ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

M.Sc. in Data Science

Text Analytics

Assignment 3 - Recurrent Neural Networks

Grammatikopoulou Maria - f3352310
Phevos A. Margonis - f3352317
Moniaki Melina - f3352321

Instructor: Ion Androutsopoulos
Grader: Foivos Charalampakos

February 28, 2024

Abstract

This study aims to compare the efficacy of Recurrent Neural Networks (RNNs) with that of the simpler Multilayer Perceptron (MLP) architecture in the context of sentiment analysis and part-of-speech tagging. Employing the IMDB Dataset for sentiment analysis and the English-GUM dataset from Universal Dependencies treebanks for part-of-speech tagging, our research underscores that despite RNNs' potential, their performance does not substantially surpass that of the simpler MLP models, highlighting the importance of model selection based on the specific linguistic task at hand.

Contents

1 Exercise 1

1.1 Introduction

Text classification is a fundamental task in natural language processing with applications ranging from sentiment analysis to topic categorization. In this report, we perform a sentiment analysis focusing on the implementation of a bi-directional stacked Recurrent Neural Network (RNN) enriched with self-attention mechanisms. An exploration of hyperparameters, such as the configuration of stacked RNNs and self-attention layers, is undertaken to optimize the model's ability to capture intricate textual patterns. We also compare its performance with baselines models.

1.2 Data set

We conducted our analysis on the IMDB Dataset, consisting of 50,000 movie reviews, each annotated with sentiment labels (positive or negative). Our initial exploration revealed a balanced data set, evenly distributed with 25,000 positive and 25,000 negative reviews.

1.3 Data Preprocessing

Initially, we removed HTML tags from our dataset that were in the form of `<br>`. The text was converted to lowercase to establish uniformity and simplify subsequent processing. Following this, we proceeded to remove punctuation marks to enhance text consistency, as well as numerical digits and any single characters that remained after the removal of apostrophes. Multiple spaces and common stop-words were also removed from the dataset to eliminate non-essential words. Moving forward, we transformed sentiment labels into binary values. Positive sentiment was encoded as 1, while negative sentiment was encoded as 0. Then, we split the data set into training (35000 samples), validation (7500 samples) and test set (7500 samples). The Spacy library was used for sentence splitting and tokenization. For each tokenized word we generated TF-IDF features and then we employed SVD for dimensionality reduction, which reduced the feature space from 5000 dimensions to 500 dimensions.

1.4 Baseline Classifiers

Majority Classifier: We implemented a Majority Baseline Classifier using the Dummy Classifier from scikit-learn. This classifier predicts the most frequent class within the training set. The subsequent classification reports offer a comprehensive summary of precision, recall, and F1 score for each class across the training, validation, and test sets correspondingly. Moreover, the PrecisionRecall AUC (PR-AUC) scores provide valuable insights into the classifier's capability to distinguish between classes in terms of precision and recall.

Table 1: Majority Classifier: Performance Metrics by Class across Data Subsets

Class	Train				Validation				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
Negative	0.50	1.00	0.67	0.75	0.50	1.00	0.66	0.75	0.49	1.00	0.66	0.75
Positive	0.00	0.00	0.00	0.75	0.00	0.00	0.00	0.75	0.00	0.00	0.00	0.75
Macro Avg	0.25	0.50	0.33	0.75	0.25	0.50	0.33	0.75	0.25	0.50	0.33	0.75

Logistic Regression Classifier: The Logistic Regression model serves as an additional baseline for our text classification task. After training the logistic regression model, we observe the following result for the training, validation and test set subsequently.

Multi-Layer Perceptron (MLP) Classifier: An MLP classifier was implemented in Exercise 9 of Part 3 to enhance the performance of the above baseline models. This MLP classifier utilizes a Sequential model with multiple layers. The architecture includes a densely connected layer with 384 units and a Rectified Linear Unit (ReLU) activation. A dropout layer with a rate of 0.2 is incorporated to mitigate overfitting. The final layer consists of a single unit with a sigmoid activation function, suitable for binary classification. The results of this classifier are included in this report to compare the experimental results of an MLP with the results of a RNN model.

