Karlsruher Institut für Technologie

Attention Neural Network-Based Abstractive Summarization and Headline Generation

Master's Thesis of

Nejmeddine Douma

at the Department of Informatics
Interactive Systems Lab (ISL)
Institute for Anthropomatics and Robotics

Reviewers:	Prof. Dr. Alexander Waibel
	Prof. Dr. Tamim Asfour
Advisor:	Dr. Jan Niehues

Submitted in partial fulfillment of the requirements
for the MSc degree in Computer Science of
Karlsruhe Institute of Technology
February 2018

Abstract

With the ever-increasing frequency of daily content creation, the need for content curators or summarizers is only at its beginning. Manually performing these tasks drive massive costs, thus the increasing demand for efficient automatic summarization systems.

Sequence-to-sequence learning has attracted intense interest over the past years. The success of end-to-end training of encoder-decoder neural networks in tasks like machine translation has sprouted active research using similar architectures in other transduction tasks such as paraphrase generation or abstractive summarization.

In this thesis, the recent advances in neural network-based abstractive summarization are reviewed, and various attention neural network-based headline generation models are proposed. One of the most significant limitations of existing solutions is limiting the input to the first one or two sentences of the article. This thesis investigates new methods trying to add to the output the important information beyond what is included in the first sentence. Methods to represent sentences and words are used in combination with single attention or dual-attention mechanisms to design new abstractive summarizers. The new models are compared experimentally to the state-of-the-art of abstractive headline generation based on various standard automatic and human evaluation measures and different tasks.

Zusammenfassung

Mit der immer zunehmenden Frequenz der täglichen Content-Erstellung steht der Bedarf an Zusammenfassung erst am Anfang. Die manuelle Ausführung dieser Aufgaben führt zu hohen Kosten und somit zu einem steigendem Bedarf an effizienten automatischen Zusammenfassungssystemen.

Das Sequence-to-Sequence-Lernen hat im Laufe der letzten Jahre starkes Interesse geweckt. Der Erfolg des End-to-End-Trainings von Encoder-Decoder Neuronalen Netzwerken in Aufgaben wie maschineller Übersetzung hat aktive Forschung mit ähnlichen Architekturen in anderen Aufgaben der Verarbeitung natürlicher Sprache, wie zum Beispiel Paraphrasengenerierung oder abstrakte Zusammenfassung, hervorgebracht.

In dieser Arbeit werden die neuen Fortschritte in der abstrakten, neuronalen netzwerkbasierten Zusammenfassung besprochen und verschiedene attention-based Neuronale Netzwerken Übershriftsgenerierungsmodelle vorgeschlagen. Eine der größten Einschränkungen existierender Lösungen besteht darin, die Eingabe auf die ersten ein oder zwei Sätze des Artikels zu beschränken. Diese Masterarbeit untersucht neue Methoden, die versuchen, der Ausgabe wichtige Informationen hinzuzufügen, die über den ersten Satz hinausgehen. Methoden zur Repräsentation von Sätzen und Wörtern werden in Kombination mit einem einfacher attention oder mit dual-attention mechanism verwendet, um neue abstrakte Zusammenfassungen zu entwerfen. Die neuen Modelle werden experimentell mit dem Stand der Technik für abstrakte Übershriftsgenerierungsmodelle vergleichen, basierend auf verschiedenen automatischen und menschlichen Standardmaßen und Aufgaben.

