Abstractive Text Summarization

Dataset

The dataset that is used to test and validate the developed text summarisation model is the “fake_or_real_news” dataset that was downloaded from the Kaggle[5] website. The aforementioned dataset consists of four columns. To be more specific, the first column is about the identification number of each of the items included in the dataset, the second one contains each text’s title, the third column is about the text concerning the news and the fourth one contains either the "REAL" or "FAKE" label. Generally, it contains 6335 entries, where the news’ (text) titles contain 6256 unique values, the texts consist of 6060 unique values and the "REAL" and "FAKE" labels related to the news articles and their titles are distributed in this dataset with a percentage of 50 each. This particular dataset was selected after careful consideration and deep research among the datasets available online that could be considered as ideal to use for the text summarisation task. Specifically, the selected dataset has not been used by many researchers for training and testing purposes of an automatic text summariser and this fact makes its choice seem quite interesting and challenging. Lastly, as a dataset, it is characterised by a very good data quality and doubtless suitability for the experiments of this project.

Data Preprocessing

The data preprocessing task is of significant importance and it is very commonly performed as part of every system that offers deep learning-based applications and implements natural language processing and text analytics techniques. In general, as far as the preprocessing stage is concerned, some important, standard tasks are tokenisation, normalisation, "noise removal", identification of words and sentences, stemming and lemmatisation, as well as stop words removal, so as to ensure and achieve the desired and appropriate format and structure of the input text, including also the necessary removal of words and sentences that are more than once mentioned in the text (also known as duplicates). Consequently, the length of the text fed as an input to the text summarisation system is decreased after the preprocessing stage.

In our case, for the purpose of preprocessing the data so as to be used for training and testing of the implemented model presented in this project, the conversion of the upper-case characters found in the texts to lower-case, trimming most of the punctuation, white space and non-letter characters removal, as well as tokenisation (which refers to the separation of all the sentences into tokens using the nltk suite of libraries) took place. Moreover, before completing the tokenisation

process, some regular expressions were used to achieve the normalisation of the strings included in the dataset, following the conversion of the characters contained within the strings from Unicode into plain ASCII.

Also, there was a need for minimising the maximum (both input X's and target Y's) sequence length to fifty characters, so as to prevent the delay that the tokeniser may have caused due to the required tokenisation of very long sequences. Moreover, stop words removal was not considered as a necessary task for the preprocessing of this particular dataset given the minimisation of the sequence length mentioned above that took place, as, if applied, the meaning of some sentences could have been significantly altered. Furthermore, an input and a target dictionary have been created, containing 10002 and 5001 words respectively, so that we could have the words included in the dataset converted into integer numbers and vice versa. Lastly, the source (input) and target (output) sequences were aligned.

Setup of the Experiments

With regard to the implementation of the model, four experiments were carried out. The first experimental model was the one where, for the attention encoder – decoder network architecture, two unidirectional recurrent neural networks with GRU units were used for each one respectively. In contrast to the first experiment, the second experimental model was that where two bidirectional recurrent neural networks with LSTM units were utilised for the attention encoder – decoder network architecture of the system.

Next, what was tested as a third experimental case was the one where, for the purpose of the attention encoder – decoder network architecture, two unidirectional recurrent neural networks with GRU units were used but also leveraging the Beam Search algorithm for the decoder to achieve better results and, then, the evaluation of the relevant results after the application of the algorithm took place. The last experiment that was done follows the same logic behind the previous experiment. In particular, precisely as the second experimental model has been performed, two bidirectional recurrent neural networks with LSTM units for the attention seq2seq encoder – decoder network architecture of the system were used but this time the Beam Search algorithm was applied and, finally, evaluated this specific experimental model case.

The following tables illustrate the exact characteristics and features of each one of the four experimental model cases described above.

Case #1 and Case #3: Unidirectional GRU RNN (without and with Beam Search respectively)

Table 2. Model Summary – Encoder's Characteristics (Case #1 and #3)

Encoder		
(embedding):	Embedding(10002, 300)	embedding with input_size = 10002 and hidden_size = 300
(gru):	GRU(300, 300)	GRU with hidden_size = 300

Table 3. Model Summary – Decoder's Characteristics (Case #1 and #3)

Attention Decoder		
(embedding):	Embedding(5001, 300)	embedding with input_size = 5001 and hidden_size = 300
(attn):	Linear(in_features=600, out_features=51, bias=True)	a linear transformation is applied to the incoming data
(attn_combine):	Linear(in_features=600, out_features=300, bias=True)	a linear transformation is applied to the embedded data
(dropout):	Dropout(p=0.1)	probability of an element to be zeroed
(gru):	GRU(300, 300)	GRU with hidden_size = 300
(out):	Linear(in_features=300, out_features=5001, bias=True)	a linear transformation is applied for the output data

Case #2 and Case #4: Bidirectional LSTM RNN (without and with Beam Search respectively)

Table 4. Model Summary – Encoder's Characteristics (Case #2 and #4)

Encoder		
(embedding):	Embedding(10002, 300)	embedding with input_size = 10002 and hidden_size = 300
(bilstm):	LSTM(300, 150, bidirectional=True)	LSTM with input_size = 300 and output_size = 150

Table 5. Model Summary – Decoder's Characteristics (Case #2 and #4)