Table 2: Logistic Classifier: Performance Metrics by Class across Data Subsets

Class	Training				Development				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
Negative	0.89	0.87	0.88	0.96	0.89	0.86	0.87	0.95	0.89	0.87	0.88	0.95
Positive	0.87	0.90	0.88	0.95	0.87	0.875	0.88	0.95	0.87	0.89	0.88	0.95
Macro Avg	0.88	0.88	0.88	0.95	0.88	0.88	0.88	0.95	0.88	0.88	0.88	0.95

Table 3: MLP Classifier: Performance Metrics by Class across Data Subsets

Class	Training				Development				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
Negative	0.99	0.99	0.99	1.00	0.89	0.87	0.88	0.95	0.89	0.86	0.88	0.95
Positive	0.99	0.99	0.99	1.00	0.87	0.90	0.88	0.95	0.87	0.90	0.88	0.95
Macro Avg	0.99	0.99	0.99	1.00	0.88	0.88	0.88	0.95	0.88	0.88	0.88	0.95

1.5 Bi-directional stacked RNN with self-attention

A bi-directional stacked Recurrent Neural Network (RNN) with self-attention mechanisms is implemented for the task of sentiment analysis. The process begins with text vectorization using a Keras TextVectorization layer, configured to handle a maximum vocabulary size (`MAX_WORDS`) and a maximum sequence length (`MAX_SEQUENCE_LENGTH`). Pre-trained word embeddings, obtained from a FastText model, are utilized to initialize the embedding layer. The RNN architecture consists of two bidirectional GRU layers with recurrent dropout, followed by layer normalization. The outputs of these layers are combined using an element-wise addition, creating a residual connection. A self-attention mechanism, implemented as a Multi-Layer Perceptron (MLP) with dropout, processes the residual states, assigning attention scores to each word in the sequence. The final attention-weighted representation is computed using these scores. The model also includes an MLP layer for further feature extraction, followed by a sigmoid-activated dense layer for binary classification. This MLP architecture closely resembles the MLP classifier architecture implemented in Exercise 9, which was fine-tuned for optimal performance.

During training, the model is compiled with binary cross-entropy loss and Adam optimizer. Checkpoints and early stopping callbacks are employed to monitor and save the model based on validation accuracy. The effectiveness of the model is evaluated on the test set, and training time is reported for reference.

Table 4: RNN Classifier: Performance Metrics by Class across Data Subsets

Class	Training				Development				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
Negative	0.95	0.94	0.95	0.99	0.89	0.89	0.89	0.96	0.89	0.89	0.89	0.96
Positive	0.94	0.95	0.95	0.99	0.89	0.89	0.89	0.95	0.89	0.89	0.89	0.96
Macro Avg	0.95	0.95	0.95	0.99	0.89	0.89	0.89	0.96	0.89	0.89	0.89	0.96

1.6 Conclusions

As a result of the RNN implementation, we observe that the model achieved impressive validation and test accuracies, reaching 89%. Therefore, the model generalizes well to unseen data. Additionally, the Precision-Recall Area Under the Curve (AUC) approached 96%, indicating a strong ability to balance precision and recall. The model cannot improve the validation accuracy past the 7th epoch, and thus the early stopping terminates the training phase at the 17th epoch.

When comparing the RNN with the baseline models, it significantly outperformed the Majority Classifier. Nevertheless, its performance is similar to MLP classifier. Notably, the Logistic Regression exhibited superior performance, achieving an impressive accuracy of 95%.

