MASTER THESIS IN ARTIFICIAL INTELLIGENCE

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Abstractive Text Summarization with Attention-based Mechanism

Nima SANJABI

Supervisors:

Marta R. COSTA-JUSSÁ

&

Lluís PADRÓ

Abstract

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 is 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). The recently proposed model which is called *Transformer* uses only the attention mechanism. It has surpassed the previous state of the art models in translation tasks. In this work, we want to experiment this new model, the Transformer, for abstractive text summarization. We will discuss its advantages and shortcomings in comparison with the available models.

Acknowledgements

I would like to appreciate my thesis adviser, professor Marta R. Costa-jussá for the inspirations and motivations that I have received from her, and for her great guidance and generous support throughout this study.

Contents

I dedicate this work to my mother, my father, and my sisters who have always supported me and without their constant encouragement during the years of my studies, this current work was not possible.

Chapter 1

Introduction

Living in the information age has changed our environment. We are surrounded by data, not in a metaphoric sense; it is a physical reality. Internet, television and radio signals are circulation in the air and we receive them by our mobile devices everywhere. Data servers in counts of millions have connected all the planet like a spider web. Only Google has more than 1 million server around the world that are online 24/7, storing new generated data. Faster telecommunication, higher bandwidths, more advanced hardware along with Web 3.0 has led an exponential growth of digital data. A considerable portion of this stream of new data is textual. No wonder that web can be named the number 1 biggest resource of textual contents, as it is cover almost most of the available text corpus, like books, journals and newspapers, e.g. in online libraries, it is also connected to millions of organizations, companies, universities and research centers, and individual users that are creating content everyday in the websites, blogs, social networks etc. While the world is covered with computers which are communicating with their own language we are still humans with biological brains . What can we do? How can we deal with living in this new world?

These information is left unusable unless we find a way to make it available for users. The idea is to decrease the volume. *Automatic Text Summarization* is an attempt to decrease the size of a corpus while keeping its valuable content.

Mainly the available methods of text summarization are in either Extractive or Abstractive. In the Extractive way the system chose a few sentences from a source corpus, as opposed to the abstractive way that breaks the corpus contents and decompose them again in a summary. Indeed the abstractive summarization is very close to how a brain receive process and regenerate information. As a result it is very difficult to design and replicate an abstractive algorithm for automatic summarization. That is why in the birth of the field around 1950s, most of the methods

were fitting in the extractive method the available algorithms, tools and resources were not sufficient to make an abstractive summarization systems.
To approach a problem one can try to solve it directly, by designing algorithms, using available methods, techniques and tools. This is like the disciplines that are used in Engineering. One might try to solve the problem indirectly. Indeed he can use the available solutions and mimic them. It is very close to Reverse Engineering. It is also the procedure of discovering the systems and mechanisms that are already available in the Nature and benefit them for our personal (as referred to any human) purposes.
In the Text Summarization terminology, the original summaries that are created by a human and are used as the metric for reassuming the accuracy of the summarizer system, are called the Golden Set. Let us look at the problem from another angle. If the main criterion for creating a good summarizer system is it being as good as a human, why we do not mimic a human approach in the first place? By this manifest, a Neural Text Summarizer has the potential to be the ultimate accurate text summarizer if we make it learn from a human style as accurate as possible. In this work, we will explore the evolution of sequential neural models from the very first RNN models to the transformer and their performance as a summarizer system. Transformer is the recent proposed neural model that has achieved state of the art results in translation task. Its applications as a summarizer system is still unexplored. We will test the transformer on two different datasets each with its own characteristics. Comparing the results with a baseline model will clearly explain the behavior of transformer model on sequential data and on summarization task.

Chapter 2

Text Summarization

Have you ever read a packet-size travel book? These kind of books are usually smaller than their original source so that a person could read them in a relatively shorter amount of time e.g. in a flight trip. But considering the smaller size of a book, does it contain all the contents of its source? Can we extract the same amount of information from both of them?

Text summarization is the process of shortening a corpus of text to a few sentences that together hold the most essential informations of the original contents in the source corpus. The source can be any text document, a book, a journal article or a radio message. The small size of a corpus makes it possible to read it easier and grasp its contents in a shorter time. So time efficiency is one of the advantages of text summarization. But it is not always about the time. When you summarize a text, you try to keep the core meaning and trim the redundant details as much as you want. A summary, hence, is more clear and more transparent. It prevents the confusion and misunderstanding; another reason to use a summary. We will explain all these characteristics of a summary with examples. But let us think about the other motives of doing summarization. So far we named time efficiency and transparency. The third lies in the concept of cost. Although in the first characteristic, time efficiency, the cost can be interpreted as the amount of time we need to read a text, the cost we focused on is for transferring. A text corpus is a packet of information. The conventional mediums to transfer a text are papers, books, letters, newspapers etc. It makes sense to say more amount of text need more paper and therefore is more costly; a linear relation. Sometimes the cost is much higher, when the resources are limited i.e. text lengths are shorter, like When you want to send a vital message. A good old example is a telegraph. Summarization is more valuable when telecommunication cost is higher.

A summary should maintain the most valuable information of a text corpus and

remove the unnecessary details. Storage limitations is one other important factor to trim the redundant data.
In brief these are the benefits of using a summary:

1. Time Efficiency.
    - Getting the idea of a text quickly (e.g. news headlines)
    - Reading a document faster (Medical Docmument Summarization (Sarkar, Nasipuri, and Ghose, 2011))
2. Clearness and Transparency
    - Removing the redundant data to guide reader to the point (e.g. user comment summarization)

2.1 Real-world Applications

In this work by *Text Summarization* we mean *Automatic Text Summarization*. Counting the available resources of textual data, surely we will find out that the internet, as the global network of billions of machines and interactive users is the biggest resource available on earth. Thus, no wonder if we find most of the applications of text summarization on the web. Therefore we divide the applications in two main categories: the general applications of text summarization. web applications of text summarization.

2.1.1 General Applications of Text Summarization

- Document summarization
- Multi-document summarization
- Information retrieval

2.1.2 Applications of Text Summarization in Web

- News snippet generation
- Searched entity description (e.g. Biography summarization)
- Title generation (news, articles, blogposts)
- User comment summarization

2.2 Methods and Technologies

2.2.1 Extractive Methods

In the Extractive methods, a summarizer tries to find and combine the most significant sentences of the corpus to form a summary. Their are some techniques to identify the principal sentences and measuring their importance namely *Topic Representation*, and *Indicator Representation* (Allahyari et al., 2017). In the Topic Representation technique, we assume a given *topic* for the text corpus. The summarizer explores the corpus to find the words that are closer to the topic and measures their frequency to score the importance of the sentences. One common frequency-based technique to measure the relevance of a term to a corpus (here that term is the topic) is TF-IDF. TF-IDF which stands for Term Frequency-Inverse Document Frequency works based on two disciplines:

- measuring the frequency and importance of a *term* in the current document (term frequency)
- compare the term frequency with the frequency of that term in whole the corpus (document frequency)

Calculations of the frequency are quite straightforward and are done by counting the words in the target document. To calculate their importance, we scan the other documents to check the presence of the word. If it is also frequent in other

documents, like the word "the"[1] that is frequent in any corpus, and therefore in not very informative and important.

2.2.2 Abstractive Methods

Abstractive Text Summarization (ATS) is the process of finding the most essential meaning of a text and re-write them in a summary. The resulted summary is an interpretation of the source. Abstractive summarization is closer to what a human usually do. He conceives the text, compares it with his memory and related information, and then re-create its core in a brief text. That is why the abstractive summarization is more challenging than the extractive method, as the model should break the source corpus apart to the very tokens and regenerate the target sentences. Achieving meaningful and grammatically correct sentences in the summaries is a big deal that demands highly precise and sophisticated models.

[1]The word "the" is the most frequent English word. Based on zif's law, this word with rank 1, has covered 7% of all the available English content. For example, this current document must have the same distribution. Zipf's law reveals a fundamental property of all the languages that the rank of every word is proportional to its frequency.

Chapter 3

Neural Text Summarization

3.1 How Our Brains Process Sequential Data

We all can remember a few lines of the poems that we used to sing around our childhood. Maybe some pieces of a novel, or a speech. These are all sequential data. A number of words in the range of tens to hundreds, and we can recall them exactly like what they were in the first place. I can remember the area of my country very well since the primary school days. It is a 7-digit number that has been saved in my memory for many years. Also a few phone numbers, my university ID number, and so on. For example, the phone number of mine that have preserved in my brain for more than 15 years is an eleven digit number. My brain recalls it in a fraction of a second, fast and accurate. But a question: can I tell my phone number backward? Or can I recall the last 5 digits? What about reciting English alphabets backward... It is very difficult if not possible. Why is that?

[1] A B C D E F G H I J K ...

$\longrightarrow$

[2] Z Y X W V U T S R Q P ...

Reciting alphabets backward [2] is not as easy as saying them in forward pass[1] because we memorize alphabets in a forward positional order.

