UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP 3nd Assignment

Student name: *Dimokritos Kolitsos*
sdi: sdi1900085

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

Contents

1. Abstract

The task involves developing a sentiment classifier for tweets related to the Greek general elections. To tackle this task I will be using bidirectional stacked RNNs with LSTM/GRU cells to develop the sentiment classifier. The end goal is for the model to be capable of classifying tweets into three sentiment categories: POSITIVE, NEUTRAL, and NEGATIVE. To achieve this, the process will be to first apply steps like data preprocessing, defining a model architecture, training the model and evaluating its performance but also experimenting with various techniques and methods to improve the model such as hyperparameter. The point is to experiment and improve the model's performance as much as possible.

2. Data processing and analysis

2.1. Pre-processing

Since the dataset is the same one we used in the previous assignments, I know that there aren't any missing (NaN) or odd(for example not a label in the labels column) data in the dataset so there is no need to do something about missing or odd data.

Since I have already implemented several good pre-processing techniques that I'm happy with the results of them for the dataset, I will keep them for this assignment aswell. Some of them include removing special characters, URLs and usernames using regular expressions. This will help the model focus on words that can have an indication of sentiment and context and not on irrelevant words and characters. Before moving on I also converted sentiment labels to numbers which will also help in the training of the machine learning model.

The processed text was then tokenized using the Natural Language Toolkit library. Tokenization is necessary as it helps the model understand the structure of the text and allows for techniques like lowercasing, stop words removal, lemmatization etc. to be applied easily.

All tokens were then converted to lowercase characters to maintain consistency, limit extra characters from needing to be included and understood and prevent possible confusion from the model in general.

I then imported a txt file with Greek stopwords which was created by fellow classmate Giorgos - Marios Patelis which he kindly shared in discord for us to use and was uploaded in kaggle by another fellow classmate. I used the file to remove stopwords from the tweet's text since I saw in the analysis that irrelevant words such as articles (ο, η το) had the highest count in the tweets. This aswell significantly helps the model focus on words that have meaning and can hint sentiment.

Afterwards I also tried applying lemmatization to the text to see if it will improve the specifier's training. I contemplated whether to try stemming, lemmatization or both but since spaCy already has a Greek language model it made my decision easier. Lemmatization involves reducing words to their base or root form, contributing to a more refined and standardized representation of the text, meaning that the the model can focus more on the essence of the words and as with the previous steps, it helps leaving less margin for confusion.

I was not happy with the results and although the lemmatization gave my 1st assignment's logistic regression model a better performance, in the previous one, applying the technique dropped the neural net model's performance significantly overall. Also with spaCy's Greek lemmatization model the processed text had little to no visible changes and the words that were changed, made little to no sense (their format etc). To add to that Giorgos - Marios Patelis shared with us yet another helpful tool, a working Greek Stemmer function, so I decided to use that instead. Although this initially didn't help with the performance of the previous assignment's model, the processed text now looks visibly more consistent and organized and the data analysis shows that, so I decided to keep this step and work with the Neural net model implementation to improve the performance of the model which in the end yielded the best results.

2.2. Vectorization

In addition to the pre-processing phase, it's crucial to highlight use of Word2Vec word embeddings to transform the processed tweet text into continuous vector representations suitable for input into a neural network model.

In the previous assignment I trained my own word2vec model using gensim's Word2Vec function, since this went well I decided to use this again. To do that, I first combined all the preprocessed text from all train,validation and test sets to one to use for the training of my model to have better training results. I then made sure that the input requirements of gensim's Word2Vec model were met by spliting the joined tokens of the prepreсessed text. Finally I trained the model on what semeed to be good parameters, I saved it so I wouldn't need to run all that again if I wanted to reuse it in the same session and finally I used it to convert my dataset's preprocessed texts to word embeddings.

To use the word embeddings to train a rnn model first I need to also convert them into PyTorch tensors and handle their padding to ensure uniform input size.

For this I defined a custom PyTorch dataset class designed to process and prepare the preprocessed data (including tweet's text and sentiments) to be ready to use as input for my model to train. With this calss I can retrieve a sample from the dataset at a specified index, convert the text data into Word2Vec embeddings, format the embeddings as tensors and return them along with the sentiment label (if it's not the test set which has no sentiments).

To prepare the data for training a sentiment classifier model creating data loaders was needed which was done using the CustomDataset class. This step will allow for efficient batching and loading the data during training, validation, and testing phases.

2.3. Analysis

For the analysis of the preprocessed data I implemented a simple word cloud graph to get a rough idea of how the dataset looks and also which are the dominant words in the dataset. Otherwise I already had a rough idea already from printing the dataset each time to check if the preprocessing techniques I implemented were working correctly and also see the difference they made each time.

2.4. Data partitioning for train, test and validation

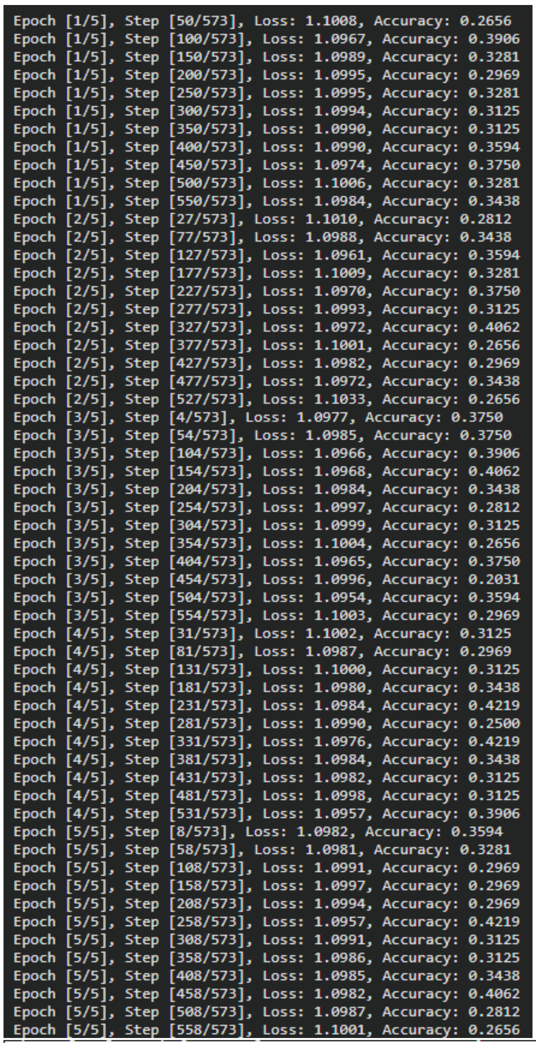
From the previous assignments I found out that repartitioning the dataset and trying out different configurations from the one already given didn't yield any significant differences in the results so I left the dataset split as given from the beginning which is the way I used it before and I'm now accustomed to it aswell.