Acknowledgements

I would like to express my special thanks to Prof. Dr. Alexander Waibel, director of the Interactive Systems Lab (ISL), for giving me the opportunity to carry out this thesis work at his organization. I express my most profound gratitude to my advisor Dr. Jan Niehues who took time out to hear and take part in useful decisions and giving necessary advice and guidance. I thank all the authors who shared their work and code. Finally, I would like to thank my family and friends for their love and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

More than 90% of the existing data in the world by 2016 was created in the 12 months preceding it [1]. For example, one place where data is being continuously created is social media networks. Each second, thousands of new pieces of content appear and get shared across the various networks [2]. This overload of information made a shift in users behavior and the way they get information and process it. Less than two decades ago, sources of long content like newspapers were the go-to solution to get written news. In 2016, 62% of US adults get news from social media[3], where content is shorter and goes straight to the point. Regardless of the field and the context, with this information overload and every source competing to win few seconds of the users' attention span, modern users look more and more for concise information, and they want it faster than ever. Therefore there is an increasing need for automatic systems capable of getting the most existing relevant textual information and outputting it in the shortest, concise and informative possible way.

1.1 Motivation

Text summarization systems are the best candidates to serve this need. However, in contrast to other successful transduction tasks like machine translation where commercial solution are widely adopted since many years, automatic text summarization systems are still a bit far from producing business-reliable outputs despite the fact that it is not a new research area Luhn (1958). Seeing the success of deep learning approaches in the task of neural machine translation, motivated a new wave of research works related to text summarization hoping to reproduce a similar progress in this field. Existing neural abstractive summarization methods are limited to taking the first sentence on an article as input and output a generated headline of the article. The choice of the first sentence is based on the observation that it tends to be the most informative sentence in the article and the closest one to the original headline. This approach suffers from practical limitations imposed by the current neural networks architecture such as the input size, training and

[1]according to a study made by IBM Marketing Cloud (retrieved on Dec 23, 2017) https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=WRL12345USEN

[2]http://www.internetlivestats.com/one-second/ (retrieved on Dec 23, 2017)

[3]according to a survey by Pew Research Center. (retrieved on Dec 23, 2017) http://www.journalism.org/2016/05/26/news-use-across-social-media-platforms-2016/

evaluation time or memory limits. By ignoring all the details in an article except the first sentence, it leaves a lot of valuable information behind that may lead to a more informative headline.

1.2 Thesis Goals

This thesis aims to study the possibility of developing a new neural network architecture able to produce headlines through a fully abstractive summarization approach taking in consideration information present anywhere in the article and not only the first sentence. We use the encoder-decoder with attention architecture provided by the Nematus Toolkit Sennrich et al. (2017), which was very successful in the task of neural machine translation and adapt it to the application of abstractive headline generation. We use this end-to-end learning architecture taking the article's first sentence as input as a baseline and enhance it by adding vectors representing all the sentences in the article to the models' input. For memory limits and extended training and evaluation time concerns, we avoid feeding every word in the article directly to the end-to-end architecture. We use pretrained Paragraph Vector Le and Mikolov (2014) to represent every sentence in the article as a vector. The encoder-decoder system in addition to the words of the article's first sentence is fed with the Paragraph Vectors of the article sentences. Different architectures combining these both different representations were tested and evaluated. Among them the new model with a dual-attention mechanism containing two separate attention modules one for each type of input.

1.3 Thesis Outline

In the next chapter, we present an overview of the field of automatic summarization and give the outcome of the background research that was carried in the various topics required to achieve the thesis goal such as words and sentences representation approaches or the fundamentals of sequence-to-sequence learning using attention mechanisms. After that, we go through a review of the related works to the specific task of abstractive headline generation. Before diving in chapter 4 into the details of the different enhancements that were added to the baseline system as well as the presentation of a new dual-attention architecture. In chapter 5, we present the datasets and implementation choices made during our experiments. Through chapter 6, both a quantitative and qualitative analysis are carried out including information about the evaluation metrics used to compare the different performances in addition to a comprehensive analysis of the achieved results and their comparison to the state-of-the-art and the various approaches related to the task of abstractive headline generation. Finally, the 7th chapter concludes the thesis and explores future developments of the existent systems.

