

Deep Learning for NLP

Student name: *Dimitrios Tsiompikas*
sdi: *7115112300036*

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	9
2.4	Embeddings	9
3	Algorithms and Experiments	10
3.1	Experiments	10
3.1.1	Table of trials	10
3.2	Hyper-parameter tuning	11
3.3	Optimization techniques	11
3.4	Multihead Attention (Bonus)	11
3.5	Evaluation	12
3.5.1	ROC Curve	12
3.5.2	Learning Curve	13
3.5.3	Confusion matrix	14
4	Results and Overall Analysis	15
4.1	Results Analysis	15
4.1.1	Best trial	17
4.2	Comparison with the first project	18
4.3	Comparison with the second project	18
5	Bibliography	19

1. Abstract

Goal of the third assignment is to train and evaluate a Recurrent Neural Network to identify sentiment (POSITIVE, NEUTRAL or NEGATIVE) of tweets based on a Tweet dataset from the 2023 Greek Elections. I will fulfill this task by following a classic ML process which is: Firstly clean the dataset from data that might harm our train , test and valid sets. Then create some visualizations to get a grasp of what I'm dealing with , using tokens , the sentiment column , word similarity and linear substructures. Train the model with the techniques mentioned below (Word2Vec, hyperparameter tuning, different techniques used in RNNs, test the different types of RNNs , namely LSTM and GRU) and finally evaluate the model with metrics and plots.

2. Data processing and analysis

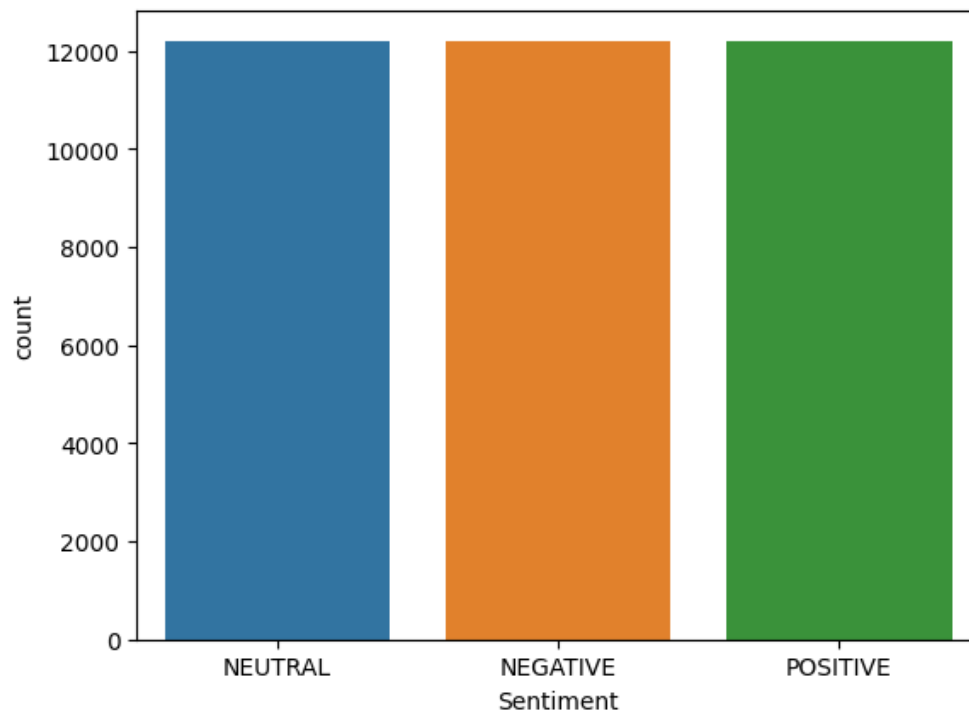
2.1. Pre-processing

For data pre-processing, I firstly checked for null values to clean them , there were none so I followed the cleanup with a hand-made tweet cleanup function that removes all mention signs , hashtags , links (URLs), punctuations and all words after mentions or hashtags (example: @akis18 , I removed akis18) as they were in English and would cause trouble with model training. I also created a stopwords list in Greek , removed all useless Greek words ,then I added English words in the list as well (taken from the tweets of all datasets) and removed them as well from the tweets in order to keep the model from having to deal with words in two languages. Lastly, I removed special symbols to further improve my model training and get the best possible results. In this part, I noticed that the dataset has some issues that cannot be fixed in such a large amount of data like wrong sentiment in specific tweets or sometimes even the wrong party. This will cause problems later on with overfitting and keeping the scores on low percentages.

