UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP using Neural Networks

Student name: *Panagiotis Zazos*
sdi: 7115112300009

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

Contents

1. Abstract

The task at hand is to create a sentiment analysis model for Twitter data about Greek Elections in the Greek language. Sentiment analysis involves classifying text into one of three categories: Negative, Neutral, Positive. The objective here is to develop a model which accurately predicts the sentiment of given tweets. To achieve this, a combination of ML and NLP techniques were used.

2. Data processing, Analysis and Preparation

2.1. Pre-processing

The data underwent a comprehensive cleaning and pre-processing to ensure the model's training on high-quality and meaningful text. Through the implementation of the following data cleaning techniques, the text has reduced the irrelevant information, noise and inconsistencies. The following data cleaning techniques were applied:

- **Text Normalization:** Tweets were preprocessed to remove URLs, hashtags, mentions, punctuations and special characters. More importantly, hashtags were split into words according to the underscore character, i.e. "ΝΔ_πολιτικό_σχόλιο" into "ΝΔ πολιτικο σχολιο".
- **Lowercasing**: All text was converted to lowercase to guarantee case insensitivity, ensuring that words were recognizable regardless of their capitalization. This step eliminates the duplication of features as well.
- **Accent Stripping:** Accents from words were stripped to maintain consistency and reduce vocabulary size.
- **Stopwords**: In the development of the sentiment analysis model utilizing Recurrent Neural Networks (RNNs), a decisio to retain stopwords within the dataset was made, diverging from the preprocessing approach taken in the second assignment. By maintaining stopwords, the natural flow of the text is being preserved, enabling the RNN to model the language more effectively.

2.2. Analysis

In addition to the initial pre-processing steps, specific adjustments were made to align the corpus with the characteristics of the pre-trained Word2Vec model. **Figure 1** shows the actual coverage of the pre-trained model over the Twitter data about Greek Elections in the Greek language.

Consistency with Pre-Trained Model Vocabulary: The pre-trained model's vocabulary does not employ stemming, therefore in order to maintain consistency, stemming was not applied to the corpus. This ensures that the word embeddings generated by the pre-trained model accurately represent the words in the tweets. The same applies to lowercasing the corpus in order to match the requirements of the pre-trained model, as capitalized words were out of the vocabulary.

Accent Removal: Despite the pre-trained model's vocabulary incorporating accented words, the decision to strip accents from the tweets were driven by the need to reduce the overall vocabulary size, which can enhance model's performance. Stripping accents can aid the model in generalizing better to new data as well. Lastly, its computational efficient to simplify the text overall.