Chapter 2

Background Research

After the success of deep leaning approaches in a variety of fields such as computer vision or speech recognition, the new architectures started spreading to other areas like machine translation and natural language processing. More than that, recent success with neural machine translation inspired new approaches for the task of automatic text summarization. In this chapter, we present an overview of the various text summarization tasks and traditional information extraction methods. Then we go through sentences and word representation methods that will be used in later chapters, before exploring the sequence to sequence learning using neural networks and how it was successful in the neural machine translation task and becoming state of the art in this field.

2.1 Automatic Summarization

Previous research work on automatic text summarization led to various types of summarization and with the continuous technological progress, new tasks keep emerging to meet the user needs.

2.1.1 Summarization types

2.1.1.1 Extractive and Abstractive summarization

When trying to categorize the different existing summarization methodes, two fundamental categories stand out which are: *extractive* and *abstractive* summarization.

Extractive summarization consists of generating a summary containing *only* words or sentences extracted from the original text. This task could be achieved through various ways. The first work on extractive summarization dates back to the 1950s. Luhn (1958) presented a way to generate abstracts of technical and magazine articles automatically. The main idea is to select the key sentences, which are the most informative ones in the article and produce an ”auto-abstract” by citing the author.

Since then, extractive summarization took a lot of attention in the published papers related to automatic text summarization and many methods of information extraction were developed. We explore the most used of them in the next subsection.

In the same way as extractive summarization, abstractive summarization tries to generate a summary that contains the main ideas of the input. However, it mainly uses words or phrase that *may not* appear in the input instead of just citing it. Unfortunately, the complexity of the task and the relative success of the developed extractive summarization techniques made research in the field of abstractive summarization way less active during an extended period.

2.1.1.2 Single- and multi-document summarization

Luhn (1958) was intended to be used to generate summaries of single-documents. Later, the emergence of new use cases such as the world wide web made the need to summarize multiple related documents at the same time more present. Single document summarization produces a summary of *only one* document as input. Its goal is to offer the user a quicker way to get the main points dealt with in the given document. However, multi-document summarization deals with generating a summary of *multiple* documents given as input. It removes redundancy and detects the most important information to provide the user a brief digest of the content of the documents.

2.1.1.3 Generic, query-focused and update summarization

Generic sumarization generates a summary without having any assumptions about the genre of the input document, the field that it deals with or the audience that the document targets. The goal is to be able to get the important information from the text and provide the reader a quick overview of the document's content. For a more user-specific summarization, query-focused summarization summarizes only information that is relevant to a user query. The summarizer considers the user input while choosing what information needs to be conveyed in the output at the step of generating the final summary. This type of summarization in more present in the context of search engines and it is deployed to enhance the user experience. One use case is to provide the user with a summary of information related to his query present in different web pages after finding them using the search engine. In some other use cases, it is required to summarize the new information since a previous point in time. This can be done through *Update summarization* which generates a summary of unseen information.

2.1.1.4 Informative and indicative summarization

A summary is called *informative* when it contains the main information included in the input document(s) and provides the user with the main ideas in the original input. In contrast, *indicative summarization* does not contain any content related information and provides the user only with a description of the input.

2.1.1.5 Keyword and headline summarization

Most of the previously mentioned summarization types aim at generating a short paragraph summary. Depending on the use case needs, it is sometimes required to have other forms than paragraph summary. This is the case for summarization types like keyword summarization and headline summarization. *Keyword summarization* aims at extracting the main words or phrases from the input. It can be used in a context of a search engine or for other specific use cases. *Headline summarization* is the task of generating a one-sentence summary of the input. And this is the task that we mainly deal with in the coming chapters.

2.1.2 Information extraction methods

As stated earlier, extractive summarization was more explored in the literature than abstractive summarization for many reasons such as being relatively more accessible and more successful. This led to the development of different methods to extract information from the input, which is one of the main components of an extractive summarizer. We will present here two unsupervised information extraction methods that will be used in later chapters.

2.1.2.1 Word probability

The likelihood of observing a given word is called the word probability. Given a word w and a training corpus T containing N words, w's word probability $p(w)$ is computed through dividing the number of w occurrences $c(w)$ by N:

$$p(w) = \frac{c(w)}{N} \tag{2.1}$$

Using this definition, Nenkova et al. (2006) presents a multinomial model to compute the likelihood of a summary given the words probability distribution:

$$L[sum; p(w_i)] = \frac{M!}{n_1!...n_r!} \prod_{i=1}^{r} p(w_i)^{n_i} \tag{2.2}$$

where M is the number of words in the summary, r is the number of unique words in the summary, and for each i, n_i is the number of times the word w_i appears in the summary $\sum_{i=1}^{r} n_i = M$ and $p(w_i)$ is the probability of w_i appearing in the summary estimated from the input documents.

Using this model, it was observed after analyzing an input consisting of 30 news articles and four human-generated summaries for each article that the likelihood of the human summaries is higher than of machine-generated summaries obtained using one of the best performing summarizers in 2006. This shows that humans tend to pick the frequent topic words in their summaries. Which confirms that frequency-based metrics are a right track for optimizing automatic summarizers.

One of the problems with the word probability model is that many words may have a high likelihood in the corpus by being present in a significant number of documents due to the Zipfian words distribution Baayen (2001), but these words may not convey the information that is specific to the input. They are called stop words and are generally prepositions, auxiliary verbs, determinants or common domain words. Since the summarization task looks for finding the topics that are specific to the input, a solution to the problem is to use TF*IDF weighting.

2.1.2.2 TF*IDF weighting

Term Frequency * Inverse Document Frequency is very used in the field of information retrieval to identify the most significant words in an input text. Instead of merely relating to the term frequency, a weighting using the number of the documents in which a term occurs is performed to measure the significance of the word in the input while reducing the influence of the Zipfian distribution of the words.

Given a corpus T of N documents and a term w, the $TF * IDF$ score of w is computed by multiplying $c(w)$, the number of w occurrences in T, by w's inverse document frequency, which is the logarithm of D divided by $d(w)$, the number of the documents in T in which w occurs:

$$\text{TF*IDF}_w = c(w) \times \log \frac{D}{d(w)} \tag{2.3}$$

Thanks to the multiplication by the inverse document frequency, the TF*IDF score of words that appear frequently but in many different documents remains low, which may be used to identify stop words more efficiently and gives space for more important topic-specific terms in the input to stand out and be better considered in the summary generation.

2.2 Words And Sentences Representations

Many ways to represent words and sentences have been explored because of the need of major machine learning algorithms for a fixed-length feature vectors as input. Two of the most used methods are bag-of-words and bag-of-n-grams. However, these two algorithms present some limits. When bag-of-words ignores the input words order, bag-of-n-grams ignores the semantics, which are both extremely important for a task like automatic text summarization. In this section, we focus more on the *Paragraph Vector* algorithm proposed by Le and Mikolov (2014) which, according to its creators, achieved the new-state-of-the-art on tasks like text classification and sentiment analysis at its release. We used this sentences embedding algorithm to feed a sequence-to-sequence neural network that will be detailed in the next section.

2.2.1 Bag-of-Words model

Since their earliest reference in Harris (1954), bag-of-words models became one of the most common fixed-length vector representations of text. The main idea is to construct a vocabulary of all the known words in the corpus and represent each document by a vector of the size of the vocabulary where for each term in the vocabulary a score is associated, which may be for example a boolean representing the presence of the word in the documents or the number of occurrences of the word in the document.

As an example, let us take the following three documents as input and compute their bag-of-words vector representation.

The input:

```
Document 1:  The start of the meeting is expected at nine.
Document 2:  The meeting main topic is the new perspectives given for the
employees.
Document 3:  Ten employees are expected to attend it.
```

To construct the vocabulary, every input document is tokenized and each new unseen token is added to the vocabulary.

Constructing the vocabulary

```
["the", "start", "of", "meeting", "is", "expected", "at", "nine", "main",
"topic", "new", "perspectives", "given", "for", "employees", "ten", "are",
"to", "attend", "it"]
```

Finally, we compute the vector representation of each document by creating a vector of the same size as the vocabulary and setting its i-th coordinate to the number of occurrences of the i-th vocabulary token in the document. For instance, the first word in the vocabulary ”the” occurs two times in the first document, so the first component of its vector representation is 2:

Computing document vectors based on terms number of occurrences

```
Document 1: [2, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Document 2: [2, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
Document 3: [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]
```

Even though this model is efficient for many tasks, it has many drawbacks. Huge vocabulary leads to huge vectors since the size of the vector is the same as the size of the vocabulary. This also leads to the problem of the sparsity of the representation. Most of the values inside the vector are zeroes. Stop words have a high number of occurrences without being topic-specific terms. This problem could be addressed using the TF*IDF weighting discussed in the previous section. Two other significant drawbacks are losing the words order. Which means that the same vector may represent two sentences conveying different meanings, and missing all semantic information. This was pointed out by Le and Mikolov (2014). Using bag-of-words model, words like ”Paris”, ”strong” and ”powerful” are equally distant despite the fact that semantically ”powerful” is closer to ”strong” than ”Paris”. And this was one of the motivations behind developing the *Paragraph Vector* model.

2.2.2 Paragraph Vector

Paragraph Vector is an unsupervised method that learns vector representation of text input of variable length. The input can be sentences, paragraphs or whole documents.

The training method aims to predict the next word in a paragraph given the concatenation of the current paragraph vector and several word vectors. The first step is learning the word vectors to use them later for inferring paragraph vectors.

2.2.2.1 Word Vectors Learning

The following algorithm for learning distributed word representation is presented in Le and Mikolov (2014) and was inspired by works related to models that are known as neural language models (Bengio et al. (2003); Collobert and Weston

(2008); Mnih and Hinton (2009); Turian et al. (2010); Mikolov et al. (2013a) and Mikolov et al. (2013b))

All the words in the vocabulary are represented by a matrix W. Each column in W is the unique vector representing the word with the same index in the vocabulary. Given a training sequence of words $w_1, w_2, w_3, ..., w_T$, the training maximizes the average log probability given by the formula:

$$\frac{1}{T}\sum_{t=k}^{T-k} \log p(w_t | w_{t-k}, ..., w_{t+k})$$

the objective achieved using the softmax layer:

$$p(w_t | w_{t-k}, ..., w_{t+k}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

Each of the y_i is an un-normalized log-probability of output word i. It is given by the formula:

$$y = b + Uh(w_{t-k}, ..., w_{t+k}; W) \tag{2.4}$$

where U, b are the softmax parameters and h is constructed by a concatenation or average of word vectors from W. An overview of the framework is given by figure 2.1

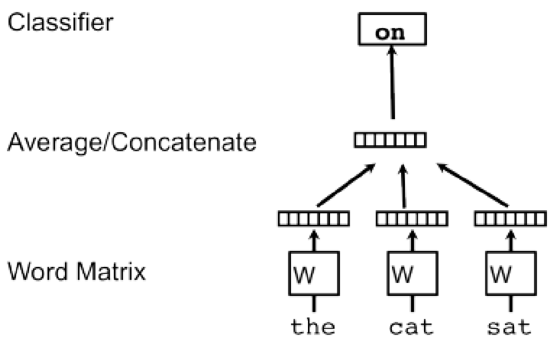

Figure 2.1: A framework for learning word vectors. Le and Mikolov (2014) Predicting "on" based on the context given by the three words ("the", "cat", and "sat").

Stochastic gradient descent with backpropagation Rumelhart et al. (1988) is used to train the neural network. Given the significant vocabulary sizes often used in practice, hierarchical softmax Morin and Bengio (2005) is more used to speed up the training compared to softmax. