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 **2**

2 Data processing and analysis **2**
2.1 Pre-processing 2
2.2 Analysis 2
2.3 Data partitioning for train, test and validation 3
2.4 Vectorization 3

3 Algorithms and Experiments **4**
3.1 Experiments 4
3.1.1 Dimensionality Reduction 4
3.1.2 FIRST BRUTE-FORCE RUN 5
3.1.3 TYPE OF CELLS 6
3.1.4 Hidden Size 8
3.1.5 STACKED RNNs 10
3.1.6 DATA REGULARIZATION 11
3.1.7 Skip Connections 13
3.1.8 Gradient Clipping 15
3.1.9 K FOLD CROSS VALIDATION 17
3.1.10 Attention Mechanism 18
3.2 Hyper-parameter tuning 21
3.3 Optimization techniques 21
3.4 Evaluation 23
3.4.1 ROC curve 23
3.4.2 F-Score 23
3.4.3 Loss 23
3.4.4 Confusion matrix 24

4 Results and Overall Analysis **24**
4.1 Results Analysis 24
4.1.1 Best trial 24
4.2 Comparison with the first project 26
4.3 Comparison with the second project 27
4.4 Comparison with the third project 27

5 Bibliography 27

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 **a bidirectional stacked RNNs with LSTM/GRU cells** for a twitter dataset about the Greek general elections. In the first assignment, we approached the same subject using logistic regression, while in the second we used deep neural networks.However, in this project we are going to experiment with a Recurrent Neural Network using the Adam optimizer and the cross-entropy loss function in order to examine if it 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, punctuation 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 cleaning is always a good technique in natural processing language so i didn'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 got 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.

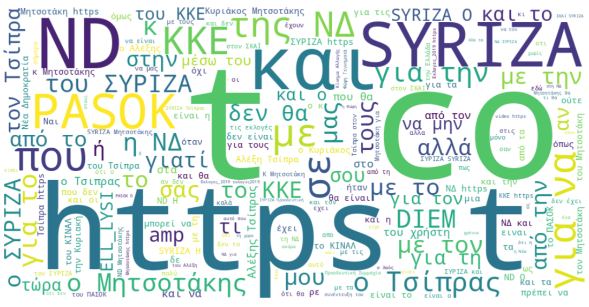


Figure 1: Word Cloud before Preprocessing

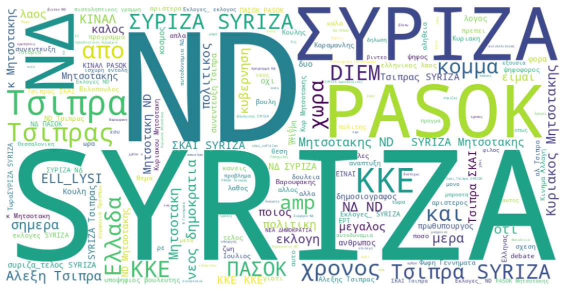


Figure 2: Word Cloud after Preprocessing

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 I concatenated the columns "Party" and "Text", in order to make the the procedure of Vectorization easier.

2.4. Vectorization

<Explain the technique used for vectorization>
For converting text data into vectors, i used 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, in the second method, implemented in attention mechanism models, instead of creating a single vector for each sentence, I now consider the entire set of word vectors. In this way, all the information that would be lost by averaging the vectors, is harnessed usefully. It is worth mentioning that, I could try to use

the word vectors without averaging them in all the experiments and not only in these that involves the attention mechanism. However, that would make the program much more complicated and thus slower, without improving model's performance, due to the bad quality of the dataset.

Note: In the first method, I considered that each dimension of the vector serves as a timestep. More specifically, my tweets are 300-dimensional, meaning that there are 300 one-dimensional timesteps. In contrast, the second technique involves treating each word in the sentence as a separate timestep.

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.

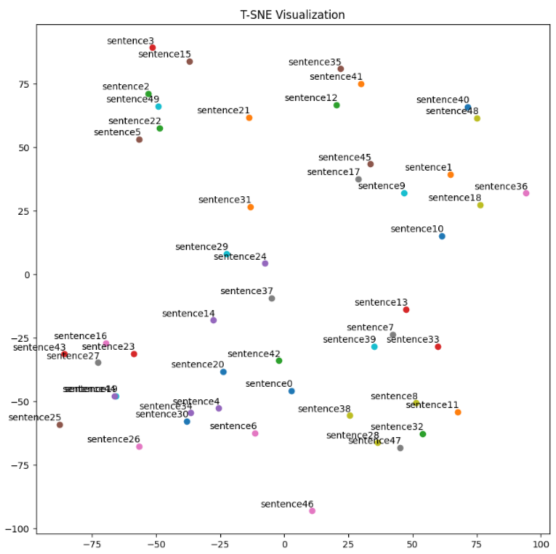


Figure 3: T-SNE Visualization

3.1.2. FIRST BRUTE-FORCE RUN. In this scenario, I created a model using a **Vanilla Recurrent Neural Network**, employing a fundamental architecture with hidden size and batch size of 8 and 64 respectively and learning rate equal to 0.01. This network is the starting point of my effort to find the best model. Upon scrutinizing the plots of both f1 and loss learning curves, we notice that f1 scores can't increase and they are low around 0.33. Moreover, losses are particularly unsteady. For this reason,further investigation is necessary in order to improve the efficiency and enhance the overall F1 scores.

Note: In this case, I used a neural network that take into consideration only the last timestep. Although, hereinafter, I am going to utilize a Reccurent Neural Network that take into account all the timesteps and not only the last one.

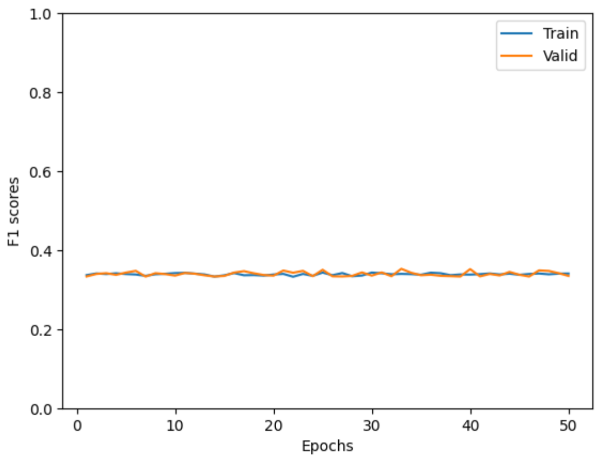


Figure 4: Brute Force: F1 Learning Curves

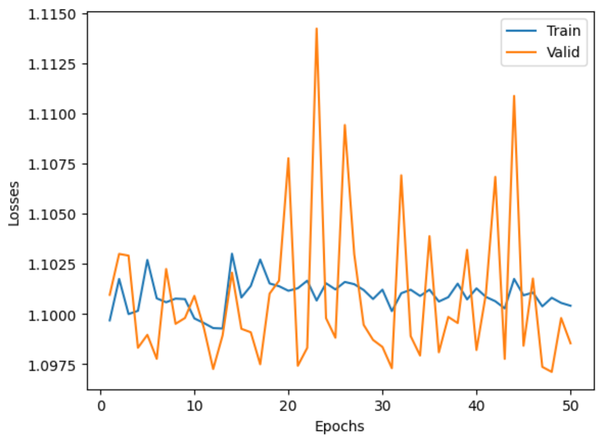


Figure 5: Brute Force: Loss Learning Curves

3.1.3. TYPE OF CELLS. During the experimentation, I tried three architectures in order to handle the data: **Vanilla**, **LSTM** and **GRU**. In theory, Vanilla is the simplest one, but suffers from the vanishing gradient problem, while on the other hand LSTM is the most complicated architecture, since it has the ability to capture information over longer sequences. GRU offers a middle ground, because it tries to avoid the vanishing gradient problem with a plainer approach than LSTM. In our case, the three architectures produced almost the same results, probably due to the bad quality of the dataset. However, in my model I selected to work with the LSTM architecture, as typically is the most suitable choice for sentiment classifiers involving longer texts.