Reciting alphabet backward is difficult. There have been long studies on the mechanism of memory formation in Neuroscience. We have a vague picture of it, but beside the total mechanism as a whole, many pieces of evidence suggest that our brains process and memorize sequential data differently. Basically what differs a sequence from random data is order. This work by Manohar, Pertzov, and Husain,

2017 discussed these ordered chunks of information. The *positional memory* in a sequence can be a *spacial* or a *temporal* order. In the first case, the information is bound to the location of stimuli. An example is remembering pages of a book in the order that have been seen. For example, some teachers advise their students to use a paper book. Because the content is ordered, and the user can memorize better by relating consecutive pages. The temporally ordered sequences are more popular because they are more prevalent in speech, text, music or even movements of a dance.

3.2 Recurrent Neural Networks

The invention of *Perceptron* back in 50's by Frank Rosenblatt was controversial. It was the ancestor of today's gigantic Neural Models, a single neuron inspired by neurons in the brain. That magical model was expected to be "the embryo of an electronic computer that expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence." Olazaran, 1996 Despite the fact that the perceptron was not able to walk and talk, but Neural Network has not been abandoned forever. The later architectures achieved better performance by using a combination of perceptrons in connected layers.
Among the revolutionary models, around the years of the 1980s inspiring by the sequential memory mechanism of the brain, Recurrent Neural Network (RNN) was proposed. An RNN tries to relate the elements of a sequent 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 (Medsker and Jain, 2001). 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 to incorporate the concept of memory, pick the state of the hidden layer combines it with the current hidden input to make the current hidden state. Figure 3.1 illustrates how the hidden state of the previous time-step combines with the input to make the new hidden state. We repeat this recursive procedure for every and each of the elements of the input sequence. To understand better you can imagine a network with as many hidden layers as the elements of the sequence. This way of depicting the RNNs is called *Back-propagation Through Time (BPTT)*.

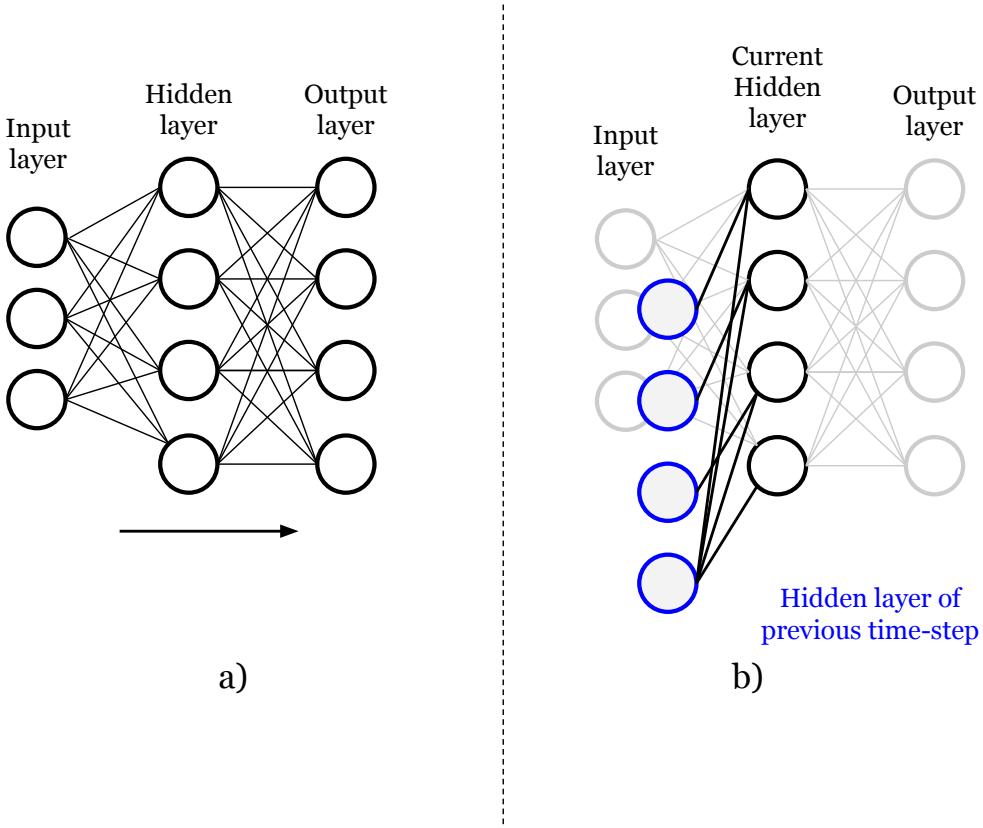


FIGURE 3.1: a) Forward-propagaton in a Neural Network. b) Feeding previous hidden layer state to the current time-step hidden layer (for clarity just some of the connections are drawn)

$$s(k+1) = f(s(k), x(k)) \tag{3.1}$$

$$y(k) = h(x(k), s(k)) \tag{3.2}$$

In equation 3.1 *x(k)*, *y(k)* and *s(k)* are input, output and state of the hidden layer in time step k respectively. Equation 3.2 calculates the output in each timestep, where h is the *model hypothesis* (Medsker and Jain, 2001).

In a conventional feed-forward Neural Network (NN) we map an input vector from the input layer to the output classes on output layer. The net then can learn this mapping for future predictions with back-propagation of the error and updating the

weights. RNN uses the same setting for placing the input and output. The only difference is that the weights of network update for each element of the sequence, while the input is combined with the previous hidden state. NN maps a pair of input and output, and RNN do in with a pair of sequences. We this feature we can train an RNN add two sequence of numbers and map them to their sum. Another application is translation which is very popular. In Figure 3.2 an Spanish sentence is mapped to its translation in English. To clarify more, sentence enters the RNN with its first word from the left.

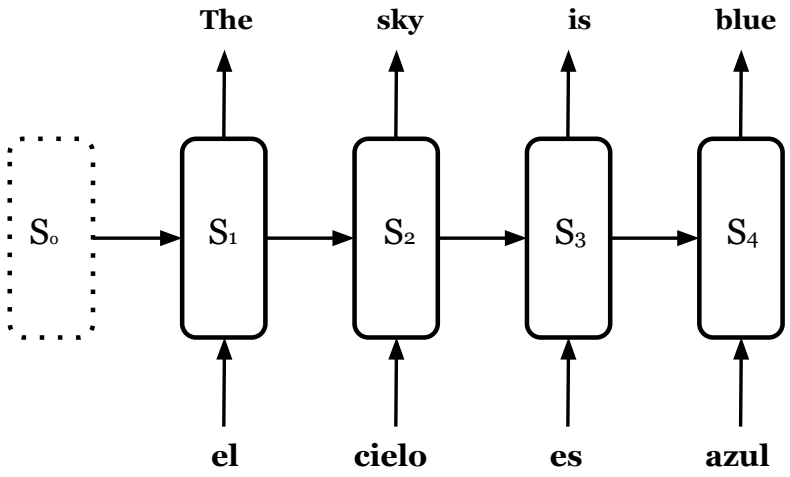


FIGURE 3.2: Propagation Through Time and mapping two text sequences in an RNN. S_0 is the initial state that could be a vector of random values.

3.2.1 Bidirectional Recurrent Neural Networks

AN RNN processes the data in timesteps. However here *"time"* is not necessarily the physical time. It only indicates that the sequence is series of a continuous sampling of a variable. The sequence might be completely time-independent e.g. a DNA sequence. RNN scan the elements of a serie to find their relation. Are they bound together and appear in a pack so that they can form a frequent pattern in the data

set? This scan has a direction and it is only backward. RNNs look at the previous timesteps to find the bounds and relations. Or in other words they have a memory of the past. One can think of an element in a sequence that is not only related to the past, but To the next step, or, the model can predict the future step (compared to memorizing 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. In the example below, the element "B" in sequence (1) and (2) comes after "A". It also comes before (C) in both sequences. Having this information gives us a more robust model. With this approach, the information we have about each element and the power of our predictions is doubled. Experiments on BRNN proved their better performance, faster learning and easier development of applications (Schuster and Paliwal, 1997)

Sequence (1): x x A B C D x x
Sequence (2): x x A B C F x x

Using arrays of bidirectional LSTM cells completes this architecture furthermore. We will study LSTMs in the section 3.4

3.3 Sequence to Sequence Models

3.3.1 Memorizing the Order

In the past section, we saw how an RNN can learn to map two sequences to each other. By converting the words to vectors, exploiting word embeddings, (Elman, 1991), we can use RNNs for text-to-text problems. For an instance, we used RNN for Spanish-English translation in the previous section. The reader should consider that the *text-to-text* title might be misguiding. In fact, the models like RNN that process sequential data, have no preference over the type of input information whether it is text or speech or a series of numbers recorded from a chain of events e.g. fluctuations of stock prices. They only learn the patterns of data and their order. Learning order is the very characteristic of these models. For example, an RNN can learn the difference between "FC Barcelona won Real Madrid," and "Real Madrid won FC Barcelona" while a simple feed-forward network is not necessarily able to do that.
Neural Text Summarization (NTS) is a special type of Text-to-text problems where

the source text sequence is much larger than the one in target. So a Summarizer Model maps a pair of long and short sequences. With this considerable difference in the lengths, putting RNN models in text-to-text problem will not provide good results. It is the bottle-neck of RNN models, unless we clip the longer sequence or pad the shorter one (Cho et al., 2014). There have been a few efforts to solve this problem. The most important one was attaching two RNNs in a row in a encoder-decoder structure. It was a genuine idea that finally led to the birth of seq2seq models, however, it was suffering from the of long-term dependency problem. As a result, despite the use of RNN models in translation tasks, where the source and target texts are aligned better, the special case of summarization problems needs a new generation of neural models.

