UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Athina Cheilakou*
sdi: sdi2000213

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

Contents

1. Project 1

1.1. Abstract

In this assignment we were tasked to create a sentiment classifier for a dataset of greek tweets using logistic regression in python. Our classifier deals with three classes: NEGATIVE, NEUTRAL, POSITIVE. For this assignment we used the SKLEARN library, SPACY library, as well as a greek stemmer created by Konstantinos Pechlivanis and Eirini Florou[1].

1.2. Data processing and analysis

1.2.1. Pre-processing. We took the following steps to ensure regularisation of the data:

- Apostrophe removal
- Make everything lowercase
- Remove all accents
- Remove mentions starting with the '@' character
- Keep only greek words. This step ensures the removal of urls and english words that might be included in hashtags
- Stopwords removal. For this step a custom list of greek stopwords was used as the spacy library did not cover our needs.
- Lemmatisation. We used the spacy lemmatisation in this step
- Stemming *

*Note that the Spacy library does not provide a stemming utility. Therefore, [1]Greek-Stemmer was used to acquire a stem for each word. Our stemmer function matches a tag provided by the Spacy library to a rough equivalent to be passed to greek stemmer. Exact matching of tags is not possible due to the structural differences between the Spacy tagging system and the ellogon tagging system used by greek stemmer (which is better suited to the particularities of the greek language). For example spacy does not differenciate between femimin and masculin nouns, or singular and plural form.

We then fitted a tfidf vectoriser on the data resulting from the pre processing phase.

1.2.2. Analysis. In this step we checked to see that our dataset is balanced, in order to choose the right metrics for the evaluation of the logistic regression algortithm. The data was balanced, so the macro avarage (for all classes) of $f1\_score$ was chosen as a suitable metric.

1.2.3. Data partitioning for train, test and validation. For hyperparameter tuning we performed cross validation by particioning the train set (*train_set.csv*) into five batches. We chose cross validation to avoid overfitting on the training data. All hyperparameter tuning was done on the train set. We used the data in *valid_set.scv* as a test set for the final model.

1.2.4. Vectorization. We used Sklearn's TfidfVectorizer. This method was chosen since it combines a count vectorization with the TFIDF transformer. IDF penalises words that appear too many times, which allows for a much better understanding of what's more important.

1.3. Algorithms and Experiments

1.3.1. Experiments. In this section we tested various hyperparameters for Sklearn's LogisticRegrssion model on data pre-processed with lemmatisation. We also experimented with different pre-processing techniques by repeating the hyper parameter tuning experiments on data passed through stemming. This was done to in order to determine which pre-processing method yeilds better results. Logistic Regression Parameters:

- Tolerance: we tested various tolerances and concluded that in both cases (lemmatisation and stemming) the algorithm converges when the tol parameter is set to 0.01.
- Inverse Regularisation Strength: in the case of lemmatisation 9.2 was the optimal value among the tests we made. In stemming it was 8.4. We also tested smaller values but the results did not seem to improve.
- Optimisation algorithms: newton-cg yeilded best results for lemmatisation and sag for stemming, which was expected as both algorithms are suitable for multiclass problems and handle multinomial loss.
- Class parameter: multinomial yielded best results in both cases, which was expected as our problem has three classes and the loss minimised in this case is the multinomial loss fit across the entire probability distribution.
- Iterations number: this test did not return usable results as the algorthm failed to converge in some cases. We choose 1000 as a suitable iterations number as we didn't encounter any issue with it.
- Vectoriser Parameters: we experimented with ignoring terms that have a document frequency strictly higher/lower than the given threshold. However results we impacted negatively so we proceeded with the default parameters.

Overall, we got better results with stemming which was ultimately chosen as the pre processing technique of the data in the final model.