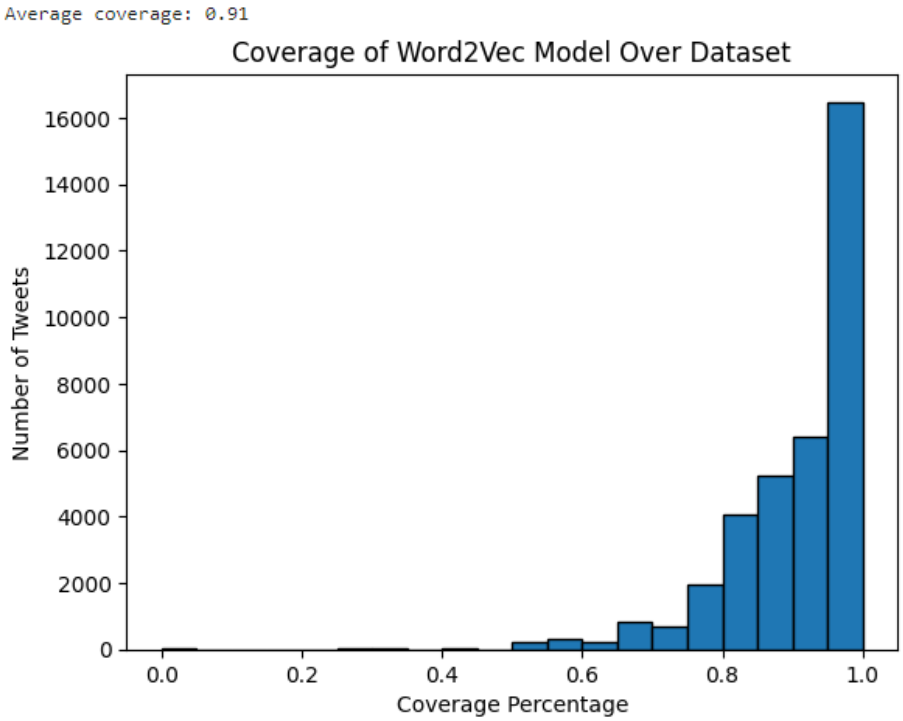

Figure 1: Coverage of word2vec_greek_model Over Dataset

2.3. Data Preparation

2.3.1. For General Neural Networks.

Conversion to embeddings: The processed tweets were tokenized and converted to embeddings using the pre-trained Greek Word2Vec model to capture the semantic information and context of the dataset.

Label Encoding: Scikit-learn's LabelEncoder is used to convert categorical sentiment labels (Positive-Neutral-Negative) into numerical form. PyTorch tensors of type torch.long, regarding Sentiment, are created by converting the encoded labels of each dataset. This datatype is selected as it is suitable for classification labels that are used in conjunction with CrossEntropyLoss loss function.

Conversion of Embeddings to PyTorch Tensors: Lastly, the embeddings are converted to PyTorch tensors, which are required for the model's training. The conversion is being achieved by stacking embeddings lists into tensors of type torch.float, which is typical for floating-point numerical computations in the neural network.

2.3.2. For RNNs.

In the context of RNNs, the sequential nature and the length of input data are critical factors that influence the model's ability to learn from textual content. Therefore, ensuring that each input sequence is represented in a uniform format is crucial, despite the tweet-length. This uniformity is achieved through the conversion of tweets to **padded embeddings**, a process which transform sequences (tweets) into fixed size inputs that the RNN can process in batches.

Embedding: Similar to **General Neural Networks** conversion to embeddings, the tweets are transformed into embeddings using the pre-trained Greek Word2Vec model and the labels are being encoded by the Label Encoder for the categorical sentiment labels.

Padding: Since tweets vary in length, padding ensures that each sequence/tweet is of equal lenght. This is accomplished by appending zero values to the end of shorter sequences. Zero values do not carry any semnatic weight, thereby minimizing their impact on th emodel's ability to learn meaningful patterns from the data.

2.3.3. For BERT Transformers.

In the application of BERT Transformers for sentiment analysis, the preparation of input data is most important to harness the model's full potential. Unlike RNNs, where sequence length uniformity is maintained through padding, BERT requires inputs to keep to its pre-defined structure, reflecting its unique approach to handling context and token relationships.

Tokenization and Encoding: BERT relies on a specialized tokenization process that breaks down text into tokens understood by the model, including the use of WordPiece tokenization. Each tweet is tokenized, and special tokens such as [CLS] for the beginning of a sentence and [SEP] for separation are appended. This process transforms the variable-length tweets into a series of tokens that BERT can interpret.

Input IDs: These are the numerical representations of tokens. Each token (word or subword) is converted into an integer ID using the tokenizer's vocabulary. The tokenizer splits the input text into tokens that are available in its vocabulary. For example, we have the sentence "Hello World". If "Hello" and "World" are both in tokenizer's vocabulary with IDs 12345 and 54321, then inputs_ids: [12345, 54321]

Attention Masks: Given BERT's attention mechanism, it's crucial to differentiate meaningful tokens from padding when processing variable-length tweets. This is where attention masks come into play, allowing the model to focus on the relevant parts of the input data. Each token receives a mask value of 1, while padding tokens are marked with a 0, ensuring the model's attention is directed appropriately.

For example, if the input_ids for "Hello World" are [12345, 54321] and the maximum length of the padding is set to 4, the sequence might be padded to [12345, 54321, 0, 0]. Correspondingly, the attention_masks would be [1, 1, 0, 0]

Fixed Sequence Length (padding): Similar to the padding approach in RNNs, BERT inputs must be standardized to a fixed length for batch processing. As stated previously, this is achieved by truncating longer sequences and padding shorter ones with zero values, with attention masks ensuring that these paddings do not affect the model's performance.