3.3.2 Seq2seq Model Architecture

In short, a Seq2seq maps 2 sequences to each other that are not necessarily in the same size, in two steps: Compressing the first sequence, and then inferring the output from it. This architecture has two side named encoder and decoder that are both LSTM layers.[1] Encoder receives the input data step by step. At each time step the state of the hidden layer is looped back and combined with the input data. At the end of this procedure, the hidden layer of last time step holds a state which has been affected by all the elements in the sequence or has retained the memory of the whole sequence in a single layer. The name of *Encoder* originates here, because it encodes a long sequence to the state of a hidden layer which is a vector. e.g. a 512-dimensional vector.

Assume we use the model for translation. The encoded version of the input sequence is a representation of its information and its principal patterns. Think of it as a compressed memory of the whole elements of the input. To make the decoder guess which translation matches with source sentence, we put the hidden state from encoder to the first timestep of the decoder. This way we can train decoder with the presence of the encoder's content, and we start it with Finally the decoder will acquire a set of weight that will generate a correct translation with the pretense of hidden state of the encoder (the memory of source sentence) in its layers.

[1] Encoder and decoder can be multilayer LSTMs. We can attach two or more parallel RNN layers just like simple Multi-layer feed-forward networks. Imagine the RNN arrays in X axis which is the timestep, and at each step multiple arrays on Y axis

$$p(y_1, ..., y_{T'}|x_1, ..., x_T) = \prod_{t=1}^{T'} p(y_t|s, y_1, ..., y_{t-1}) \tag{3.3}$$

The equation 3.3 explains the probability distribution of output sequence, given the input sequence (Sutskever, Vinyals, and Le, 2014). T and $T^{'}$ are the length of input sequences x and output sequence y respectively. s is the state of the last hidden layer in the encoder. Figure 3.3 we show an encoder-decoder sequence2sequence model. Our choice for RNN model was an aligned pair of the source text sequence in Spanish and its translation in English. Here we examine a summarization. To see how the model can infer the output from the input sequence, follow the steps below:

1. encoder receives a sample sequence from the source dataset.
2. by using the hidden state of last time step as the representation of input sequence, the encoder converts the input to a vector.
3. we put that vector in the first time step of the decoder.
4. Having the hidden state of the encoder in the first timestep of decoder, it gives us an output, which is a vector. We find the closest word vector in our dictionary to the output word vector. To measure how the output of that timestep matches the target word we can calculate a score like equation 3.4 from their multiplication. s_t and v_t are the hidden state of decoder and output layer's weight vector in timestep t.
5. Now that we have the scores of outputs, we can calculate the probability of every output word. We use a normalization technique called *softmax* in the equation 3.5. $e(w_i)$ is the score of the current output word (Mikolov et al., 2010). Note that just like regular RNNs we loop back the hidden state and combine it with the output, with an important difference; here the input word in each time step is the output of the previous timestep as in illustrated by dot-lines in the Figure 3.3.
6. in each the model select the most probable word in the output and put in in the input of the next layer. The algorithm will stop after entering the </s> sign which indicates the end of the sequence is reached.

$$e(w_t) = s_t v_t \tag{3.4}$$

$$p(w_i|w_1, ..., w_{i-1}, s_t) = \frac{\exp(e(w_i))}{\sum_j \exp(e(j))} \tag{3.5}$$

To train this model we use the same BPTT. The same technique that we used for RNNs. By each training example, the models-output and target sequence are compared. The error is sent back through hidden layers in time steps. We calculate the cost function based on accumulated error for all the training example, and the first derivative of the cost function with respect to all the weights. Then by using an optimization technique like Stochastic Gradient Descent (SGD) we try to minimize the cost in each mini-batch and update the parameters (weights) of the model.

3.4 Long Short-term Memory

RNNs were a breakthrough in the field of Neural Networks. They opened the NNs to sequential data and introduced the concept of memory. But they had a few shortcomings. We discussed that they are not able to learn unbalanced pairs of sequences in which the target length is different from the source. Even with the encoder-decoder configuration, they had very limited usage because of the lack of long-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 the 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. Their

benefit is that they can learn long-term memories. To understand the short and long-term memories consider this two sequences below:

Sequence (1): x x x A B x x
Sequence (2): x A x x x B x

In sequence 1, the character "B" appears immediately after "A" (x elements could be any character). "AB" is a pattern in our sequence. By feeding more examples of data samples that include this pattern, the RNN memorize that after "A" there must be a "B". It is a short-term memory. So if the model receives the "AB" pattern again it will remember its order and can predict the corresponding output.
In sequence 2, the character "B" comes 4 time-steps after the character "A". A simple RNN directly copies the internal state of the previous state and combines it with the input. If the model receives new data during this 4 steps, they will destroy the memory of the "A" character. To learn a long pattern like "A x x x B" we need a new mechanism.
LSTM cells are rather complex structures based on RNNs (Hochreiter and Schmidhuber, 1997). We just explain two main parts of them that are two handles for keeping or forgetting the memory. A line that all the states can freely pass across it till the last timestep, and a forget gate. The forget gate itself is controlled by a 1 simple feed forward network, usually one layer that is fed with the previous hidden state. By training these gates, the LSTM will learn when to keep, and when to forget. It all depends on the pattern of the input data. For any input pattern if it is more frequent in the dataset, network will learn to keep its memory. LSTM, unlike the RNN, does it regardless of the length of pattern, long or short term memories will retain if they are more frequent.

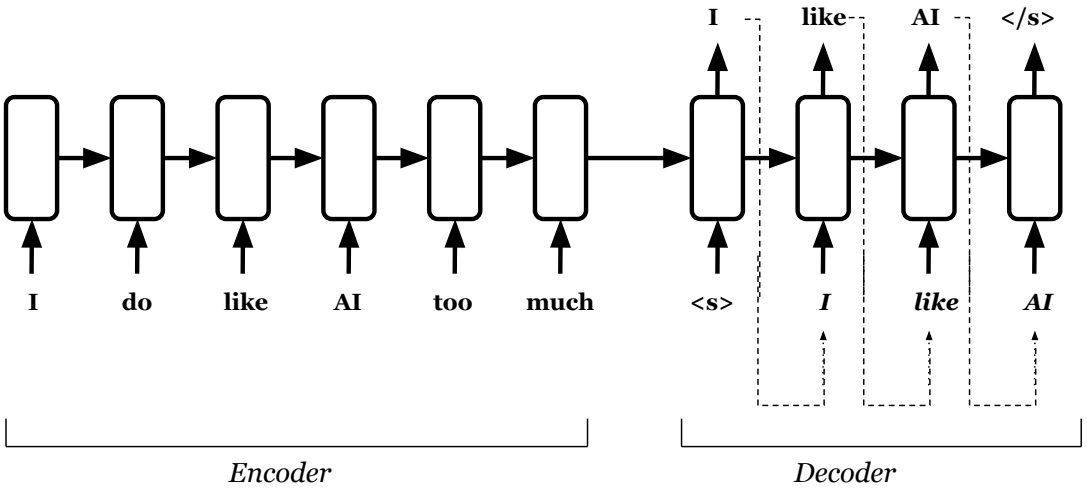


FIGURE 3.3: Abstractive text summarization by mapping unaligned text sequences in a Seq2seq model

Chapter 4

Attention Is All You Need

4.0.1 Improving Seq2seq Model with Attention Mechanism

The performance of sequential models improved to a higher extend by when they equipped to the attention mechanism. A seq2seq model is combined with an encoder and a decoder. The decoder receives all the contents of the encoder compressed in one vector and trains the target with its presence. It is a very clever idea, however, when the source sequence is too long, its content will fade in the coded vector. It makes sense to think that hypothetically the average vector of two big enough corpus are very similar to each-other. To solve this problem, attention mechanism tries to get multiple encoded vectors from the source and not only one. This way with a very long sequence, the seq1seq model can align the source and target sequences, while retaining the most important contents of the source. Figure 4.1 shows how a seq2seq model focuses on the different parts of the source sequence. The procedure is as below: The encoder process the input sequence till the last element. Decoder copies the hidden state of the encoder in the last step to its first timestep. It compares this new state with all the hidden states of the encoder to find their similarity. The hidden states which are more similar will get a higher weight. The decoder then uses the weighed average states of the encoder and combines it with its current state. With the new hidden state, decoder can infer the first output. It repeats this procedure for all the target timesteps. The rest of procedure is like a conventional seq2seq model.
The alignment of source and target sequences can be illustrated with on a matrix. Figure 4.2 shows a sample of this alignment the first elements (I) has the most similarity, therefore will get the highest attention weight, while the "I" and "AI" has no semantic correlation, so the comparison weight is close to zero.