Attention Decoder		
(embedding):	Embedding(5001, 300)	embedding with input_size = 5001 and hidden_size = 300
(attn):	Linear(in_features=600, out_features=51, bias=True)	a linear transformation is applied to the incoming data
(attn_combine):	Linear(in_features=600, out_features=300, bias=True)	a linear transformation is applied to the embedded data
(dropout):	Dropout(p=0.1)	probability of an element to be zeroed
(bilstm):	LSTM(300, 150, bidirectional=True)	LSTM with input_size = 300 and output_size = 150
(out):	Linear(in_features=300, out_features=5001, bias=True)	a linear transformation is applied for the output data

Results of the Experiments

As already mentioned in the previous Chapter, four experimental models have been implemented. It should be mentioned that, in simple terms, there is a “distance” between the results – output distribution that the model was expected to have as an output and to what the actual output distribution is. That “distance” is indicated by the cross entropy loss. With the help of the matplotlib library, plotting was performed using the saved loss values during the training phase.

In the figures below, the representation of the cross entropy loss in correspondence with the iterations during the training phase is depicted.

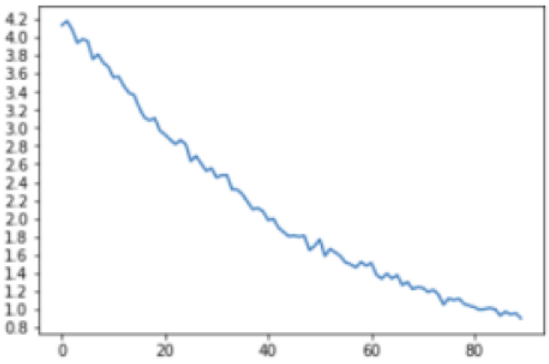

Figure 10. Case #1

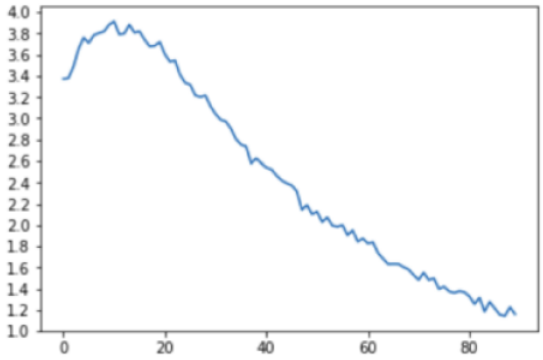

Figure 11. Case #2

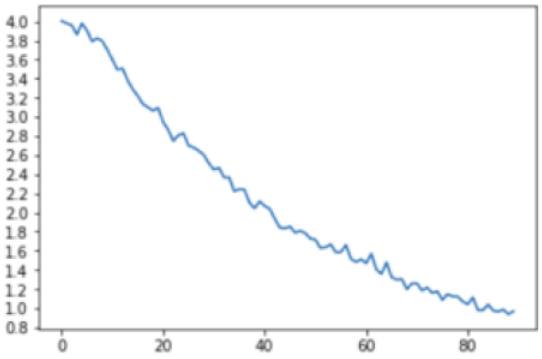

Figure 12. Case #3

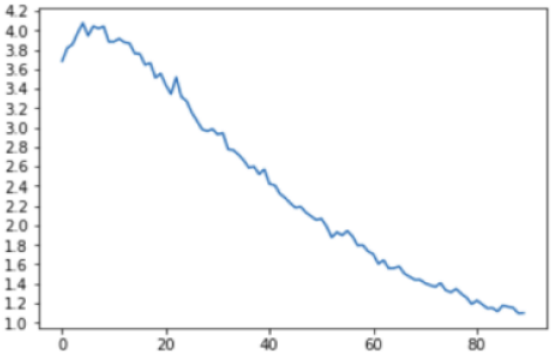

Figure 13. Case #4

In the loss plot diagrams above, the x-axis corresponds to the iterations made during the training phase (which are 90000) and the y-axis corresponds to the respective loss values. We observe that at the beginning of the training phase the loss is quite high, which is quite reasonable. Nevertheless, as it approaches the end of the training phase, the loss is steadily decreasing significantly in proportion to the increase in training iterations. All the experimental model cases seem to have the

same downward trend in cross entropy loss with the difference that, in cases #2 and #4, while at the beginning of the training phase they start both with a lower loss rate than those in model cases #1 and #3, they have slightly increased cross entropy losses in the run up to a little before the end of the training phase. However, only the model case #1 has the lowest loss rates with a steady downward trend from start to finish compared to the other three model cases. Then, with descending order from the best to the worst case model, case #3, case #4 and model case #2 follow.

The table below illustrates the time it took to each of the four experimental models in order to complete the training phase, as well as the ROUGE-1 - F1 scores we have achieved during the validation testing and evaluation process. The first model we test (case #1), according to the ROUGE-1 - F1 score as it appears on the table, has the best performance compared to the other 3 models which are following with a downward course after the first one. Thus, the descending order again from the best to the worst case model is case #3, case #4 and model case #2.

Table 6. Models' training phase execution time and ROUGE-1 – F1 scores

Model	Training Execution Time	ROUGE-1 – F1 score
Case 1	184m	12
Case 2	716m	7.92
Case 3	215m	9.97
Case 4	708m	8.11