3. Model Architecture and Training

3.1. General Neural Network

3.1.1. Neural Network Design.

Layer Configuration: The model consists of three linear layers. The first two are following by Batch Normalization and Dropout and the third is the output layer.
Batch Normalization is applied after the first and second linear layers to normalize the output of the previous layer, providing improvement to the stability in model's training, as it makes the network less sensitive to the initial weights.
Initially, dropout has been set to 0.5, but after some experimentation, the dropout rate of 0.3 served a better accuracy throughout the epochs. Dropout randomly deactivates a fraction of neurons during training, after the first and second linear layers, which helps in preventing overfitting and basically ensures that the network does not rely on any specific set of neurons.

Activation Functions: ReLU (Rectified Linear Unit) is used after the first and the second linear layers. This activation function offers non-linearity, allowing the model to learn more complex patterns.

Output Layer: The final layer maps the features to the number of sentiment classes. The reason behind the absence of an activation function is that the raw outputs are used in the CrossEntropyLoss function during training.

3.1.2. Model Parameter Selection and Evaluation.

The insights gained from a grid search are instrumental in fine-tuning the model's parameters.

Grid Search for Optimal Parameters: In the pursuit of the optimal hyperparameters for the sentiment analysis model, a grid search was conducted. This approach involved experimenting with commonly used combinations of learning rates, batch sizes and number of epochs:

- **Learning Rates:** 0.01, 0.001 and 0.0001
- **Batch Sizes:** 32, 68 and 128
- **Learning Rates:** 50 and 100

Observations and Decisions: The combination yielding the highest average validation accuracy (approx 0.39) was found with a learning rate of 0.0001, batch size 64 and 100 epochs. However, it was observed that the model's accuracy plateaued fairly quickly in the training phase, in the range of 0.39 to 0.40.
In contrast, a different parameter set comprising a learning rate of 0.001, batch size 128 and 100 epochs, while achieving a similar average validation accuracy (approx 0.387), demonstrated higher ceiling and a way better average accuracy during training, reaching up to 0.47 and an average of 0.45.

The decision to select the latter parameter set (learning rate = 0.001, batch size = 128, epochs = 100) was driven by its ability to achieve higher training accuracy. This indicates a better learning capacity of the model, even though the validation accuracies were similar to both parameter sets. However, one can also address the danger of overfitting on the training corpus. That's a risk that this model's implementation has taken into account and built upon.
StepLR Scheduler was implemented to dynamically adjust the learning rate by 10% every 20 epochs, offering a more static and naive way to interpret the scheduler. This gradual reduction allows the model to make larger updates to the weights initially and finer adjustments in later stages. Note that there are better and more dynamic ways to compute the step size (frequency) and the gamma (the percentage of decrease) of the scheduler.
Lastly, **Adam** optimizer was chosen for its adaptability and efficiency, as its suitable for common problems in NLP with sparse gradients and noisy data. In contrast to SGD optimizer, the Adam optimizer is less sensitive to the initial learning rate and other hyperparameters. For this, the grid search that was cited earlier did not include many different parameters to tune, only a few of learning rates and batch sizes.

3.2. RNN

3.2.1. Neural Network Design.

Cell types and Bidirectionality: The model can use different types of RNN cells (**RNN**, **LSTM**, **GRU**), which are specified during initialization. Bidirectionality in RNNs is also supported, which processes context from both forward and backward directions. Often this leads to better performance as capturing the context from both sides of the tweet is useful in sentiment analysis problems as this.

Dropout and Batch First: Dropout is incorporated to prevent overfitting. Since this technique is only effective when multiple layers are used, as RNN modules apply dropout only between layers, a configuration has been established to handle this as well. The 'batch_first=True' argument in the RNN initialization ensures that the input and output tensors are expected to have the batch size as the first dimension, e.g. (batch, time_step, input_size).

Output Layer and Final States: The final output layer is a fully connected linear layer that maps RNN output to the sentiment classes. Finally, the model correctly handles the final hidden states, concatenating the last forward and backward states if RNN is bidirectional, or taking the last state if it's not.