3. Algorithms and Experiments

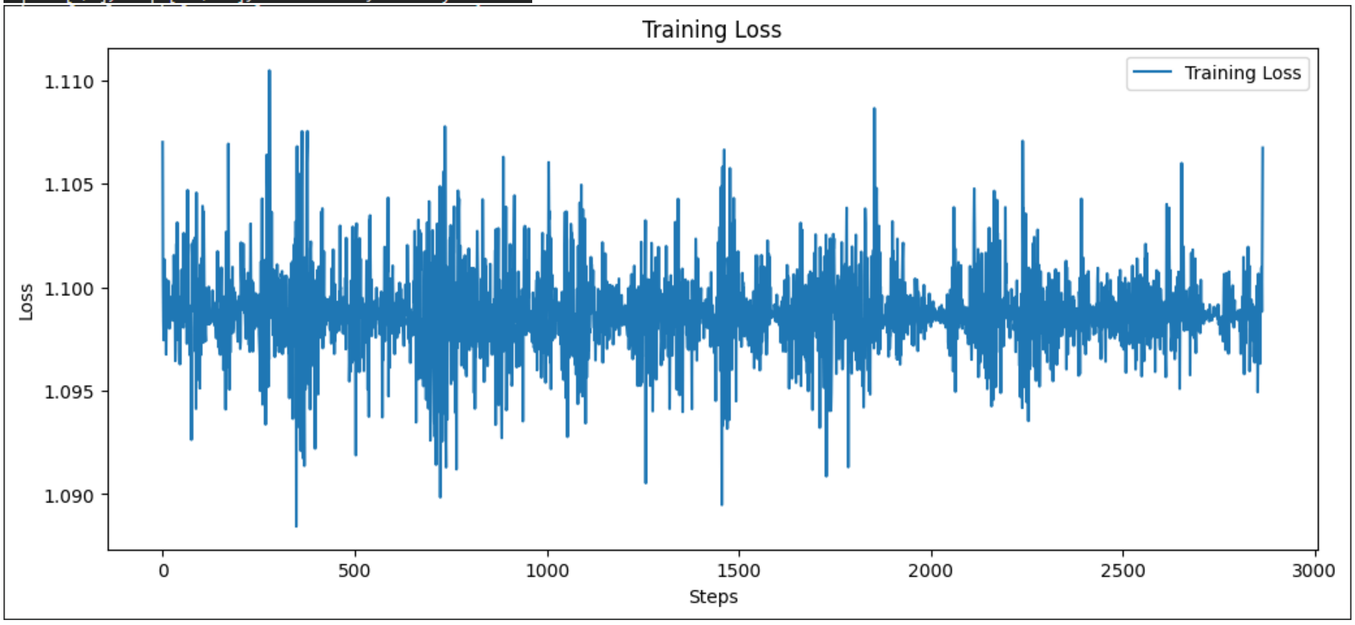
3.1. Experiments

As it was the process with the previous assignments, my first goal was to have a working model that can yield some results before experimenting and trying different setups with the various states of the implementation such as data preprocessing techniques, vectorization, model definition and training along with hyperparameter tuning etc.. In this assignment I thought this would be easier since I had already experimented enough with the particular dataset to have a good base of the first part of the implementation that had to do with processing the dataset and turning it to a state it can be used to train a model, so I could focus more on the second half of the implementation which was the model setup. The problem was that with my word embeddings setup from the previous assignment many things were going wrong until I managed to have the processed data split to tensors and fed into the model correctly so a lot more time than I thought was spent on managing the dataset embeddings and tensors.

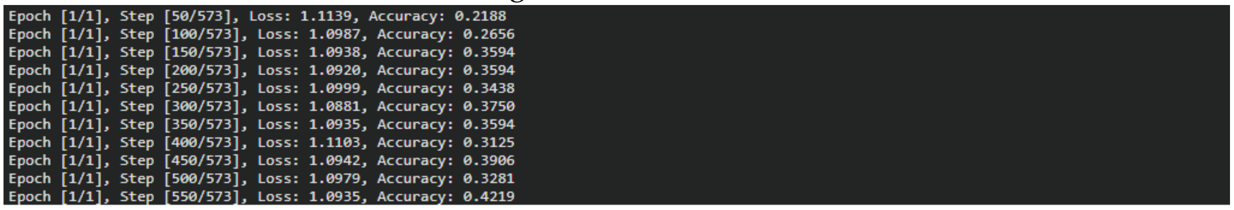
At last I managed to configure a much better way to handle the data using the data class I described above and I was ready to shift my attention to the model implementation. Of course to see that the data can be properly used to train a model, I had already defined a basic model and a basic training loop. I wasn't exactly satisfied with my first training loop and the way I visualized the results though so this step needed some experimenting as well and in the end I ended up using a very similar loop than the one used in the sample notebook provided in the tutorial of the homework. Here are the results I got one of the first times I run it with everything else as basic as possible:



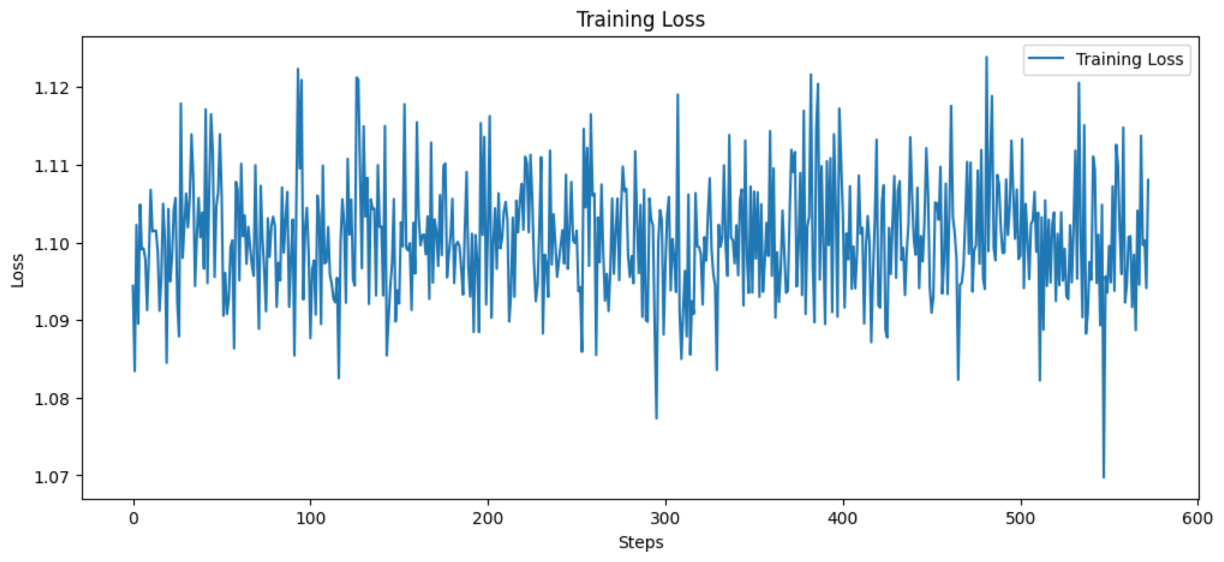
```
Epoch [1/5], Step [50/573], Loss: 1.1008, Accuracy: 0.2656
Epoch [1/5], Step [100/573], Loss: 1.0967, Accuracy: 0.3906
Epoch [1/5], Step [150/573], Loss: 1.0989, Accuracy: 0.3281
Epoch [1/5], Step [200/573], Loss: 1.0995, Accuracy: 0.2969
Epoch [1/5], Step [250/573], Loss: 1.0995, Accuracy: 0.3281
Epoch [1/5], Step [300/573], Loss: 1.0994, Accuracy: 0.3125
Epoch [1/5], Step [350/573], Loss: 1.0990, Accuracy: 0.3125
Epoch [1/5], Step [400/573], Loss: 1.0990, Accuracy: 0.3594
Epoch [1/5], Step [450/573], Loss: 1.0974, Accuracy: 0.3750
Epoch [1/5], Step [500/573], Loss: 1.1006, Accuracy: 0.3281
Epoch [1/5], Step [550/573], Loss: 1.0984, Accuracy: 0.3438
Epoch [2/5], Step [27/573], Loss: 1.1010, Accuracy: 0.2812
Epoch [2/5], Step [77/573], Loss: 1.0988, Accuracy: 0.3438
Epoch [2/5], Step [127/573], Loss: 1.0961, Accuracy: 0.3594
Epoch [2/5], Step [177/573], Loss: 1.1009, Accuracy: 0.3281
Epoch [2/5], Step [227/573], Loss: 1.0970, Accuracy: 0.3750
Epoch [2/5], Step [277/573], Loss: 1.0993, Accuracy: 0.3125
Epoch [2/5], Step [327/573], Loss: 1.0972, Accuracy: 0.4062
Epoch [2/5], Step [377/573], Loss: 1.1001, Accuracy: 0.2656
Epoch [2/5], Step [427/573], Loss: 1.0982, Accuracy: 0.2969
Epoch [2/5], Step [477/573], Loss: 1.0972, Accuracy: 0.3438
Epoch [2/5], Step [527/573], Loss: 1.1033, Accuracy: 0.2656
Epoch [3/5], Step [4/573], Loss: 1.0977, Accuracy: 0.3750
Epoch [3/5], Step [54/573], Loss: 1.0985, Accuracy: 0.3750
Epoch [3/5], Step [104/573], Loss: 1.0966, Accuracy: 0.3906
Epoch [3/5], Step [154/573], Loss: 1.0968, Accuracy: 0.4062
Epoch [3/5], Step [204/573], Loss: 1.0984, Accuracy: 0.3438
Epoch [3/5], Step [254/573], Loss: 1.0997, Accuracy: 0.2812
Epoch [3/5], Step [304/573], Loss: 1.0999, Accuracy: 0.3125
Epoch [3/5], Step [354/573], Loss: 1.1004, Accuracy: 0.2656
Epoch [3/5], Step [404/573], Loss: 1.0965, Accuracy: 0.3750
Epoch [3/5], Step [454/573], Loss: 1.0996, Accuracy: 0.2031
Epoch [3/5], Step [504/573], Loss: 1.0954, Accuracy: 0.3594
Epoch [3/5], Step [554/573], Loss: 1.1003, Accuracy: 0.2969
Epoch [4/5], Step [31/573], Loss: 1.1002, Accuracy: 0.3125
Epoch [4/5], Step [81/573], Loss: 1.0987, Accuracy: 0.2969
Epoch [4/5], Step [131/573], Loss: 1.1000, Accuracy: 0.3125
Epoch [4/5], Step [181/573], Loss: 1.0980, Accuracy: 0.3438
Epoch [4/5], Step [231/573], Loss: 1.0984, Accuracy: 0.4219
Epoch [4/5], Step [281/573], Loss: 1.0990, Accuracy: 0.2500
Epoch [4/5], Step [331/573], Loss: 1.0976, Accuracy: 0.4219
Epoch [4/5], Step [381/573], Loss: 1.0984, Accuracy: 0.3438
Epoch [4/5], Step [431/573], Loss: 1.0982, Accuracy: 0.3125
Epoch [4/5], Step [481/573], Loss: 1.0998, Accuracy: 0.3125
Epoch [4/5], Step [531/573], Loss: 1.0957, Accuracy: 0.3906
Epoch [5/5], Step [8/573], Loss: 1.0982, Accuracy: 0.3594
Epoch [5/5], Step [58/573], Loss: 1.0981, Accuracy: 0.3281
Epoch [5/5], Step [108/573], Loss: 1.0991, Accuracy: 0.2969
Epoch [5/5], Step [158/573], Loss: 1.0997, Accuracy: 0.2969
Epoch [5/5], Step [208/573], Loss: 1.0994, Accuracy: 0.2969
Epoch [5/5], Step [258/573], Loss: 1.0957, Accuracy: 0.4219
Epoch [5/5], Step [308/573], Loss: 1.0991, Accuracy: 0.3125
Epoch [5/5], Step [358/573], Loss: 1.0986, Accuracy: 0.3125
Epoch [5/5], Step [408/573], Loss: 1.0985, Accuracy: 0.3438
Epoch [5/5], Step [458/573], Loss: 1.0982, Accuracy: 0.4062
Epoch [5/5], Step [508/573], Loss: 1.0987, Accuracy: 0.2812
Epoch [5/5], Step [558/573], Loss: 1.1001, Accuracy: 0.2656
```