2.2. Analysis

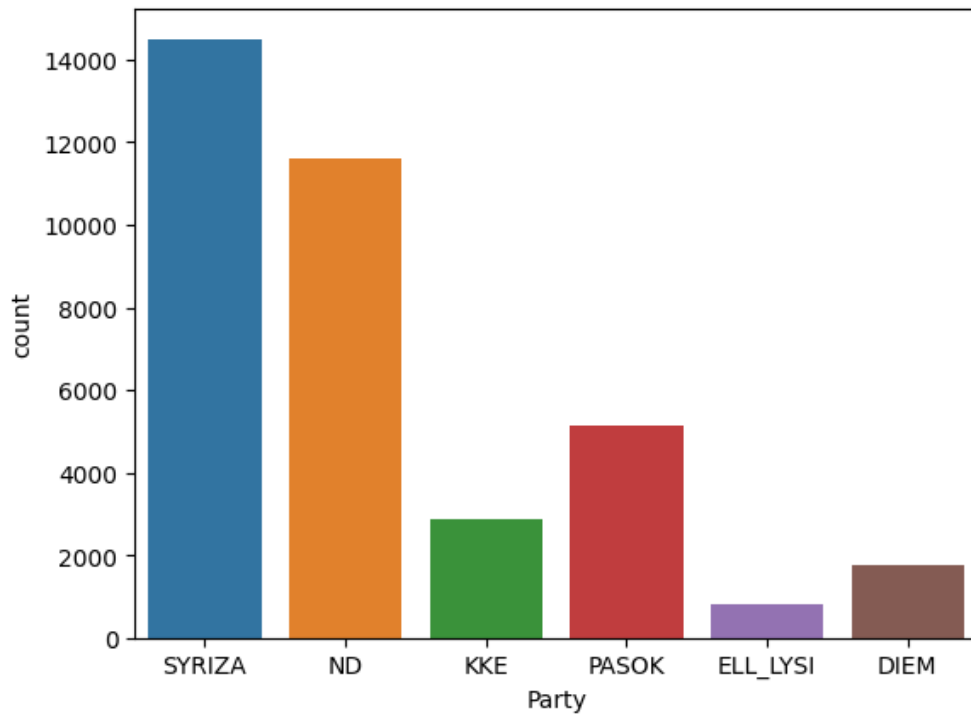
Firstly, I began with analyzing the dataset columns:

Figure 1: Sentiment countplot



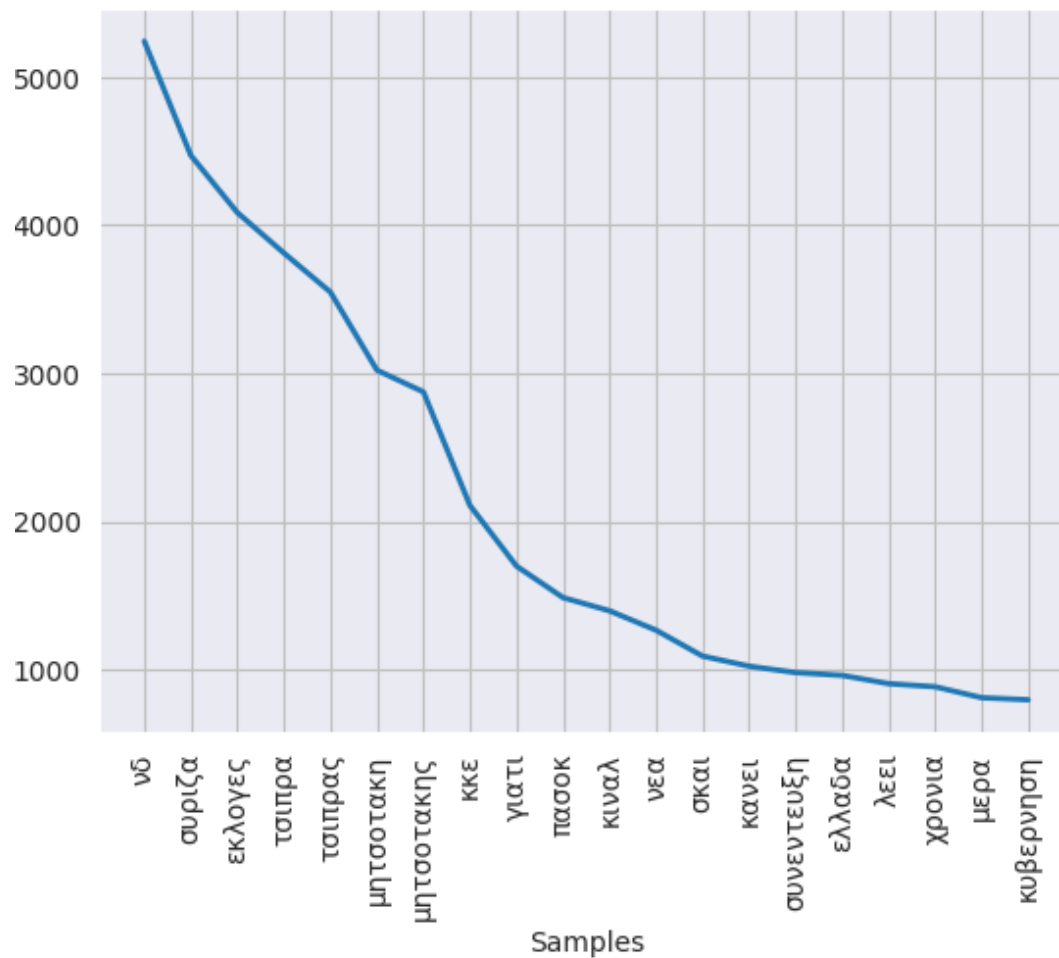
We have the same count for all sentiments.

Figure 2: Party countplot



Here, we can see that the most prevalent party in the tweets is SYRIZA which makes sense since SYRIZA was the political opposition for ND.

Figure 3: Word frequency diagram from Text column



From this diagram we can see the most frequent tokens in the Text column. We can see that the first ones are related to Alexis Tsipras, Kyriakos Mitsotakis, ND and SYRIZA as they were the two main competitors for the 2023 elections. Afterwards, parties with lower percentages follow such as KINAL and KKE and some other words related to elections or politics.

Figure 5: Most similar words to the word Mitsotakis


Most semantically similar to word Mitsotakis:

word	similarity score
=====	
μητσοτακης...	0.9547027349472046
χωρων	0.9515827298164368
ζητησε	0.9509779810905457
διαφορετικος	0.9440712332725525
απελευθερωσε	0.9418005347251892
μακρυνιτσα	0.9397727847099304
συνοδος	0.9377662539482117
υπερηφανους	0.9341276288032532
αβραμοπουλος	0.9335430860519409
αποδειχθει	0.9329411387443542

Here we can see some words that are similar to the word Mitsotakis using the word embeddings, the first word is indeed similar since it's the same thing, the rest might correspond to something that Mitsotakis has done, or words that follow the word Mitsotakis right after in tweets.

Figure 6: Most similar words to the word Tsipras

Most semantically similar to word Tsipras:

word	similarity score
=====	
ξεβρακωμα	0.9591887593269348
	0.9571133852005005
συμβιβαστηκαμε	0.9513048529624939
σπαει	0.9492200613021851
μετανιωνω	0.9466826319694519
σκαϊτσιπρας	0.9450240135192871
τσιπραςσυνταγμα	0.944040834903717
πρωινες	0.9435499310493469
ανακοινωνει	0.9427955746650696
ζαγορα	0.9409149885177612

Here we can see words that are similar to the word Tsipras , again, using the word embeddings. Apart from every word that contains the same substring (tsipras) the rest of the words seem to correlate to a TV channel named SKAI in Greece which is known to be more ND-favored among Greek citizens and probably gets backlash to all tweets regarding Alexis Tsipras hence the similarity.

