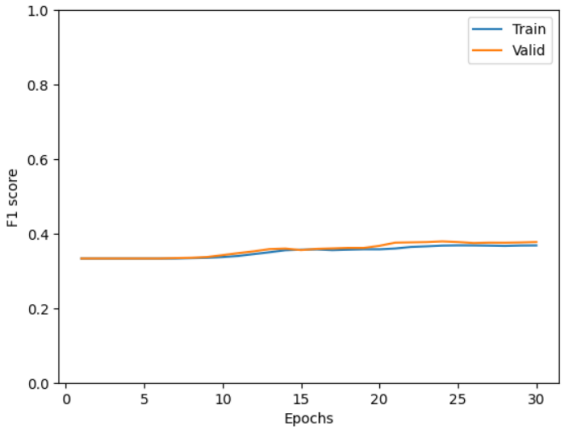

Figure 16: Batch mode: F1 Learning Curves for batch size equal to the dataset

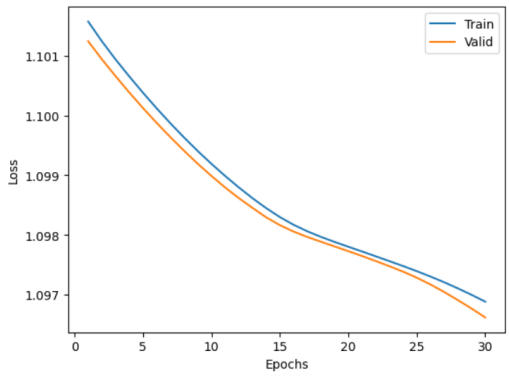

Figure 17: Batch mode: Loss Learning Curves for batch size equal to the dataset

3.1.5. K FOLD CROSS VALIDATION. The idea of k fold cross validation is simple. The data-set is divided into K subsets and the model is trained and evaluated K times, each time using a different fold as the validation set and the remaining folds as the training set. This method is particularly useful when we have a small dataset, which does not happen in our case. Considering our scenario, we use k fold cross validation

in order to check if we have any anomalies in the dataset (because now we use all the dataset as training) and as we can notice from the F1 learining curves below this doesn't occur.

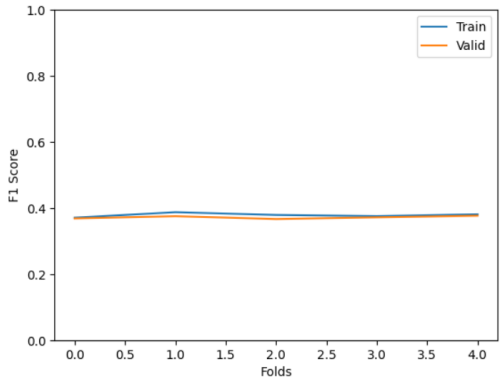

Figure 18: F1 Learning Curves

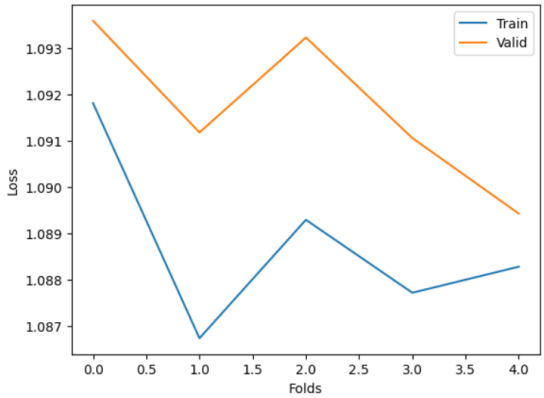

Figure 19: Loss Learning Curves

3.1.6. DATA REGULARIZATION. Regularization is a technique used in machine learning in order to reduce overfitting. For the assigment, I emplemented two regularization methods: **Dropout** and **L2**.
Dropout regularization works as follow: Before each iteration a random number of nodes is chosen to be dropped out, because this way prevents the model from depending on specific neurons. It is worth mentioning that the probability of a neuron to be dropped out depends on the dropout rate, which is determined by the programmer. As for the effects in my model,as i expected, the dropout rate didn't change the final results, because overfitting doesn't occur.(the gap that we notice after 100 epochs is very small to be considered as overfit). For practice reasons I tried on various dropout rates to hidden layers. Here are the results obtained by applying a dropout rate of 0.3 to all the hidden layers.

