Abstractive text summarization using Attention based Neural Network

1) Introduction

In the age of information, the massive amount of data produced every day will remain unuseful for humans unless it makes it available with new tools and technologies. Abstractive Text Summarization tries to get the essential content of a text corpus and compress it to a shorter text while keeping its meaning and maintaining its semantic and grammatical correctness. Neural architectures are becoming dominant in the Abstractive Text Summarization. The use of deep learning architectures in natural language processing entered a new era after the sequence to sequence models in the recent decade. These models are founded on a couple of recurrent neural networks connecting the input and output data in an encoder-decoder architecture. Better results were possible by adding the Attention Mechanism to the RNN layers. Many works have shown the competitive performance of these new architectures in Machine Translation (ML).

Websites such as Amazon and Yelp allow customers to leave reviews for various products. There are usually hundreds of reviews for a single product; each review could be lengthy and repetitive. Therefore automatic review summarization has a vast potential in that it could help customers make quick decisions on certain products. In addition, summarization could be applied to reviews and other entities, such as emails and news articles, etc.

2) Summarization Methods

There are mainly two ways to make the summary. Extractive and Abstractive.
Extractive
Select relevant phrases of the input document and concatenate them to form a summary (like "copy-and-paste").

- **Pros**: They are pretty robust since they use existing natural-language phrases taken straight from the input.
- **Cons**: They lack flexibility since they cannot use novel words or connectors. They also cannot paraphrase like people sometimes do.

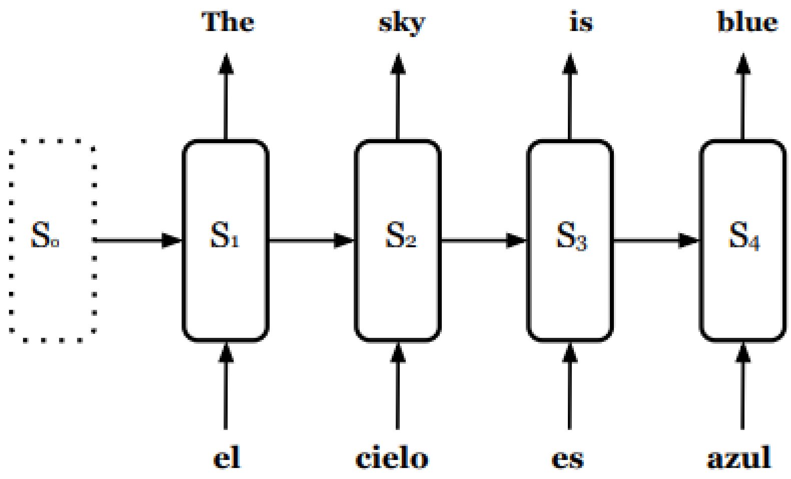
Abstractive
Generate a summary that keeps original intent. It is just like humans do.

- **Pros**: They can use words that were not in the original input. It enables us to make more fluent and natural summaries.
- **Cons**: It is also a much more complex problem as you now require the model to generate coherent phrases and connectors.

3) Neural Text Summarization

Recurrent Neural Networks

An RNN tries to relate the elements of a sequence to each other so that a network retains the previous data's memory. To this end, RNN re-uses its output state and feeds it back to its hidden layer. Using the hidden layers state might cause misunderstanding. To clarify, we compare the conventional feed-forward architecture with RNN. In a feed-forward setting, the raw data inputs are inserted in the input layer. With forward propagation, the state of the following layers will change one by one. If we insert another instance of the input in the input layer, the states of the hidden layer will completely change. RNN incorporates the concept of memory, picking the state of the hidden layer combines it with the currently hidden input to make the current hidden state. We repeat this recursive procedure for each of the input sequence elements.



Bidirectional Recurrent Neural Networks

RNNs look at the previous timesteps to find the bounds and relations. This scan has a direction, and it is only backward. Alternatively, in other words, they have a memory of only the past. To this end, a Bidirectional Recurrent Neural Network (BRNN) is utilized to analyze the dependencies of each element in a sequence to the previous and the next one.

