Abstractive Text Summarization using NLP

Our objective is to build a text summarizer where the input is a long sequence of words (in a text body), and the output is a short summary (which is a sequence as well). So, we can model this as a Many-to-Many Seq2Seq problem.

In the training phase, we will first set up the encoder and decoder. We will then train the model to predict the target sequence offset by one timestep.

Encoder

An Encoder Long Short Term Memory model (LSTM) reads the entire input sequence wherein, at each timestep, one word is fed into the encoder. It then processes the information at every timestep and captures the contextual information present in the input sequence.

The hidden state (h_i) and cell state (c_i) of the last time step are used to initialize the decoder since the encoder and decoder are two different sets of the LSTM architecture.

Decoder

The decoder is also an LSTM network which reads the entire target sequence word-by-word and predicts the same sequence offset by one timestep. The decoder is trained to predict the next word in the sequence given the previous word.

Inference Phase

After training, the model is tested on new source sequences for which the target sequence is unknown. So, we need to set up the inference architecture to decode a test sequence:

1. Encode the entire input sequence and initialize the decoder with internal states of the encoder
2. Run the decoder for one timestep with the internal states
3. The output will be the probability for the next word. The word with the maximum probability will be selected
4. Pass the sampled word as an input to the decoder in the next timestep and update the internal states with the current time step
5. Repeat steps 3 – 5 until we hit the maximum length of the target sequence

Global Attention

The attention is placed on all the source positions. In other words, all the hidden states of the encoder are considered for deriving the attended context vector.

Local Attention

The attention is placed on only a few source positions. Only a few hidden states of the encoder are considered for deriving the attended context vector.

Understanding the Problem Statement

Customer reviews can often be long and descriptive. Analyzing these reviews manually, as you can imagine, is really time-consuming. This is where the brilliance of Natural Language Processing can be applied to generate a summary for long reviews.

We will be working on a really cool dataset. Our objective here is to generate a summary for the Amazon Fine Food reviews using the abstraction-based approach we learned about above.

You can download the dataset from here.

Custom Attention Layer

Keras does not officially support attention layer. So, we can either implement our own attention layer or use a third-party implementation. We will go with the latter option for this article. We can download the attention layer from here.

Read the dataset

This dataset consists of reviews of fine foods from Amazon. The data spans a period of more than 10 years, including all ~500,000 reviews up to October 2012. These reviews include product and user information, ratings, plain text review, and summary. It also includes reviews from all other Amazon categories.

We'll take a sample of 100,000 reviews to reduce the training time of our model. We can also use the entire dataset for training our model.

Preprocessing

Performing basic preprocessing steps is very important before we get to the model building part. Using messy and unclean text data is a potentially disastrous move. So, we will drop all the unwanted symbols, characters, etc. from the text that do not affect the objective of our problem.

We will perform the below preprocessing tasks for our data:

- Convert everything to lowercase
- Remove HTML tags
- Contraction mapping
- Remove (‘s)
- Remove any text inside the parenthesis ( )
- Eliminate punctuation and special characters
- Remove stopwords
- Remove short words

Understanding the distribution of the sequences

We will analyze the length of the reviews and the summary to get an overall idea about the distribution of length of the text. This will help us fix the maximum length of the sequence.

We need to split our dataset into training and validation set. We’ll use 90% of the dataset as the training data and evaluate the performance on the remaining 10% (holdout set):

Preparing the Tokenizer

A tokenizer builds the vocabulary and converts a word sequence to an integer sequence. Go ahead and build tokenizers for text and summary:

a) Text tokenizer
b) Summary tokenizer

Model building

- Return Sequences = True: When the return sequences parameter is set to True, LSTM produces the hidden state and cell state for every timestep
- Return State = True: When return state = True, LSTM produces the hidden state and cell state of the last timestep only
- Initial State: This is used to initialize the internal states of the LSTM for the first timestep

- Stacked LSTM: Stacked LSTM has multiple layers of LSTM stacked on top of each other. This leads to a better representation of the sequence. I encourage you to experiment with the multiple layers of the LSTM stacked on top of each other (it's a great way to learn this)

We will monitor the validation loss (val_loss). Our model will stop training once the validation loss increases.

How can we Improve the Model's Performance Even Further?

- Increase the training dataset size and build the model. The generalization capability of a deep learning model enhances with an increase in the training dataset size
- Implementing Bi-Directional LSTM which is capable of capturing the context from both the directions and results in a better context vector
- Using the beam search strategy for decoding the test sequence instead of using the greedy approach (argmax)
- Evaluate the performance of your model based on the BLEU score
- Implementing pointer-generator networks and coverage mechanisms