Deep Learning project:

Text summarization using Auto-encoder

Realized by: Sarah gharbi, Bassem el Ghoul and Firas Bargui

**Idea**:

Part1:we will use an encoder-decoder sequence-to-sequence model to summarize the text. In this model, an encoder accepts the actual title and body, trains the model to create an encoded representation, and sends it to a decoder which decodes the encoded representation into a reliable title. In this part, we will cover:

1- Import the dataset

2- Cleaning Data

3- Determine the max permissible sequence lengths

4- Select bodies and titles

5- Tokenizing the text

6- Removing empty data

7- Creating the model

8- Training the model

9- Prediction

Part2: we used the pegasus model by Google to perform abstractive summarization. The model uses Transformers Encoder-Decoder architecture. The encoder outputs masked tokens while the decoder generates Gap sentences. In this part, we will cover:

1- Install dependencies

2-Import and load model

3-Perform abstractive summarization

**Resume**:

Part1:

1-Import the dataset:

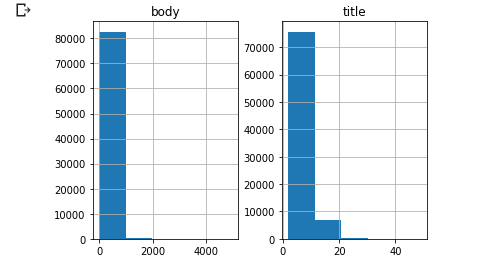
Import the dataset using panda’s read\_csv()

2-Cleaning Data:

the data may have non-alphabetic characters so we can use the regular expressions library to remove these characters before training the model.

3-Determine the max permissible sequence lengths:

in this sequence, we will determine the range of length of words where the maximum number of body and title fall into.



4-Select bodies and titles:

select plausible bodies and titles which are below the maximum lengths as defined in the previous sequence.

5-Tokenizing the text:

First of all we split the data into train and test data using train\_test\_split from slearn.model\_selection, then we prepare and tokenize the text data.

6-removing empty data:

we will remove empty titles and their associated bodies from the data.

7-Creating the model:

we will define the encoder and decoder networks.

Encoder:

the encoder accepts an input equal to the maximum body length estimated in sequence 3. This is followed by three LSTM networks wherein each layer returns the LSTM output, as well as the hidden and cell states observed at the previous time steps.

Decoder:

The decoder defines an embedding layer followed by an LSTM network. The initial state of the LSTM network is the last hidden cell state taken from the encoder. The output of the LSTM is given to a Dense layer wrapped in a TimeDistributed layer with an attached softmax activation function.

Altogether, the model accepts encoder (Body) and decoder (Title) as input and it outputs the Title. The prediction happens through predicting the upcoming word of the Title from the previous word of the Title.

8-Training the model:

We compile the model and use model.fit() to fit the training data. We can plot the training and validation loss metrics observed during the training phase.

9-Prediction:

First, we will reverse map the indices to the words then we define the encoder and decoder inference models to start the prediction.

An encoder inference model accepts the body and returns the output generated from the three LSTMs, and hidden cell states. A decoder inference model accepts the start of the sequence identifier (sostok) and predicts the upcoming word, eventually leading to predicting the whole title.

Part2:

1- Install dependencies:

We will install all our dependencies to be able to use the Pegasus model. Specifically, we’ll use a library called HuggingFace Transformers, Pytorch, and a text tokenizer known as SentencePiece.

2-Import and load model:

We import 2 classes:

the PegasusForConditionalGeneration and PegasusTokenizer.

The PegasusTokenizer converts the sentences into tokens. This allows us to pass it to our DL model. The PegasusForConditionalGeneration will allow us to use our model.

3-Perform abstractive summarization:

Now we can pass a bunch of the body through the Pegasus model and see how the model performs abstractive summarization on the text.