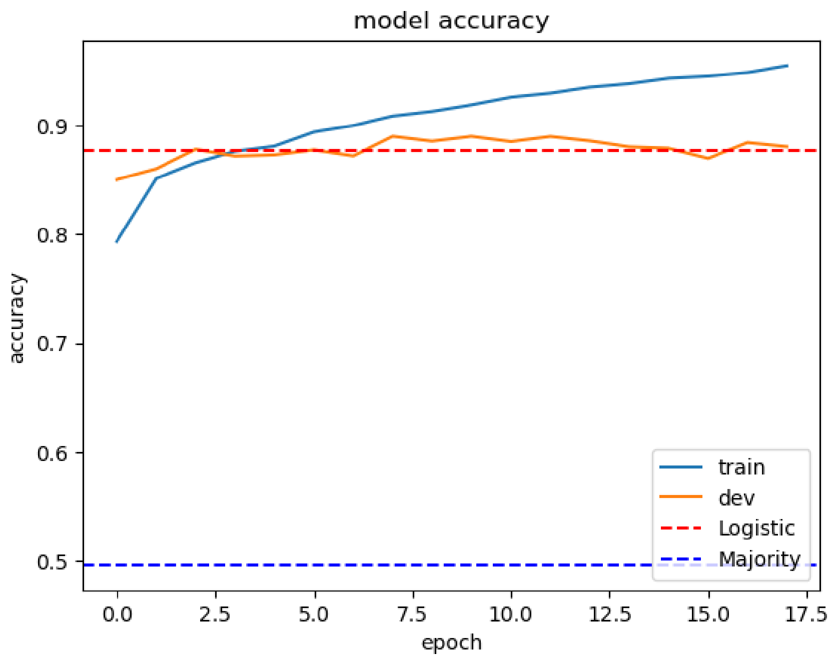

Figure 1: Accuracy Curves for Training and Validation Data as a Function of Training Epoch

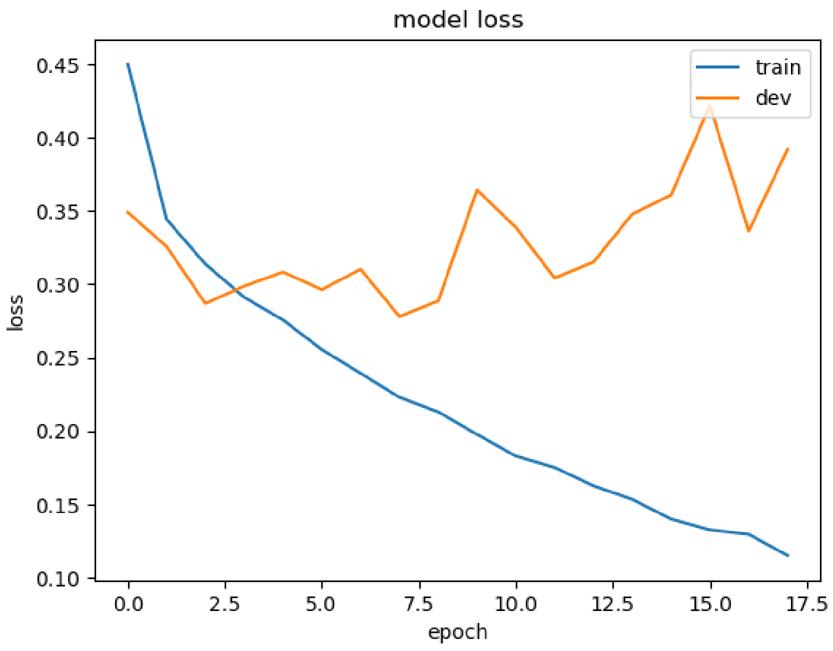

Figure 2: Loss Curves for Training and Validation Data as a Function of Training Epoch

2 Exercise 2

2.1 Introduction

This study aims to address the problem of Part-of-Speech tagging using Recurrent Neural Networks (RNNs) that are capable of understanding the context of each word. For the purpose of comparison with the simple Multilayer Perceptron (MLP) network presented in an earlier study, the final classification layers in the model's architecture have been retained unchanged.

2.2 Dataset

The language of choice for this exercise is English. A collection of annotated text corpora across multiple languages, standardized to represent grammatical information and syntactic relations, is provided by the Universal Dependencies treebanks. From this collection, the English corpus that will serve as the foundation for this study, provided by Georgetown University Multilayer, can be found here. This corpus contains 10,761 sentences, 184,373 tokens[1], and 187,417 syntactic words. More details about the dataset's intricacies can be further explored here.

Universal Dependencies have standardized CONLLU as their file format of choice. The main difference from a plain TEXT file is the inclusion of metadata for each token like POS Tags, Features, Relations, etc., and for that reason, the dedicated Python library conllu is required to parse them.

While parsing the data, the features that are extracted are (a) the token to be classified, (b) its preceding and subsequent words, forming a context window of $n = 3$, and (c) the token's UPOS label. The UPOS labels denote 17 distinct classes (i.e., ADJ, ADP, ADV, AUX). To account for the first and last word in a sentence, and by extension their context window, a pseudo-token *PAD* is prepended and appended accordingly. Hence, a DataFrame is created where each row holds the above information for each available token. Lastly, the DataFrame is stored as CSV to streamline the following steps. This process is repeated for all three available Train-Validation-Test datasets.