Figure 7: Linear substructure visualization using Word Embeddings



Here I used the words "μητσοτακης, τσιπρας, συριζα, νδ, κιναλ, κκε" as my vocabulary for the Linear substructure visualization because they're very common words in the tweets. The distancing between words is apparent due to issues in the dataset.

2.3. Data partitioning for train, test and validation

I've used the train set for training, test set to predict the labels and valid set to create the metrics.

2.4. Embeddings

For this assignment we had to use word embeddings to create the input tensors for our neural networks. I used Word2Vec library from gensim, I tokenized each tweet (Text column of each dataset) after preprocessing, created sentences for each dataset, trained my word2vec model with them and then created word embeddings for each word (with vectors of size 100). Afterwards, I convert the tweets into embeddings, one-hot encode the labels of the train and valid data sets to add them to the tensors and also created a small dictionary to map the one hot vectors to numbers 0,1,2 so I can create the submission.csv later on. Finally, I add the tweet embeddings to torch tensors in order to feed them to the NNs.

3. Algorithms and Experiments

3.1. Experiments

Firstly, I created a bi-directional stacked RNN architecture, which uses the cell type string to differentiate between LSTM, RNN and GRU NNs. I made one initial LSTM run to get the learning curve for one epoch, to see how the loss function behaves in each step of evaluation. The results are very distorted of course, because the dataset has the aforementioned issues that have been discussed during all exercises. Best number of epochs this time was 60 since this gave me the best possible results for every experiment.

Furthermore, I took the initiative to test all 3 architectures to see their results. Due to dataset issues, the performance is a bit random regarding the F1-score and LSTM, GRU and RNN have similar performance overall. This is why I chose LSTM to be used for my next experiments. I also tested the following techniques: Skip connections, Gradient clipping and the Attention mechanism as mentioned in the paper "Attention is all you need" but they did not provide much different results unfortunately.

Lastly, using my results from the previous homework, I got the best hyperparameters from there and through experimentation I did not have to use Optuna this time. For each experiment, I used the Adam optimizer and CrossEntropyLoss function as mentioned in the assignment. Below you will find the table with my results from the experiments.

3.1.1. Table of trials.

Trial	Learning rate	hidden layers num	clip value	heads num	Score
LSTM	1e-3	2	-	-	39%
plain RNN	1e-3	2	-	-	39.3%
GRU	1e-3	2	-	-	40.1%
Skip connections	1e-3	2	-	-	38.7%
Gradient clipping	1e-3	2	1.0	-	39.2%
Attention	1e-3	2	-	4	39.7%

Table 1: Trials

3.2. Hyper-parameter tuning

I used hyperparameters from the assignment as mentioned by the instructors and through experimentation I found the best hyperparams for all my experiments. The best values I found for these hyperparameters are:

Learning rate = 1e-3

Dropout rate = 0.2

Optimizer = Adam

Number of hidden layers (used for stacked RNN) = 2

Bidirectionality = true (Bidirectional RNN)

Clip value = 1.0 (used for Gradient clipping)

Number of heads (Used for Multihead attention mechanism) = 4

Providing the best score which was 40.1%.

3.3. Optimization techniques

I mostly used experimentation and evaluated the hyperparameters through the classification reports to find out what was best for my NNs. Stable performance was achieved using the above, however the scores do not change that much due to dataset issues.

3.4. Multihead Attention (Bonus)

In this assignment, I also used the attention mechanism as described in the paper "Attention is all you need". PyTorch has a ready-to-go implementation of the algorithm used in the paper (essentially a nn.MultiheadAttention layer), that can be used in a NN architecture. Multihead attention allows for multiple sets of attention weights (called heads) to be applied to the input sequence to output a weighted sum.