3.2.2. Model Parameter Selection and Evaluation (Optuna).

Optuna Usage: Employing a hyperparameter otpimization framework, like Optuna, was critical in the pursuit of an optimal set of hyperparameters, since selecting the most efficient configuration of the model is tricky. Optuna facilitates an efficient search for the best hyperparameter values by automating multiple trials and evaluating the performance of each trial's configurations and provides several visualization options that help understand and analyse the results of the optimization process.

During the optimization process, each trial proposed a unique combination of hyperparameters, including:

- The learning rate (lr) range (0.00001, 0.1)
- The size of the hidden layer (hidden_size) [64, 128, 256]
- The dropout rate (dropout) range (0.1, 0.5)
- The RNN cell types (cell_type) ['LSTM', 'GRU']
- The number of layers (num_layers) range (1, 3)

Trials: Each trial's model was trained for 10 epochs. After each epoch, the model's performance was assessed on both the training and validation datasets, providing insights into the model's learning progression and generalization capabilities. As commented in the code for reference, the average validation accuracy across epochs served as the objective function for Optuna's optimization, ensuring that the search was focused on enhancing the model's performance on unseen data. This extensive training was expedited by using the computational prowess of an NVIDIA RTX 3060 GPU, which significantly reduces the training time per trial. After 30 trials, each representing a different model configuration, the study concludes with a selection of the best performing hyperparameters.

3.3. BERT Transformers

3.3.1. Neural Network Design.

Pre-trained Model Utilization: The core of the architecture leverages the pre-trained BERT models, namely GreekBERT and DistilGREEK-BERT from Hugging Face Transformers library, which are adaptations of BERT pre-trained specifically for the Greek language. These models encapsulate a deep understanding of Greek syntax and semantics, making them highly suitable for sentiment analysis on Greek Twitter data related to general elections.

Pre-trained Architecture: The pre-trained models come with a specific architecture that includes layers for handling input tokenization, attention mechanisms, and output classification. This architecture also includes dropout layers and other regularization techniques as part of its design to prevent overfitting.

Fine-Tuning: Generally, fine-tuning involves adapting these pre-trained models to the specific task (sentiment analysis) of one's project by training on the dataset. This usually involves adding a classification layer on top of the pre-trained model, but luckily, Hugging Face's model classes like "**AutoModelForSequenceClassification**" handle this for us.

An Important Note: The refinement of the model parameters was conducted utilizing the Optuna framework, specifically applied to the GreekBERT. Concurrently,

DistilGREEK-BERT, a variant of GreekBERT, was employed to generate outcomes utilizing the hyperparameters derived from the primary GreekBERT optimization process. This decision was informed by real-life constraints on available time for development. Under different circumstances, a comprehensive and distinct optimization for DistilGREEK-BERT would be pursued to thoroughly investigate and leverage its unique modeling characteristics.

Hyperparameter Tuning: While the inner workings of the model, like dropout rates within BERT layers are preset, the optimization occurred by experimenting with hyperparameters like the learning rate, batch size and # training epochs using the same hyperparameter otpimization framework for the entirety of the assignment, Optuna.

Trials: Each trial was trained for either 2 or 4 epochs, as these values were standardized by BERT transformers. As with RNN's evaluation process, the average validation accuracy across epochs served as the objective function for Optuna. Likewise, the training was expedited by using the same GPU, the NVIDIA RTX 3060, which run for 10 trials.
During the optimization process, each trial proposed a unique combination of hyperparameters, including:

- The learning rate (lr) range (0.00001, 0.00005)
- The batch size (batch_size) [16, 32, 64]
- The number of epochs (num_epochs) [2, 4]

The number of epochs, as well as the range of the learning rates (log-uniform distribution), are typical for fine-tuning BERT models and are recommended by the original BERT paper, as they have been found to work well across variety of tasks.