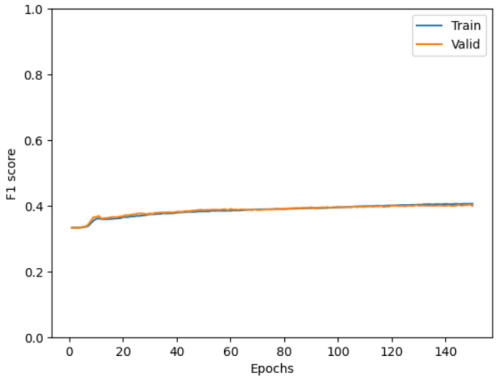

Figure 20: F1 Learning Curves for batch size 2048

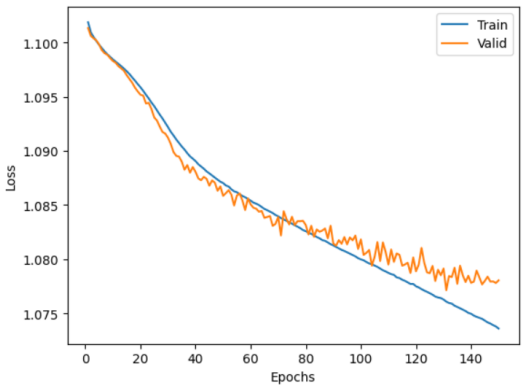

Figure 21: Loss Learning Curves for batch size 2048

L2 adds the squared values of the coefficients as a penalty to the cost function , in order to prevent the creation of large weights and in this way the network is less complex. However,as we explained before, overfitting doesn't occur, so the results remain the same as before. In my final model i apply a 0.3 dropout rate, beacause it is always a good idea to utilize dropout regularization.

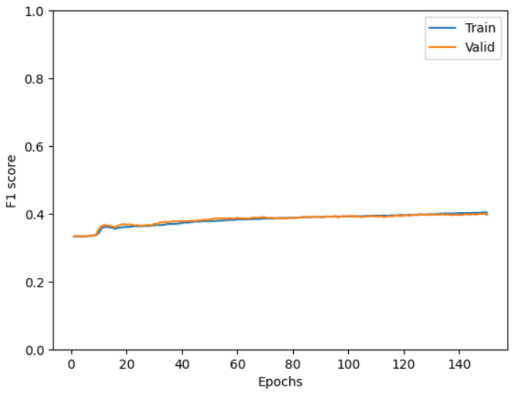

Figure 22: F1 Learning Curves for batch size 2048 and weight decay 1e-4

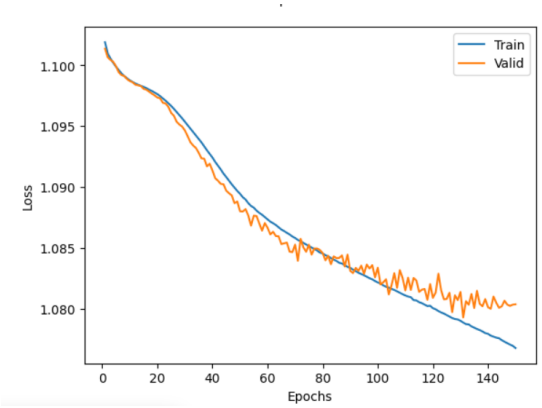

Figure 23: Loss Learning Curves for batch size 2048 and weight decay 1e-4

3.2. Hyper-parameter tuning

<Describe the results and how you configured the model. What happens with under-over-fitting??> Hyperparameters are configurations that influence the behavior of the algorithm, the performance of the model and the size of over and underfitting.In other words, different hyperparameter values can lead to completely different classifiers.In my model, the principal hyperparameters I tried to tune were the learning and dropout rate, the weight decay parameter and the number of batches, layers and nodes. As for the learning rate I experimented the values 1e-3 , 1e-4 and 1e-5 and I ended up that the best value is 1e-4, because 1e-3 created overfitting and 1e-5 required a long time for the model to reach a good solution. It is worth mentioning, that I used a learning rate scheduler in order to improve the result, but his contribution was not particularly useful so I removed it from my final model. In regard to the dropout rate and weight

decay parameter, as I mentioned in the data regularization part, I tried out several values, and I used optuna framework too, but none of the values greatly improved the outcome. Moreover, for the batch size I used the mini-batch mode and i simply run the optuna framework discovering that the best value for batch size is 2048. Last but not least, I concluded that a network with many layers and neurons doesn't improve the overall performance and that the best combination is the one with 4 layers and 64,32,16,8 nodes in each layer respectively.