Looking at these early results we clearly seet that the loss values are fluctuating alot and are generally high. This suggests that the model is having difficulty learning from the training data since high loss values can indicate that the model's predictions are far from the actual labels. The accuracy on the training set also confirms this, it's also fluctuating and even if it's good/better in some cases than the accuracy we achieved in the previous assignments given the state of our dataset, most of the times it still is relatively low. With the state of the dataset I have come to expect the issue of remarkably low accuracy but also the high fluctuation and inconsistency of the results lead me to believe that there's a lot of room for improvement. So I sat down for sometime and played with different model configurations and different hyperparameters manually. Here's a somewhat better result I managed to achieve:

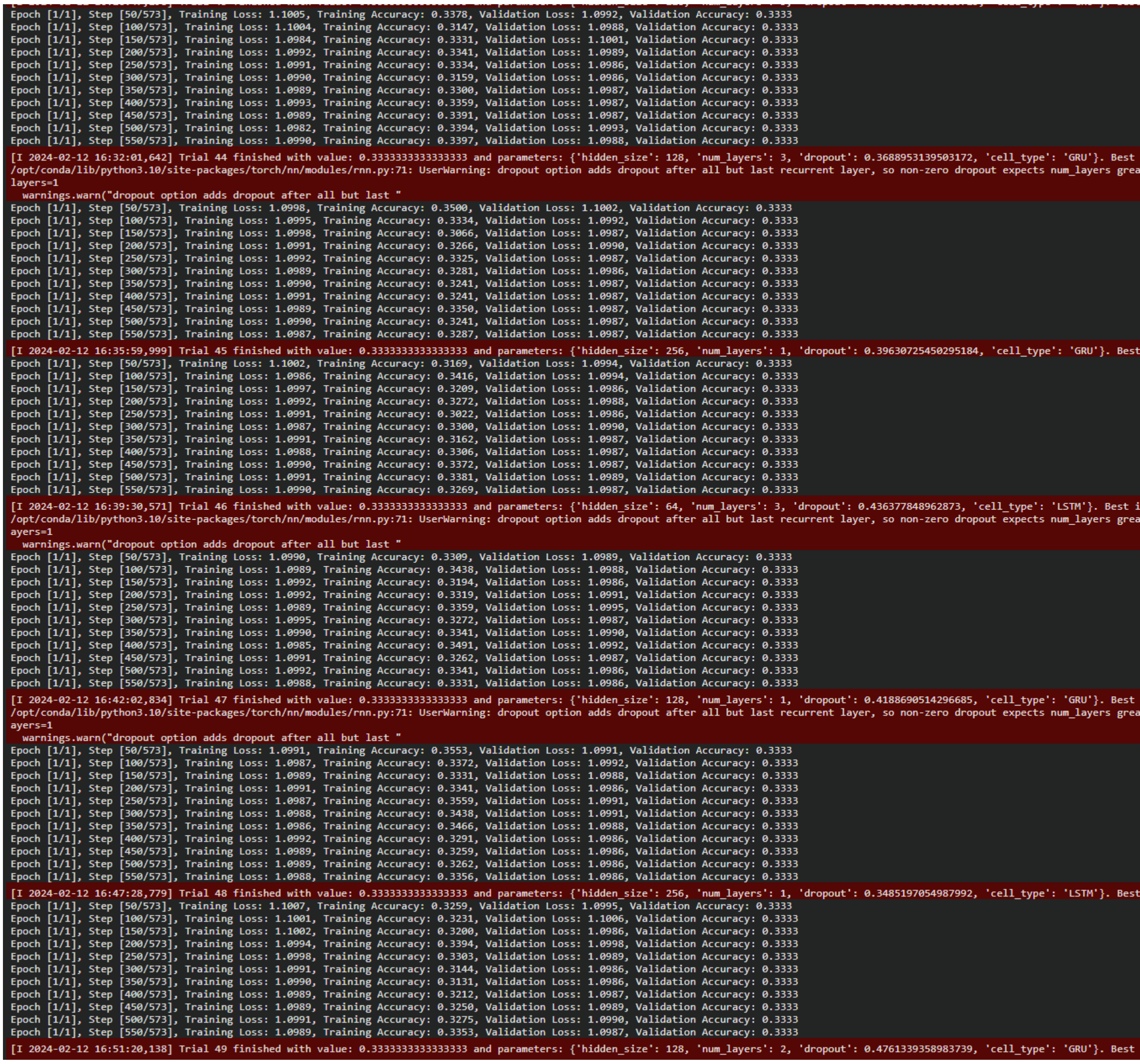


```
Epoch [1/1], Step [50/573], Loss: 1.1139, Accuracy: 0.2188
Epoch [1/1], Step [100/573], Loss: 1.0987, Accuracy: 0.2656
Epoch [1/1], Step [150/573], Loss: 1.0938, Accuracy: 0.3594
Epoch [1/1], Step [200/573], Loss: 1.0920, Accuracy: 0.3594
Epoch [1/1], Step [250/573], Loss: 1.0999, Accuracy: 0.3438
Epoch [1/1], Step [300/573], Loss: 1.0881, Accuracy: 0.3750
Epoch [1/1], Step [350/573], Loss: 1.0935, Accuracy: 0.3594
Epoch [1/1], Step [400/573], Loss: 1.1103, Accuracy: 0.3125
Epoch [1/1], Step [450/573], Loss: 1.0942, Accuracy: 0.3906
Epoch [1/1], Step [500/573], Loss: 1.0979, Accuracy: 0.3281
Epoch [1/1], Step [550/573], Loss: 1.0935, Accuracy: 0.4219
```



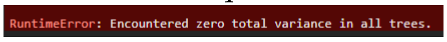
And here's a slightly better one along 5 epoch:
.50.6.5

We see 'kinda' more consistent results at least when we're talking about the accuracy but still the loss is relatively high and fluctuating. Since there are many parameters to play with and tune it was about time I brought forth the "big guns" and started playing with hyperparameter tuning. In the previous assignments I always ended up using grid search to help finetune my model and find the best hyperparameters for it to perform at it's best and since there were usually big boost in the performance of my models, I was excited to see what optuna was capable of. Until now, no matter what changes I did (and trust me I did a lot) to my model or the way I fed my data into it, the results on the validation set where almost constantly 0.3333 (with some 0.3336, 0.3330, 0.3334 etc. outliers). After managing to come up with an implementation of hyperparameter tuning with optuna and getting it to work, I was excited to let it work with my model so I let it do 50 different trials (big mistake...) which took a whole day only to refuse to print out any graphs (such as optimization history and parameter importance) because my model literally kept having the exact same 0.33333 with optuna training (although I had managed to get some slightly (but only just) different results from the validation set.. Here's an example of the results I got from that run:



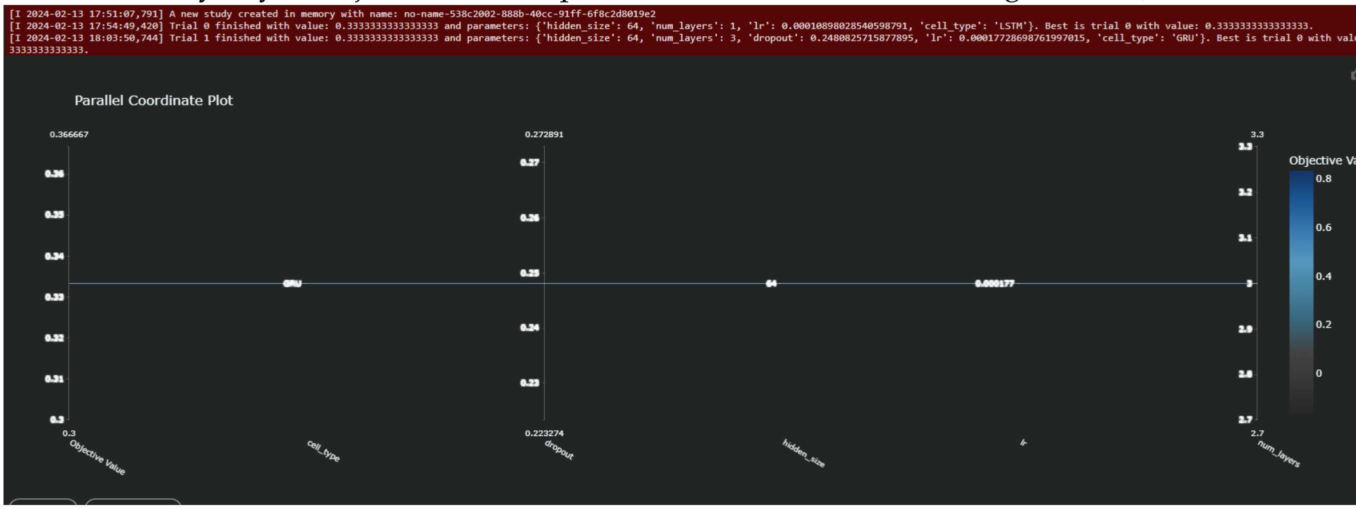
```
Epoch [1/1], Step [50/573], Training Loss: 1.1005, Training Accuracy: 0.3378, Validation Loss: 1.0992, Validation Accuracy: 0.3333
Epoch [1/1], Step [100/573], Training Loss: 1.1004, Training Accuracy: 0.3147, Validation Loss: 1.0988, Validation Accuracy: 0.3333
Epoch [1/1], Step [150/573], Training Loss: 1.0984, Training Accuracy: 0.3331, Validation Loss: 1.1001, Validation Accuracy: 0.3333
Epoch [1/1], Step [200/573], Training Loss: 1.0992, Training Accuracy: 0.3341, Validation Loss: 1.0989, Validation Accuracy: 0.3333
Epoch [1/1], Step [250/573], Training Loss: 1.0991, Training Accuracy: 0.3334, Validation Loss: 1.0986, Validation Accuracy: 0.3333
Epoch [1/1], Step [300/573], Training Loss: 1.0990, Training Accuracy: 0.3159, Validation Loss: 1.0986, Validation Accuracy: 0.3333
Epoch [1/1], Step [350/573], Training Loss: 1.0989, Training Accuracy: 0.3300, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [400/573], Training Loss: 1.0993, Training Accuracy: 0.3359, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [450/573], Training Loss: 1.0989, Training Accuracy: 0.3391, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [500/573], Training Loss: 1.0982, Training Accuracy: 0.3394, Validation Loss: 1.0993, Validation Accuracy: 0.3333
Epoch [1/1], Step [550/573], Training Loss: 1.0990, Training Accuracy: 0.3397, Validation Loss: 1.0988, Validation Accuracy: 0.3333
[I 2024-02-12 16:32:01,642] Trial 44 finished with value: 0.3333333333333333 and parameters: {'hidden_size': 128, 'num_layers': 3, 'dropout': 0.3688953139503172, 'cell_type': 'GRU'}. Best
/opt/conda/lib/python3.10/site-packages/torch/nn/modules/rnn.py:71: UserWarning: dropout option adds dropout after all but last recurrent layer, so non-zero dropout expects num_layers grea
layers=1
  warnings.warn("dropout option adds dropout after all but last "
Epoch [1/1], Step [50/573], Training Loss: 1.0998, Training Accuracy: 0.3500, Validation Loss: 1.1002, Validation Accuracy: 0.3333
Epoch [1/1], Step [100/573], Training Loss: 1.0995, Training Accuracy: 0.3334, Validation Loss: 1.0992, Validation Accuracy: 0.3333
Epoch [1/1], Step [150/573], Training Loss: 1.0998, Training Accuracy: 0.3066, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [200/573], Training Loss: 1.0991, Training Accuracy: 0.3266, Validation Loss: 1.0990, Validation Accuracy: 0.3333
Epoch [1/1], Step [250/573], Training Loss: 1.0992, Training Accuracy: 0.3325, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [300/573], Training Loss: 1.0989, Training Accuracy: 0.3281, Validation Loss: 1.0986, Validation Accuracy: 0.3333
Epoch [1/1], Step [350/573], Training Loss: 1.0990, Training Accuracy: 0.3241, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [400/573], Training Loss: 1.0991, Training Accuracy: 0.3241, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [450/573], Training Loss: 1.0989, Training Accuracy: 0.3350, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [500/573], Training Loss: 1.0990, Training Accuracy: 0.3241, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [550/573], Training Loss: 1.0987, Training Accuracy: 0.3287, Validation Loss: 1.0987, Validation Accuracy: 0.3333
[I 2024-02-12 16:35:59,999] Trial 45 finished with value: 0.3333333333333333 and parameters: {'hidden_size': 256, 'num_layers': 1, 'dropout': 0.39630725450295184, 'cell_type': 'GRU'}. Best
Epoch [1/1], Step [50/573], Training Loss: 1.1002, Training Accuracy: 0.3169, Validation Loss: 1.0994, Validation Accuracy: 0.3333
Epoch [1/1], Step [100/573], Training Loss: 1.0986, Training Accuracy: 0.3416, Validation Loss: 1.0994, Validation Accuracy: 0.3333
Epoch [1/1], Step [150/573], Training Loss: 1.0997, Training Accuracy: 0.3209, Validation Loss: 1.0986, Validation Accuracy: 0.3333
Epoch [1/1], Step [200/573], Training Loss: 1.0992, Training Accuracy: 0.3272, Validation Loss: 1.0988, Validation Accuracy: 0.3333
Epoch [1/1], Step [250/573], Training Loss: 1.0991, Training Accuracy: 0.3022, Validation Loss: 1.0986, Validation Accuracy: 0.3333
Epoch [1/1], Step [300/573], Training Loss: 1.0987, Training Accuracy: 0.3300, Validation Loss: 1.0990, Validation Accuracy: 0.3333
Epoch [1/1], Step [350/573], Training Loss: 1.0991, Training Accuracy: 0.3162, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [400/573], Training Loss: 1.0988, Training Accuracy: 0.3306, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [450/573], Training Loss: 1.0990, Training Accuracy: 0.3372, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [500/573], Training Loss: 1.0991, Training Accuracy: 0.3381, Validation Loss: 1.0989, Validation Accuracy: 0.3333
Epoch [1/1], Step [550/573], Training Loss: 1.0990, Training Accuracy: 0.3269, Validation Loss: 1.0987, Validation Accuracy: 0.3333
[I 2024-02-12 16:39:30,571] Trial 46 finished with value: 0.3333333333333333 and parameters: {'hidden_size': 64, 'num_layers': 3, 'dropout': 0.436377848962873, 'cell_type': 'LSTM'}. Best i
/opt/conda/lib/python3.10/site-packages/torch/nn/modules/rnn.py:71: UserWarning: dropout option adds dropout after all but last recurrent layer, so non-zero dropout expects num_layers grea
ayers=1