4. Results and Overall Analysis

4.1. General Neural Network

4.1.1. Results.

Figure 2: Training-Validation Loss Over Time

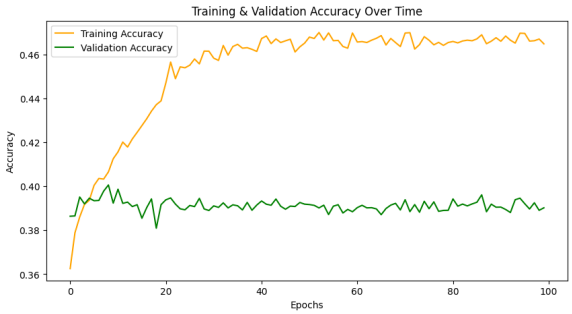

Figure 3: Training-Validation Accuracy Over Time

Training-Validation Loss: While the training loss curve starts quite high, it sees a significant drop in the initial epochs, indicating that the model is learning from the data quite effectively. As the epochs progress, the training loss continues to decrease, until it reaches a point of tiny deviations.
Validation loss on the other hand, does not decrease. It fluctuates around a certain value without a clear downward trend. Therefore, the model is not improving its performance on the validation set.
Training-Validation Accuracy: Training accuracy shows a substantial increase initially and then continues to rise at a slower pace. Thus the model's predictions are becoming more accurate as training proceeds.
On the other hand, validation accuracy quickly plateaus and fluctuates within a narrow boundary. This behavior indicates that while the model is becoming more proficient at classifying the training data, its ability to generalize to unseen tweets is not significantly improving.

4.1.2. Confusion Matrices.

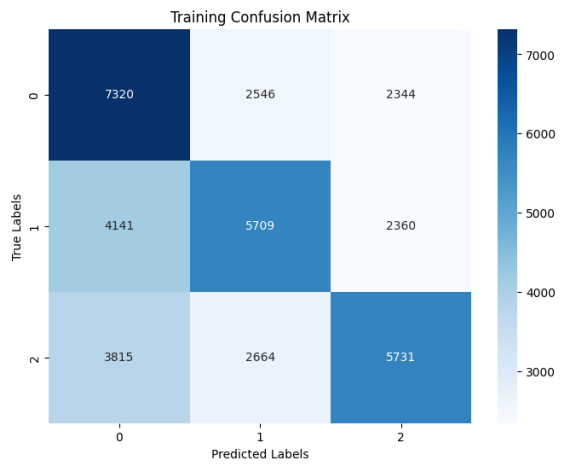

Figure 4: Training Confusion Matrix

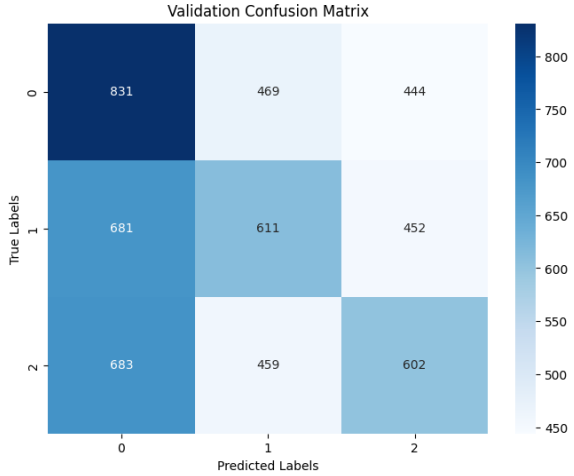

Figure 5: Validation Confusion Matrix

A confusion matrix is a table used to describe the performance of a classification model on a set of data for which the true values are known. Let's preface with the encoding of the categories:

- **0: Negative**
- **1: Neutral**
- **2: Positive**

Diagonal Values (True Positives) for training dataset Figure 4:

- For Negatives tweets, the model correctly predicted 7320 instances.
- For Neutral tweets, the model correctly predicted 5709 instances.

- For Positive tweets, the model correctly predicted 5731 instances.

Off-diagonal Values (Misclassifications):

- Negatives were incorrectly predicted as Neutrals 2546 times, and as Positives 2344 times.
- Neutrals were incorrectly predicted as Negatives 4141 times, and as Positives 2360 times.
- Positives were incorrectly predicted as Negatives 3815 times, and as Neutrals 2664 times.

Diagonal Values (True Positives) for validation dataset Figure 5:

- For Negatives tweets, the model correctly predicted 831 instances.
- For Neutral tweets, the model correctly predicted 611 instances.
- For Positive tweets, the model correctly predicted 602 instances.

Off-diagonal Values (Misclassifications):

- Negatives were incorrectly predicted as Neutrals 469 times, and as Positives 444 times.
- Neutrals were incorrectly predicted as Negatives 681 times, and as Positives 452 times.
- Positives were incorrectly predicted as Negatives 683 times, and as Neutrals 459 times.