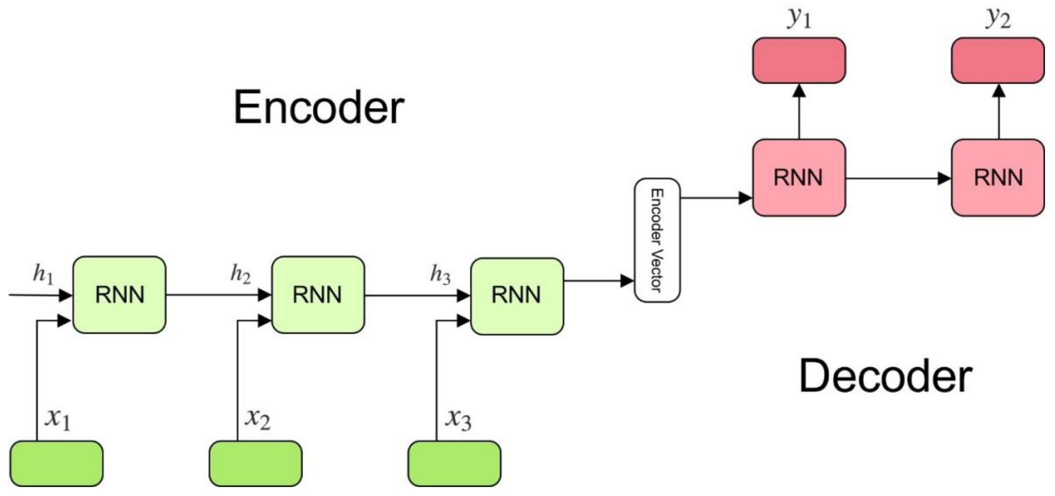
Long Short-term Memory

In each timestep, an RNN loops back the state of its hidden layer. So in the next step, it has a memory of the last step. Imagine a sequence of length ten as our input. In the 2nd step, the RNN can remember state 1. However, as it progresses in the length of the sequence, each time a new input combines with the looped back state, so in step 10, the network has a memory of all the previous steps with the notion that they do not have the same strength. The first step almost dissolves after this long repetitive incoming of new inputs. Therefore an RNN can remember the immediate past better, or with a better interpretation, it has a short-term memory. Equipping these RNN architectures to long-term memory was the fundamental idea of LSTM cells.

Sequence to Sequence Model

A sequence to sequence model aims to map a fixed-length input with a fixed-length output where the input and output length may differ.

How does the Sequence to Sequence Model work?



The model consists of 3 parts: encoder, intermediate (encoder) vector, and decoder.

Encoder

- A stack of several recurrent units (LSTM or GRU cells for better performance) where each accepts a single element of the input sequence, collects information for that element, and propagates it forward.
- In question-answering problems, the input sequence collects all words from the question. Each word is represented as x_i, where i is the order of that word.
- The hidden states h_i are computed using the formula:

$$h_t = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$$

This simple formula represents the result of an ordinary
recurrent neural network. As you can see, we apply the appropriate weights to the previously hidden state h_(t-1) and the input vector x_t.

Encoder Vector

- This is the final hidden state produced from the encoder part of the model. It is calculated using the formula above.
- This vector aims to encapsulate the information for all input elements to help the decoder make accurate predictions.
- It acts as the initial hidden state of the decoder part of the model.

Decoder

- A stack of several recurrent units predicts an output y_t at a time step t.
- Each recurrent unit accepts a hidden state from the previous unit and produces an output and its hidden state.
- In the question-answering problem, the output sequence collects all words from the answer. Each word is represented as y_i, where i is the order of that word.
- Any hidden state h_i is computed using the formula:

$$h_t = f(W^{(hh)}h_{t-1})$$

As you can see, we are just using the previous hidden state to compute the next one.

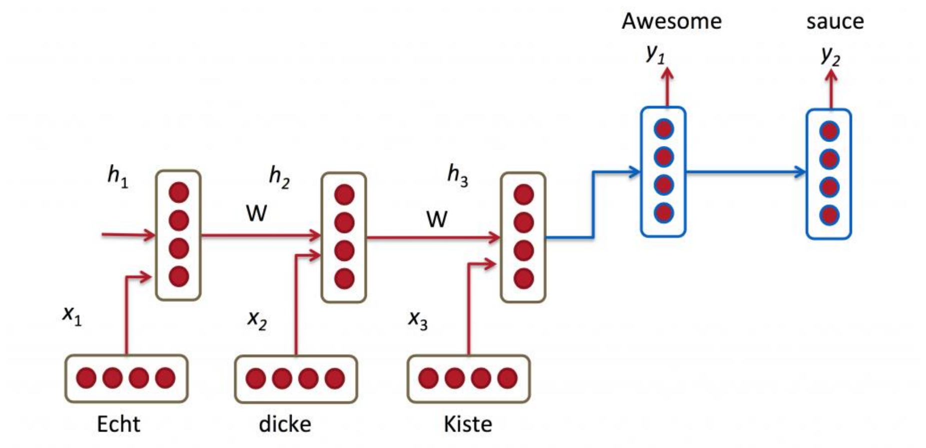
- The output y_t at time step t is computed using the formula:

$$y_t = softmax(W^S h_t)$$

We calculate the outputs using the hidden state at the current time step and the respective weight W(S). Softmax is used to create a probability vector that will help us determine the final output (e.g., the word in the question-answering problem).
The power of this model lies in the fact that it can map sequences of different lengths to each other.

