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

In this work, we describe the process of resolving abstactive text summarization task as sequence-to-sequence problem. We trained encoder-decoder LSTM-based model with attention mechanism for generating captions of news articles.

1 Introduction

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].

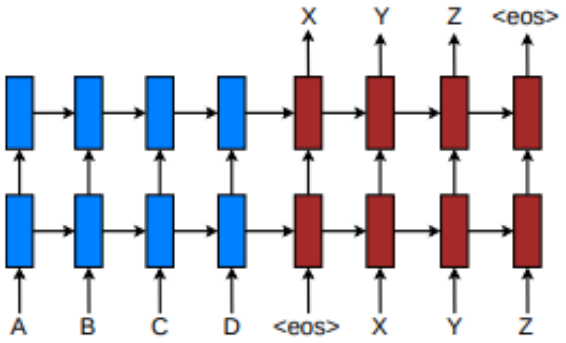

Image source: [4]

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.

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

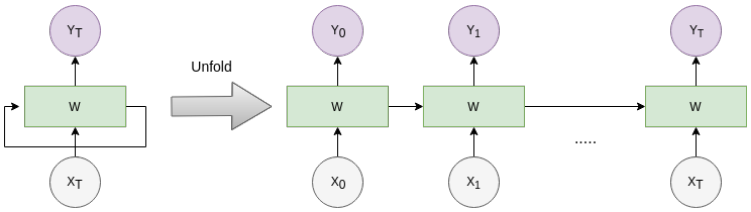

Image source: Created by author
Figure 2: Recurrent Neural Network

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. We used the next repository as a reference point of how to build the whole process: `https://github.com/udibr/headlines`

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. We run the stacked LSTMs over the sequence of vectors and store the last hidden state $v \in R^H$

[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.

- 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

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 $g : R^H \rightarrow R^V$ 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. The softmax function results in a vector of likelihood for each word in the vocabulary to be the next in the sequence.

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).

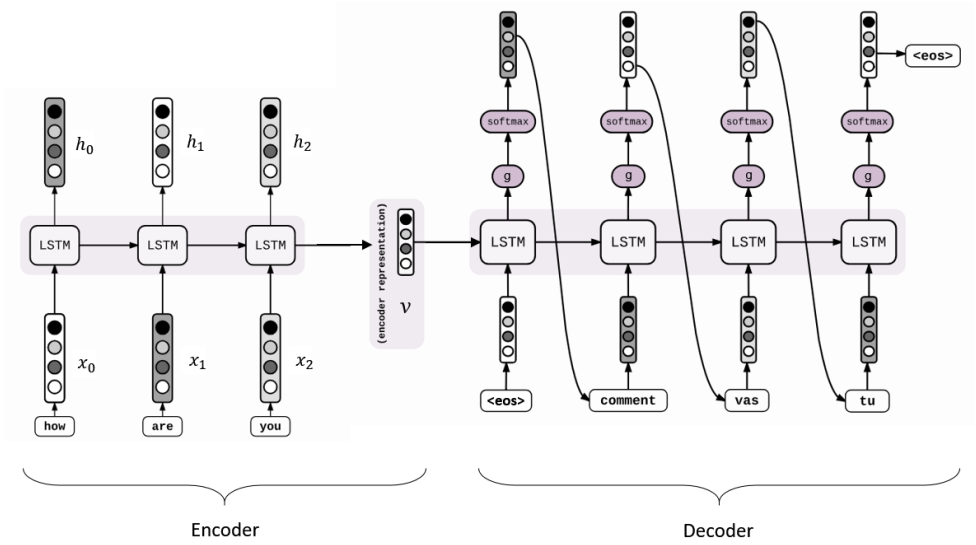

Image source: Created by author, inspired by
`https://guillaumegenthial.github.io/sequence-to-sequence.html`

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.

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 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].

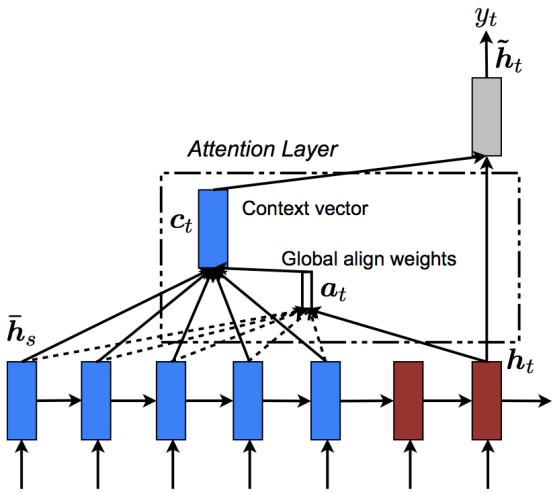

Image source: [4]

Figure 4: Global Attention in an Encoder-Decoder Recurrent Neural Network

2.2.4 Training details

We used Google Colaboratory to process data, train and text model. Colaboratory is a research tool for machine learning education and research. It is a Jupyter notebook environment that requires no setup to use. Colaboratory is free and allows a group of users edit one Jupyter notebook simultaneously. Google Colab has 12Gb free RAM and allows using one Tesla K80 GPU with Keras and Tensorflow.

Our solution is implemented on Python3, the model is built on Keras with Tensorflow backend.