  warnings.warn("dropout option adds dropout after all but last "
Epoch [1/1], Step [50/573], Training Loss: 1.0990, Training Accuracy: 0.3309, Validation Loss: 1.0989, Validation Accuracy: 0.3333
Epoch [1/1], Step [100/573], Training Loss: 1.0989, Training Accuracy: 0.3438, Validation Loss: 1.0988, Validation Accuracy: 0.3333
Epoch [1/1], Step [150/573], Training Loss: 1.0992, Training Accuracy: 0.3194, Validation Loss: 1.0986, Validation Accuracy: 0.3333
Epoch [1/1], Step [200/573], Training Loss: 1.0992, Training Accuracy: 0.3319, Validation Loss: 1.0991, Validation Accuracy: 0.3333
Epoch [1/1], Step [250/573], Training Loss: 1.0989, Training Accuracy: 0.3359, Validation Loss: 1.0995, Validation Accuracy: 0.3333
Epoch [1/1], Step [300/573], Training Loss: 1.0995, Training Accuracy: 0.3272, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [350/573], Training Loss: 1.0990, Training Accuracy: 0.3341, Validation Loss: 1.0990, Validation Accuracy: 0.3333
Epoch [1/1], Step [400/573], Training Loss: 1.0985, Training Accuracy: 0.3491, Validation Loss: 1.0992, Validation Accuracy: 0.3333
Epoch [1/1], Step [450/573], Training Loss: 1.0991, Training Accuracy: 0.3262, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [500/573], Training Loss: 1.0992, Training Accuracy: 0.3341, Validation Loss: 1.0986, Validation Accuracy: 0.3333
Epoch [1/1], Step [550/573], Training Loss: 1.0988, Training Accuracy: 0.3331, Validation Loss: 1.0986, Validation Accuracy: 0.3333
[I 2024-02-12 16:42:02,834] Trial 47 finished with value: 0.3333333333333333 and parameters: {'hidden_size': 128, 'num_layers': 1, 'dropout': 0.4188690514296685, 'cell_type': 'GRU'}. Best
/opt/conda/lib/python3.10/site-packages/torch/nn/modules/rnn.py:71: UserWarning: dropout option adds dropout after all but last recurrent layer, so non-zero dropout expects num_layers grea
ayers=1
  warnings.warn("dropout option adds dropout after all but last "
Epoch [1/1], Step [50/573], Training Loss: 1.0991, Training Accuracy: 0.3553, Validation Loss: 1.0991, Validation Accuracy: 0.3333
Epoch [1/1], Step [100/573], Training Loss: 1.0987, Training Accuracy: 0.3372, Validation Loss: 1.0992, Validation Accuracy: 0.3333
Epoch [1/1], Step [150/573], Training Loss: 1.0989, Training Accuracy: 0.3331, Validation Loss: 1.0988, Validation Accuracy: 0.3333
Epoch [1/1], Step [200/573], Training Loss: 1.0991, Training Accuracy: 0.3341, Validation Loss: 1.0986, Validation Accuracy: 0.3333
Epoch [1/1], Step [250/573], Training Loss: 1.0987, Training Accuracy: 0.3559, Validation Loss: 1.0991, Validation Accuracy: 0.3333
Epoch [1/1], Step [300/573], Training Loss: 1.0988, Training Accuracy: 0.3438, Validation Loss: 1.0991, Validation Accuracy: 0.3333
Epoch [1/1], Step [350/573], Training Loss: 1.0986, Training Accuracy: 0.3466, Validation Loss: 1.0988, Validation Accuracy: 0.3333
Epoch [1/1], Step [400/573], Training Loss: 1.0992, Training Accuracy: 0.3291, Validation Loss: 1.0986, Validation Accuracy: 0.3333
Epoch [1/1], Step [450/573], Training Loss: 1.0989, Training Accuracy: 0.3259, Validation Loss: 1.0986, Validation Accuracy: 0.3333
Epoch [1/1], Step [500/573], Training Loss: 1.0989, Training Accuracy: 0.3262, Validation Loss: 1.0986, Validation Accuracy: 0.3333
Epoch [1/1], Step [550/573], Training Loss: 1.0988, Training Accuracy: 0.3356, Validation Loss: 1.0986, Validation Accuracy: 0.3333
[I 2024-02-12 16:47:28,779] Trial 48 finished with value: 0.3333333333333333 and parameters: {'hidden_size': 256, 'num_layers': 1, 'dropout': 0.3485197054987992, 'cell_type': 'LSTM'}. Best
Epoch [1/1], Step [50/573], Training Loss: 1.1007, Training Accuracy: 0.3259, Validation Loss: 1.0995, Validation Accuracy: 0.3333
Epoch [1/1], Step [100/573], Training Loss: 1.1001, Training Accuracy: 0.3231, Validation Loss: 1.1006, Validation Accuracy: 0.3333
Epoch [1/1], Step [150/573], Training Loss: 1.1002, Training Accuracy: 0.3200, Validation Loss: 1.0986, Validation Accuracy: 0.3333
Epoch [1/1], Step [200/573], Training Loss: 1.0994, Training Accuracy: 0.3394, Validation Loss: 1.0998, Validation Accuracy: 0.3333
Epoch [1/1], Step [250/573], Training Loss: 1.0998, Training Accuracy: 0.3303, Validation Loss: 1.0989, Validation Accuracy: 0.3333
Epoch [1/1], Step [300/573], Training Loss: 1.0991, Training Accuracy: 0.3144, Validation Loss: 1.0986, Validation Accuracy: 0.3333
Epoch [1/1], Step [350/573], Training Loss: 1.0990, Training Accuracy: 0.3131, Validation Loss: 1.0986, Validation Accuracy: 0.3333
Epoch [1/1], Step [400/573], Training Loss: 1.0989, Training Accuracy: 0.3212, Validation Loss: 1.0987, Validation Accuracy: 0.3333
Epoch [1/1], Step [450/573], Training Loss: 1.0989, Training Accuracy: 0.3250, Validation Loss: 1.0989, Validation Accuracy: 0.3333
Epoch [1/1], Step [500/573], Training Loss: 1.0991, Training Accuracy: 0.3275, Validation Loss: 1.0990, Validation Accuracy: 0.3333
Epoch [1/1], Step [550/573], Training Loss: 1.0989, Training Accuracy: 0.3353, Validation Loss: 1.0987, Validation Accuracy: 0.3333
[I 2024-02-12 16:51:20,138] Trial 49 finished with value: 0.3333333333333333 and parameters: {'hidden_size': 128, 'num_layers': 2, 'dropout': 0.4761339358983739, 'cell_type': 'GRU'}. Best
```

