Ukrainian Catholic University

Applied Sciences Faculty

Data Science Master Programme

Abstractive text summarization with Recurrent Neural Networks

Machine Learning project report

Authors:
Hanna Pylieva
Yuriy Mykhalchuk
Irynei Baran

June 10, 2018

Abstract

Invent a meaningful abstract.

In this work, we cast text summarization as a sequence-to-sequence problem and apply the attentional encoder-decoder RNN that has been shown to be successful for Neural Machine Translation (NMT) [1].

1 Introduction

Invent a meaningful introduction. Per Shelpuk: Good project will be dedicated to an important problem and have a clear vision of what value it can bring to the potential users. All stages will be explored and analyzed, the approach for each of them is selected thoughtfully, compared to the alternatives and clearly explained. The results are evaluated and explained (explanation should provide additional information, not just restating the results or the code in English).

In the modern Internet age, textual data is ever increasing. According to this each Internet user will highly benefit from condensing data while preserving the information and meaning. This idea is a driver of growing interest among the research community for developing new approaches to automatically summarize the text. Automatic text summarization system generates a short summary that captures the main ideas of an input text. Since the advent of text summarization in 1950s, researchers have been trying to improve techniques for generating machine summaries which are not worse than human made summaries [2].

There are two prominent types of summarization algorithms.

- Extractive summarization copies parts of the source text through some measure of importance and then combines those part/sentences together to render a summary. Importance of sentence is based on linguistic and statistical features.

- Abstractive summarization generates new phrases, possibly rephrasing or using words that were not in the original text. Naturally abstractive approaches are harder as it involves robust natural language processing. For perfect abstractive summary, the model has to first understand the document and then express that understanding in succinct form possibly using new words and phrases. Abstractive summarization has complex capabilities like generalization, paraphrasing and incorporating real-world knowledge [3].

Majority of the work has traditionally focused on extractive approaches due to the easy of defining hard-coded rules to select important sentences than generate new ones. Also, it promises grammatically correct and coherent summary. But they often don't summarize long and complex texts well as they are very restrictive.

Abstractive methods, on the other hand, provide highly powerful and promising results. That is why in this project we implemented an algorithm for abstractive text summarization to build solid understanding if this approach and discover how it can be improved.

2 Model

2.1 Background

Models for abstractive text summarization fall under a larger deep learning category called sequence-to-sequence models, which map from an input sequence to a target sequence. This approach is illustrated on Figure 1. It was initially used for Neural Machine Translation as described in [4].

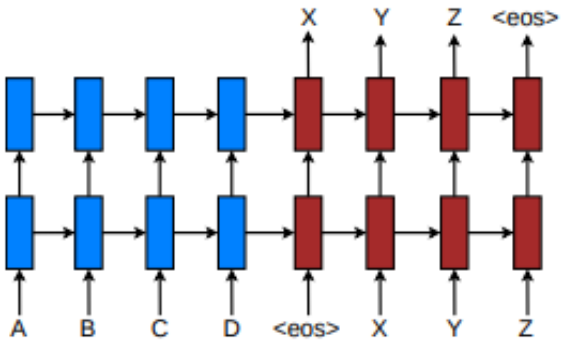

Image source: [4]

Figure 1: The model reads an input sentence "ABC" and produces "XYZ" as the output sentence. The model stops making predictions after outputting the <eos> token. It then starts emiting one target word at a time.

An effective and standard approach to build sequence-to-sequence models is using Encoder-Decoder architecture that act as an encoder and a decoder pair. The encoder reads the entire input sequence and encodes it into an internal representation, often a fixed-length vector called the context vector. The decoder reads the encoded input sequence and generates the output sequence. Both the encoder and the decoder submodels are trained jointly, i.e. at the same time.

Naturally we need the input and output to be of variable length. This can not be reached with ordinary Deep Neural Networks (DNNs) which require that the dimensionality of the inputs and outputs is known and fixed.

The Recurrent Neural Networks (RNNs) are generalization of feedforward neural networks to sequences. Given a sequence of inputs $(x_1, ..., x_T)$, a standard RNN computes a sequence of outputs $(y_1, ..., y_T)$ by iterating the following equation [5] :

$$\begin{aligned} h_t &= f(W^{hx}x_t + W^{hh}h_{t-1}) \\ y_t &= W^{yh}h_t \end{aligned} \tag{1}$$

where f - activation function, x_t - input vector, h_t - hidden layer vector, y_t - output vector, W - weight matrix

An RNN can map sequences to sequences provided that the alignment between the inputs and the outputs is known ahead. In other words RNN needs a defined one-to-one correspondence between a sequence of inputs and sequence of outputs which can be understood better from RNN unfolded representation on Figure 2.