The results of the three approaches for batch size=2048, hidden size=16 and learning rate=0.001 are the following:

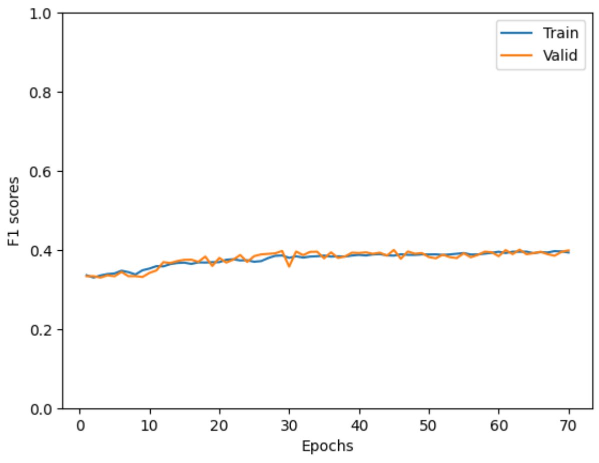


Figure 6: RNN: F1 Learning Curves

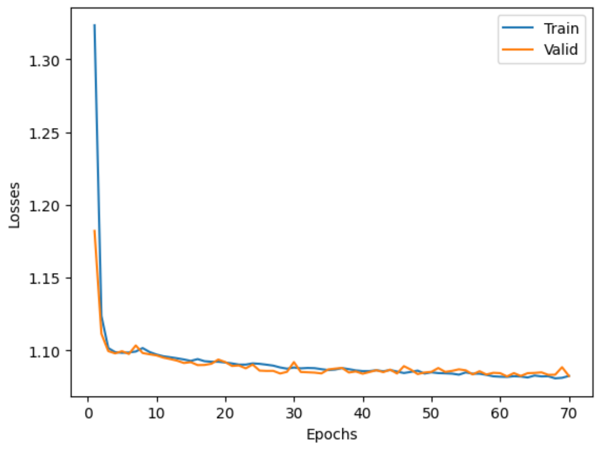


Figure 7: RNN: Loss Learning Curves

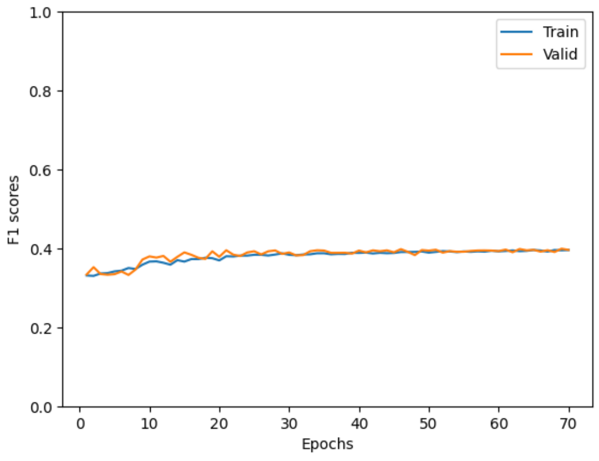


Figure 8: LSTM: F1 Learning Curves

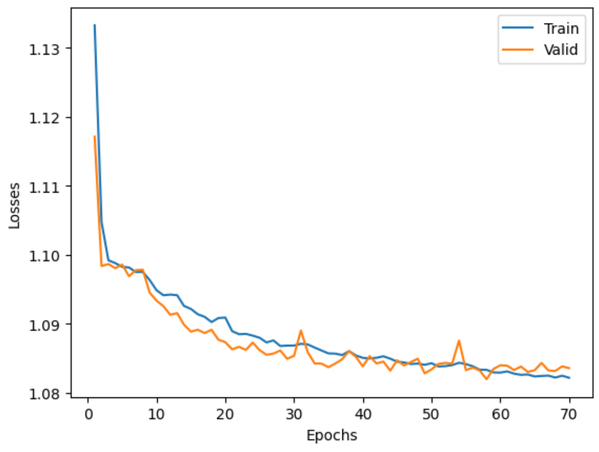


Figure 9: LSTM: Loss Learning Curves

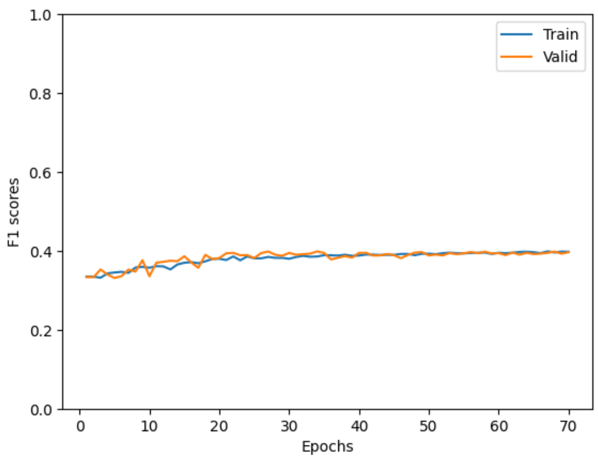


Figure 10: GRU: F1 Learning Curves

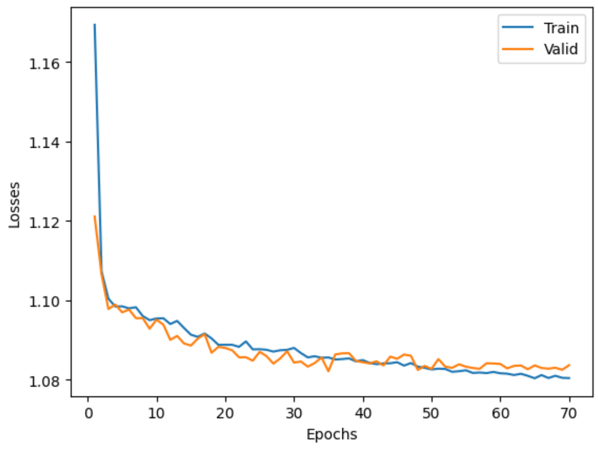


Figure 11: GRU: Loss Learning Curves

3.1.4. Hidden Size. In a recurrent neural network, the hidden size defines the **size of the hidden state**. In other words, it refers to the number of hidden neurons in the recurrent layer and the choice of the appropriate hidden size is essential for the correct functioning of the neural network. On paper, larger hidden sizes allow the model to capture more complex patterns in the data and remember longer sequences in them. However, they can create problems, such as overfitting or the exploding gradient issue. In my model, I experimented with various hidden sizes ranging from 8 to 256. Impressively, as the hidden size increased, the resulting graphs didn't change significantly, despite that the executed time was become grater. For this reason, I decided that the optimal choice was to set hidden size equal to 16, because it creates the best possible results at the best time. Here are indicative the results of hidden size 16 and 128 for batch size 2048 and learning rate 0.001.