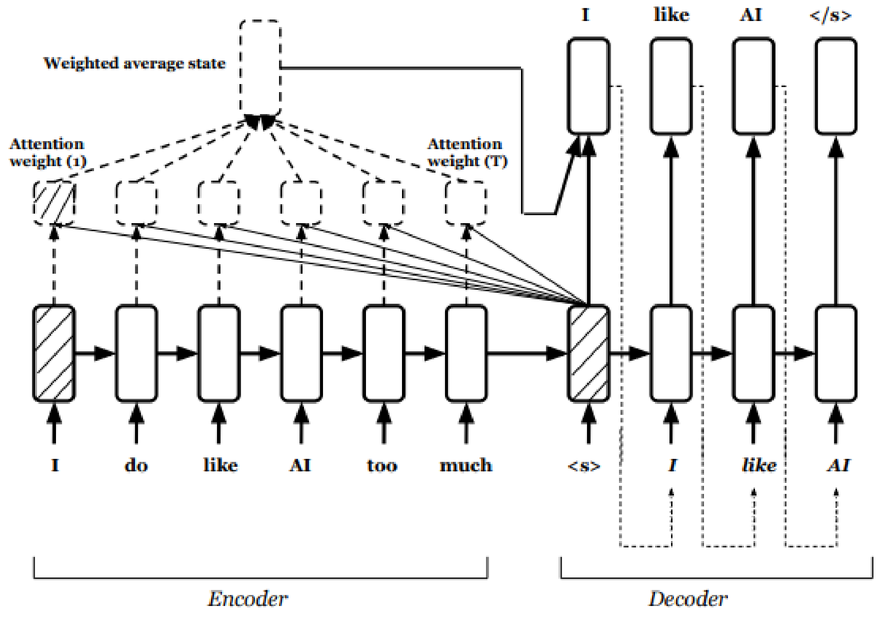
4) Attention networks

What problem does Attention solve?

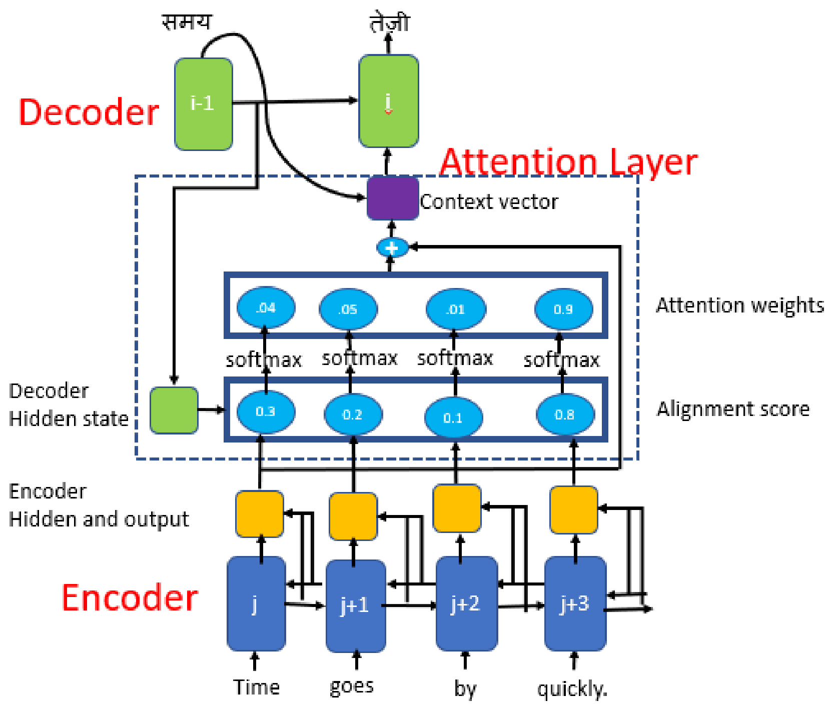


In the picture above, “Echt,” “Dicke,” and “Kiste” words are fed into an encoder, and after a special signal (not shown), the decoder starts producing a translated sentence. The decoder keeps generating words until a special end of sentence token is produced. Here, the vectors represent the internal state of the encoder.