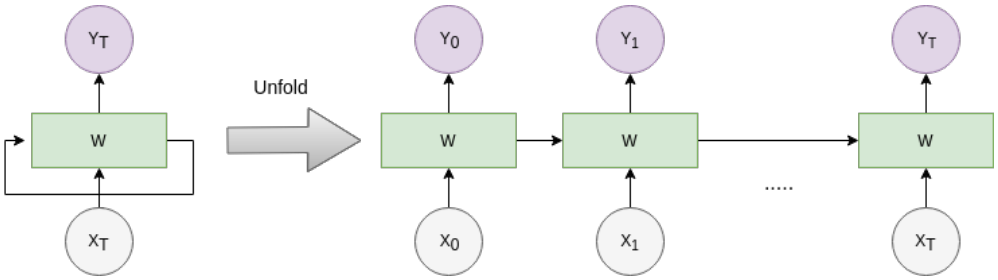

Image source: Created by author
Figure 2: Recurrent Neural Network

However, it is not clear how to apply an RNN to problems whose input and the output sequences have different lengths with complicated and non-monotonic relationships.

The simplest strategy for general sequence learning is to map the input sequence to a fixed-sized vector using one RNN, and then to map that vector to the target sequence with another RNN (the approach is represented in [1]). Since the RNN is provided with all the relevant information this will work. Whereas the need to store information over extended time interval will lead to difficulties with training RNN (vanishing/ exploding gradient problem explained in [6]). Here Long Short-Term Memory (LSTM) layers will come in handy as they are known to learn problems with long range temporal dependencies and will succeed in this setting.

2.1.1 LSTM

The goal of LSTM is to estimate the conditional probability of output sequence by input sequence $p(y_1, ..., y_{T'}|x_1, ..., x_T)$ when lengths of those sequences are possibly $T' \neq T$. An LSTM computes this conditional probability by first obtaining the fixed dimensional representation v of the input sequence $(x_1, ..., x_T)$ given by the last hidden state of the LSTM, and then computing the probability of $y_1, ..., y_{T'}$ with a standard LSTM-LM [1] formulation whose initial hidden state is set to the representation v of $x_1, ..., x_T$:

$$p(y_1, ..., y_{T'}|x_1, ..., x_T) = \prod_{t=1}^{T'} p(y_t|v, y_1, ..., y_{t-1})$$

In this equation, each $p(y_t|v, y_1, ..., y_{t-1})$ distribution is represented with a softmax over all the words in the vocabulary [5].

Taking into consideration all above and having processed prominent researches on abstractive summarization topic we've built an encoder-decoder recurrent neural network with LSTM units and attention mechanism to generate headlines from the text of news articles. We took the model described in [7] as baseline, but performed some modifications according to available software restrictions.

[1]Language Model (LM) is a probability distribution over sequences of words. LSTM-LM is an algorithm of using an LSTM (and softmax function) to predict the next word given your previous words.

2.2 Overview

In our model encoder and decoder get more precise descriptions.

2.2.1 Encoder

The input of encoder consists of article's text of restricted size followed by an end-of-sequence symbol (EOS, in code we used '$< eos >$') followed by header. Each word of input is first passed through an embedding layer which transforms it into a distributed representation. In our case encoder consists of 2 LSTMs, both on hidden layer. We run the stacked LSTMs over the sequence of vectors and store the last hidden state $v \in R^H$ - this is a fixed-size encoder representation, which will be used by decoder to generate the target sequence word by word.

2.2.2 Decoder

Our decoder consists of 1 LSTM cell. On each decoding step we feed v as hidden state and the word generated on previous step (EOS symbol on the first step) into decoder as input. Then a function $g : R^H \to R^V$ is applied so that $s_t := g(s_{t-1}, [v, y_{t-1}]) \in R^V$ is a vector of the same size as vocabulary ($s_0 := g([v, < eos >])$. Then we predict next word y_t in output sequence by applying softmax to s_j. Softmax function results in a vector of likelihood for each word in the vocabulary to be the next in the sequence.

Does this sentence easy to understand?

The task of decoder is to find the most likely output sequence for the input received and processed by encoder. Due to big vocabulary size the search problem through all the possible output sequences based on their likelihood is exponential in length of the output sequence and is NP-complete. That is why it is common to use approximate methods. We used beam search and were keeping in memory 10 best hypotheses (output sequences) to eventually opt for the best combination of words (Figure 3).

The described process corresponds to testing phase. On training phase instead of feeding in as input of new generative step the freshly generated word, the expected word from the actual headline is fed in. This leads to a disconnect between training and testing. To overcome this disconnect, during training we randomly feed in a generated word, instead of the expected word, as suggested in [8]. Specifically, we do this for 10% of cases. This approach is called "teacher forcing".

2.2.3 Attention

In the vanilla encoder-decoder architecture encoder reads the input sequence of words and compresses all the information into a fixed-length vector of hidden states. This can lead to significant information loss and inadequate summarization in case of long input sequences, because hundreds of words will be represented by several of them. Attention mechanism is aimed to fix this problem. It speeds up the learning and lifts the skill of the model on sequence to sequence prediction problems.