Then all sums of the weights are concatenated and linearly transformed to produce the final output. This allows the model to focus on different parts of the input sequence simultaneously, capturing different aspects and relationships within the data. Unfortunately, due to the dataset issues mentioned above, the difference in F1-Score wasn't that tremendous, it did help LSTM get a bit better, but not much difference after all.

3.5. Evaluation

The predictions were evaluated using the valid set sentiment column, I put all NNs to evaluation mode, compared my predicted results with the actual values from the y batch of the valid dataset and I used the following metrics:

Classification report: this produces several metrics all-in-one , namely Recall, Precision, F1-score and Accuracy.

Recall: this shows how many true results are returned from the predictions.

Precision: this shows how relevant are the results from the predictions.

F1-score: this uses recall and precision to find out how accurate the model is with its predictions. It ranges from 0 to 1 and the closer it is to 1 the better.

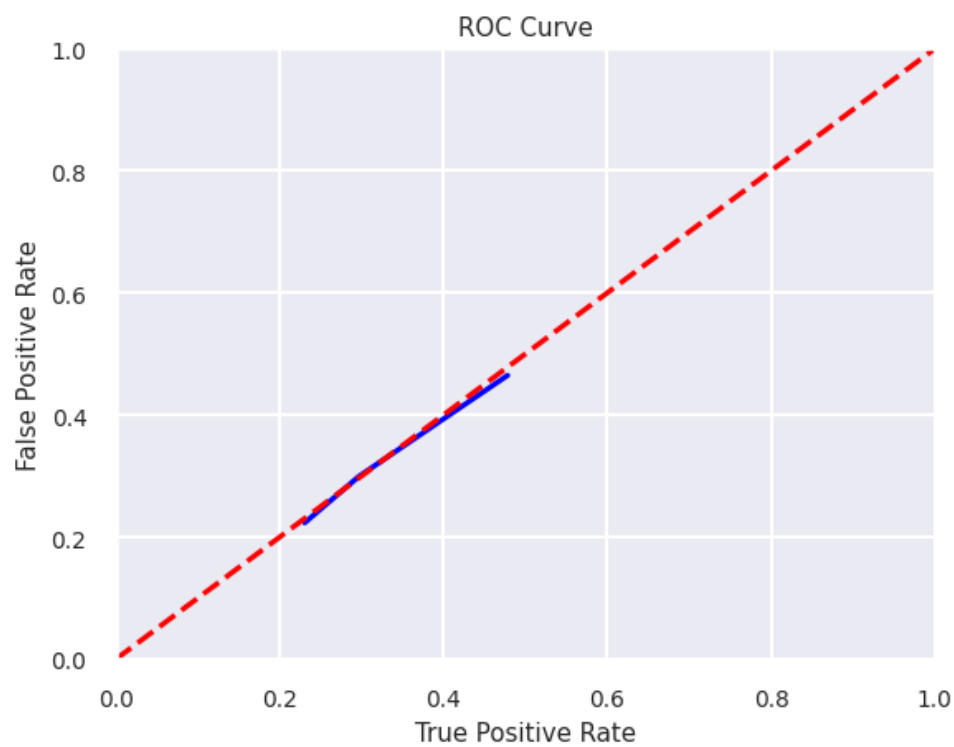
Accuracy: this calculates the accuracy of the model by calculating the fraction of number of true results to the number of total results.

Confusion matrix: This is a table that shows the True positive , true negative, false positive and false negative counts of values that the model has found. It's used to calculate the metrics above.

Below I will add figures showcasing each of the results.