Table of trials

Trial	Optimal Parameter	Lem Score	Stem Score
tolerance	0.01	0.3798	0.3819
inverse regularisation strength	9.2, 8.4	0.3829	0.3852
optimisation algorithm	newton-cg, sag	0.3831	0.3853
class parameter	multinomial	0.3830	0.3853
iterations number	1000	0.3830	0.3843
vectoriser parameters	-	0.3660	0.3665

Table 1: Trials

1.3.2. Hyper-parameter tuning. The final configuration of the model was: penalty = l2, tolerance = 0.01, inverse regularisation strength = 8.4, soptimisation algorithm = sag , maximum number of iterations = 1000 and multi class parameter = multinomial.
On the whole, the results of the final testing on the validation data with our configured model were very close to those we saw during training,in both cases not yeilding an $f1\_score$ greater than 0.4 . Judging from these results and the fact that we explicitly used cross-validation during testing, we rule out the possibility of overfitting the training data. However there is strong likelyhood that our model underfits the data. As we can see in the learning curve, adding new instances to the training set makes it increasingly hard for the model to fit the training data, both because the data is noisy and because it is not linear. So the accuracy on the training data goes down, but does not reach a plateau, which might have to do with there not being enough data to train the model sufficiently.

1.3.3. Evaluation. In this project we used f1 score to evaluate the performance of our classifier, as our goal was to strike a balance between precision and recall.

- Precision is a measure of how many of the positive predictions made are correct (true positives).
- Recall is a measure of how many of the positive cases the classifier correctly predicted, over all the positive cases in the data. By using f1 score we aimed a creating a balanced model that makes best use of the data available.

1.3.4. Learning Curve.

1.4. Results and Overall Analysis

1.4.1. Results Analysis. In the end, the results we got were modest, but we suspect that this was due to the language of our data not being fully supported by the available natural language processing libraries. As mentioned above, we had to work with an impromptu stemmer, as the tags returned by the spacy library did not match those of the [1]Greek-Stemmer. Perhaps, our hyperparameter tuning was also insufficient. Given more time, we would experiment with hyperparameter optimisation frameworks such as optuna or scikit-optimize We would also experiment more with the penalty parameter.

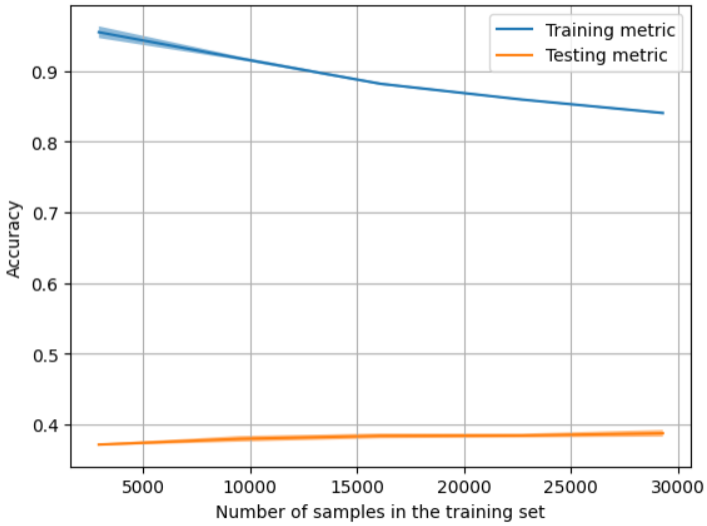

Figure 1: Learning Curve

1.4.2. Results of Final Model on Validation Data. Can be seen on the above image

	precision	recall	f1-score	support
0	0.41	0.40	0.41	1744
1	0.40	0.40	0.40	1744
2	0.38	0.40	0.39	1744
accuracy			0.40	5232
macro avg	0.40	0.40	0.40	5232
weighted avg	0.40	0.40	0.40	5232

Figure 2: classification report

2. Project 2

2.1. Abstract

In this project we were tasked to create a sentiment classifier for the dataset of the previous project this time using a Neural Network. As in the first project our classifier deals with three classes: NEGATIVE, NEUTRAL, POSITIVE. For this assignment we used the SKLEARN library, SPACY library, PYTORCH library, as well as a greek stemmer created by Konstantinos Pechlivanis and Eirini Florou[1]. We also used the Word2Vec model from the GENSIM library for data vectoriastion. For hyperparameter tuning we opted to use the optimisation framework Optuna.

2.2. Model Overview

Our Neural Network, has 2 hidden layers. It implements two regularization techniques: batch normalisation and dropout. In addition, it can be trained and tested with different activation functions and hidden layers sizes, for tuning.

2.3. Data processing and analysis

2.3.1. Pre-Processing. We applied the same pre-processing steps as in the first project. Please refer to the appropriate section for details.