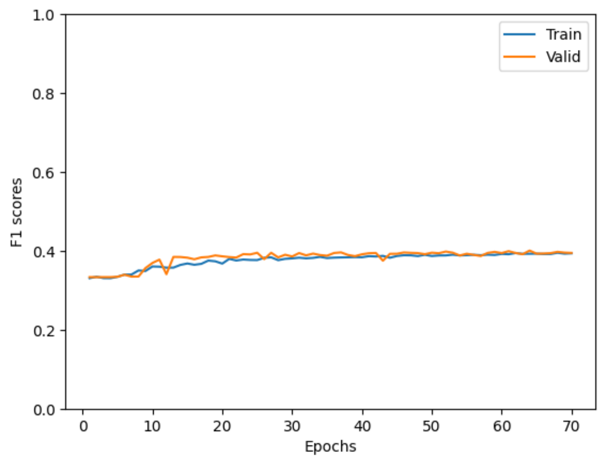


Figure 12: Hidden Size 16: F1 Learning Curves

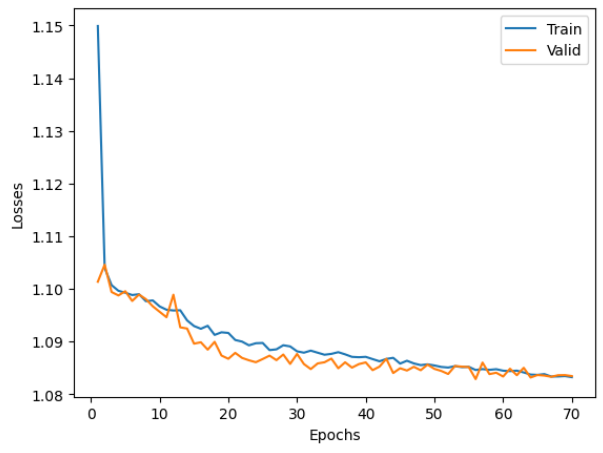


Figure 13: Hidden Size 16: Loss Learning Curves

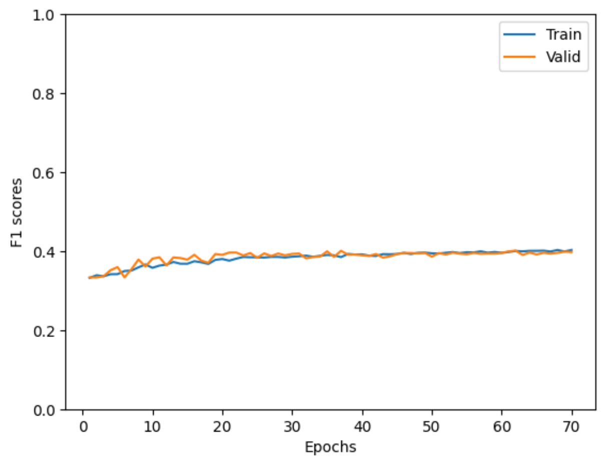


Figure 14: Hidden Size 128: F1 Learning Curves

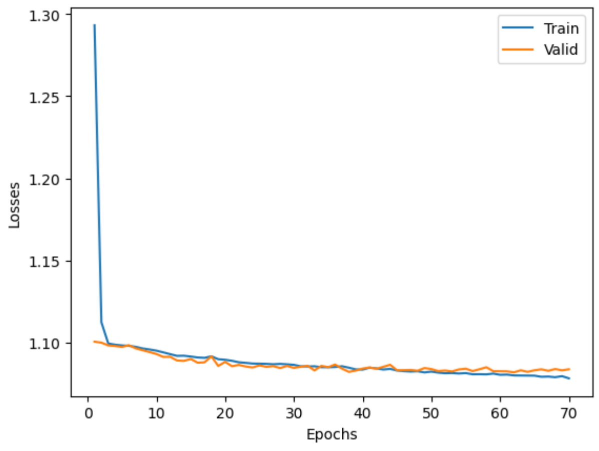


Figure 15: Hidden Size 128: Loss Learning Curves

3.1.5. STACKED RNNs. Stacking multiple recurrent neural networks (RNNs) is an ordinary method in deep learning to capture complex relationships in the data. As for our neural network, I tried to use various stacked RNNs layers, in order to examine the difference in their results. As it is obvious, the more layers we had the more the time increased because the model got deeper and deeper.Although, in all cases, the f1 and loss learning curves remained almost the same, with the only difference being that more layers required more epochs to learn. Therefore, I concluded that in our case the most suitable is the one layer for now, since it requires the less time. Below are listed the plots of using 1 and 3 layers for batch size 2048, hidden size 16 and learning rate 0.001.

Note: Later I will try again more layers, in order to implement the skip connections method.

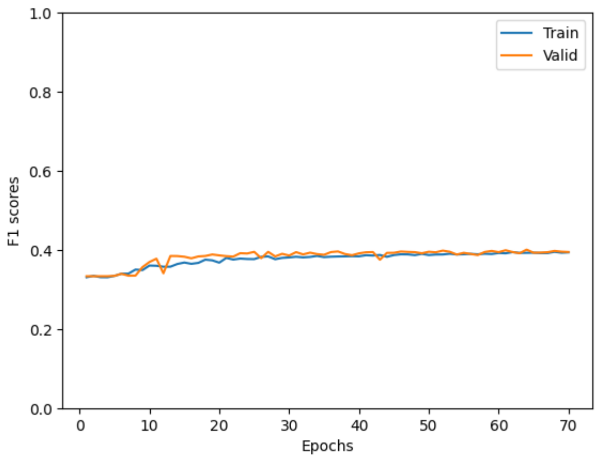


Figure 16: One RNN Layer: F1 Learning Curves

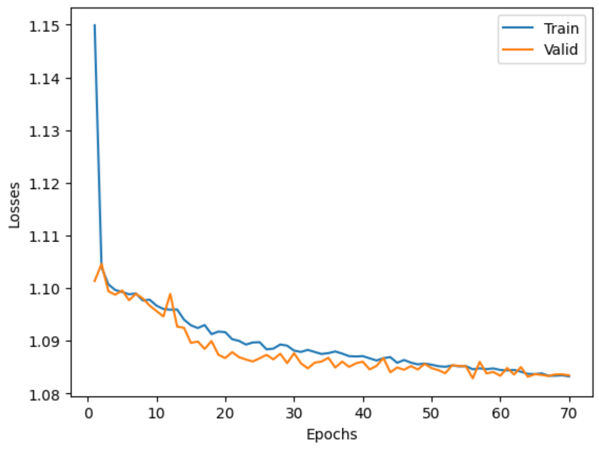


Figure 17: One RNN Layer: Loss Learning Curves

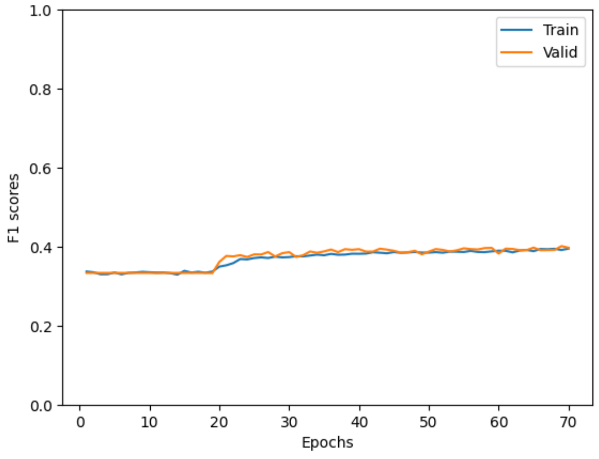


Figure 18: Three RNN Layers: F1 Learning Curves

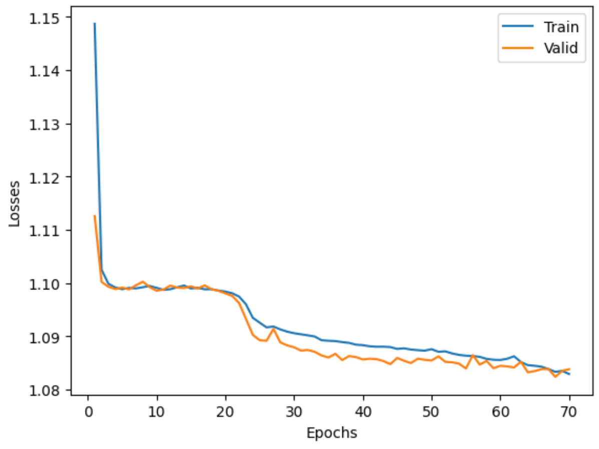


Figure 19: Three RNN Layers: Loss Learning Curves

3.1.6. DATA REGULARIZATION. Regularization is a technique used in machine learning in order to reduce overfitting. For the assignment, I implemented the **Dropout** regularization method.

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 we can notice below, after 90 epochs occurs overfitting, since training loss is lower than validation loss. In order to handle the issue, I experimented with various dropout rates in range [0.1 , 0.5] using the Optuna framework and I ended up that 0.3 is the rate that deals with the problem in the best way.

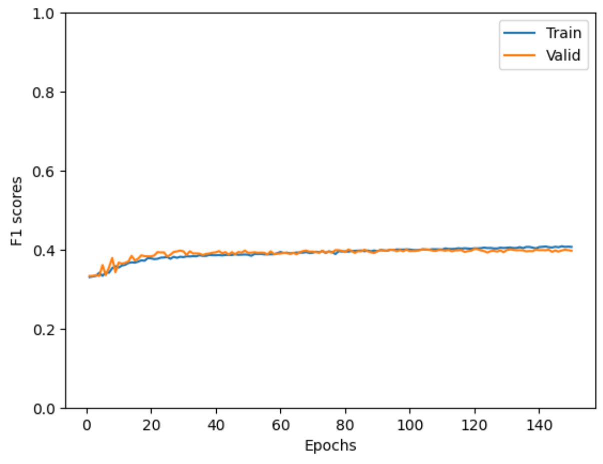


Figure 20: Dropout Rate = 0: F1 Learning Curves

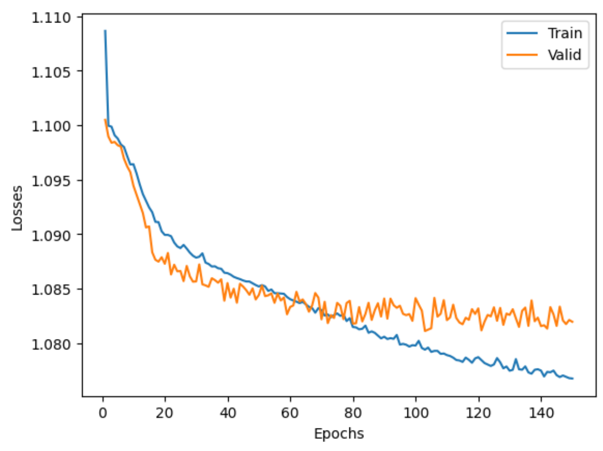


Figure 21: Dropout Rate = 0: Loss Learning Curves

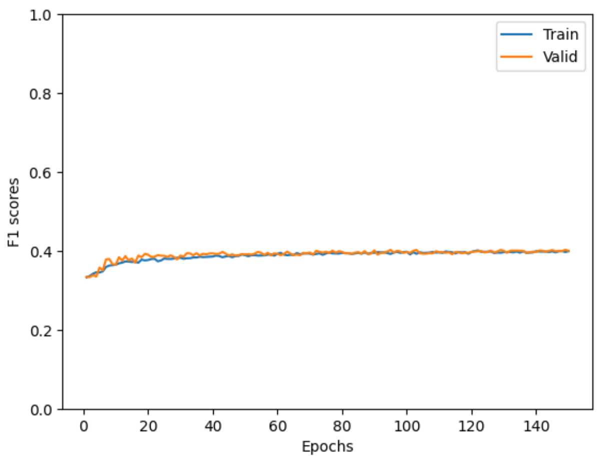


Figure 22: Dropout Rate = 0.3: F1 Learning Curves

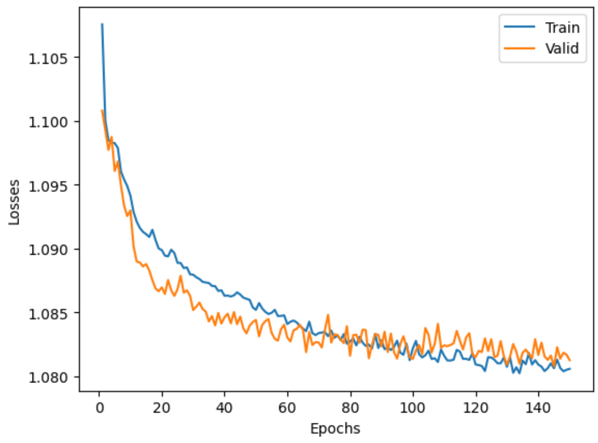
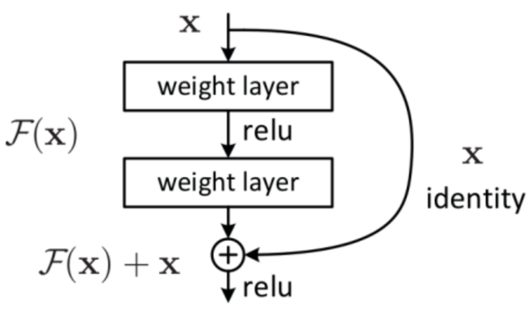


Figure 23: Dropout Rate = 0.3: F1 Learning Curves

3.1.7. Skip Connections. Skip connection, in neural networks design, links a layer's output to a non-sequential layer and is helpful in order to avoid problems like the **vanishing gradient issue**. There are many different approaches of the skip connections method. For example, the technique can be applied to all layers or to some of them. Additionally, the connection can involve the output of all the previous layers or just the output of the immediate preceding layer. Personally, I decided to skip all the layers with the previous output, as illustrated in the image below:



However, applying this method to my neural network didn't change significantly the scores, despite that our model suffers from the vanishing gradient problem. The bad dataset is probably responsible for this result. Despite the observed outcome, I decided to maintain the methodology in my final model, under the consideration that with a better dataset, its relevance would become essential.