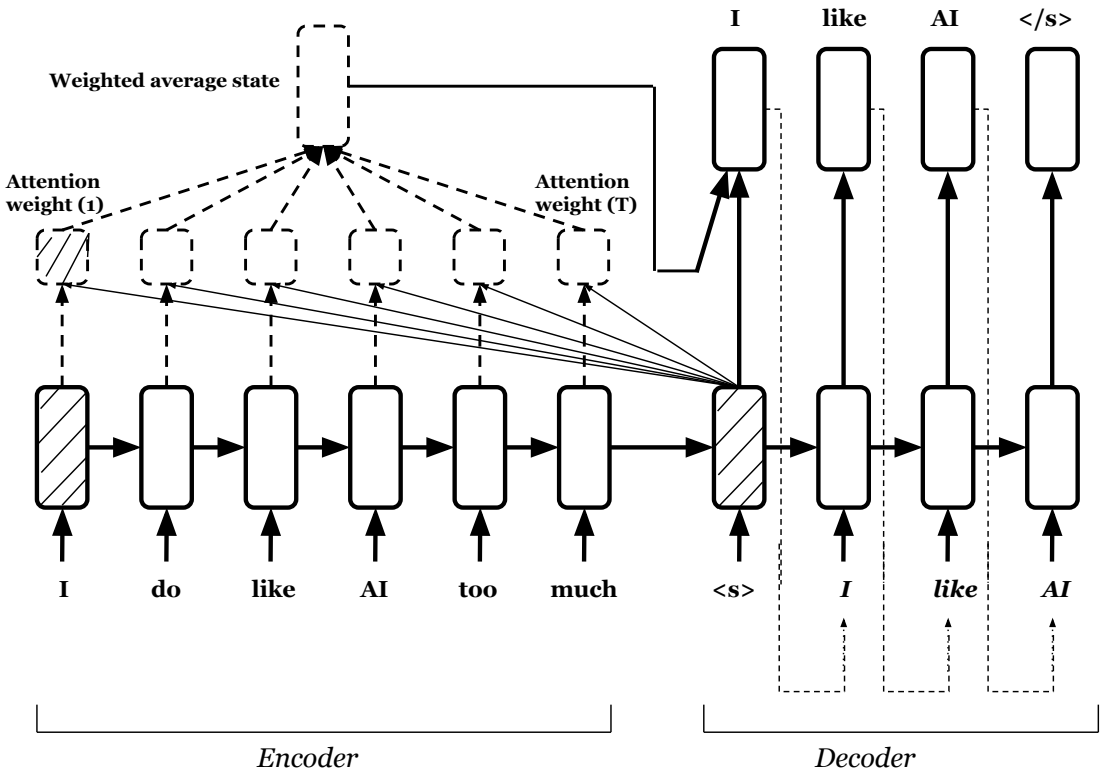


FIGURE 4.1: Seq2Seq model with attention

4.1 Attention Is All You Need/ The Transformer

name of this chapter is copies from the article by Vaswani et al., 2017 that has introduced the transformer. The transformer is among the newest proposed deep learning models with a high potential. It has more handles to control the learning behaviors like over over-fitting a. The model has removed the conventional RNN layers and does not need to be trained with BPTT. While the seq2seq models use RNN to process the sequential data, transformer keeps the position of the inputs and outputs by using a position-wise embedding mechanism. The main paper has used *sine* and *cosine* functions in which the position of the input is a multiplier on the size of sin amplitude and the dimensions of the embeddings (e.g. each of 128 dimensions of a word vector are the different frequencies of the sine function) So the transformer is basically an encoder-decoder with positional embedding of inputs and fully connected NN that learns the alignments of input and output sequences. The long and costly procedure of BPTT have always been a weakness of sequential

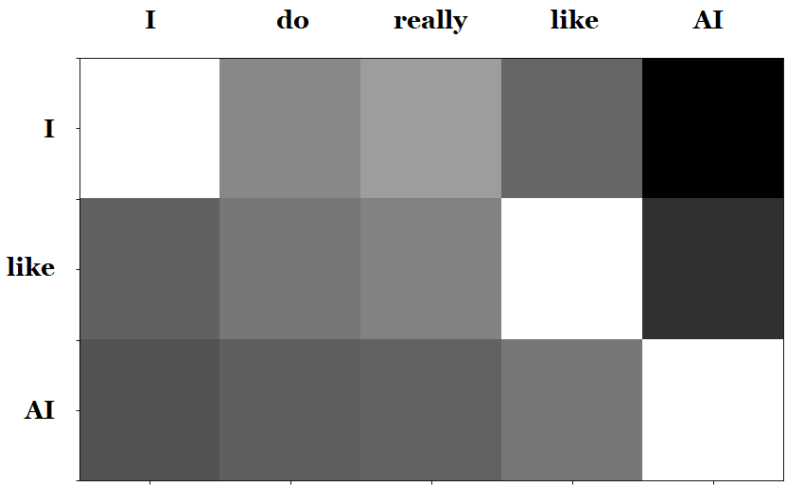


FIGURE 4.2: Alignment of source and target sequences

neural models. Transformer is simply a step higher than its previous generations by simply substitute the RNN layers with a new structure.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

Transformer deploys the attention mechanism in a similar way to seq2seq model. In the later one, the hidden states of the decoder were compared to hidden states of the decoder. In the transformer model, this applies to the positional encoding in decoder and encoder.

Chapter 5

Evaluation Framework

5.1 System Parameters

5.1.1 Software Configuration

All the models that we have used in this work are implemented in the Python programming language. (Python is the most used programming for Deep Learning applications. Among the other available infrastructure, we can name R. For example, using TensorFlow libraries in R[1]). Deep Learning models grow rapidly in size with the addition of hidden layers, and increasing the size of each layer. In sequential neural models, it is also proportional to the length of the sequence. Usually, a model will use tens of millions of variables. For example one of our Seq2seq models with 4 layers of LSTMs each with 1024 modes, has 42,789,205 parameters in total. The computations on these models are a series of repeated loops inside the codes. Mostly it is because of the back-propagation procedure and model cost optimization calculations with e.g. SGD. This huge amount of computation is very costly and time-consuming. In the recent years the only solutions for performing huge computations, was using supercomputers which use arrays of CPUs to do many computations in parallel. The next dramatic change happened just after the first release of CUDA parallel computing platform by Nvidia[2]. The idea was to use a GPU instead of CPU to perform parallel calculations. A GPU usually has a lower Clock-rate but having many cores (e.g. 3584 units in Nvidia Titan X Graphic Card). For compatibility reasons, two versions of python are used. 2.7 for TensorFlow and 3.5 for Pytorch. For this purpose, we needed two isolated installations of python.

[1] see `https://tensorflow.rstudio.com/`

[2] see `https://developer.nvidia.com/about-cuda`

We recommend using a Virtual Environemtnt[3] Virtualenv[4]. The codes are executed inside a terminal. Details of the models are described in the following section. An example of running the bash file is shown below.

5.1.2 Baseline Model (Seq2seq with attention)

Among the models and configuration that have been used for NMS applications, the Seq2seq and pointer Generator Network have succeeded the previous models. The seq2seq model is more popular and is broadly available in many Deep Learning libraries like Tensorflow, Pytorch[5], Theano, Keras and many others. Due to this excellent variety and availability, we chose seq2seq as our base-line model. The implementation we preferred to use is seq2seq from OpenNMTpy [6] that is ported version of OpenNMT project to Pytorch. OpenNMT is an open-source project that originated from NLP group of Harvard University (*OpenNMT Neural Machine Translation*).
The openNMTpy altogether has 3 main modules. The preprocessing module, training, and evaluation module. Each have a wide set of hyper-parameters and is highly customizable. All the hyper-parameters have a default value. Using the train modulde we can degine and customize our models. This module also let us define the training procedure and adjust the optimization parameter values. We will discuss the preprocessing module after the section for datasets.

Baseline Training Hyper-parameters
Train module has 70 hyper-parameters each for a specific porpus.

General hyper-parameters Including paths to input data, and output mode, log file, GPU id etc.

Model Setting Hyper-parameters They define the model. Type of encoder-decoder, number of layers, size of hidden layers, attention-type, size of word embeddings, number of heads, keys and values (in thecase of transformer) etc.

[3]Here a Virtual Environment is an isolated space on an Operating System that is used to install a python distribution with its packages and their dependencies without interfering with any other installation of python. As an example, you can have both the python 2.7 and python 3.5 in a single system. To use each of them you need to activate the corresponding virtual environment in which your python distribution is installed.