2.3 Pre-processing

Initially, the data are parsed from the CONLLU files and stored in a DataFrame. This frame comprises two columns. The first column, *text*, contains one sentence per row in the form of a string. Similarly, the second column, *label*, contains the respective UPOS labels as a string. This process is repeated for all Train-Validation-Test data, resulting in approximately 8000, 1000, and 1000 sentences, respectively. For speed and efficiency, the DataFrames are stored as CSV files, which will be used as the primary data source.

Upon loading the CSV files, several exploratory queries are performed. Firstly, in the training set, duplicate entries are found, usually corresponding to the valediction part of a document. Since they offer no extra information, they are dropped. Secondly, statistics regarding the length of each sentence are calculated, revealing that 90% of them comprise approximately 35 words. This information will later influence the chosen value for the maximum context window that the RNNs can handle.

For faster training and inference times, a context window of 32 words (max sequence length) is chosen. This means that if a sentence exceeds 32 words, only the first 32 will be considered. Conversely, sentences with fewer words will be padded to meet the max sequence length requirement.

Following this, both texts and labels must be converted to a format that the model can handle. For speed and simplicity, the TextVectorization layer from Keras is used to tokenize the labels into sequences of integers. An important detail is that this layer reserves the first two positions for UNKNOWN and PAD tokens, meaning the target labels will take values in the range of 2-19 instead of 0-17.

Having vectorized the label values, we proceed with the texts. Here, a TextVectorization layer is initialized to accept sentences, tokenize them, convert the tokens to lowercase, and match them to their integer values found in the vocabulary adapted from the training set. This preconfigured layer has been stored as an object that will later be included in the model as an on-the-fly Text-to-Vector converter.

Finally, utilizing the Vocabulary created in the last step, along with the Fasttext model, every word in the Vocabulary is converted to its respective embedding, thus creating the embedding matrix. This matrix reserves the first two positions for the PAD and UNK tokens, which take a zero-vector embedding.

2.4 Baselines

Before proceeding with the creation of the RNN, a Majority classifier is built to act as a baseline for comparison. This classifier tags each word with the most frequent tag it encountered in the training data. For words not encountered in the training data, the baseline returns the most frequent tag (over all words) from the training data. This simplistic approach achieves a validation score of about 88%. The results can be found in Table 5.

Furthermore, the MLP classifier presented in the last study is included as a strong baseline, and its classification abilities can be found in Table 6.

[1]https://www.datacamp.com/blog/what-is-tokenization

Table 5: Majority Classifier: Performance Metrics by Class across Data Subsets

Class	Train				Validation				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
ADJ	0.91	0.93	0.92	-	0.90	0.81	0.85	-	0.88	0.80	0.84	-
ADP	0.90	0.89	0.90	-	0.91	0.91	0.91	-	0.91	0.91	0.91	-
ADV	0.89	0.85	0.87	-	0.87	0.80	0.84	-	0.89	0.77	0.82	-
AUX	0.87	0.95	0.91	-	0.88	0.96	0.92	-	0.87	0.94	0.91	-
CCONJ	0.99	1.00	0.99	-	0.98	1.00	0.99	-	0.98	1.00	0.99	-
DET	0.95	0.98	0.96	-	0.95	0.98	0.97	-	0.96	0.98	0.97	-
INTJ	0.73	0.76	0.74	-	0.75	0.74	0.74	-	0.78	0.68	0.72	-
NOUN	0.92	0.95	0.93	-	0.74	0.94	0.83	-	0.68	0.93	0.78	-
NUM	0.95	1.00	0.98	-	0.95	0.86	0.91	-	0.97	0.81	0.89	-
PART	0.69	0.90	0.78	-	0.75	0.92	0.83	-	0.69	0.90	0.78	-
PRON	0.92	0.96	0.94	-	0.93	0.96	0.94	-	0.92	0.97	0.94	-
PROPN	0.94	0.82	0.88	-	0.78	0.40	0.53	-	0.79	0.35	0.49	-
PUNCT	0.99	1.00	1.00	-	1.00	1.00	1.00	-	0.99	1.00	1.00	-
SCONJ	0.85	0.33	0.48	-	0.87	0.36	0.51	-	0.88	0.38	0.54	-
SYM	0.95	0.68	0.79	-	1.00	0.36	0.53	-	0.82	0.53	0.64	-
VERB	0.93	0.89	0.91	-	0.92	0.77	0.84	-	0.90	0.76	0.83	-
X	0.97	0.73	0.84	-	1.00	0.50	0.67	-	0.88	0.50	0.64	-
Macro Avg	0.90	0.86	0.87	-	0.89	0.78	0.81	-	0.87	0.78	0.81	-