The attention mechanism is used when outputting each word in the decoder (Figure 4). For each output word the attention mechanism computes a weight over each of the input words that determines how much attention should be paid to that input word. The weights

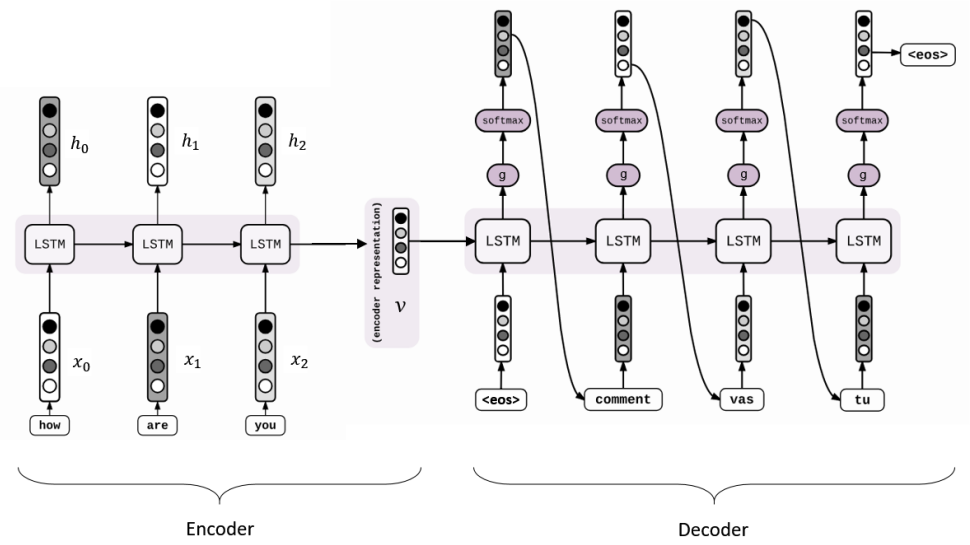

Image source: Created by author, inspired by
`https://guillaumegenthial.github.io/sequence-to-sequence.html`

Figure 3: LSTM-based sequence to sequence model on example of NMT. Note: v is actually fed on input of decoder on each step to prevent neural network from "forgetting" about it till the end of output sequence.

sum up to 1, and are used to compute a weighted average of the last hidden layers generated after processing each of the input words. This weighted average, referred to as the context, is then fed into the softmax layer along with the last hidden layer from the current step of the decoding. As shown on encoder's hidden states are weighted [7].

2.2.4 Training details

Google Collaboratory for each team member to contribute simultaneously. To store files - used one account. We reduce the sample iput size as otherwise the

3 Dataset

3.1 Overview

Where the dataset is from. What it contains.

3.2 Preprocessing

Start with reformating that strange JSON that theyy had. Then, all the stuff we do in vocabulary-embedding part

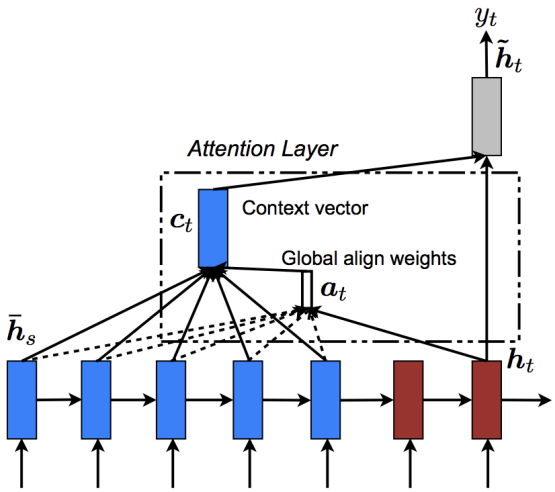

Image source: [4]
Figure 4: Global Attention in an Encoder-Decoder Recurrent Neural Network

4 Results

4.1 Evaluation

Which metric we used, the best BLEU we had

4.2 Results

What our model outputs , explain why result are 'not perfect'

References

[1] Bahdanau,D.; Cho,K.; Bengio Y. *Neural machine translation by jointly learning to align and translate*. CoRR, abs/1409.0473, 2014. `http://arxiv.org/abs/1409.0473`.

[2] Gambhir, M. and Gupta, V. *Recent automatic text summarization techniques: a survey*. Artificial Intelligence Review, 47(1):1-66, 2017.

[3] Singhal,S. and Bhattacharya, A. *Abstractive Text Summarization*. Department of Computer Science IIT Kanpur, 2017.

[4] Minh Thang Luong; Hieu Pham; Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. CoRR, abs/1508.04025, 2015. `http://arxiv.org/abs/1508.04025`

[5] Ilya Sutskever, Oriol Vinyals, Quoc V. Le *Sequence to Sequence Learning with Neural Networks*. CoRR, abs/1409.3215, 2014. `http://arxiv.org/abs/1409.3215`

[6] S. Hochreiter and J. Schmidhuber. *Long short-term memory*. Neural Computation, 1997.

[7] Lopyrev, K.*Generating News Headlines with Recurrent Neural Networks*. CoRR, abs/1512.01712, 2015.

[8] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. *Scheduled sampling for sequence prediction with recurrent neural networks*. CoRR, abs/1506.03099, 2015. `http://arxiv.org/abs/1506.03099`