2.3.2. Data Partitioning for train, test and validation. Contrary to the first project we didn't use k-fold cross validation for the huperparameter tuning stage, as it was deemed too time consuming to implement in this case (although it would be optimal). Instead, we concatenated the *train_set* and *valid_set* and then split the data into 80% training data, 10% validation data and 10% test data.

2.3.3. Vectorisation. We vectorised the input data using a 100-element Word2Vec vector described here: Gensim Models Word2Vec. These vectors capture information about the meaning of the word and their usage in context, in other words two contextually similar words will have similar vectors.
We first pre-process and stem the data. We then extract the Word2Vec vector (embedding) of each word in a review. The embeddings are then added and divided by the number of words in that review. This gives us an average vector of the review. We tackle out of vocabulary words by assigning them a vector with zero values.

2.4. Algorithms and Experiments

2.4.1. Experiments. In this section we tested various hyperparameters for our Neural Network model on data pre-processed with stemming. We performed 50 trials in total and 25 epochs per trial (arbitrarily chosen small number for quick testing), using $f1\_score$ as the parameter we want to optimize. This time we focused on learning and using the Optuna framework, in order to streamline hyperparameter tuning, something we were not able to do in the first project. We considered doing a multiple objective optimisation (minimizing loss while maximizing $f1\_score$), which led to added run time but did not account for a boost in model performance, so we decided not to follow this approach.

Neural Network parameters:

- Optimizer Function: we experimented with various optimizer functions such as Adam, Adamax, RMSprop, NAdam. We got the best results using the Adamax optimiser, although this result seemed to vary depending on the runs
- Activation Function: [ReLU, ELU, SELU, LeakyReLU]
- Neuron Size: [128,512]
- Learning Rate: [0.01, 0,3] with a step of 0.02
- Dropout Rate: [0.05, 0.4] with a step of 0.05

2.4.2. Hyper-parameter tuning. According to our tests the best results were obtained using the Adamax optimizer and LeakyReLU activation function, with a learning rate of 0.2 and dropout rate of 0.2.

2.4.3. Evaluation. In this project we used f1 score to evaluate the performance of our classifier, as our goal was to strike a balance between precision and recall.

2.5. Results and Overall Analysis

2.5.1. Results Analysis. We reached an $f1\_score$ of 38% (this time macro avaraged $f1\_score$). In comparison to the first projet we got slightly worse results. We suspect this is due to the complexity of our model (many hyperparameters to tune, large hyperparameter space) and the naive approach we followed in order to tune them (50 trials is a relatively small number and we also entrusted the choice of parameters for each test to the optuna framework). Overall the results remain very modest due to the nature of our dataset (impromptu stemmer needed for the greek language, same as in the first project, some poorly labeled tweets)
Possible improvements could result from adding another hidden layer to our model and experimenting with techniques for the initialisation of the weights in our model, as presented in the lecture slides.

2.5.2. Learning Curve. Below we present the learning curve for our model. The blue line represents $f1\_score$ during testing, and the orange one represents $f1\_score$ during validation:

2.6. Comparison with the first project

Overall we got a slighlty worse f1 score than in the first project. We assume that this is due to our own methods being inadequate, as we expected the more sophisticated neural networks to produce a better result than plain linear regression.