Table 6: MLP Classifier: Performance Metrics by Class across Data Subsets

Class	Train				Validation				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
ADJ	0.96	0.98	0.97	0.99	0.91	0.93	0.92	0.97	0.88	0.92	0.90	0.96
ADP	0.98	0.99	0.99	0.99	0.95	0.97	0.96	0.99	0.95	0.98	0.97	0.99
ADV	0.98	0.94	0.96	0.99	0.94	0.87	0.91	0.96	0.96	0.87	0.91	0.97
AUX	0.99	0.99	0.99	0.99	0.98	0.97	0.98	0.99	0.98	0.97	0.98	0.99
CCONJ	0.93	0.94	0.93	0.98	0.83	0.87	0.85	0.91	0.79	0.84	0.81	0.89
DET	0.99	0.99	0.99	0.99	0.95	0.96	0.96	0.99	0.95	0.96	0.96	0.99
INTJ	0.89	0.87	0.88	0.93	0.82	0.76	0.79	0.89	0.88	0.81	0.85	0.89
NOUN	0.96	0.98	0.97	0.99	0.93	0.95	0.94	0.97	0.89	0.94	0.92	0.96
NUM	0.99	0.97	0.98	0.99	0.99	0.96	0.97	0.98	0.98	0.96	0.97	0.99
PART	0.99	0.97	0.98	0.99	0.93	0.90	0.92	0.96	0.92	0.83	0.87	0.94
PRON	0.99	1.00	0.99	1.00	0.99	0.99	0.99	0.99	0.98	0.99	0.98	0.99
PROPN	0.89	0.88	0.88	0.95	0.79	0.73	0.76	0.83	0.80	0.70	0.75	0.83
PUNCT	0.99	0.99	0.99	0.99	0.98	0.98	0.98	0.99	0.98	0.98	0.98	0.99
SCONJ	0.96	0.91	0.94	0.98	0.86	0.77	0.81	0.89	0.84	0.81	0.83	0.90
SYM	0.96	0.83	0.89	0.92	1.00	0.64	0.78	0.97	0.96	0.68	0.79	0.79
VERB	0.98	0.98	0.98	0.99	0.95	0.95	0.95	0.98	0.95	0.94	0.94	0.98
X	0.97	0.34	0.50	0.67	0.80	0.33	0.47	0.44	1.00	0.39	0.56	0.52
Macro Avg	0.96	0.91	0.93	0.96	0.92	0.86	0.88	0.92	0.92	0.86	0.88	0.92

2.5 Recurrent Neural Network Model

Aiming to improve upon those baselines, a Recurrent Neural Network model is implemented. The design utilizes the Functional API to add:

- A one-dimensional input layer that accepts a sentence in the form of a string.
- A Vectorization layer, that transforms the input string into a sequence of integers.
- An Embedding layer that matches each integer word to its embedding from the embedding matrix.
- A Dropout layer with a 0.33 probability.
- Four Bidirectional LSTM layers with residual connections.
- Three Layer-Normalization layers in-between.

Finally, the intact Multilayer Perceptron from the previous study is included for comparison purposes, with:

- A hidden Dense layer of 128 neurons and a ReLU activation function.
- A Dropout layer with a 0.4 probability.
- A hidden Dense layer of 128 neurons and a Tanh activation function.
- A Dropout layer with a 0.1 probability.
- An output layer with 19 neurons and a SoftMax activation function.

The above RNN layers have been created for educational purposes to represent a stacked bidirectional RNN and to test the efficiency of CUDA GPU kernels for these applications. Another important aspect of this architecture is the formulation of the Embedding layer.