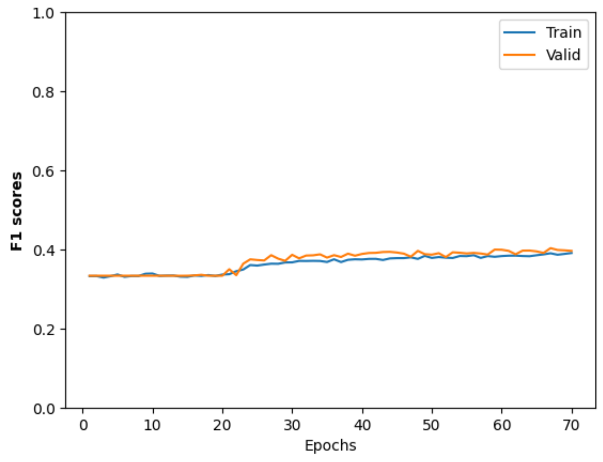


Figure 24: Without Skip: F1 Learning Curves

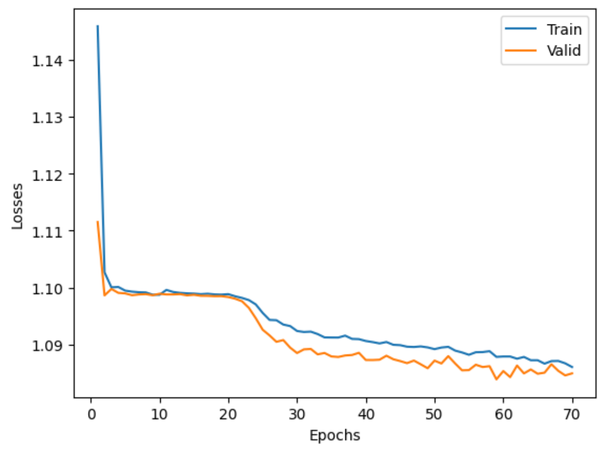


Figure 25: Without SKip: Loss Learning Curves

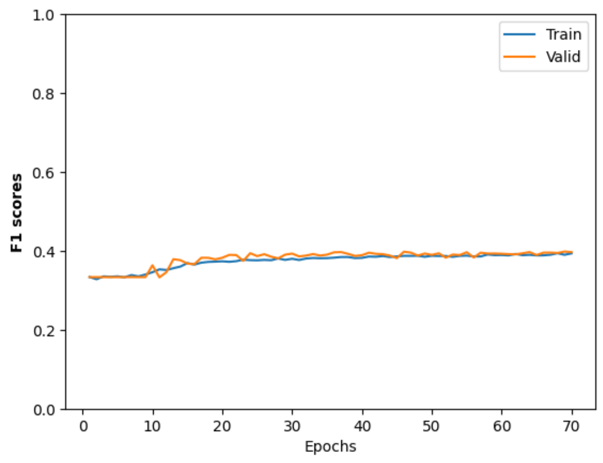


Figure 26: With Skip: F1 Learning Curves

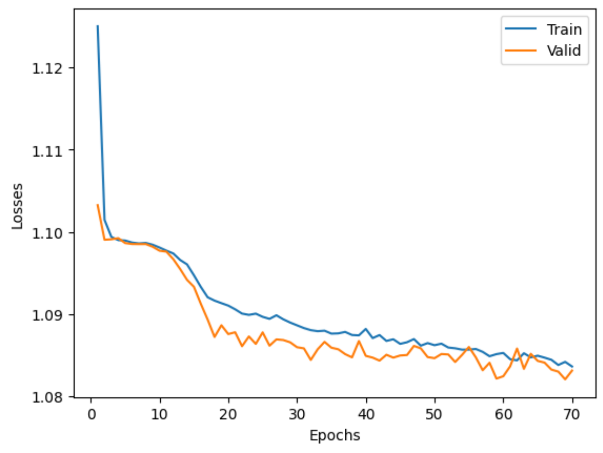


Figure 27: With Skip: F1 Learning Curves

3.1.8. Gradient Clipping. Gradient clipping is a method, which is utilized in the training of a neural network to avoid the exploding gradient problem, leading to more stable and reliable networks. The idea behind gradient clipping is very simple. More specifically, it establishes a threshold value and then scales down gradients whose norms exceeds this threshold. As for me, I attempted to implement gradient clipping in my model, but unfortunately, there was no noticeable improvement, as it has happened in almost all the experiments so far. This probably happens due to the fact that our model suffers from the vanishing gradient issue and not from the exploding one, because the gradients close to zero. However, I opted to retain the gradient clipping method in my final model, considering it is a wise technique to mitigate potential issues. To determine the optimal clip value, I employed the Optuna framework.

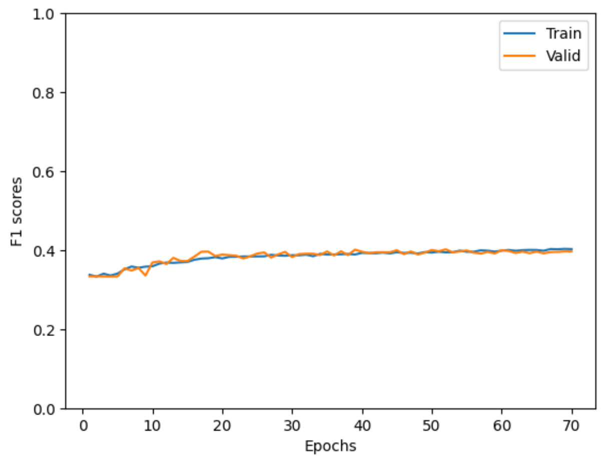


Figure 28: Without Clipping: F1 Learning Curves

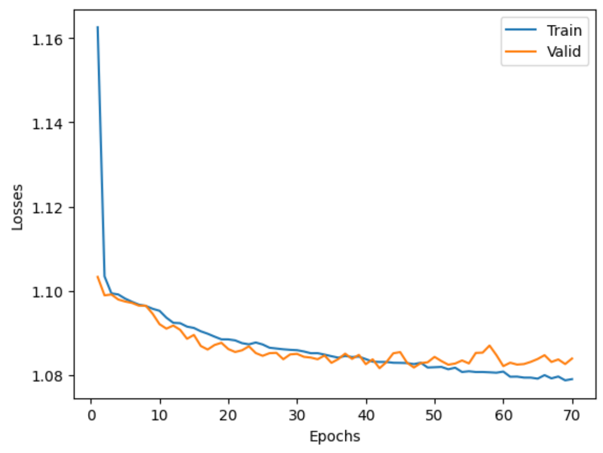


Figure 29: Without Clipping: Loss Learning Curves

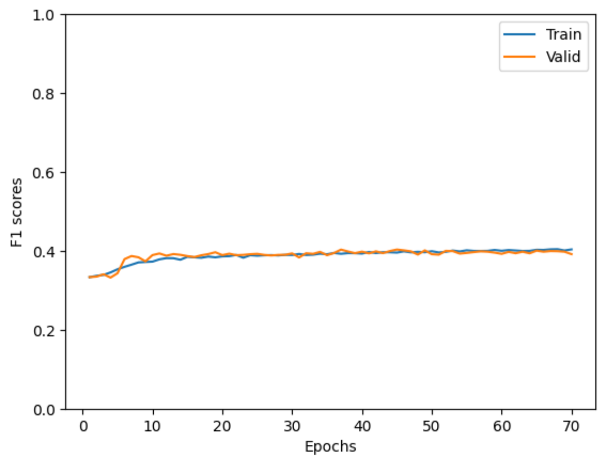


Figure 30: With Clipping: F1 Learning Curves

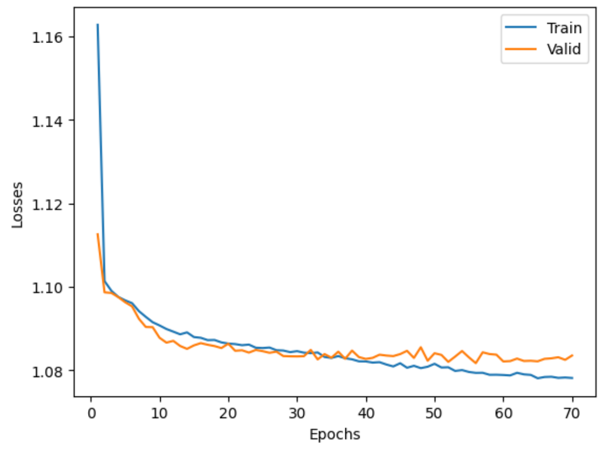
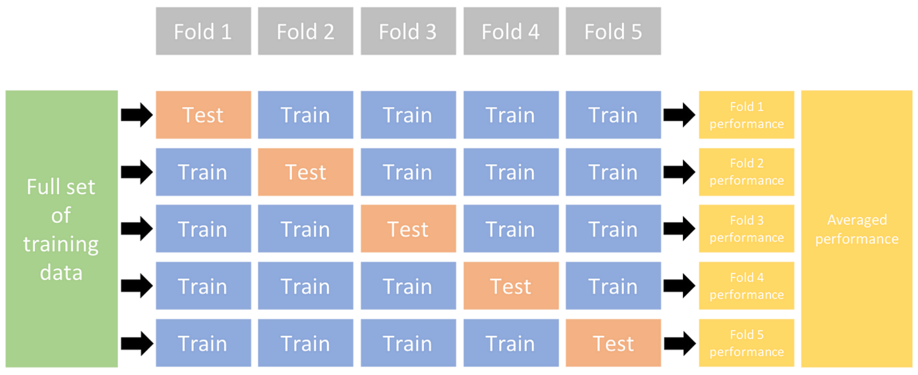


Figure 31: With Clipping: F1 Learning Curves

3.1.9. 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 for the validation set and the remaining folds for the training set, as shown in the following diagram:



Cross validation 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 learning curves below this doesn't occur.