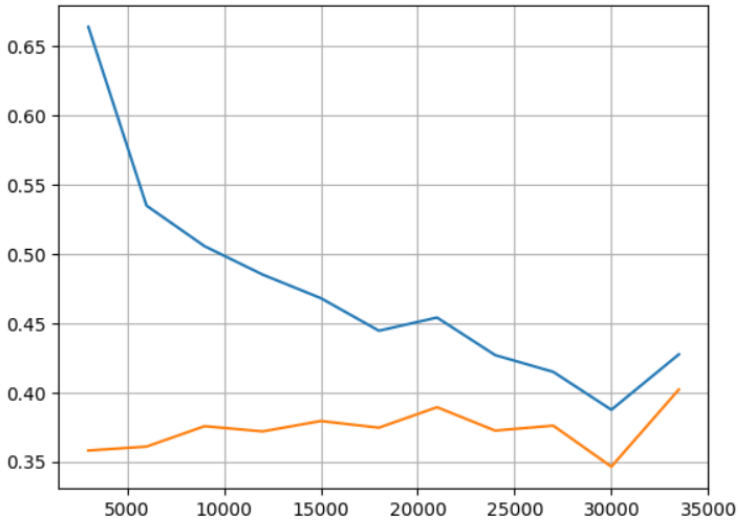

Figure 3: Learning Curve

3. Project 3

3.1. Abstract

In this project we were tasked to create a sentiment classifier for the dataset of the previous projects this time bidirectional stacked Recursive Neural Networks with LSTM/GRU cells. As in the two previous projects our classifier deals with three classes: NEGATIVE, NEUTRAL, POSITIVE. For this assignment we used the SKLEARN library, PYTORCH library, as well as a greek stemmer created by Giorgos Marios Patelis based on the work of Georgios Ntais. We also used the Word2Vec model from the GENSIM library for data vectoriastion.

3.2. Data processing and analysis

3.2.1. Pre-Processing. We applied the same pre-processing steps as in the first and second project, although this time we replaced the [1]Greek-Stemmer used in the two previous projects with one made by Giorgos Marios Patelis, a collegue of ours. This was a huge step forwards as we abandoned the impromptu matching of tags between the spacy library and the old stemmer. Instead, the new stemmer uses a streamlined process based on the work of Georgios Ntais.
After pre-processing we transformed the data into a 100-element vector using the Word2Vec model from the Gensim library

3.2.2. Data Partitioning for train, test and validation. We partitioned the data in the same way as in the second project. Please refer to the appropriate section for details.

3.2.3. Vectorisation. We used 100-element word vectors as in the second project. Please refer to the appropriate section for details.

3.3. Algorithms and Experiments

3.3.1. Experiments. In this section we tested various hyperparameters for our Neural Network model on data pre-processed with stemming. We performed 50 trials in total and 25 epochs per trial (arbitrarily chosen small number for quick testing), using $f1\_score$ as the parameter we want to optimize. Similarly to the second project, we used the Optuna framework in order to streamline hyperparameter tuning.

Recursive Neural Network parameters:

- Cell Types: We experimented both with LSTM and GRU cells, but LSTMs yielded the best results.
- Number of hidden layers per RNN: Hidden layers per RNN range from 2 to 128, although small numbers of connections where favoured during hyperparameter tuning
- Number of stacked RNNs layers: We experimented with networks of 2 to 10 layers of stacked RNNs, yeilding the best results with few layers.
- Nonlinearity function: We tested both options,tanh, relu, choosing tanh in the end.
- Skip Connections: We experimented with networks utilising skip connections, and with ones that didn't. The final model uses skip connections.
- Learning Rate: We experimented with learning rates ranging from 0.01 to 0,3 with a step of 0.02. The smallest learning rate was chosen, which was not expected.
- Dropout Rate:Both small (0.05) and larger(0.4) dropout rates where tested. In the end a relatively small rate was chosen (0.15)

3.3.2. Hyper-parameter tuning. The final configuration of the model was: num_layers: 3, hidden_size: 19, learning_rate: 0.01, cell_type: LSTM, clip: 1.0, skip: True, nonlinearity: tanh, dropout_rate: 0.15000000000000002, the best value: 0.33164322489138803

On the whole, the results of the optuna testing seemed to be worse than the final testing on the validation data with our configured model, as we achieved an $f1\_score$ of 0.39 during validation. Judging from these results and the fact that we explicitly used skip connections and dropout during testing, we rule out the possibility of overfitting the training data. However there is strong likelyhood that our model underfits the data. As we can see in the learning curve, adding new instances to the training set makes it increasingly hard for the model to fit the training data, both because the data is noisy and because it is hard to capture contexual relationships in greek tweets. So the accuracy on the training data bounces up and down unpredictably, but does not reach convergence.

3.3.3. Evaluation. In this project we used f1 score to evaluate the performance of our classifier, as our goal was to strike a balance between precision and recall.