[4]see `https://virtualenv.pypa.io/en/stable/`

[5]see `http://pytorch.org/`

[6]see `http://opennmt.net/OpenNMT-py/main.html`

Training and Optimization Hyper-parameters Include batch-size, number of epochs, learning rate initial value, learning rate decay rate, wurmup steps etc.
Table 5.1 shows the actual Hyper-parameters that are adjusted in our setting with a brieaf description for each. Among them `dropout`, `adagrad` and `max_grad_norm` methods need more explanation. We describe them below:

Dropout
One of the common problems in machine learning applications is over-fitting. It happens when the model learn a sample dataset really well and shows a high accuracy, but can not generalize it to the new test data, so that the validation accuracy is much lower. Dropout is a regularization technique used in neural networks to compensate the over-fitting. In a dropout strategy some of the neural with are randomly disabled (or droped) with a given probability in each update. For example with a dropout of 0.1, 10% of all hidden layer nodes drop after each weight update (Srivastava et al., 2014).

Adagrade
The commonly used optimization algorithm for minimizing cost function is SGD. SGD use the derivatives of the weights of the network and update them with a single laring rate. Adagrade unlike SGD use a learning rate specific to each weight.Eperiment of Dean et al., 2012 has proved that Adagrade improves the performance of SGD in big deep learning models.

Max_grad_norm
Training RNN networks usually suffers from two problems. Vanishing and Exploding gradients. They occur when the weight in the layers are very smaller or very bigger that 1. As a result the the output y which is a product of all the weights might vanish (gets very small) or explode. To prevent this problem a way is to renormalize the weights. In this experiment this value is set to 2.

5.1.3 Target Model (Transformer)

Our main focus on this work is on the new proposed neural architecture called Transformer by Vaswani et al., 2017 that is discussed in the section 4.1. The transformer is an Attention-based model that has replaced the RNN layers that are a

Hyper-parameter	Value	Description
data	1	path to preprocessed data
copy_attn	True	uses copy attention mechanism
word_vec_size	128	the size of word embeddings
rnn_size	512	size of hidden layers in LSTM cells
layer	1	number of hidden layers (in each time-step)
encoder_type	brnn	type on encoder (RNN, BRNN, or Transformer)
epochs	25	number of epochs
max_grad_norm	2	the threshold for gradient clipping
dropout	0	droup out regularization multiplier
batch_size	16	number of instances for a weight update
optim	adagrad	uses adagrade optimizser (default is SGD)
learning_rate	0.15	learning rate of the gradient descent algorithm
seed	777	initialization for random variables

TABLE 5.1: Baseline model hyper-parameters

common part of sequential models with a new structure. As a result, the new model stands on a new configuration and has new parameters that demands a precise study. As the date of this work (April, 2018) there are at least 2 well-designed and documented implementations available, each one with some different characteristic and performance measures, OpenNMTpy[7] that uses pytorch libraries, and TensorFlow tensor2tensor[8]. However, these models are not perfect, as they are all open-source projects under development. Many individual Programmers and Developers contribute to complete these models every day and piece by piece. In this work, we chose OpenNMTpy transformer. Although tensor2tensor has the benefit of reporting a few metrics that OpenNMTpy does not have, like ROUGE and BLEU but we still choose OpenNMTy, because it gave us best output results. In case of tensor2tensor, the decoder module (or translator) has an issue which is not removed yet.

OpenNMTpy Transformer is an NMT model available in the OpenNMTpy project. The project is under development. i.e. they had referred to their transformer "experimental". But the changes are very quick in the field of Deep Learning . During the preparation of this work, they have released a full working transformer model along with full documentation (as of April, 2018).

[7] see https://github.com/OpenNMT/OpenNMT-py

[8] see https://github.com/tensorflow/tensor2tensor

Hyper-parameter	Value	Description
layers	4	(see baseline hyper-parameters)
rnn_size	512	(see baseline hyper-parameters)
word_vec_size	512	(see baseline hyper-parameters)
max_grad_norm	0	(see baseline hyper-parameters)
optim	adam	(see baseline hyper-parameters)
encoder_type	transformer	(see baseline hyper-parameters)
decoder_type	transformer	(see baseline hyper-parameters)
potition_encoding	True	uses a sin encoder replplacing RNN structure
dropout	0.2	(see baseline hyper-parameters)
epochs	25	(see baseline hyper-parameters)
warmup_steps	8000	(see baseline hyper-parameters)
learning_rate	2	(see baseline hyper-parameters)
decay_method	noam	is a way of changing learning rate.
adam_beta2	0.998	a method for using adaptive learning rates
batch_size	4096	(see baseline hyper-parameters)
batch_type	tokens	can be a sentence or token
share_embeddings	True	share encoder and decoder word vectors
copy_attn	True	(see baseline hyper-parameters)

TABLE 5.2: Transformer model hyper-parameters

Transformer Training Hyper-parameters

It uses the same modules of our baseline model with different parameters. The key hyper-parameter is `encoder_type` and `decoder_type` which must be both set to `transformer`. To see the other configurations see table 5.2 that contains our transformer initial settings. Most of the hyper-parameters are common with the ones for the baseline. but their value in the transformer setting might be different.

Steps vs. Epochs

In machine learning terminology an **epoch** is equal to training the model with all the examples in the dataset. For example, an epoch in a Deep Learning model is equal to a forward and backward pass of all the training examples.

Each batch contains a subset of all the samples in the dataset. The batch-size unit is usually a complete sequence, e.g. a sentence. Some models like Tensor2Tensor use Tokens instead of sentences. As an effect, these models report the training progress with steps and not epochs. The benefit of this approach is that we can freely fill a batch with more of small sentences or fewer long sentences.

5.1.4 Hardware Configuration

All the experiment are done on GPUs. For this purpose, we have used the infrastructure of the high-performance computing center of the UPC in Barcelona. The user can access the cluster node via SSH connection. SLURM job scheduling system receives the command from the users and execute the desired applications. Each user can request a desired number of GPUs, CPUs and storage space. As the OpenNMTpy models do not support multi-GPU computing, we only used 1 unit in each experiment. Although for some experiments with tensor2tensor arrays of 4 and 8 GPUs had been used. Each GPU unit is a Nvidia Titan X with 12 Giga-bytes of RAM, equipped to CUDA 9, with 3584 CUDA cores and Compute Capability 6.1. To use the best of a GPU the model should have in a certain size limited by the GPU RAM capacity. A big model with millions of parameter could cause a Out of Memory (OOM) error, while a small model will perform slower. To limit the number of model parameters during training we can adjust the Batch-size.

5.2 Datasets

Text summarization by definition is producing a smaller piece of text out of a larger one. Both sides, aka source and target, have a specific average length. Usually, there is a ratio between the length of target and source. It supposed to be at least 2 (5.1). (Radev, Hovy, and McKeown, 2002)

$$\frac{\text{target length}}{\text{source length}} \leq \frac{1}{2} \tag{5.1}$$

A ratio is a unit-less number. We should also know the absolute length of our text samples and the total size of the dataset. The absolute length ranges are especially important when we use neural models that contain RNN layers. We chose two different datasets varying in ratio and average lengths *Amazon reviews dataset* and *CNN-Dailymail* articles.

	Total Size	Total tokens	Avg src len	Avg tgt len	tgt-src ratio
CNNDM	280,000	220,000,000	750	75	1/10
Amazon	500,000	30,000,000	40	4	1/10

TABLE 5.3: Length distribution of text samples in CNN-DM and Amazon datasets

5.2.1 CNN-dailymail Dataset

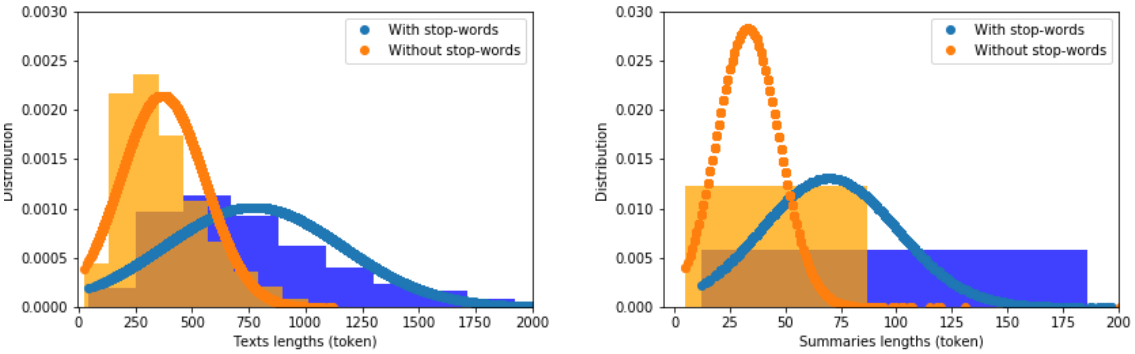
CNN-dailymail dataset is a collection of the articles mostly news, interviews that have been published on the two popular websites *CNN.com* and *dailymail.com*. Like the common styles on newspapers and journals, each article contains some highlighted sections that together form the summary of the whole article. The raw dataset includes the text contents of web-pages saved in separate HTML files. This dataset that originally has been gathered in the work of Hermann et al., 2015, has become a standard source for training and evaluating text summarizer models and has been used in many later studies including the works of Nallapati et al., 2016 and See, Liu, and Manning, 2017. The code (Manning et al., 2014) that See, Liu, and Manning, 2017 later in 2017 used to pre-process and tokenize the data is the Stanford coreNLP Java toolkit that is available online (*standford coreNLP Natural Language Software*). Some of the articles are very long with more than 2000 words. In practice, most experiments use a clipped version of articles to reduce computation cost. Figure 5.1 shows the length distribution of articles and summaries. Their distribution is close to Normal. We did not remove the stopwords [9] to be able to compare or model with prior experiments.