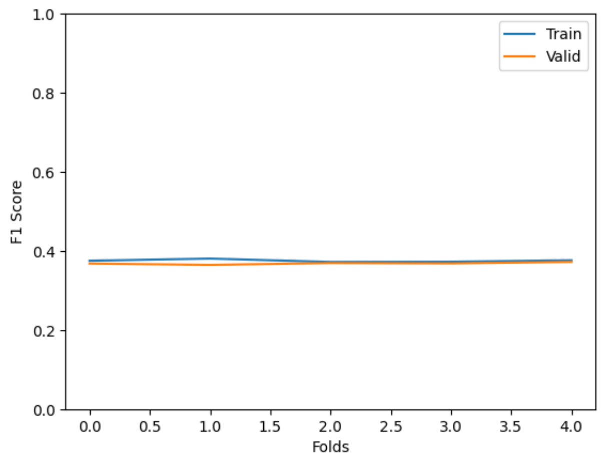


Figure 32: K Fold Cross Val: F1 Learning Curves

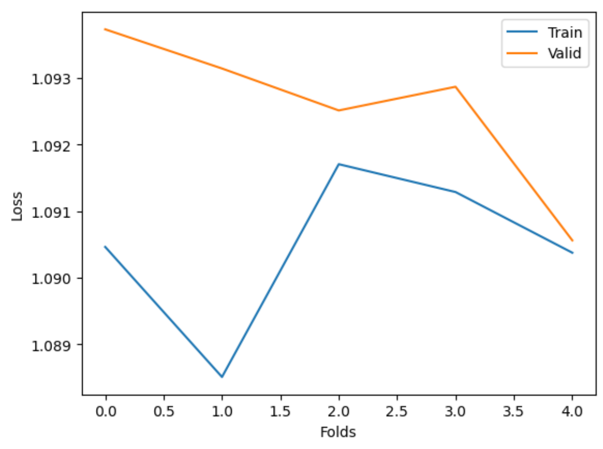


Figure 33: K Fold Cross Val: Loss Learning Curves

3.1.10. Attention Mechanism. The Recurrent Neural Networks we've employed thus far suffer from a significant drawback: They try to remember the whole input sequence through one single hidden unit.The challenge lies in the fact that compressing all the information into one hidden unit may lead to loss of information, especially for long sequences. For this reason, I will use the attention mechanism, which permits the network to take into consideration different parts of the input sequence when making predictions. To incorporate the attention mechanism into my architecture, I followed the following steps:

- **Vectorization**: In order to use the attention mechanism, I had to use the second vectorization technique, mention in the vectorization part in order to utilize usefully the information of all the words.

- **Attention Class**: It is obvious that in order to apply an attention technique, I had to create a class, which would represent a dot-product attention mechanism module for the Recurrent Neural Network. The idea of the mechanism is very simple: It takes as argument the RNN output and the last hidden state and upon completing the necessary mathematical computations, it calculates the required weights and the final context.

  Note: The term **context** refers to the information from the input sequence that is relevant or important for predicting a particular element in the output sequence.

- **RNN Class**: This section contains the fundamental implementation of the neural network, specifically featuring:

  - The **encoder part**, where the input sequences are passed through the RNN layers.
  - The **attention part**, where the context vector is calculated in the way mentioned above.
  - The **decoder part**. In fact the concept of a decoder, as seen in sequence-to-sequence models, is not explicitly implemented. However, the layers that are responsible for transforming the hidden states into the final output can be considered as performing functions akin to a traditional decoder.

In order to examine the functionality of the mechanism, I performed numerous experiments. Below, I present the two most important:

- **Basic Attention Mechanism Approach**: This experiment constitutes the most straightforward application of the mechanism. Specifically, methods like skip connections, dropout regularization, and gradient clipping were intentionally omitted to observe how the model performs without the usage of these techniques. The results indicate overfitting, because the training loss is decreased significantly and the validation loss is increased notable. However, despite the overfitting, the f1 scores demonstrate noteworthy performance, so our focus will now be on mitigating the observed overfitting noticed in both learning curves.

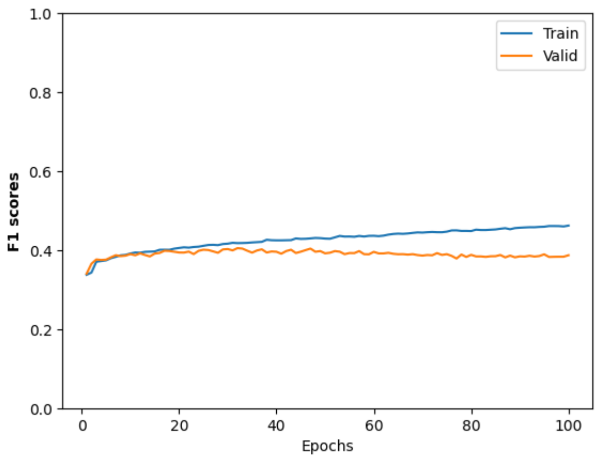


Figure 34: Basic Attention Mechanism: F1 Learning Curves

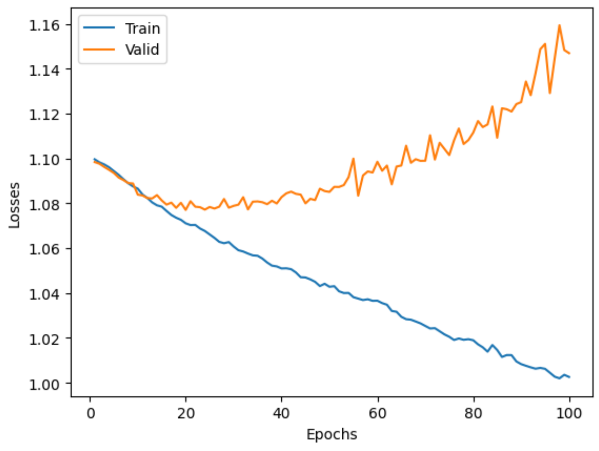


Figure 35: Basic Attention Mechanism: Loss Learning Curves

- **Advanced Attention Mechanism Approach**: As I mentioned before, in order to decrease overfitting and improve the overall model's performance, I will incorporate the skip connections and data regularization methods and i will try a smaller learning rate. As we expected, the significant overfitting has been eliminated and the model has quite better performance compared to cases where the attention mechanism was not employed, since the validation loss is decreased to 1.075. For this reason, this model is what I will use for my final submission.

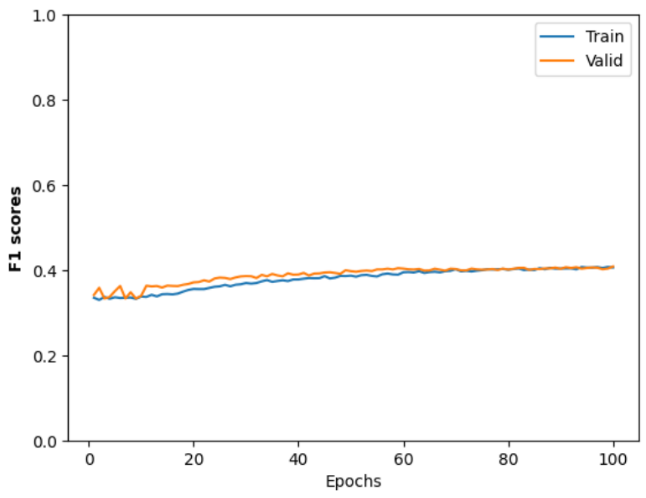


Figure 36: Advanced Attention Mechanism: F1 Learning Curves

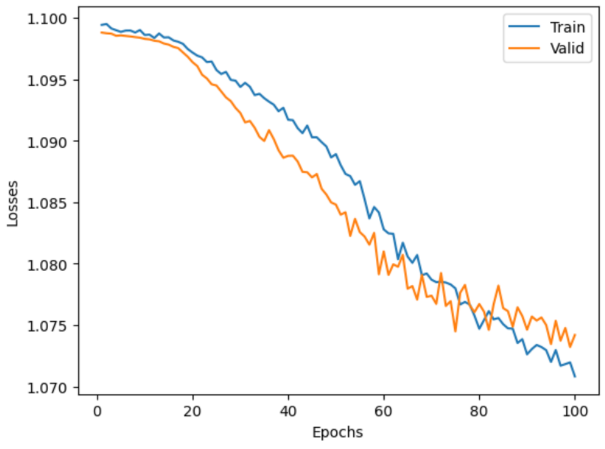


Figure 37: Advanced Attention Mechanism: Loss Learning Curves

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 hidden size, the number of stacked RNNs, the dropout rate, the clip value, the learning rate and the batch size. Regarding to the first four aforementioned hyperparameters, there is a detailed explanation of how i tuned each one of them in the experimentation part of the report. As for the rest two, the approach I followed was the following one:

Learning Rate: Respecting the learning rate, I experimented the values 0.0001, 0.001 and 0.01 and I ended up that when attention mechanism is not employed, the best value is 0.001, because 0.01 creates overfitting and 0.0001 requires a long time for the model to reach a good solution. Nevertheless, when attention is applied, setting the learning rate to 0.001 causes overfitting. Consequently, adjusting the learning rate to 0.0001 becomes crucial to deal this issue. 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.