3.4. Results and Overall Analysis

3.4.1. Results Analysis. We run the 50-test Optuna study multiple times yielding different results in each run. Taking all the executions into account we concluded that best results are obtained by relatively small LSTM rnns(few layers, and small number of hidden connections), with skip connections, dropout and gradient clipping applied. We speculate that this is due to the unpredictable nature of our corpus. In other words models with more connections and weights struggled to fit the inconsistent nature of greek tweets (the language in these tweets varies, from grammatically correct sentences e.g. "Αυτό που είναι συγκλονιστικό είναι η ψυχασθένεια του Τσίπρα" to hashtags and phrases understood only given the subtext e.g. #απολυμανση_χοριοι #απεντομωση_χοριος #απολυμανσεις #χοριος #Alphatv #Την Κυριακη #Κουλης #Τσιπρα). The overall struggle of the network to fit our data is also evident in the learning curve as the lines indicate that convergence is not achieved and that accuracy bounces unpredictably between epochs.
Based on our results, we concluded that these types of networks are better suited to grammatically coherent corpuses e.g. news articles, as they capture contextual relationships, something that is a hard to achieve in short "coded" messages such as tweets. In the end, we reached an f1 score of 0.3998 in our validation set. We are satisfied with our results, given our limitations.

3.4.2. Learning Curve and Optuna Plots. Below we present the learning curve for our model. The blue line represents $f1\_score$ during testing, and the orange one represents $f1\_score$ during validation:

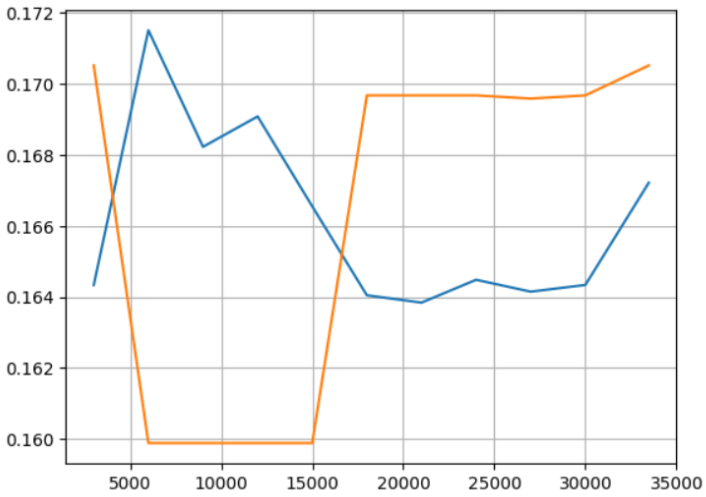

Figure 4: Learning Curve

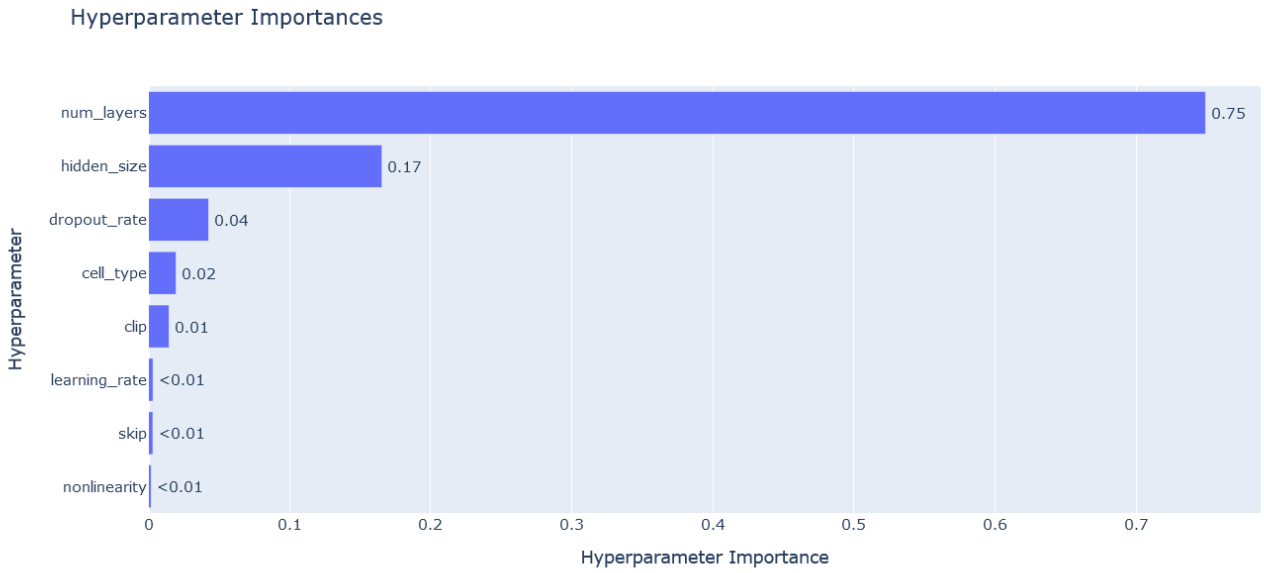

Figure 5: Hyperparameter importance regarding the duration of tests

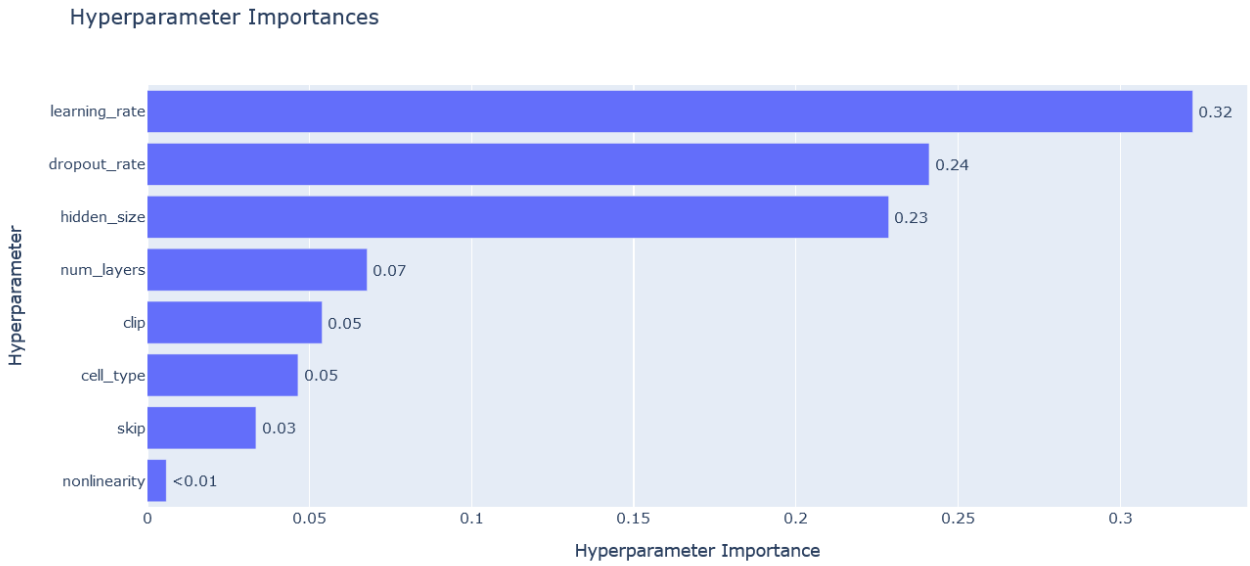

Figure 6: Hyperparameter importance regarding f1 score

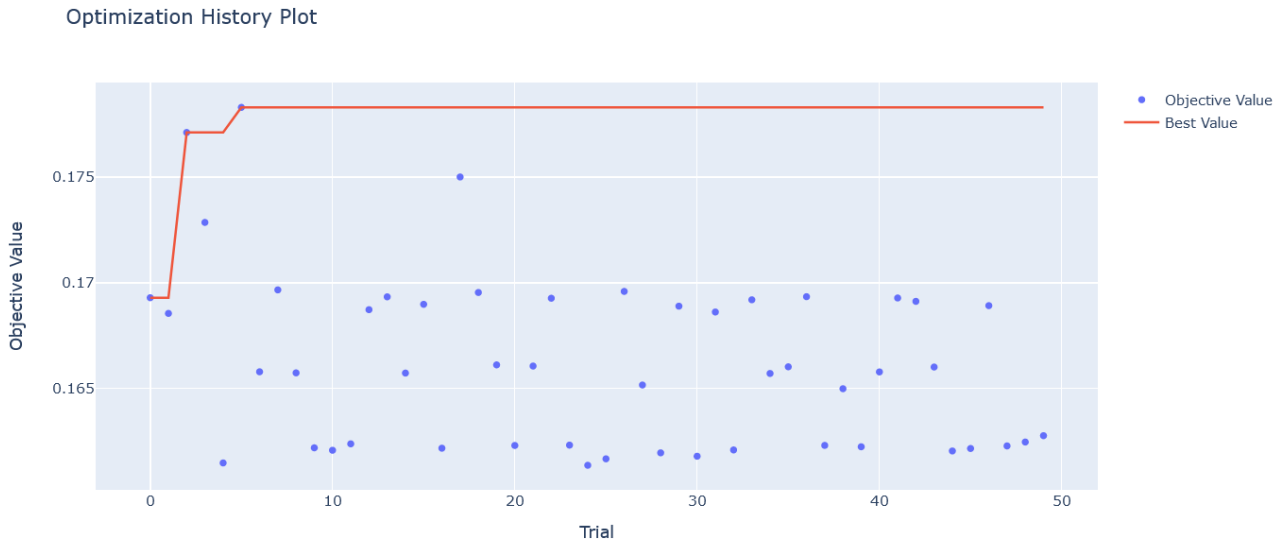

Figure 7: Optimisation History

3.4.3. Comments on the Plots. Regarding Figue 4 the unpredictable behavior of the learning curve is discussed in the above sections. As we can see in Figure 5, the number of layers affects the duration of tests the most, which is expected as the propagation of the gradients through the network taking recursion into account is the most time consuming aspect of the calculations. Thus, time is increased proportionately to the increase of layers in the network.
In Figure 6 we observe that f1 score is mostly affected by the learning rate of our network followed by the dropout rate, which is natural as the learning rate can affect wether convergence is reached during training and dropout is our major line of defense against overfitting the data.
The number of hidden layers seems to play a big role two, which is expected as the number of hidden connections affects how well our model can adapt to the nature of our data.
In Figure 7, the Objective Value on the y-axis is the f1 score of the model. From looking at the Best Value line, we can see that 50 trials was probably much less than was necessary for our noisy data, as only one dot meets the red line (Best Value). Additionally, the nearly right angle of the Best Value line shows that Optuna found good values for our data in well under twenty trials and was unable to improve on those results that much in continuing to tune up to 50 trials. This means that Optuna struggled to predict the direction for the parameter optimisation, and reached the best value early probably at random, which is in accordance to the unpredictable nature of our dataset.

3.5. Comparison with the first project