According to RAM limitation and politics of Google Colaboratory to switch off processes which run for mote than 12 hours (to prevent cryptocurrency mining on the tool), we had

issues during training as it was insufficient for our purposes. Mostly the models like the one described are run during 3-8 days in case of one GPU as they use a big input dataset and deep LSTMs. As we had only 1 GPU, the maximum model characteristics which allowed us run at least one epoch in a time without fails were the following:

- Maximum length of description (in words): 25 . Description is article. So we took only 25 first words from each article.
- Maximum length of header (in words): 25. Header is caption of article.
- Number of RNN (LSTM) layers: 3
- Dimensionality of output from RNN: 128
- Batch-size: 64
- Vocabulary size: 40000
- Size of input dataset(number of articles): 100000 (10 times less than the size of dowloaded dataset from the source)
- Learning rate: started with 0.01, then changed to 0.0001

As we experienced "Runtime died" frequently, our model saves weights after each epoch. Each epoch took up to 1 hour a The best loss we reached on validation set was 7.8181 (started from 20.0). We managed to run our model for at most 8 epochs without failures.

3 Dataset

3.1 Overview

We used The Signal Media One-Million News Articles Dataset. It contains 1 million articles that are mainly English, but they also include non-English and multi-lingual articles. Sources of these articles include major ones, such as Reuters, in addition to local news sources and blogs.

The file is in JSONL format, where each line is a JSON object representing an article. Each article has the following fields:

- id: a unique identifier for the article
- title: the title of the article
- content: the textual content of the article (may occasionally contain HTML and JavaScript content)
- source: the name of the article source (e.g. Reuters)
- published: the publication date of the article
- media-type: either "News" or "Blog"

We extracted only "content" and "title" fields, then placed them into .pkl to make reads more convenient.

3.2 Preprocessing

During data preprocessing we did the following:

1. Convert the original file into standard JSON format and leave only "content" and "title" fields.
2. Leave only two sentences in descriptions
3. Tokenize data using nltk's Punkt tokenizer
4. Get vocabulary and words counts of all words in dataset
5. Index all words in vocabulary Create mappings from words to indexes and vice versa
6. Read GloVe embeddings (with dimetionality of vectors 100), set embeddings for lowercased words the same as for original ones if the lowercased version has no embedding
7. Initialize embedding matrix for 40k the most popular words uniformly at random
8. For each word among those 40k which has embedding in Glove set this embedding to an appropriate row in the embedding matrix
9. For all words which are not the most popular and are present in Glove find its embeddings
10. For embeddings from the previous step find the nearest embedding from the most popular words and assign it to this non popular word (this helped to find embeddings for 187k additional words)
11. Replace words with their indices in headlines and descriptions
12. Write indexed headlines and descriptions into .pkl with Pickle
13. Write embeddings, mappings between words from our vocabulary and their indices, mappings between words from our vocabulary and words from Glove to files

4 Results

4.1 Evaluation

The performance of the model was measured using BLEU [9] evaluation metric over the holdout set. The BLEU evaluation metric looks at what fraction of n-grams of different lengths from the expected headlines are actually output by the model. It also considers the number of words generated in comparison to the number of words used in the expected headlines.

4.2 Analysis

The examples of meaningful headers generated by our model are represented in Table 1.

There are a lot of nonsense headers which our model generates, so there is obviously a big area for improvement of our model. Training on physical GPU or more reliable Cloud than Google Colaboratory for more epochs should improve the results significantly.

What we've mentioned for now is that our model likes to put out-of-vocabulary words in caption. For example, one of generated headers for the third example in table was "Business

Table 1: Example predictions

Text	Actual Headline	Predicted Headline
Sydney-based strategic integrated communications agency, Zadro, has moved to the next level; due to the business expansion, they are commencing a new chapter in their growth by relocating to a bigger, better and brighter office space in Surry Hills. Felicity Zadro, Managing Director, Zadro, says the move is indicative of the way the agency has flourished since its inception eight years ago.	Integrated communications agency, Zadro, moves to the next level.	Launches in Sydney-based Zadro.
Shares of Toronto-Dominion Bank (NYSE: TD) have been given an average recommendation of Buy by the eleven research firms that are currently covering the firm, Market Beat Ratings reports. One equities research analyst has rated the stock with a sell rating, three have assigned a hold rating and six have given a buy rating to the company.	Toronto-Dominion Bank Receives $56.75 Average Target Price from Analysts (NYSE: TD)	Report of Products Market Toronto-Dominion.
Microsoft has acquired Double Labs, an Android app startup which develops the popular Echo Notification Lockscreen for Android devices. This app had received between 1 million and 5 million downloads so far in Play Store.	Microsoft Acquires Popular Android App Echo Notification Lockscreen	Lockscreen Management Announces

Cash Lockscreenˆ the : Lockscreenˆ and (to and Lockscreenˆ) , 2015 Nikhil after key issued price Agency". Such behavior is caused by high attention on such words. Also model does not recognize a multy-word entities, i.e. in the same example we have "Echo Notification Lockscreen" entity, but in headers we mostly only "Lockscreen" appears as "echo" and "notification" are known words.

What is important to note, our model doesn't rephrase the ground truth headline, but generates its own.

Our code for data preprocessing, model testing and training can be found [here].

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

[9] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu.*BLEU: A method for automatic evaluation of machine translation*. In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, ACL '02: 311–318, Stroudsburg, 2002.