Batch Size: In regard to the batch size, I simply ran the Optuna framework and examining the results that were created with various batch sizes, I concluded that the best value for batch size is 2048. More details about the methodology I followed using the framework will be given at the Optimization part below.

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 an optimization framework used in machine learning, aimimg to discover the optimal hyperparameters in order to improve the effectiveness of a model. Personally, I used this

framework to discover the optimal batch size, dropout rate and clip value and in this way i avoided all the experiments i should have try in order to determine their values. Another point worth mentioning is that for the best utilization of the framework, I used some of the visualization features optuna provides, in order to analyze the optimization results visually. More specifically:

- **Slice Plot**: From this plot, we can easily understand which hyperparameter values create the best scores. More precisely, a clip value of approximately 0.4 and a dropout rate of 0.1 are considered ideal. Nevertheless, in the construction of my final model, I selected a dropout rate of 0.3. This decision was informed by the fact that the final model is much more complicated than the one used in the Optuna application. As a result, a bigger dropout rate is essential in order to avoid overfitting. Moreover, Optuna suggests a batch size of 512. However, during experimentation I realized that this batch size increased the run time without improving significantly the results, so I decided that the best choice is a batch size of 2048.

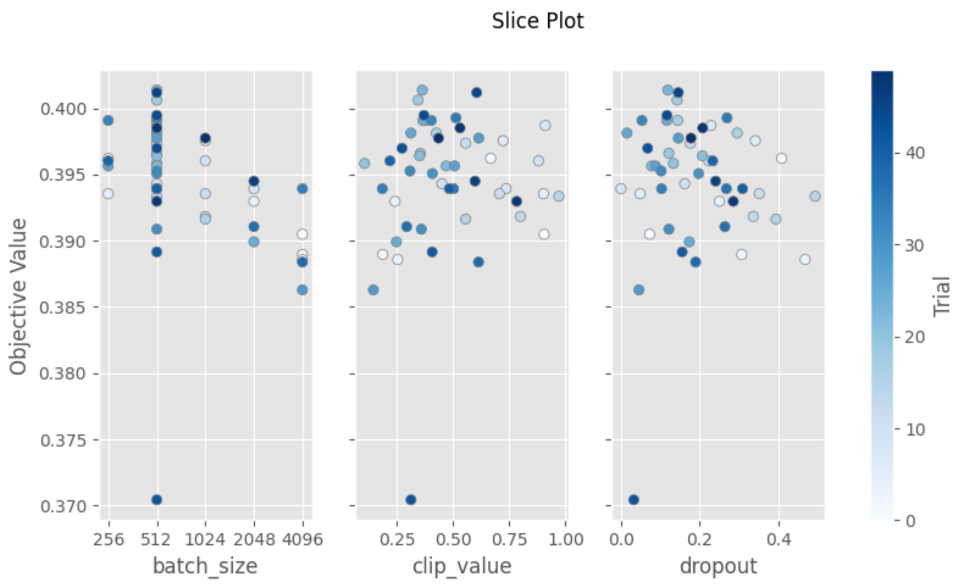
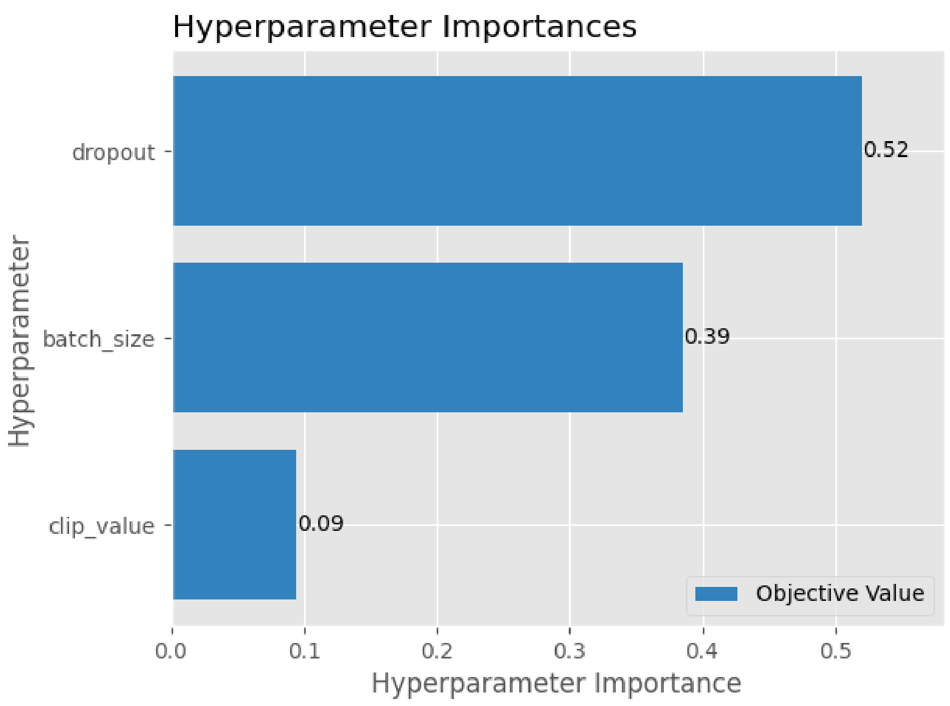


Figure 38: Slice Plot

- **Hyperparameters Importance**: The hyperparameters importances graph refers to the impact of each hyperparameter on the model's performance. The graph reveals that the dropout rate assigns a significant influence of 52 percent, while the batch size has a modest impact of 0.39 percent. In contrast, the clip value has a minor effect on the model, since the assigned importance is 0.09 percent.



Note:If you want to run the Optuna code in my Notebook change the *use_optuna* variable to True.

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 0.57-0.59, 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 to 0.41, which indicates a moderate performance.

3.4.3. Loss. A loss learning curve is a graphical representation of a machine learning model's performance over epochs and depicts the difference between the predicted values and the actual values. In my final model, i used the CrossEntropy loss function and both losses are decreased and reach approximately 1.073 for training and 1.076 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 matrices below, we notice that the model can predict correct 55 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 a bidirectional stacked RNNs 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 try to focus more on how the model corresponds when using the Vanilla Recurrent Neural Network, rather than mainly concentrating on the other two cell architectures."

4.1.1. Best trial. Given all the experiments, I end up that the best trial was the one that combines the following: hidden size of 16, the skip connections method within a framework of 3 stacked RNN layers, gradient clipping with a clip value of 0.4, dropout regularization with a dropout rate of 0.3, integration of an Attention mechanism, a batch size of 2048, utilization of LSTM cell type and a learning rate of 0.0001, because it creates the best results at the best time.

