Text Summarizer using Deep learning

# Text Summarization in NLP:

Automatic text summarization is the task of producing a concise and fluent summary while preserving key information content and overall meaning.

There are broadly two different approaches that are used for text summarization:

* Extractive Summarization
* Abstractive Summarization

## Extractive Summarization:

Identify the important sentences or phrases from the original text and extract only those from the text.

## Abstractive Summarization:

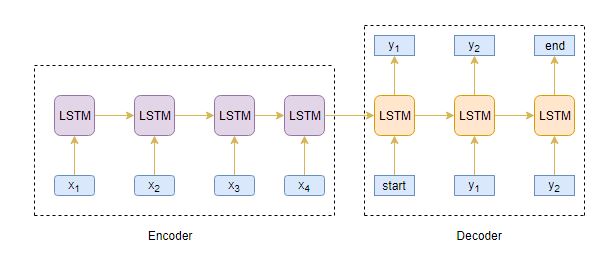
Generate new sentences from the original text. This is in contrast to the extractive approach we saw earlier where we used only the sentences those were present. The sentences generated through abstractive summarization might not be present in the original text.

**Note:** We have used Abstractive method for Text Summarization using deep learning techniques.

## Introduction to Sequence-to-Sequence (Seq2Seq) Modeling:

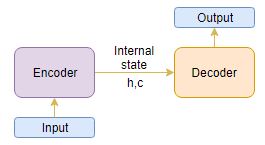
We can build a Seq2Seq model on any problem which involves sequential information. This includes Sentiment classification, Neural Machine Translation, and Named Entity Recognition – some very common applications of sequential information.

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.**



## Understanding the Encoder-Decoder architecture:

The Encoder-Decoder architecture is mainly used to solve the sequence-to-sequence (Seq2Seq) problems where the input and output sequences is of different lengths.



Generally, variants of Recurrent Neural Networks (RNNs), i.e. Gated Recurrent Neural Network (GRU) or Long Short Term Memory (LSTM), are preferred as the encoder and decoder components. This is because they are capable of capturing long term dependencies by overcoming the problem of vanishing gradient.

We can set up the Encoder-Decoder in 2 phases:

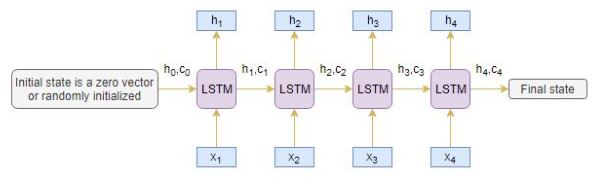
* Training phase
* Inference phase

## Training Phase:

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 time step. Let us see in detail on how to set up the encoder and decoder.

## Encoder:

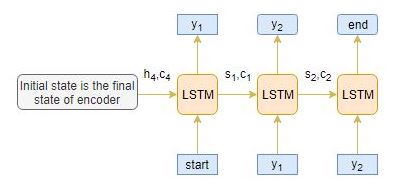
An Encoder Long Short Term Memory model (LSTM) reads the entire input sequence wherein, at each time step, one word is fed into the encoder. It then processes the information at every time step and captures the contextual information present in the input sequence.



The hidden state (hi) and cell state (ci) of the last time step are used to initialize the decoder. Remember, this is because the encoder and decoder are two different sets of the LSTM architecture.

## Decoder:

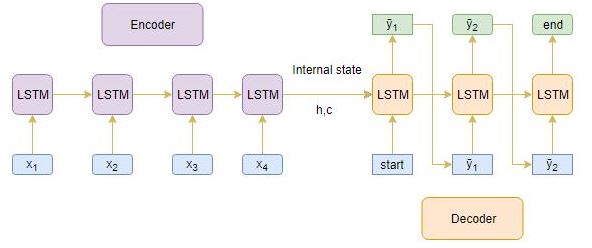
The decoder is also an LSTM network which reads the entire target sequence word-by-word and predicts the same sequence offset by one time step. The decoder is trained to predict the next word in the sequence given the previous word.



<Start> and <End> are the special tokens which are added to the target sequence before feeding it into the decoder. The target sequence is unknown while decoding the test sequence. So, we start predicting the target sequence by passing the first word into the decoder which would be always the <start> token. And the <end> token signals the end of the sentence.

## 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:



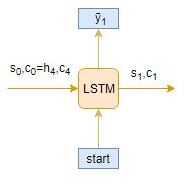
Here are the steps to decode the test sequence:

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

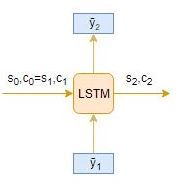
Let’s take an example where the test sequence is given by [x1, x2, x3, x4]. How will the inference process work for this test sequence? I want you to think about it before you look at my thoughts below.

1. Encode the test sequence into internal state vectors
2. Observe how the decoder predicts the target sequence at each time step.

## TimeStep t1:



## TimeStep t2:



## TimeStep t3:

