Abstractive text summarization using Attention based Neural Network

1)Introduction

In the age of information, the huge amount of data that is produced every day will remain unuseful for humans unless making it available with new tools and technologies. Abstractive Text Summarization tries to get the most 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 appearance of the sequence to sequence models in the recent decade. These models are basically founded on a couple of recurrent neural networks that connect 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 huge potential in that it could help customers to make quick decisions on certain products. In addition, summarization could be applied to not only reviews but also 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 quite robust since they use existing natural-language phrases that are taken straight from the input.
- **Cons**: But 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's just like humans do.

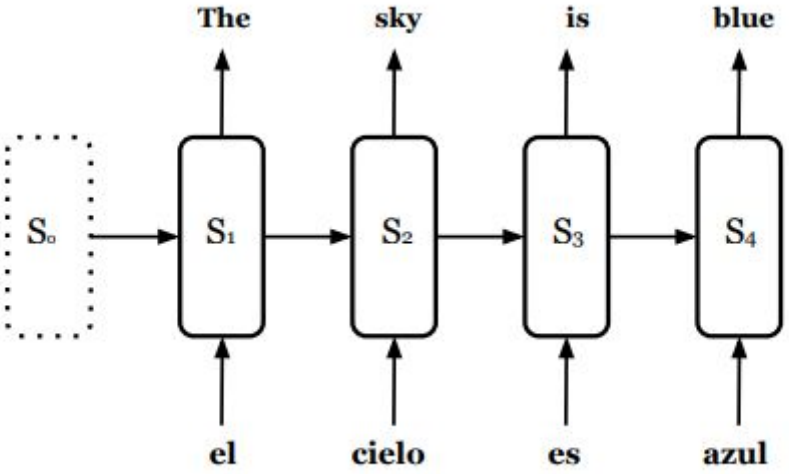
- **Pros**: They can use words that were not in the original input. It enables us to make more fluent and natural summaries.
- **Cons**: But it is also a much harder problem as you now require the model to generate coherent phrases and connectors.

3)Neural Text Summarization

Recurrent Neural Networks

An RNN tries to relate the elements of a sequence to each-other so that a network retain the memory of the previous data. To this end, RNN re-uses its output state and feed 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 inputs which are raw data inserted in the input layer. With forward propagation, the state of next 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 current hidden input to make the current hidden state. We repeat this recursive procedure for every and each of the elements of the input sequence.



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.Or 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.

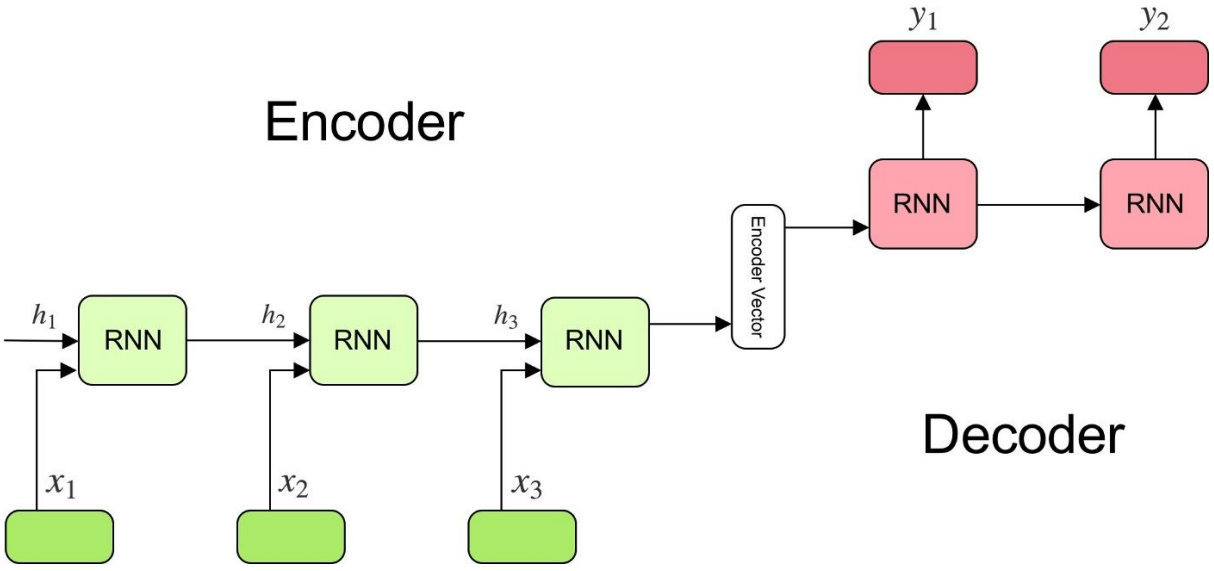
Long Short-term Memory

In each time-step an RNN loops back the state of its hidden layer. So in the next step, it has a memory of the past step. Imagine a sequence of length 10 as our input. In the 2nd step the RNN can remember the state 1. But 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 key 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 length of the input and output 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 is a collection of 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 just apply the appropriate weights to the previous 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 in order 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 where each predicts an output y_t at a time step t.
- Each recurrent unit accepts a hidden state from the previous unit and produces and output as well as its own hidden state.
- In the question-answering problem, the output sequence is a collection of 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.