Before instead of at least some graphs to indicate me to some direction of what hyperparameters suit my model even in it's current state and also giving me something to show, it ended up with this..



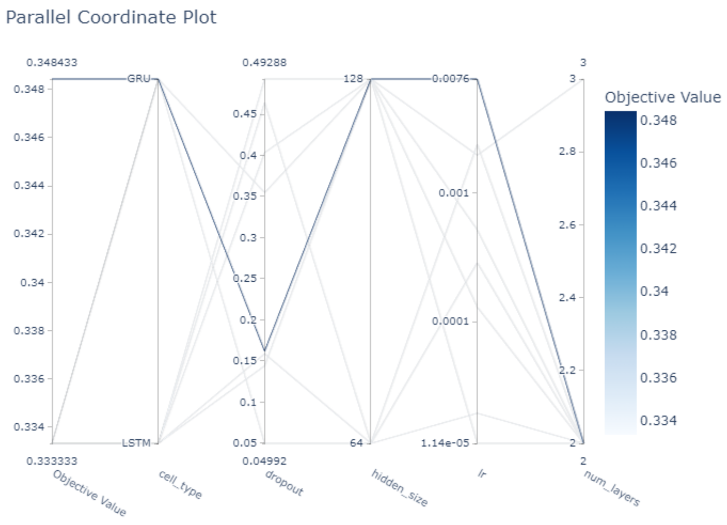
```
RuntimeError: Encountered zero total variance in all trees.
```

After playing around a lot with all different sections of my implementation, trying to tweek how I define and use my dataset class and making sure the margin for error from this aspect is as little as possible (there were sometimes that I had some text sections turn out to be only zeroes but this was happening with my previous assignments and the results were fine), trying to play with the way I define the rnn model and also with the hyperparameters and also playing around and trying many different configurations of code for training and validating the model, I really could not shake that constant 0.333333 accuracy especially when I was trying to apply hyperparameter tuning with optuna.. I even tried running it again with multiple trials to let it maybe find some different configuration which would work better with my model but to no avail.. the results were consistent (too consistent..) Here's an example of a small run with optuna (I forgot to keep a screenshot from a run with more steps but the results were identical anyway.. so I just run a 2 step one so I at least had something to show)



```
[I 2024-02-13 17:51:07,791] A new study created in memory with name: no-name-538c2002-888b-40cc-91ff-6f8c2d8019e2
[I 2024-02-13 17:54:49,420] Trial 0 finished with value: 0.3333333333333333 and parameters: {'hidden_size': 64, 'num_layers': 1, 'lr': 0.00010898028540598791, 'cell_type': 'LSTM'}. Best is trial 0 with value: 0.3333333333333333.
[I 2024-02-13 18:03:50,744] Trial 1 finished with value: 0.3333333333333333 and parameters: {'hidden_size': 64, 'num_layers': 3, 'dropout': 0.2480825715877895, 'lr': 0.00017728698761997015, 'cell_type': 'GRU'}. Best is trial 0 with val
3333333333333.
```

In essence, the problem is that the model only predicts one sentiment which is something that from what I've heard is a common problem which multiple other classmates have faced during this and also the previous assignment (which was similar in terms of that we had to train neural net models). After trying out multiple different ways to implement my model and also given the unpredictable and unstable nature state of the dataset which could possibly play a role in this, I was ready to give up on trying to achieve different results and admit defeat. Then after trying a last configuration and letting optuna take it's time to get a bit more that 2 trials (after not getting the desireble results I wanted and wasting a full day that first time with the 50 trials I only run about 2-5 each time testing different configurations which admittedly also did not help) and with some different hyperparameters the model started achieving different results than constant 0.333333 (admitedly though still not by much going up to 0.348 but that can be called significant given the nature of our dataset) Here's optuna's results:



3.2. Hyper-parameter tuning

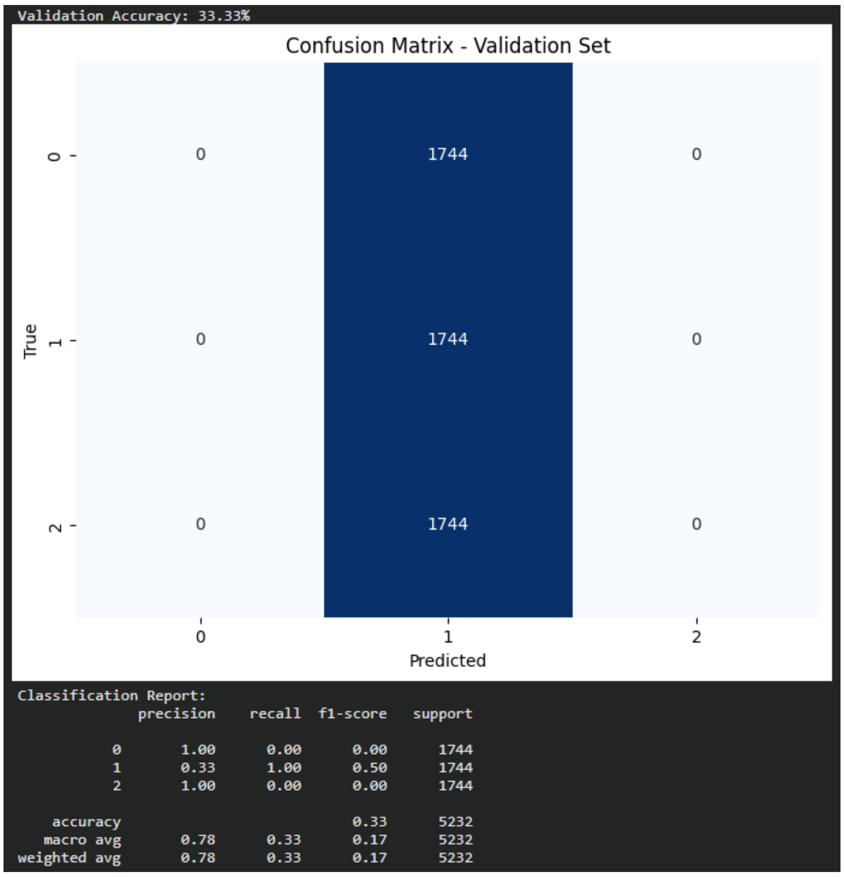
In an attempt to optimize the performance of the rnn sentiment classifier model (and also to find hyperparameters that might fix the fact that the model only predicts one sentiment) I implemented hyperparameter tuning using optuna to search for the most effective combination of hyperparameters.

Optuna, a hyperparameter optimization framework, was used to search for the optimal values of hyperparameters such as hidden size, number of layers, dropout rate, learning rate, and cell type (LSTM or GRU) for our sentiment analysis model. It employs an optimization algorithm to intelligently sample and evaluate different hyperparameter configurations, iteratively refining its search based on past evaluations to converge towards the optimal solution.

Once the hyperparameter tuning process is completed, I visualized the optimization history using plots such as the optimization history plot and the parallel coordinate plot. These visualizations would provide really useful insight into how the performance of the model changed over the course of the optimization process.

Overall, hyperparameter tuning is an extremely important technique that *usually* plays a key role to optimize a sentiment classifier model and improve it's performance.

3.3. Evaluation



```
Validation Accuracy: 33.33%
```

	0	1	2
0	0	1744	0
1	0	1744	0
2	0	1744	0

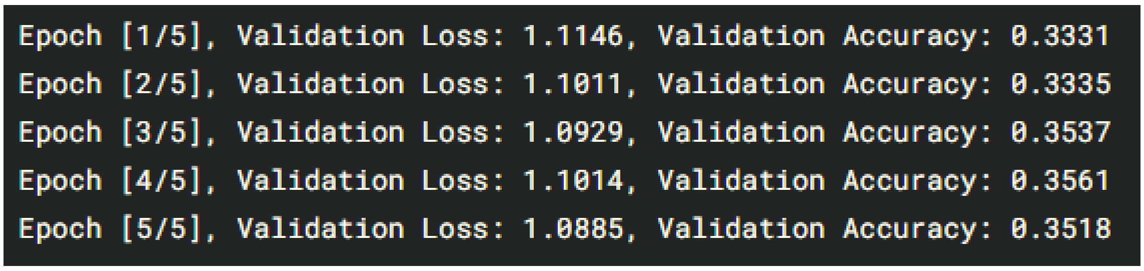
```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.00      0.00      1744
           1       0.33      1.00      0.50      1744
           2       1.00      0.00      0.00      1744

    accuracy                           0.33      5232
   macro avg       0.78      0.33      0.17      5232
weighted avg       0.78      0.33      0.17      5232
```

More is showcased in section 3.1

4. Results and Overall Analysis

4.1. Results Analysis

This section was pretty much covered by section 3.1



```
Epoch [1/5], Validation Loss: 1.1146, Validation Accuracy: 0.3331
Epoch [2/5], Validation Loss: 1.1011, Validation Accuracy: 0.3335
Epoch [3/5], Validation Loss: 1.0929, Validation Accuracy: 0.3537
Epoch [4/5], Validation Loss: 1.1014, Validation Accuracy: 0.3561
Epoch [5/5], Validation Loss: 1.0885, Validation Accuracy: 0.3518
```

4.1.1. Best trial.

4.2. Comparison with the first and second projects

In comparison to the first 2 assignments I can say that the process was at least for me a bit more complicated and is definitely no justification for my own incompetence that I did not manage better results, but I would still like to mention that the state of the dataset has not helped at all with the development of the sentiment classifier models in these assignments since most of the time it was far too difficult to experiment and test/try different implementations and techniques when the model's performance and results wouldn't necessarily reflect any improvement or change.

In the end in terms of results they where more or less the same or very similar across all implementations and for me at least a simpler logistic regression model would have the same or even slightly better performance (but basically the same more or less) with the neural network models that are in theory far more sophisticated.

5. Bibliography

References

[1] Data Magic (by Sunny Kusawa). How to use kaggle notebook | kaggle tutorial | machine learning | data magic.

[2] codebasics. What is word2vec? a simple explanation | deep learning tutorial 41 (tensorflow, keras & python).

[3] codebasics. Word2vec part 2 | implement word2vec in gensim | | deep learning tutorial 42 with python.

[4] DigitalSreeni. 189 - hyperparameter tuning for dropout, neurons, batch size, epochs, and weight constraint.

[5] Pradip Nichite. Build text classification model using word2vec | gensim | nlp | python | code.

[6] utsav aggarwal. Text preprocessing | tokenization | cleaning | stemming | stopwords | lemmatization.

[1] How to use Kaggle Notebook | Kaggle Tutorial | Machine Learning | Data Magic
[6] Text Preprocessing | tokenization | cleaning | stemming | stopwords | lemmatization
[2] What is Word2Vec? A Simple Explanation | Deep Learning Tutorial 41 (Tensorflow, Keras & Python)

[3] Word2Vec Part 2 | Implement word2vec in gensim | | Deep Learning Tutorial 42 with Python

[5] Build Text Classification Model using Word2Vec | Gensim | NLP | Python | Code

[4] 189 - Hyperparameter tuning for dropout, neurons, batch size, epochs, and weight constraint

StackOverflow What is the best way to remove accents (normalize) in a Python unicode string?

Greek Stemmer

FastText pretrainned models

NLPL word embeddings repository