Overall, our RNN model reached similar f1 scores with the linear regresssion one we tested in the first project. It seems that this f1 score is close to the maximum that can be achieved for our data set, given its unpredictable nature.

3.6. Comparison with the second project

Overall, our RNN model performed better than the simple Neural Network model. This can be attributed to the fact that RNNs are better suited for natural language processing and also to the fact that we did not perform sufficient testing in our second Project.

4. Project 4

4.1. Abstract

4.2. Data processing and analysis

4.2.1. Pre-Processing.

4.2.2. Data Partitioning for train, test and validation.

4.2.3. Vectorisation.

4.3. Algorithms and Experiments

4.3.1. Experiments.

4.3.2. Hyper-parameter tuning.

4.3.3. Evaluation.

4.4. Results and Overall Analysis

4.4.1. Results Analysis.

4.4.2. Learning Curve.

4.5. Comparison with the first project

4.6. Comparison with the second project

4.7. Comparison with the third project

5. Bibliography

References

[1] Eirini Florou Pechlivanis Konstantinos. Greek-stemmer, December 2021.

Stanford Stemming and Lemmatisation
analyticsvidhya.com Stemming and Lemmatisation
databasecamp.de Stemming and Lemmatisation
Tutorial On Spacy Part Of Speech Tagging
Scikit-learn Cross Validation
Scikit-learn Logistic Regression
Differences between micro and macro avareging
Data Scaling
Learning Curve Plot
*F*1_*score*
Project 2:
Skip Gram neural network from scrath
Gensim Models Word2Vec
Sentiment analysis using Word2Vec
Cross Validation for Pytorch Neural Network
Testing Different Activation Functions in Pytorch
Cross Entropy Loss Function
Multi Objective Optimisation using Optuna
Implementing Dropout in Pytorch
Kaggle Introduction to neural networks
Batch Normalisation using Pytorch
Torch.nn.Sequential
Project 3:
Torch RNN
Skip Connections
Attention
Torch Multihead Attention
torch LSTM
Torch GRU
Optuna Framework
Torch Gradient Clipping