3.3. Optimization techniques

<Describe the optimization techniques you tried. Like optimization frameworks you used.>
For the exercise implementation i used the Optuna framework.Optuna is a optimization framework used in machine learning, aimimg to discover the optimal hyperparameters in order to improve the effictiveness of a model. Personally, I used this framework to discover the optimal hyperparameters for batch size, dropout and weight decay and in this way i avoid all the experiments i should have try in order to determine these values.
Note 1:If you want to run the Optuna code in my Notebook change the variable *use_optuna* to True.
Note 2: After I used the framework, I chose the trial that produced the best plots and i did not take into consideration only the best trial optuna returned to me.

3.4. Evaluation

<How will you evaluate the predictions? Detail and explain the scores used (what's fscore?). Provide the results in a matrix/plots> <Provide and comment diagrams and curves>

For evaluating my model, I used F1 learning curves, loss learning curves, ROC curves and Confusion matrices.

3.4.1. ROC curve. ROC curve is another evaluating measure,which is created by plotting the true positive rate opposed to the false positive rate at various thresholds,while AUC is the area under the ROC curve.In our final model AUC is around 0.56-0.59, which means that the ability of our model to predict is a bit better than random guessing, but still remains low.

3.4.2. F-Score. F-score is a classification algorithm's performance metric, that combines precision and recall into a single measure and it ranges from 0 to 1, where a higher value indicates better performance.After the above experimentation, i observed that in the worst models i tested the F1 score ranges around 0.35, while in my final model it starts from 0.33 and reaches about 0.40, which indicates a moderate performance.

3.4.3. Loss. A loss learning curve is a graphical representation of the performance of a machine learning model over epochs and depicts the difference between the predicted

values and the actual values.In my final model, i use the CrossEntropy loss function and both losses are decreased and reach approximately 1.074 for training and 1.081 for validation and this reveals that my model has a similar performance to seen and unseen data.

3.4.4. Confusion matrix. A confusion matrix represents the prediction's results in matrix form and it shows how many prediction are correct and incorrect on a validation or a test set.Taking into consideration the confusion matrice below, we notice that the model can predict correct 54 percent of negative sentiments and 46 percent of positive, but struggles to identify the neutral elements, since it has an accuracy of 19 percent.In other words, what we conclude from these results is that all three results and especially the neutral ones are below the desired limit.

Note: In my paper, you can find the ROC Curves and Confusion matrices plots only for my final model, because incorporating the diagrams of all the experiments into my paper would create a sense of complexity making it tedious to read.

4. Results and Overall Analysis

4.1. Results Analysis

<Comment your results so far. Is this a good/bad performance? What was expected? Could you do more experiments? And if yes what would you try?> <Provide and comment diagrams and curves>

In conclusion, the best accuracy the sentiment classifier using deep neural networks can reach is about 0.4.This is because, a significant number of Sentiments in the dataset is wrong.However, the goal of the Project was to experiment with various techniques and hyperparameters.Obviously, there are many techniques and experiments, I don't present in this paper or in the Notebook.For example, I could use more vectorization techniques like Skip-gram or i could implement the GridSearchCV function for hyperparameter tuning.

4.1.1. Best trial. Given all the experiments, I end up that the best trial was the one that combines the following: CrossEntropy loss function, ReLU activation function, Adam optimizer, 1e-4 learning rate, 2048 batch size, 4 hidden layers, 64,32,16,8 nodes in each layer respectively and 0.2 dropout(it is not necessary), because it creates the best results at the best time.

