PROJECT REPORT

Webpage Text Summarization

Submitted By:
Mahesh Lomrar
B.Tech(Computer Science)
IIT Bombay

Under the Guidance of:
Vinod Mishra,
Harshit Gautam
Samsung R&D Delhi

May-June 2020

Samsung R&D Institute India - Delhi (SRI - Delhi)
| Tower A, 2nd Floor, 2 - A, Sector 126, Noida - 201303 |
EP-F12116E5FF7B4103BD364A416085B428

1. Introduction:

Summarization is the ability to explain a larger piece of literature in short and covering most of the meaning the context addresses. There are mainly two ways to make summaries:

1.1 Extractive

Select relevant phrases from the input document and concatenate them to form a summary. The selection is based on the words in the sentences where predefined weights assigned to the important words.

1.2 Abstractive

Abstractive summarization includes heuristic approaches to train the system in making an attempt to understand the whole context and generate the summary based on that understanding. This is a more human-like way of generating summaries.

2. Dataset:

For this task, I used the dataset. The columns which were unnecessary for this task (Source, Time, Date) are removed. The dataset has ~55k samples of news-headlines pairs. There are ~76k unique words in the news data and ~29k in the headlines. The sample with the longest news sequence consist of 469 words and the average length of news is 368 words, while that of the headlines is 96 and 63 words respectively. This data cannot be directly used in the model and hence need to be preprocessed.

3. Preprocessing:

Basic preprocessing required to train the NLP model.

1. For recognizing the start and end of target sequences, we pad them with start ("<go>") and end ("<stop>") tokens.
2. Fit tokenizer to filter punctuation marks, converts text to lowercase.
3. Map words with tokens, and convert textual data to tokens which can be directly inputted to model.
4. Padding/truncating the sequences to a fixed length so that we can have a generalized input to the model.
5. Shuffle and batch the data so that it can be easily fetched while training the model.

6. Used TensorFlow's Dataset API for faster computation of operations.

4. Utility Functions:

Components of the transformer which directly contribute to the model.

4.1 Positional Encodings

Positional encodings basically induce the notion of ordering among the input words.

4.2 Masking

Padding mask is responsible for ignoring the external padding added to the sequences which are smaller than *maxlen*.
Look-ahead mask is responsible for ignoring the words that occur after a given current word in the target sequence from contributing the prediction of the current word.

5. Model

5.1 Scaled-Dot Product

Compute weights for the attention layer using softmax activation.

5.2 Point-wise-feed-forward-network

This is a regular feed-forward network which is used after almost every sub-layer and is
Used identically.

5.3 Multi-Head Attention

Defined as TensorFlow Custom layer. Here input is splitted into multiple heads, compute the attention weights using scaled dot-product attention and finally, concat output from all heads.

5.4 The Encode and Decoder Blocks

These consist of Multi-Head Attention layers followed by residual dropout connections and linear transformation layers.

5.5 The Actual Encoder and Decoder

This layer is responsible for interacting with the input and output directly. The input embeddings are obtained, then added to the positional encodings.

5.6 Stacking the layers in Model

Stack all the intermediate layers in a Custom Model class inherited from tf.keras.model class. Append the output layer having vocab size units to the decoder. The output layer of this class will be used either for inference or for loss calculation while training.

6. Training the model

The model is trained with SparseCategoricalCrossEntropy loss as regular CategoricalCrossEntropy loss would require the target as a one-hot encoding of each word in the target sequence which would be a huge array.
The Hyper-parameters values that I used for training are:

Num-layers: 4
D_model: 4
Dff: 512
Num_heads: 8
Epochs: 20
BUFFER_SIZE: 20000
BATCH_SIZE: 64
Encoder_maxlen: 400
Decoder_maxlen: 75
Encoder_vocab_size: 76362
Decoder_vocab_size: 29661

7. Inference

The test should also form a similar dataset as that of training for better results.
Implemented a custom inference step as the model is used to receiving target sequence as an input at the decoder. While testing, we need to use the previous predictions as input to the decoder.

8. Conclusion

Successfully implemented the well-known Transformer model using the TensorFlow API.

9. References

1. https://github.com/icoxfog417/awesome-text-summarization
2. https://colab.research.google.com/drive/1l39vWjZ5jRUimSQDoUcuWGIoNjLjA2zu#scrollTo=L_apacxCvZZ7
3. https://www.tensorflow.org/tutorials/text/transformer
4. https://www.tensorflow.org/api_docs
5. Dataset: https://www.kaggle.com/shashichander009/inshorts-news-data