5.2.2 Amazon Fine Food Reviews

Our second dataset, Amazon reviews, 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. So in this dataset unlike the CNN-dailymail, the summaries are very short(McAuley and Leskovec, 2013) The data set is available on `kaggle.com` [10]. Figure 5.2 shows the length distribution of amazon texts and summaries. In side (A) for the texts the diagram has two plots from which the orange

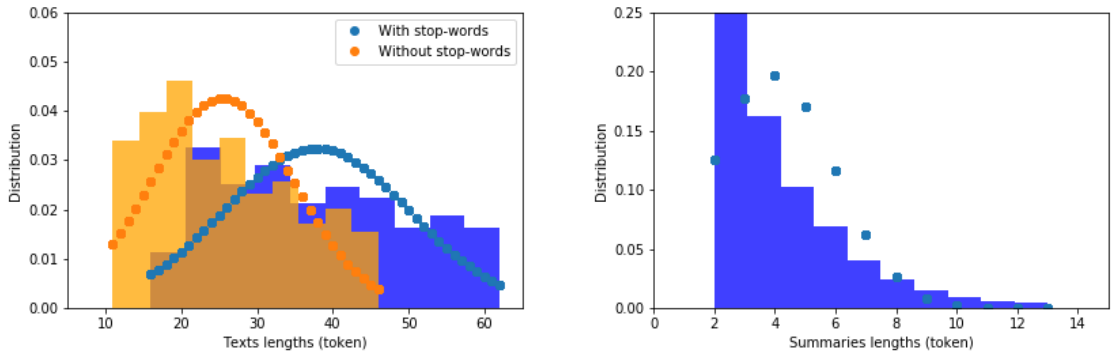
[9] Stopwords are some insignificant tokens like prepositions, auxiliary verbs, and pronouns that do not add much to the meaning of the corpus.

[10] `https://www.kaggle.com/snap/amazon-fine-food-reviews`



(A) CNNDM Texts (B) CNNDM Summaries

FIGURE 5.1: Length distribution of Texts and Summaries in CNN-Dailymail dataset (in tokens)



(A) Amazon Reviews Texts (B) Amazon Reviews Summaries

FIGURE 5.2: Length distribution of Texts and Summaries in Amazon reviews dataset (in tokens)

one is the distribution of texts tokens without stopwords. In Amazon experiment, we just removed stopwords from the texts. We kept them in summaries because they are very short, e.g. 2 words, removing stopwords will affect their meaning.

5.3 Data Preprocessing

Data preprocessing is a crucial and decisive step in machine learning applications (Srividhya and Anitha, 2010). A mere flaw in a dataset can ruin hours and days of training procedure. For example in a sequence-to-sequence problem where the data consists of a pair of source and target, pairs are sorted and indexed in two

same size lists. Shifting, adding or removing a single record in each list will damage the balance of all the pairs so that each source instance will bind to a wrong target instance. In a text processing application some of the preprocessing steps are as follows:

- Aligning source and target
- Removing Whitespace
- Removing bad characters
- Encoding correction
- Removing duplicates
- Removing stopwords (depending on the problem)

We took two difference preprocessing strategies for CNNDM and Amazon reviews datasets:

Preprocessing CNNDM dataset

The articles are too long. They are sometimes 3 times bigger than the average. Figure 5.1 shows the distribution of the source and target lengths. We truncated the source texts to 400 tokens and the targets to 100 tokens. This way guarantees that almost 70 percent of instances have equal size and the rest are smaller. We will experiment the effect of sequence truncation better in the experiments on Amazon reviews dataset. Some redundant character and symbols, hyperlinks, HTML tags and non-latin alphabets are removed. We kept the stopwords to be able to compare the results with previous studies.

Preprocessing Amazon reviews dataset

In Amazon reviews dataset the target summaries are very short. They are on average 4, and at most 10 tokens. Removing the stopwords, in this case, will damage the summaries and make their meaning unclear. We experimented the source texts on two modes of with and without stopwords. The length of the original source text without stop words is 25 woken on average. The original dataset contains 500,000 instances. A considerable portion of the data is duplicated (some users has posted their comments more than once). We limited the source reviews to 100 tokens and a minimum size of 20, to have a better alignment between source and target. After removing the duplicates and filtering the very short and very long sequences the size

of the dataset was reduced to 210,000. In our setting we performed three experiment on the original length, truncated to 40 tokens and truncated to 20 tokens.

Chapter 6

Experiments

6.1 Experiments with CNN-dailymail

Experiment on Baseline
CNNDM is a relatively big dataset. It contains more than 200 million words. Training our models on this dataset takes days. For example we trained our seq2seq model for 2 days until convergence. It reached 53% of validation accuracy and converged after it. Because of the long training time we limited or chooses in tunning the models. It is worth to say that validation accuracy is not a proper metric for summarization tasks. So far Rouge scoring system has been used in many studies as the only metric to evaluate a summarizer model. As the OpenNMTpy models do not report Rough scores, our strategy is to train the model till the convergence of the validation accuracy. And then calculating the Rouge-F1 score.

Experiment on Transformer
A set of hidden layer sizes of 256, 512, and 1024 are examined. The validation accuracy is shown in Figure 6.1a This experiment can show us a comparative performance on long sequences. You can see that the curve is relatively smooth and without sharp fluctuations. This is due to the big size of the dataset and long sequences which are all almost the same size after truncation. hidden layers with 256 and 512 neurons show good accuracy. They passed 50% validation accuracy in the first epoch. While a hidden layer with 1024 neurons is too big for this dataset. The validation accuracy collapsed in the first epoch at 25% which is very low. This is the highest overfitting that is usually observed from big models. In total, the hidden layer size of 512 gave us the best validation accuracy.

Reference summary
the large hadron collider lhc begins again after a two-year shutdown . the restart was delayed in march .

Baseline output
lhc generates up to 600 million particles per second , with a beam circulating for 10 hours , traveling more than 6 billion miles 27 km the distance from earth to neptune and back again .

Transformer output
the large hadron collider lhc is ready for action following a two-year shutdown in march . scientists at the european organization for nuclear research cern completed final tests , enabling the first beams to circulating inside the lhc 's 17 mile 27 km ring .

TABLE 6.1: Sample (1) - Output summaries of baseline and transformer. The transformer hold the essential contents of the reference summary.

6.1.1 Results

Figure 6.2a and 6.2b shows the training metrics of both models. You can see how transformer outperformed the baseline on CNNDM dataset. Now that we have these two converged training on models we can evaluate the models with test dataset and calculate rouge scores. Table 6.3 contains the detailed information of this comparison. You can also see two sample output summaries from both models compared to the reference summary. The outputs of the transformer are surprisingly much more accurate. They hold the main content of the summary. For example in the sample (2) 6.2 a part of the reference summary (" furious 7 " is opening around the globe this weekend) with 11 tokens is exactly repeated in the output summary of the transformer .

6.2 Experiments with Amazon Reviews

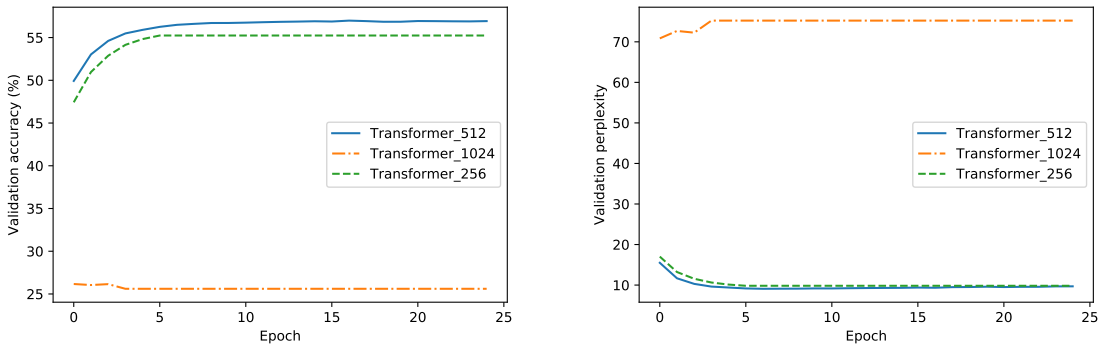
Amazon Fine Food Reviews dataset contains samples of a brief review on a product and its title. As discussed in chapter 5, the size of dataset after preprocessing reduced to 210,000. The source and target pairs are relatively smaller than the ones from CNNDM. The total tokens size of Amazon reviews dataset is 7 times smaller that CNNDM. The short training time due to this smaller size helped us experiment both of our models with a wide variety of hyper-parameters on this dataset.