The Embedding layer receives a Tensor of shape (batch size, fixed sequence length) and is responsible for converting it into a 3D Tensor (batch size, fixed sequence length, embedding). At this point, the designer must make a crucial decision regarding the padded sequences. Typically, padded sequences should be masked with `mask_zero=True` so that the model can ignore them during training. During training and evaluation (`model.evaluate()`), the model can correctly identify the padded tokens and ignore them. However, the problem arises during inference (`model.predict()`), as the model cannot properly identify and thus mask these tokens. At this stage, the model attempts to infer the class of the padded tokens without prior knowledge of their existence, resulting in the model assigning a fixed class, in this case, class *4*, to all *pad* tokens. This issue complicates the final evaluation of the model (classification reports and AUC scores) because it prevents an objective comparison between the actual and predicted classes. A different problem formulation or architecture might overcome this problem while still retaining the mask.

Alternatively, setting `mask_zero` to False forces the model to consider the padding during its training phase, represented by zeros, to extract their proper embeddings from the embedding matrix, treat the zeros as a viable class, and thus learn to predict them. This approach is problematic for two reasons. Firstly, it will include the overrepresented class zero in the model's metrics report, inflating the metrics and providing a non-objective view of the model's performance. Secondly, the model may misclassify other classes as class zero, deflating the precision-recall metrics for other classes.

To address this discrepancy, manual masking must be implemented in both scenarios to remove the padded tokens from the evaluation process. The following approach creates the mask from the encoded target classes. In a production setting, this decision is biased because target labels are not available, necessitating derivation from the encoded text sequences (which would require a different architecture, possibly excluding the vectorization layer inside the model). Ultimately, `mask_zero=True` is chosen as it offers the most objective approach despite the need for manual masking.

Other settings included:

- The removal of recurrent dropout to enable the use of cuDNN GPU cells.
- The loss function was set to `sparse_categorical_crossentropy`, appropriate for labels that are sequences of integers rather than one-hot encoded labels.

- The Adam algorithm was chosen as the optimizer, with a learning rate of 0.001.
- The batch size was set to 256, with a maximum of 50 epochs.
- EarlyStopping was employed with a patience of 10 epochs.

As a result, the model achieves a validation accuracy of 93.63%. The final architecture of the model can be seen in Figure 5 , and its performance can be scrutinized in Table 7 . Lastly, the learning curves are presented in Figure 3 and Figure 4 , respectively.

Table 7: RNN Classifier: Performance Metrics by Class across Data Subsets

Class	Train				Validation				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
ADJ	1.00	1.00	1.00	1.00	0.88	0.88	0.88	0.95	0.86	0.87	0.86	0.94
ADP	1.00	1.00	1.00	1.00	0.96	0.98	0.97	0.99	0.97	0.98	0.97	1.00
ADV	1.00	0.98	0.99	1.00	0.88	0.87	0.88	0.95	0.89	0.89	0.89	0.95
AUX	1.00	1.00	1.00	1.00	0.96	0.99	0.98	1.00	0.94	0.98	0.96	1.00
CCONJ	1.00	1.00	1.00	1.00	0.99	1.00	0.99	1.00	0.99	0.99	0.99	1.00
DET	1.00	1.00	1.00	1.00	1.00	0.99	0.99	1.00	0.99	0.99	0.99	1.00
INTJ	0.92	1.00	0.96	1.00	0.82	0.94	0.88	0.95	0.81	0.94	0.87	0.96
NOUN	1.00	1.00	1.00	1.00	0.89	0.93	0.91	0.97	0.85	0.93	0.89	0.96
NUM	1.00	1.00	1.00	1.00	0.98	0.85	0.91	0.93	0.97	0.83	0.90	0.91
PART	1.00	1.00	1.00	1.00	0.87	0.98	0.92	0.99	0.73	0.96	0.83	0.96
PRON	1.00	1.00	1.00	1.00	0.99	0.99	0.99	1.00	0.99	0.99	0.99	1.00
PROPN	1.00	0.99	0.99	1.00	0.77	0.63	0.70	0.79	0.82	0.59	0.68	0.80
PUNCT	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
SCONJ	0.99	1.00	0.99	1.00	0.87	0.81	0.84	0.90	0.85	0.87	0.86	0.91
SYM	1.00	1.00	1.00	1.00	1.00	0.71	0.83	1.00	0.87	0.81	0.84	0.82
VERB	1.00	1.00	1.00	1.00	0.94	0.91	0.93	0.98	0.93	0.90	0.92	0.97
X	0.99	0.92	0.95	0.99	0.80	0.36	0.50	0.41	0.67	0.25	0.36	0.38
Macro Avg	0.99	0.99	0.99	1.00	0.92	0.87	0.89	0.93	0.89	0.87	0.87	0.92