The issue of overfitting is can be seen using the matrices as well. This is indicated by the higher numbers on the diagonals of the training confusion matrix compared to those of the validation confusion matrix. There is also the struggle to distinguish features that separate Neutral (1) tweets from Negative (0) and Positive (2). Lastly, the model appears to be biased towards Negative tweets on both training and validation, possibly because Negative tweets have more distinct features and there are more instances of these in the dataset that is provided.

4.1.3. Possible Improvements. Since the model's accuracy does not improve on par with training, it may be useful to experiment with different hyperparameters, model architectures, maybe a different strategy on layers management or incorporation of regularization techniques. Even a more advanced learning rate scheduler might yield better generalization. The goal here is to achieve better generalization
One more way to improve the model is to adapt a data augmentation technique. It is known that the dataset that has been provided is problematic, due to too many incorrect labels on both training and validation datasets. Data augmentation or the addition of more varied and more accurate tweets with their proper sentiment values could potentially improve the model's generalization capabilities.

4.1.4. Best Scores.

Loss: 1.026, Accuracy: 0.466, Classification Report:

	precision	recall	f1-score	support
0	0.45	0.50	0.48	12210
1	0.47	0.42	0.45	12210
2	0.48	0.47	0.47	12210
accuracy			0.47	36630
macro avg	0.47	0.47	0.46	36630
weighted avg	0.47	0.47	0.46	36630

Figure 6: Training Classification Report

Loss: 1.085, Accuracy: 0.391, Classification Report:

	precision	recall	f1-score	support
0	0.38	0.48	0.42	1744
1	0.40	0.35	0.37	1744
2	0.40	0.35	0.37	1744
accuracy			0.39	5232
macro avg	0.39	0.39	0.39	5232
weighted avg	0.39	0.39	0.39	5232

Figure 7: Validation Classification Report

4.2. RNN

4.2.1. Results.

Figure 8: Training-Validation Loss Over Time

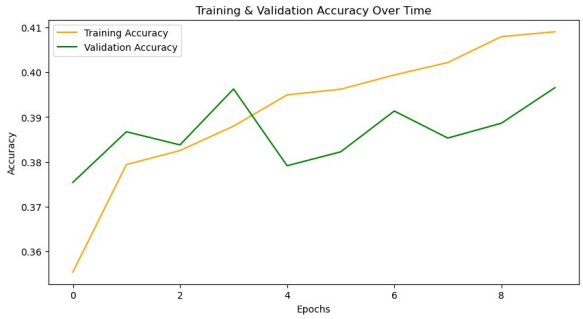

Figure 9: Training-Validation Accuracy Over Time

Training-Validation Loss: The loss curves for both training and validation sets exhibit a steady decline, with the training loss decreasing more smoothly.
Training-Validation Accuracy: The curves of the training and validation accuracy over time indicate a consistent but gradual improvement in training accuracy, but less so in validation accuracy, where the variability is greater and the consistency lower.
While there might be room for further model tuning, this is the best achievable approach given this dataset.

4.2.2. Optuna Visualizations.

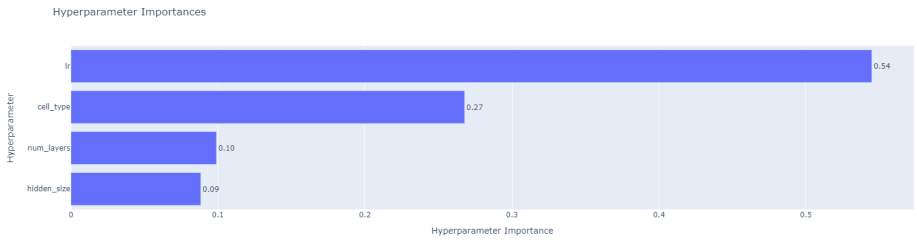

Figure 10: RNN Hyperparameter Importance by Optuna

The hyperparameter importance visualization provided by Optuna reveals that the learning rate (lr) is by far the most influential factor affecting model performance. This highlights the critical role that the learning rate plays in the training of RNNs, thus careful selection of this parameter must be considered. However, which cell type was going to be used, either GRU or LSTM, is definitely worth the consideration.