Reference summary
the final film featuring the late paul walker , " furious 7 " is opening around the globe this weekend . it 's worldwide debut may approach or cross $ 300 million by the end of easter sunday .

Baseline output
universal 's " furious 7 " continues to build momentum at the box office for a weekend debut in the $ 135 million - $ 138 million range , the largest opening in north america since fall 2013 .

Transformer output
" furious 7 " is opening around the globe this weekend and earned a $ 60 million internationally on wednesday and thursday for a possible worldwide debut approaching or crossing $ 300 million by the end of easter sunday .

TABLE 6.2: Sample (2) - Output summaries of baseline and transformer. The transformer holds the essential contents of the reference summary.

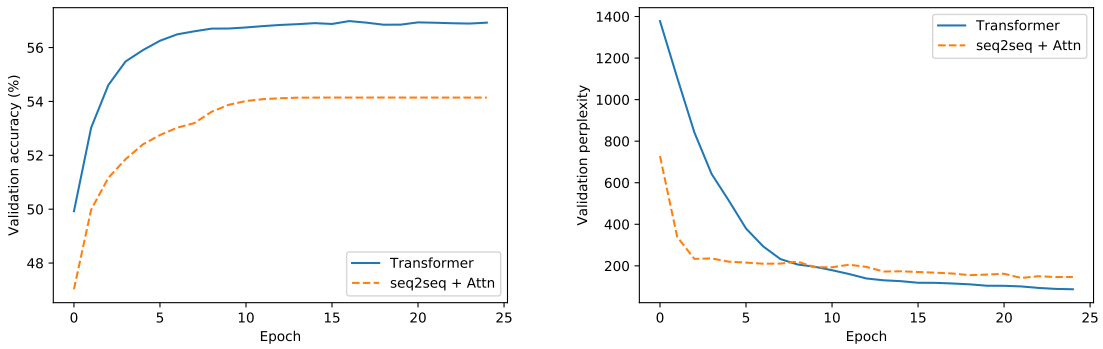


(A) Transformer training accuracy (B) Transformer training perplexity

FIGURE 6.1: Training CNNDM on Transformer with default hidden layer size 512 (blue) hidden layer size 1024 (dash-dotted orange) and hidden layer size 512 (dashed green).

Model	Baseline	Transformer
Number of parameters	42,789,205	81,468,757
Total training epochs	25	25
Epoch time	6500 s	1500 s
Validation accuracy	54.14	56.92
Validation perplexity	11.21	9.68
Rouge-F1 score	26.1	27.4

TABLE 6.3: Baseline and Transformer performance on CNNDM dataset



(A) Transformer vs baseline (acc) (B) Transformer vs baseline (ppl)

FIGURE 6.2: Training CNNDM on Transformer (blue) and baseline (dashed orange).

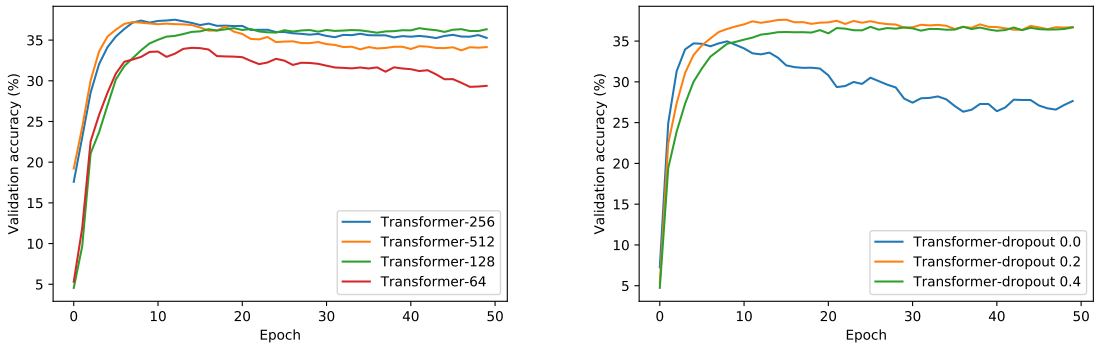
Just like the experiments on CNNDM dataset we trained the baseline with a set of default recommended hyper-parameters. Then we tuned the transformer model to see how the output Rouge scores will change with the better validation accuracies. At each experiment, all the hyper-parameters are kept fixed instead of a target hyper-parameter. We tried different parameters like label-smoothing, but only report the ones that produced a significant difference in the training.

6.2.1 Experiment 1 - Hidden Layer Size

A good approach for tunning is to start from a small model size. Then increase the size and complexity step by step. We tried a set of 4 different layer sizes: 64, 128, 256, 512. Among them, a layer size of 128 gave us the best result (Figure 6.3a). Unlike the experiment on CNNDM with had the best performance with a 512-dimensional hidden layer. This is due to the long sequences of CNNDM (Although the total size of both datasets is almost equal).

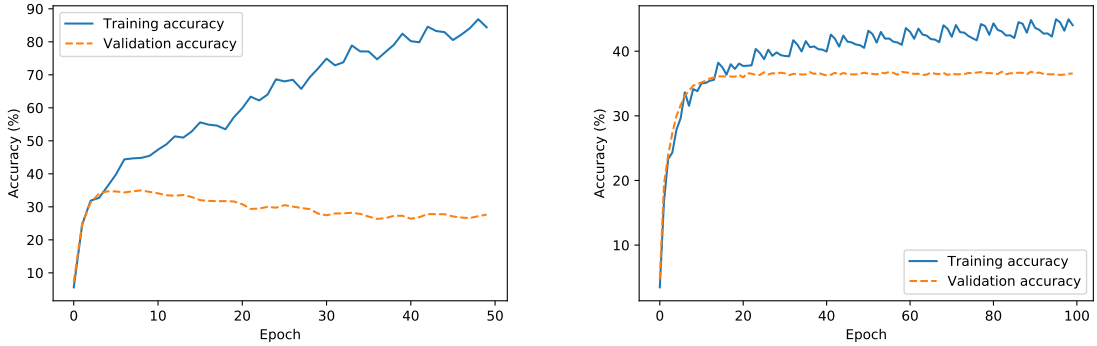
6.2.2 Experiment 2 - Dropout

We found that dropout has a crucial effect on the training process. Figure 6.3b show how dropout can control the validation accuracy. For a better explanation see Figure 6.4 where the gap between Training accuracy and validation accuracy is reduced with a dropout value of 0.4.



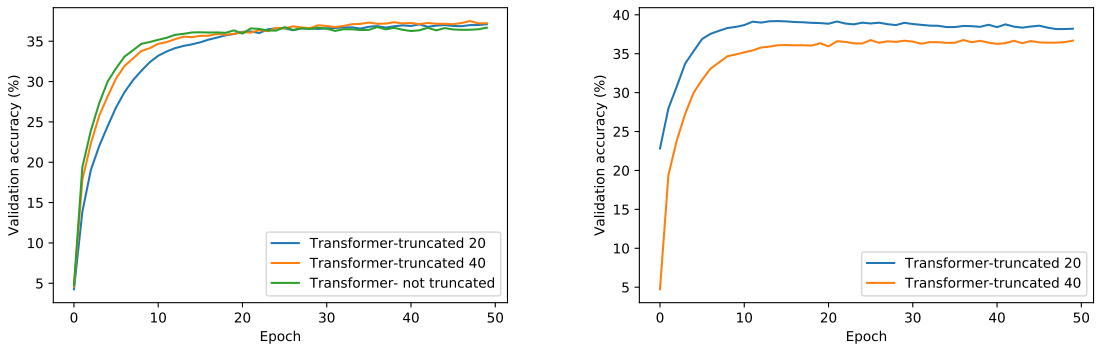
(A) Different hidden size (B) different dropout value

FIGURE 6.3: Tunning transformer model with different hidden size and different dropout values.



(A) Dropout 0.0 (B) Dropout 0.4

FIGURE 6.4: Increasing dropout value reduces the gap of Training accuracy and validation accuracy.



(A) Truncation effect (B) Removing stopwords effect

FIGURE 6.5: Truncation and removing stopwords effects on validation accuracy.

6.2.3 Experiment 3 - Truncation Effect

In this experiment, we cut the source sentences to a certain token number. Our choices are 40 tokens which will truncate 30% of source instances, and 20 tokens that will truncate almost 70% of source sentences and will provide a very uniform training set. But you should consider that truncation will damage the grammatical correctness of the sentences. Figure 6.5 illustrates that truncation had almost no effect on the validation accuracy. However, we will see that truncation can change rouge scores.

6.2.4 Experiment 4 - Removing Stopwords Effect

As described in chapter 5, we had removed the stopwords from source data, because it usually just add noise to the training process. Figure 6.5b shows that having stopwords in the source data can increase the validation accuracy. Although in this case we found no correlation between rouge scores and validation accuracy.