2.6 Conclusions

1. The use of the Fasttext model for embedding extraction eliminated the need for custom embeddings, as was necessary in the previous study.
2. The updated version of the Majority Classifier shows surprisingly accurate results.
3. In this case, the RNN network acts as a mechanism that provides enriched, more context-aware embeddings for each word.
4. The RNN network does not offer any competitive advantage over the simple *NER-Window* approach of the previous study, where the embeddings of the three neighboring words were concatenated to form the context window. This result suggests that the UPOS tag of a word is dependent only upon its two neighboring words.

Contributions

In this assignment, the work was divided between the members of our team. The first part of exercise 1, up to and including the baseline classifiers, was implemented by *Melina Moniaki*, while the rest was designed by *Maria Grammatikopoulou.* Lastly, exercise 2 was implemented by *Phevos A. Margonis.*

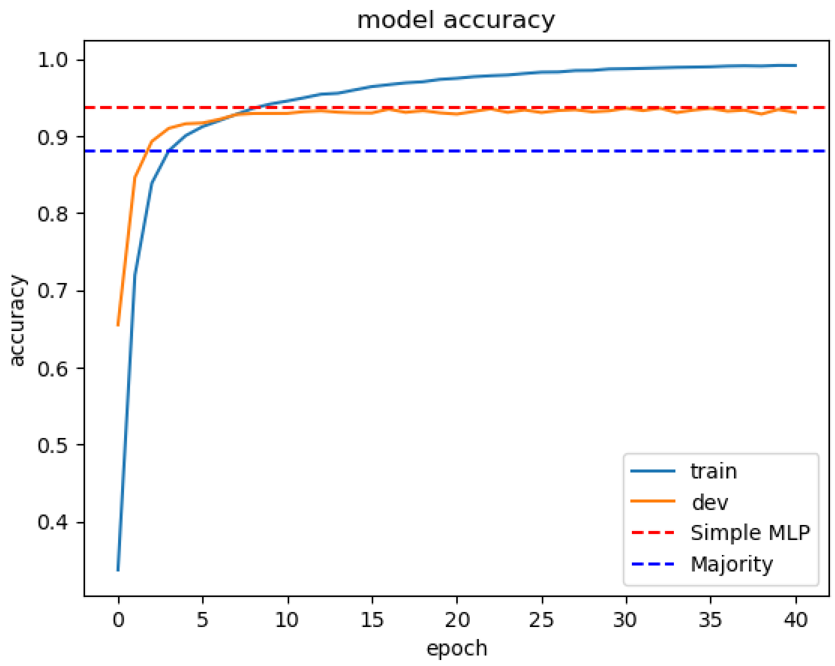

Figure 3: Accuracy Curves for Training and Validation Data as a Function of Training Epoch

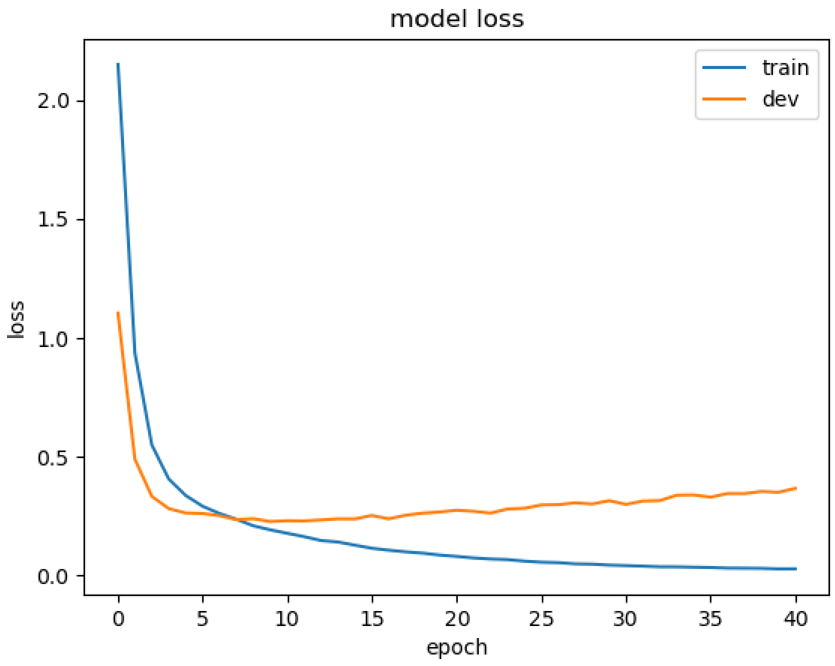

Figure 4: Loss Curves for Training and Validation Data as a Function of Training Epoch