4.2.3. Best Scores.

```
Best trial:
  Value: 0.39273736971180606
  Params:
    lr: 0.0005180452353320056
    hidden_size: 64
    num_layers: 2
    dropout: 0.23928385238272737
    cell_type: LSTM
```

Figure 11: Best Validation scores achieved with Optuna

The best trial achieved a validation accuracy of approximately 0.395 and the optimal hyperparameters (approx values) identified through this trial are as follows:

- Learning Rate (lr): 0.0005
- Hidden Size (hidden_size): 64
- Number of Layers (num_layers): 2
- Dropout (dropout): 0.24
- Cell Type (cell_type): LSTM

```
Epoch 1 | Training Loss: 1.0963, Accuracy: 0.3558
Epoch 2 | Training Loss: 1.0904, Accuracy: 0.3757
Epoch 3 | Training Loss: 1.0853, Accuracy: 0.3818
Epoch 4 | Training Loss: 1.0805, Accuracy: 0.3884
Epoch 5 | Training Loss: 1.0774, Accuracy: 0.3912
Epoch 6 | Training Loss: 1.0734, Accuracy: 0.3966
Epoch 7 | Training Loss: 1.0711, Accuracy: 0.3973
Epoch 8 | Training Loss: 1.0672, Accuracy: 0.4024
Epoch 9 | Training Loss: 1.0662, Accuracy: 0.4047
Epoch 10 | Training Loss: 1.0622, Accuracy: 0.4104
```

Figure 12: Best model's accuracy scores

The hyperparameter configuration of the model that Optuna decided upon showed a training accuracy of approximately 0.41, as shown in **Figure 12**

4.3. BERT Transformers

4.3.1. Results.

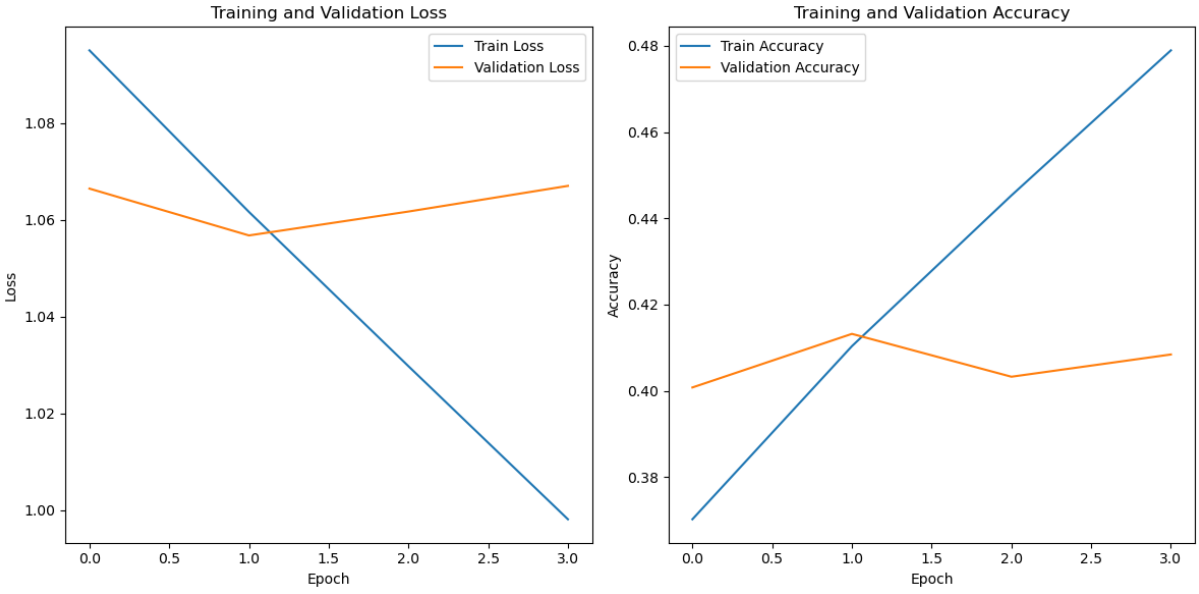

Figure 13: BERT Training-Validation Loss/Accuracy OverTime

Training-Validation Loss: The training loss shows a sharp decrease as epochs progress, indicating that the model is effectively learning from the training data. On the other hand, though, while the validation loss initially decreases, it starts to plateau and slightly increase as training continues, alerting about overfitting.
Training-Validation Accuracy: The training accuracy consistently increases across epochs. Same with the validation loss, the validation accuracy initially increases but stagnates and decreases. The model's ability to generalizing is not particularly improving beyond a certain point.