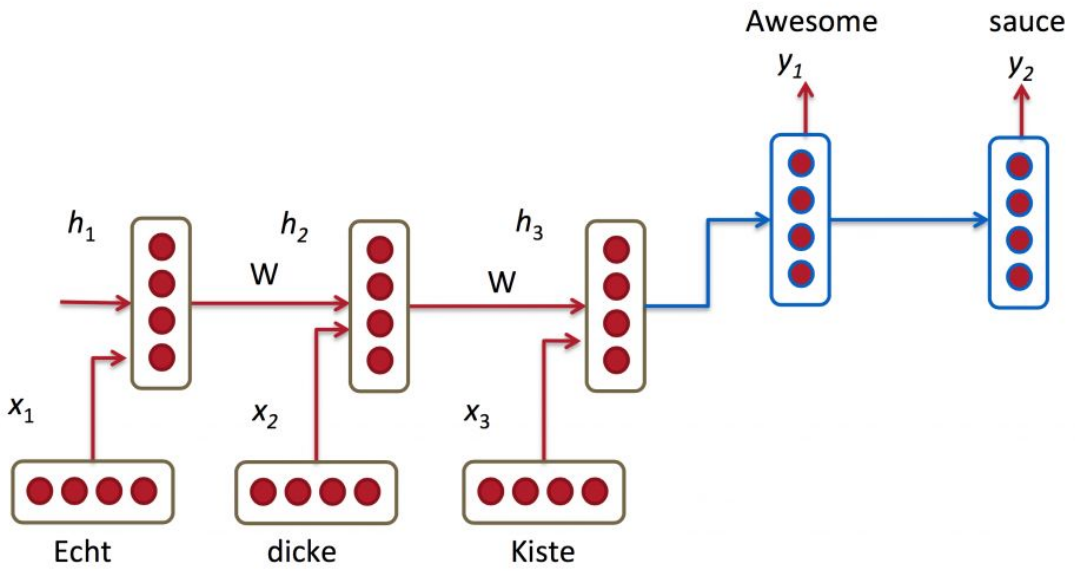
- The output y_t at time step t is computed using the formula:

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

We calculate the outputs using the hidden state at the current time step together with the respective weight W(S). Softmax is used to create a probability vector which will help us determine the final output (e.g. 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

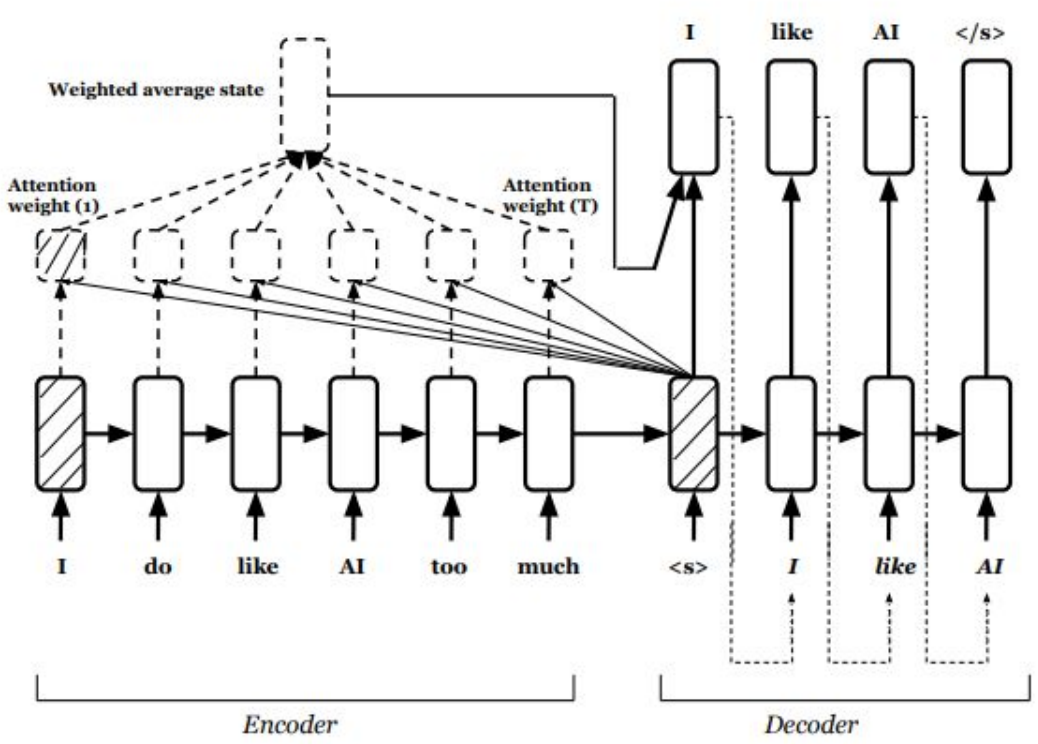
What problem does Attention solve?



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 h vectors represent the internal state of the encoder.

Here, The y‘s are our translated words produced by the decoder, and the x‘s are our source sentence words. The above illustration uses a bidirectional

recurrent network, but that's not important and you can just ignore the inverse direction. The important part is that each decoder output word y_t now 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.



5)Model and Implementation

Data:

Amazon Fine Food Reviews**:** It consists of a collection of 500,000 reviews from Amazon users that is gathered in a 10 year period. 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. 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