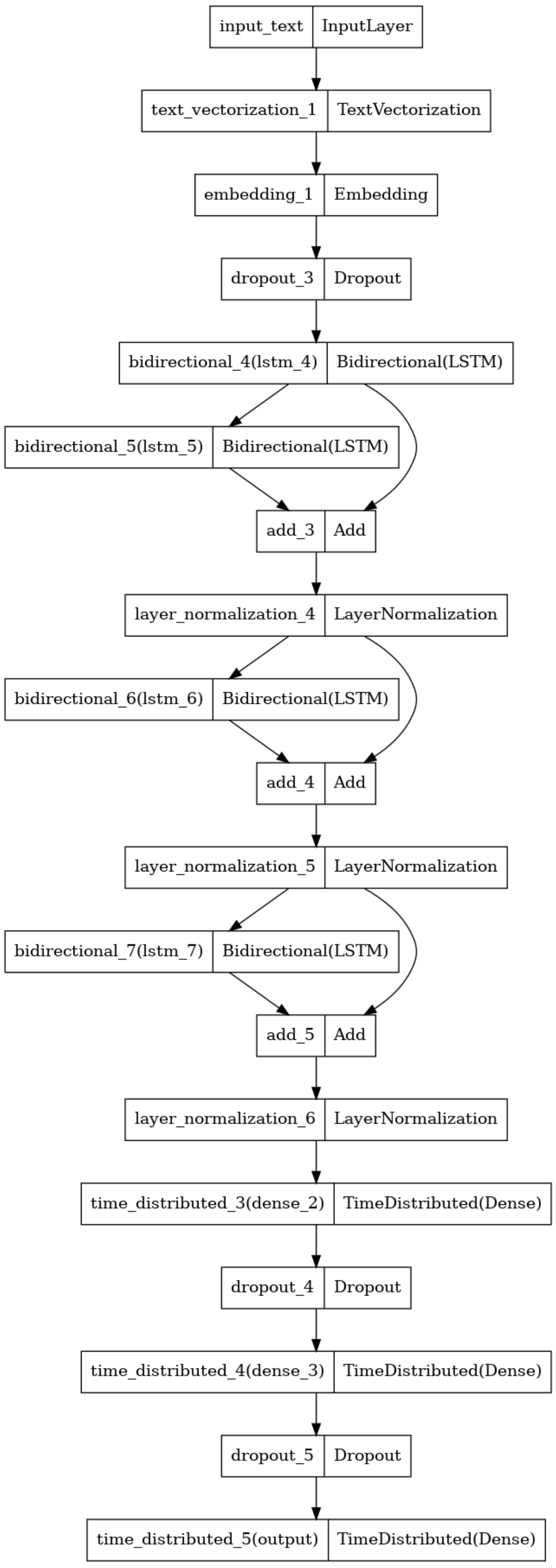
input_text | InputLayer
text_vectorization_1 | TextVectorization
embedding_1 | Embedding
dropout_3 | Dropout
bidirectional_4(lstm_4) | Bidirectional(LSTM)
bidirectional_5(lstm_5) | Bidirectional(LSTM)
add_3 | Add
layer_normalization_4 | LayerNormalization
bidirectional_6(lstm_6) | Bidirectional(LSTM)
add_4 | Add
layer_normalization_5 | LayerNormalization
bidirectional_7(lstm_7) | Bidirectional(LSTM)
add_5 | Add
layer_normalization_6 | LayerNormalization
time_distributed_3(dense_2) | TimeDistributed(Dense)
dropout_4 | Dropout
time_distributed_4(dense_3) | TimeDistributed(Dense)
dropout_5 | Dropout
time_distributed_5(output) | TimeDistributed(Dense)

Figure 5: RNN Model Architecture