6.2.5 Results

After tuning transformer we compare its results with baseline. The baseline training performance of Amazon reviews in shown in figure **??**. For each hyper-parameter value the model is tested and a rouge score is calculated. Table 6.4 lists all the experiments on Amazon reviews dataset. Unlike the CNNDM dataset which has longer sequences, here baseline model has performed better on the Amazon reviews

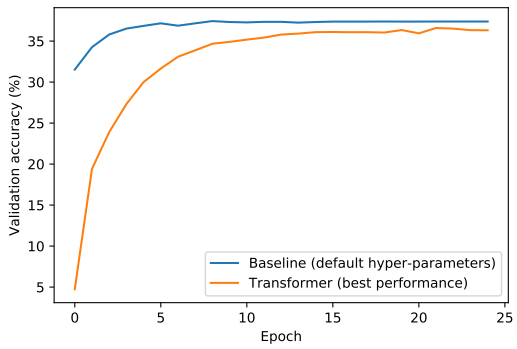


FIGURE 6.6: Baseline performance on Amazon reviews dataset compared with transformer.

Model	Hyper-parametersa and settings	Rouge-F1
Baseline	Default	17.87
Transformer	2 layer 128- dropout 0.2 - no truncation- no stopwords	12.95
Transformer	2layer 128- dropout 0.4 - no truncation- no stopwords	13.91
Transformer	2layer 128- dropout 0.4 - truncated to 20- no stopwords	15.34
Transformer	2 layer 128- dropout 0.4 - truncated to 40- no stopwords	14.76
Transformer	2 layer 128- dropout 0.4 - no truncation- with stopwords	13.21

TABLE 6.4: Baseline and Transformer performance on Amazon reviews dataset

dataset with a Rouge-F1 score of 17.87. It tells that baseline is more adaptable with different datasets. To tune the transformer on amazon changing the number of attention heads might help. i.e. it is embedded in tensor2tensor transformer model as an argument, but it is available in our OpenNMTpy transformer model.
Adding stopwords reduced the Rouge-F1 score from 13.91 to 13.21 as it was expected. We got the highest Rouge score by truncating the source sequences to 20 tokens.
In general, rouge scores on Amazon reviews are lower compared to the ones from CNNDM. It is due to the very short summaries of Amazon dataset. If fact they are very abstractive. Some of the summaries are under 3 words, and some are one or two words that do not exist in its corresponding source text. Some of the sample summaries are listed in the table 6.5.

Reference	my dogs seem to like it
Baseline	dogs love it
Transformer	the dogs love it
Reference	meh
Baseline	worst coffee ever
Transformer	worst coffee ever
Reference	great coffee
Baseline	a great coffee
Transformer	great everyday coffee

TABLE 6.5: Output summaries of baseline and transformer from Amazon reviews dataset. The reference summaries are very abstractive.

Chapter 7

Discussion

7.1 Observations

- choosing a right tuning strategy is very important. Tuning deep learning model is very tricky, but following a few rules of thumb will reduce its time . In this work we used a greedy approach.First tip is to do the fastest experiments first. Start from some small models with low complexity e.g. few number of layers and small hidden layer size. When you got a good training accuracy and overfits the dataset, and if the validation accuracy is not following the training accuracy, use a regularization technique. Having the proper model-size you can tune other hyper-parameters to achieve the best possible model.

- ABS is a more challenging problem compared it extractive counter-part. When dataset samples are too small, it carries more abstractive content. Sometimes the summaries tokens are not available in the source. It makes the prediction very difficult and as a result, our summarizer systems do not show good performance measures on them. Amazon dataset is an example which gave us a very lower Rouge score in comparison with CNNDM dataset.

- Removing stopwords actually reduce the input noise to the model. We got better results from the Amazon dataset without stopwords. The stopwords can be more harmful in a small dataset. As they are very frequent and the model can overfit on them.

- Rouge scores so far have been the golden standard to evaluate summarizer systems. Rouge is a statistical metric that works with comparing the frequency of tokens in source and target. Consequently, It cannot evaluate the grammatical and semantic correctness of the outputs. It also fails to deal with Out Of Dictionary Words. A better metric that can evaluate a summarizer systems from all aspects is needed.

7.2 Conclusions

ABS is a text-to-text problem, very close to translation, with a big difference that the source and target sequences are not properly aligned. Therefore, it demands sophisticated models and high-performance algorithms to deal with its complexity. We discussed that RNN networks as the first sequential models were not able to handle the non-aligned inputs, hence were not useful for summarization tasks. Seq2seq neural architectures succeed the RNN models in many sequence-to-sequence problems including text summarization. They improved furthermore with the addition of attention mechanism. The Transformer is a quite new model that works only with the disciplines of attention. It is relatively faster because of the removed BPTT that allow them to process long sequences in a shorter amount of time. Although its performance speed depends on the setting and dataset structure, so that in some cases it might perform slower. The model had shown state of the art performance on translation and summarization tasks. This architecture has a high potential to substitute previous successors. To make use of this potential, its rather more complex architecture should be explored more in different problems and experiments with a variety of configurations. We saw how the transformer surpassed our seq2seq baseline model in CNNDM summarization task. Comparing the results with the experiments on Amazon Reviews dataset showed that its more complicated hyper-parameters should be precisely tuned depending on the characteristics of the dataset.
A highly essential requirement for a summarizer model is a precise metric to direct the training process. So far Rough has been the most reliable metric for abstractive text summarization.

Bibliography

Allahyari, Mehdi et al. (2017). "Text summarization techniques: A brief survey". In: *arXiv preprint arXiv:1707.02268*.

Cho, Kyunghyun et al. (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078*.

Dean, Jeffrey et al. (2012). "Large scale distributed deep networks". In: *Advances in neural information processing systems*, pp. 1223–1231.

Elman, Jeffrey L (1991). "Distributed representations, simple recurrent networks, and grammatical structure". In: *Machine learning* 7.2-3, pp. 195–225.

Hermann, Karl Moritz et al. (2015). "Teaching machines to read and comprehend". In: *Advances in Neural Information Processing Systems*, pp. 1693–1701.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Manning, Christopher D. et al. (2014). "The Stanford CoreNLP Natural Language Processing Toolkit". In: *Association for Computational Linguistics (ACL) System Demonstrations*, pp. 55–60. URL: http://www.aclweb.org/anthology/P/P14/P14-5010.

Manohar, Sanjay G, Yoni Pertzov, and Masud Husain (2017). "Short-term memory for spatial, sequential and duration information". In: *Current opinion in behavioral sciences* 17, pp. 20–26.

McAuley, Julian John and Jure Leskovec (2013). "From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews". In: *Proceedings of the 22nd international conference on World Wide Web*. ACM, pp. 897–908.

Medsker, LR and LC Jain (2001). "Recurrent neural networks". In: *Design and Applications* 5.

Mikolov, Tomáš et al. (2010). "Recurrent neural network based language model". In: *Eleventh Annual Conference of the International Speech Communication Association*.

Nallapati, Ramesh et al. (2016). "Abstractive text summarization using sequence-to-sequence rnns and beyond". In: *arXiv preprint arXiv:1602.06023*.

Olazaran, Mikel (1996). "A sociological study of the official history of the perceptrons controversy". In: *Social Studies of Science* 26.3, pp. 611–659.

OpenNMT Neural Machine Translation. [Online]. Available from: `http://opennmt.net/`.

Radev, Dragomir R, Eduard Hovy, and Kathleen McKeown (2002). "Introduction to the special issue on summarization". In: *Computational linguistics* 28.4, pp. 399–408.

Sarkar, Kamal, Mita Nasipuri, and Suranjan Ghose (2011). "Using machine learning for medical document summarization". In: *International Journal of Database Theory and Application* 4.1, pp. 31–48.

Schuster, Mike and Kuldip K Paliwal (1997). "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11, pp. 2673–2681.

See, Abigail, Peter J Liu, and Christopher D Manning (2017). "Get to the point: Summarization with pointer-generator networks". In: *arXiv preprint arXiv:1704.04368*.

Srivastava, Nitish et al. (2014). "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Srividhya, V and R Anitha (2010). "Evaluating preprocessing techniques in text categorization". In: *International journal of computer science and application* 47.11, pp. 49–51.

standford coreNLP Natural Language Software. [Online]. Available from: `https://stanfordnlp.github.io/CoreNLP/`.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*, pp. 3104–3112.

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in Neural Information Processing Systems*, pp. 6000–6010.

List of Figures

List of Tables

List of Abbreviations

ABS	**A**bstractive **T**ext **S**ummarization
NN	**N**eural **N**etwork
MT	**M**achine **T**tanslation
NTS	**N**eural **T**ext **S**ummarization
RNN	**R**ecurrent **N**eural **N**etwork
BRNN	**B**idirectional **R**ecurrent **N**eural **N**etwork
LSTM	**L**ong **S**hort **T**erm **M**emory
BPTT	**B**ack **P**ropagation **T**hrough **T**ime
SGD	**S**tochastic **G**radient **D**escent
OOM	**O**ut **O**f **M**emory