3.5.1. ROC Curve.

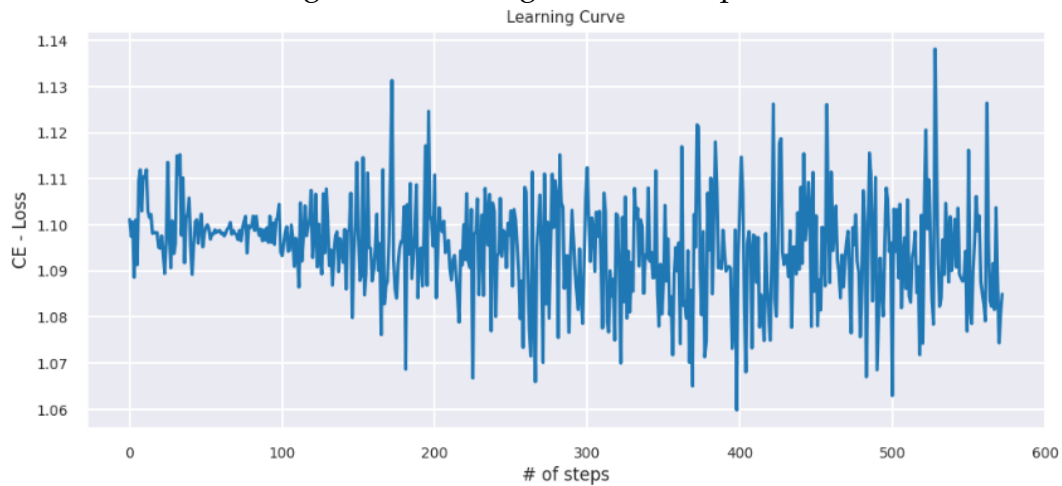
Figure 8: ROC Curve with best hyperparams



3.5.2. Learning Curve.

Here we observe that there is way too much distortion in the graph (Loss is fluctuating with each evaluation step and we can see apparent overfitting due to the dataset issues) .

Figure 9: Learning curve for 1 epoch



3.5.3. Confusion matrix.

Figure 10: Confusion Matrix with best hyperparameters (best trial)

```
[[811 527 406]
 [823 522 399]
 [846 508 390]]
```

4. Results and Overall Analysis

Figure 11: LSTM Multihead attention scores

```
===== LSTM (ATTENTION) NN SCORES =====
precision    recall  f1-score   support

NEGATIVE      0.38      0.62      0.48      1744
NEUTRAL       0.41      0.29      0.34      1744
POSITIVE      0.42      0.28      0.34      1744

accuracy              0.40      5232
macro avg           0.40      0.40      0.38      5232
weighted avg       0.40      0.40      0.38      5232

accuracy: 0.39755351681957185
f1: 0.39755351681957185
total f1: [0.47537754 0.33670715 0.33776868]
```

Figure 12: LSTM Gradient clipping scores

```
===== LSTM (GRADIENT CLIPPING) NN SCORES =====
precision    recall  f1-score   support

NEGATIVE      0.37      0.61      0.46      1744
NEUTRAL       0.44      0.22      0.30      1744
POSITIVE      0.40      0.35      0.37      1744

accuracy              0.39      5232
macro avg           0.41      0.39      0.38      5232
weighted avg       0.41      0.39      0.38      5232

accuracy: 0.39239296636085624
f1: 0.3923929663608562
total f1: [0.46194111 0.29640947 0.37166513]
```

4.1. Results Analysis

Figure 13: LSTM NN scores

```

===== LSTM NN SCORES =====
      precision    recall  f1-score   support

    NEGATIVE      0.39      0.52      0.44      1744
     NEUTRAL      0.41      0.22      0.29      1744
    POSITIVE      0.38      0.42      0.40      1744

 accuracy          0.39          5232
 macro avg      0.39      0.39      0.38      5232
weighted avg      0.39      0.39      0.38      5232

accuracy: 0.3900993883792049
f1: 0.3900993883792049
total f1: [0.44493392 0.29090909 0.40184632]

```

Figure 14: plain RNN scores

```

===== RNN SCORES =====
      precision    recall  f1-score   support

    NEGATIVE      0.39      0.52      0.44      1744
     NEUTRAL      0.40      0.36      0.38      1744
    POSITIVE      0.40      0.29      0.34      1744

 accuracy          0.39          5232
 macro avg      0.39      0.39      0.39      5232
weighted avg      0.39      0.39      0.39      5232

accuracy: 0.3929663608562691
f1: 0.3929663608562691
total f1: [0.44379562 0.38012571 0.33786923]

```

In conclusion, the results of the experiment were pretty underwhelming again. Since the dataset problems were beyond repair the model only produced results up to 40%. On average, Neural networks usually produce around 80-95% with proper tuning and data cleanup. So the expected results would be in that amount of percentages.