4.3.2. Optuna Visualizations.

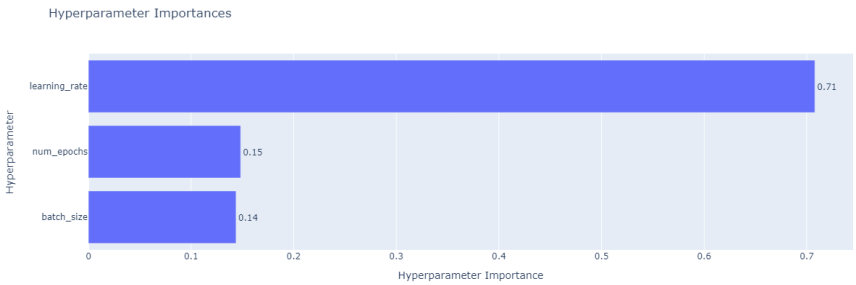

Figure 14: BERT Hyperparameter Importance by Optuna

The hyperparameter importance visualization provided by Optuna for the BERT transformers reveals, similarly to RNN's, that the learning rate (lr) is again has the most significant impact on the model's performance. Number of epochs, as well as the batch size, held a moderate impact, each scoring about 15%.

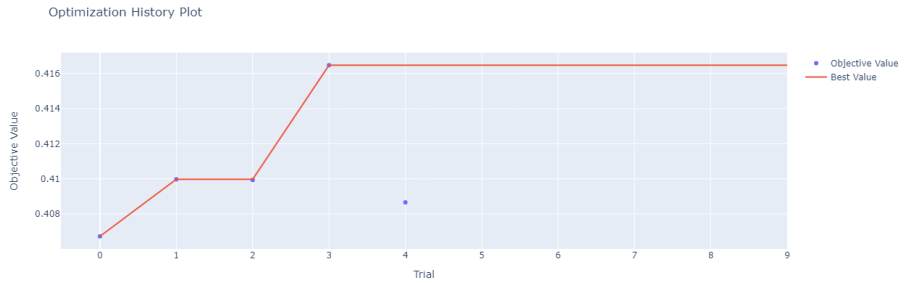

Figure 15: BERT Optimization History by Optuna

On the final iteration of the 10 trials studied by Optuna, the 3rd trial had achieved the highest validation accuracy. After that, trials were being pruned as decided by the hyperparameter framework, as it appears that subsequent trials did not yield better results. This, however, is not always the case, as the trials randomly choose the combinations, thus raising the number of trials for Optuna to study is always a good option. Still, this would take significantly more computational time.

4.3.3. Best Scores.

Optuna's best trial: The best trial achieved a validation accuracy of approximately 0.41 and the optimal hyperparameters (approx values) identified through this trial are as follows:

- Learning Rate (lr): 0.0001
- Batch Size (batch_size): 16
- Number of Epochs (num_epochs): 4

Classification report scores

Training Classification Report

	precision	recall	f1-score	support
NEGATIVE	0.49	0.65	0.56	12210
NEUTRAL	0.57	0.42	0.49	12210
POSITIVE	0.55	0.52	0.54	12210
accuracy			0.53	36630
macro avg	0.54	0.53	0.53	36630
weighted avg	0.54	0.53	0.53	36630

Validation Classification Report

	precision	recall	f1-score	support
NEGATIVE	0.40	0.53	0.45	1744
NEUTRAL	0.41	0.30	0.35	1744
POSITIVE	0.42	0.40	0.41	1744
accuracy			0.41	5232
macro avg	0.41	0.41	0.40	5232
weighted avg	0.41	0.41	0.40	5232

Figure 16: BERT Training - Validation classification report

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>