<Showcase best trial>

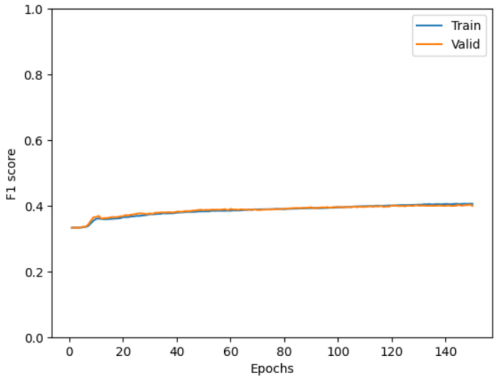

Figure 24: Best trial: F1 Learning Curves

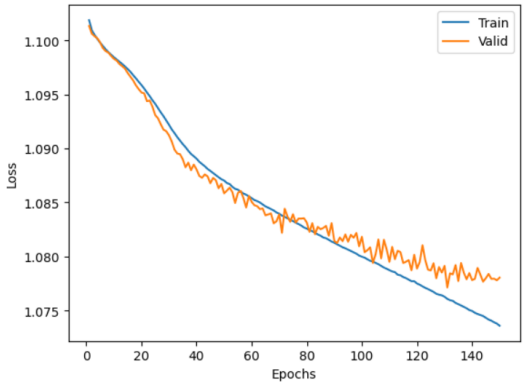

Figure 25: Best trial: Loss Learning Curves

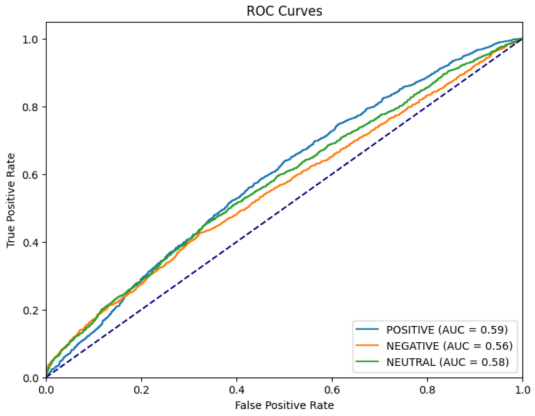

Figure 26: Best trial: ROC Curves

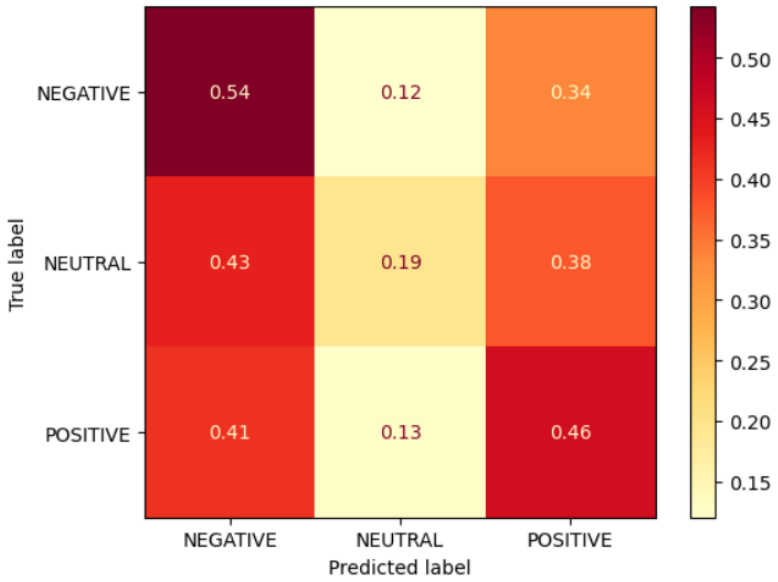

Figure 27: Best trial: Confusion Matrices

4.2. Comparison with the first project

<Use only for projects 2,3,4>
<Comment the results. Why the results are better/worse/the same?> This approach is quite better than that of the first assignment. First of all, the vectorization now is much more targeted and the words aren't be represented by random numbers. Additionally, deep neural networks are more capable of capturing complex, non-linear relationships in data than logistic regression classifiers.The aforementioned excellence can also be seen in the results. In the previous project the model corresponded well to seen data, but it had low performance to unseen ones, creating overfitting ,in contrast to this assigment where the model reacts to both in the same way.

4.3. Comparison with the second project

<Use only for projects 3,4>
<Comment the results. Why the results are better/worse/the same?>

4.4. Comparison with the third project

<Use only for project 4>
<Comment the results. Why the results are better/worse/the same?>

5. Bibliography

References

[1] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

<You should link and cite whatever you used or "got inspired" from the web. Use the / cite command and add the paper/website/tutorial in refs.bib>
<Example of citing a source is like this:> [1] <More about bibtex>

REFERENCES

- Sklearn Documentation: <https://scikit-learn.org/stable/>
- Emoji Removal: <https://gist.github.com/slowkow/7a7f61f495e3dbb7e3d767f97bd7304b>
- PyTorch Documentation: <https://pytorch.org/docs/stable/index.html>
- Dropout Regularization: <https://thepythoncode.com/article/dropout-regularization-in-pytorch>
- KFold Cross Validation: <https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-use-k-fold-cross-validation-with-pytorch.md>