I also noticed that the training loss and validation loss were fluctuating (going up and down) due to overfitting after 10-15 epochs, I tried ReLU hidden layers for regularization to fix as much as I could, but it would still overfit. Some more techniques I could have used include: Used GLoVe pre-trained embeddings, combined the techniques all together in one NN architecture to see the results. I believe I did a handful of experiments taking note of each percentage so the only thing I'd do additionally is using the techniques mentioned previously. Below, you'll also find the skip connection results.

Figure 15: LSTM skip connections scores

```

===== LSTM (SKIP CONNECTIONS) NN SCORES =====
              precision    recall  f1-score   support

    NEGATIVE         0.40      0.42      0.41       1744
     NEUTRAL         0.38      0.47      0.42       1744
    POSITIVE         0.39      0.27      0.32       1744

 accuracy                   0.39       5232
  macro avg              0.39      0.39      0.38       5232
 weighted avg            0.39      0.39      0.38       5232

accuracy: 0.38704128440366975
f1: 0.38704128440366975
total f1: [0.40988614 0.41880559 0.31625683]

```

4.1.1. Best trial. Below, you will find my metrics for my best trial. Essentially, after testing the problem thoroughly, I concluded in using the following techniques: GRU stacked bi-directional NN, with 0.001 learning rate, dropout rate = 0.2, Adam as the optimizer, 2 hidden layers after hyperparameter tuning. Best f1-score resulted in 40.1%.

Figure 16: Best trial classification report, with micro f1 score and accuracy

```

===== GRU SCORES =====
              precision    recall  f1-score   support

    NEGATIVE         0.39      0.56      0.46       1744
     NEUTRAL         0.41      0.36      0.38       1744
    POSITIVE         0.41      0.28      0.34       1744

 accuracy                   0.40       5232
  macro avg              0.40      0.40      0.39       5232
 weighted avg            0.40      0.40      0.39       5232

accuracy: 0.4013761467889908
f1: 0.4013761467889908
total f1: [0.4592803 0.3847319 0.33684927]

```

Figure 17: Confusion Matrix with best hyperparameters (best trial)

```

[[811 527 406]
 [823 522 399]
 [846 508 390]]

```

4.2. Comparison with the first project

The results in this project are pretty much the same, due to the false labeling in the dataset , even after conducting several experiments with different types of RNNs and techniques like skip connections, Gradient clipping and multihead attention.

4.3. Comparison with the second project

The results in this project are pretty much the same (RNNs produce the same result as the classic FFNNs we used in the previous exercise), due to the false labeling in the dataset , even after conducting several experiments with different types of RNNs and techniques like skip connections, Gradient clipping and multihead attention.

5. Bibliography

References

- [1] Rohit Agrawal. Using fine-tuned Gensim Word2Vec Embeddings with Torchtext and Pytorch. <https://rohit-agrawal.medium.com/using-fine-tuned-gensim-word2vec-embeddings-with-torchtext-and-pytorch-17eea2883cd>, 2020.
 - [2] PyTorch contributors. Multihead attention mechanism using PyTorch. <https://pytorch.org/docs/stable/generated/torch.nn.MultiheadAttention.html>, 2023.
 - [3] lucidv01d. Calculate metrics through confusion matrix. <https://stackoverflow.com/questions/31324218/scikit-learn-how-to-obtain-true-positive-true-negative-false-positive-and-fal>, 2017.
 - [4] Neri Van Otten. Word2Vec for text classification. <https://spotintelligence.com/2023/02/15/word2vec-for-text-classification/#Conclusion>, 2023.
 - [5] Milind Soorya. Introduction to Word Frequency in NLP using python. <https://www.milindsoorya.com/blog/introduction-to-word-frequencies-in-nlp#data-processing>, 2021.
- [5] [3] [1] [4] [2]
 [Word frequency with NLTK] [Metric calculation for ROC curve using confusion matrix] [Using fine-tuned Gensim Word2Vec Embeddings with Torchtext and Pytorch] [Word2vec text classification methods] [PyTorch multi head attention documentation]