Here, The ‘s are our translated words produced by the decoder, and the ‘s are our source sentence words. The above illustration uses a recurrent bidirectional network, but that is not important, and you can ignore the inverse direction. The important part is that each decoder output word y_t depends on a **weighted combination of all the input states**, not just the last state. The a’s are weights that define how much of each input state should be considered for each output.



Bahdanau attention mechanism



The attention layer consists of

- Alignment score
- Attention weights

- Context vector

Alignment score

The alignment score maps how well the inputs around position “j” and the output at position “i” match. The score is based on the previous decoder’s hidden state, $s_{(i-1)}$, just before predicting the target word and the hidden state, h_j of the input sentence.

$$e_{ij.} = a(s_{i-1}, h_j) \qquad \text{Alignment Score}$$

The decoder decides which part of the source sentence it needs to pay attention to instead of encoding all the information of the source sentence into a fixed-length vector.
The alignment vector that has the same length as the source sequence and is computed at every time step of the decoder
In our example, to predict the second target word, तेज़ी, we will generate a high score for the input word quickly.

Attention weights

We apply a softmax activation function to the alignment scores to obtain the attention weights.

$$\alpha_{i_j} = {\exp(e_{ij})}\Big/{\sum_{k=1}^{Tx} \exp(e_{ik})} \qquad \text{Attention weight}$$

Softmax activation function will get the probabilities whose sum will be equal to 1, This will help to represent the weight of influence for each of the input sequences. Higher the attention weight of the input sequence, the higher will be its influence on predicting the target word.
In our example, we see a higher attention weight value for the input word quickly to predict the target word, तेज़ी

Context Vector

The context vector is used to compute the final output of the decoder. The context vector c_i is the weighted sum of attention weights and the encoder hidden states (h_1, h_2, …,h_{tx}), which maps to the input sentence.

$$C_i = \sum_{j=1}^{Tx} \alpha_{ij} h_j$$ Context vector

Predicting the target word
To predict the target word, the decoder uses

- Context vector(c_i),
- Decoder's output from the previous time step (y_{i-1})and
- Previous decoder's hidden state(s_{i-1})

$$s_i = f(s_{i-1}, c_i, y_{i-1})$$

Decoder's hidden state at time step i

5) Model and Implementation

Amazon Fine Food Reviews**:** It consists of a collection of 500,000 reviews from Amazon users that is gathered in ten years. Each review with an average length of 75 words has a title written by the user. We can consider the titles as the summary of the whole review.
To build our model, we will use a two-layered bidirectional RNN with LSTMs on the input data and two layers, each with an LSTM using attention on the target data. The decoder contains two layers of LSTM with dropouts. This model uses Conceptnet Numerbatch's pre-trained word vectors.
Used Tensorflow for building the deep learning architecture.NLTK was used for preprocessing data.

6) Output:

Examples of reviews and summaries:

- Review(1): The coffee tasted great and was at such a good price! I highly recommend this to everyone!
- Summary(1): great coffee
- Review(2): This is the worst cheese that I have ever bought! I will never buy it again and I hope you won't either!
- Summary(2): omg gross gross

- Review(3): love individual oatmeal cups found years ago sam quit selling sound big lots quit selling found target expensive buy individually trilled get entire case time go anywhere need water microwave spoon know quaker flavor packets
- Summary(3): love it

7) Reference

- M. Banko, V. O. Mittal, and M. J. Witbrock, "Headline Generation Based on Statistical Translation,". In Proceedings of the 38th Annual Meeting on Association for Computational Linguistics, pages 318–325. Association for Computational Linguistics, 2000.
- A. M. Rush, S. Chopra, and J. Weston, "A Neural Attention Model for Abstractive Sentence Summarization,". In EMNLP, 2015.
- S. Chopra, M. Auli, and A. M. Rush, "Abstractive sentence summarization with attentive recurrent neural networks,". In North American Chapter of the Association for Computational Linguistics, 2016.
- R. Nallapati, B. Zhou, C. dos Santos, C. Gulcehre, and B. Xiang, "Abstractive text summarization using sequence-to-sequence RNNs and beyond,". In Computational Natural Language Learning, 2016.