<Showcase best trial>

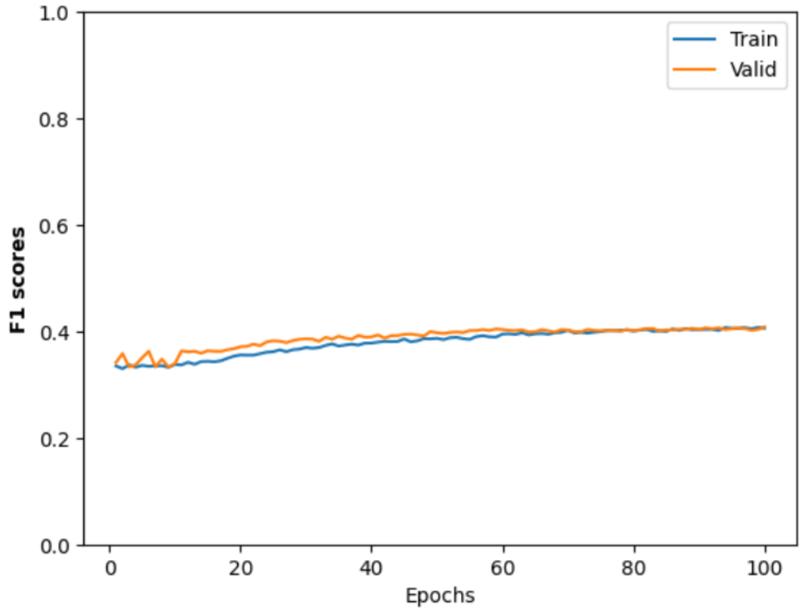


Figure 39: Best trial: F1 Learning Curves

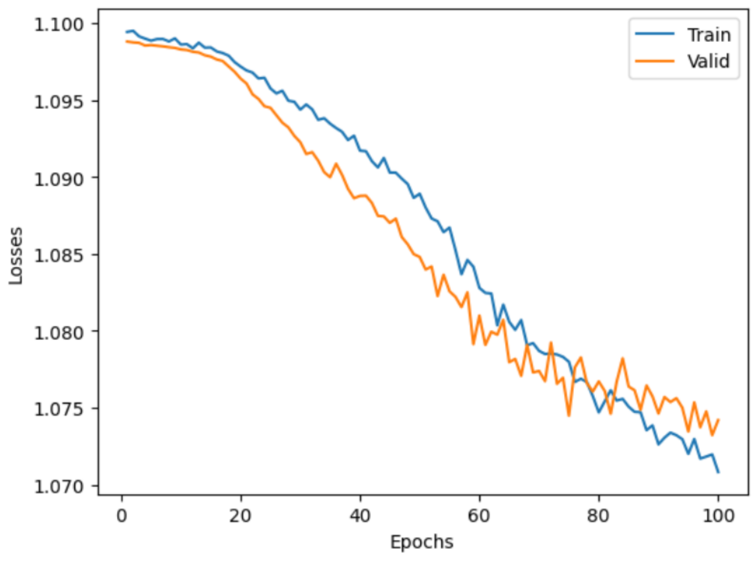


Figure 40: Best trial: Loss Learning Curves

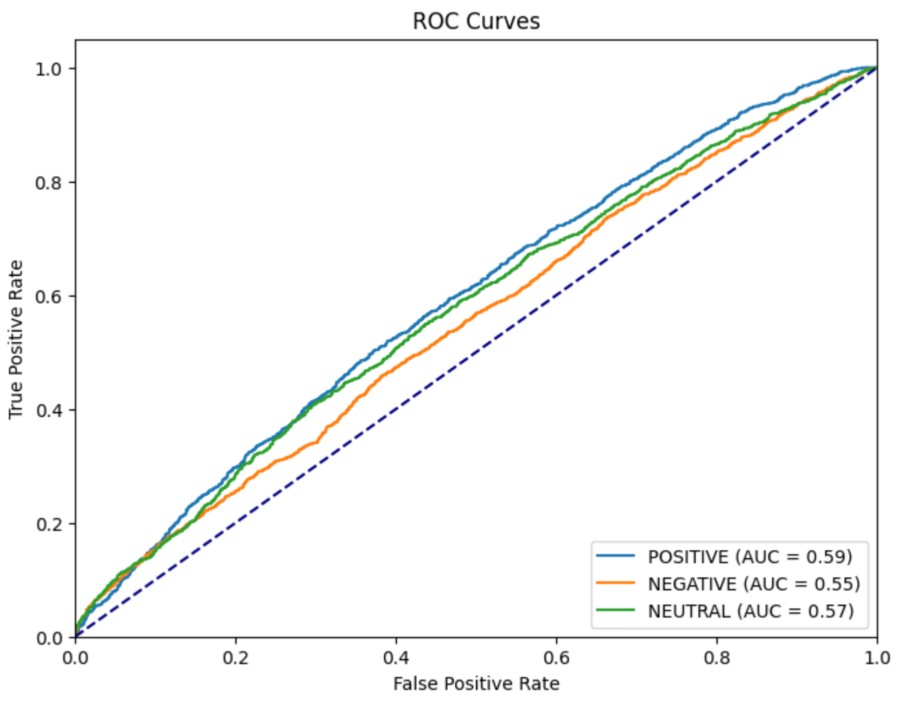


Figure 41: Best trial: ROC Curves

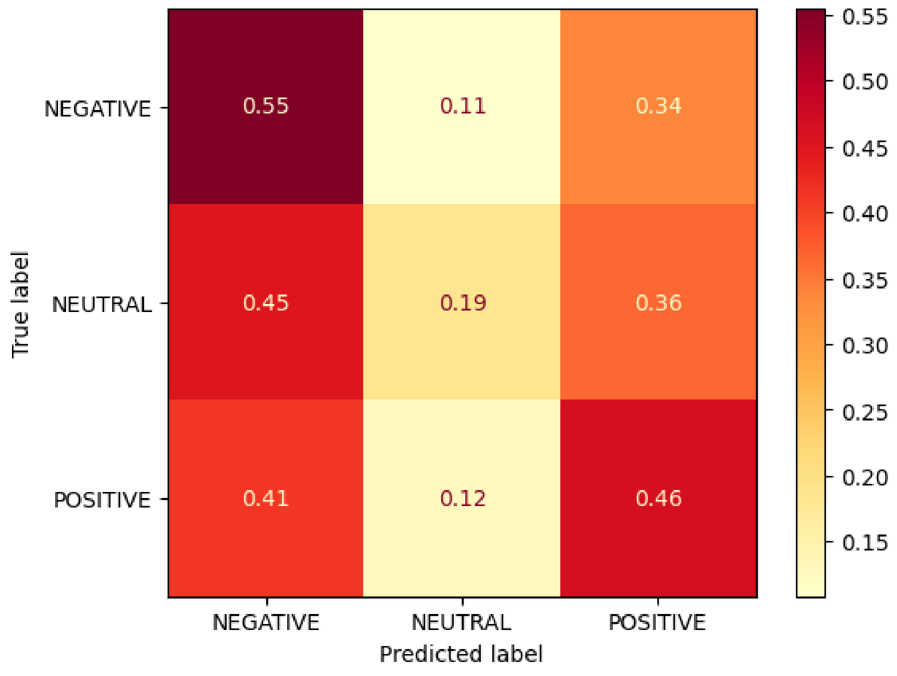


Figure 42: Best trial: Confusion Matrices

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, sentiment classifiers using stacked RNNs 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 first project the model corresponded well to seen data, but it had low performance to unseen ones, creating overfitting ,in contrast to this assignment where the model reacts to both in the same way.

4.3. Comparison with the second project

This assignment closely resembles the second project (the third is a bit better), because both neural networks create similar results. However, in theory, stacked RNNs were expected to improve the model more significantly than deep neural networks, because they are capable to capture long-term dependencies effectively. The reason why RNNs didn't enhance the model isn't clear. The bad dataset is probably the primary factor contributing to the outcome, but we cannot substantiate this estimation with certainty.

<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>
- KFold Cross Validation: <https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-use-k-fold-cross-validation-with-pytorch.md>
- Attention Mechanism: <https://discuss.pytorch.org/t/attention-for-sequence-classification-using-a-lstm/26044/2>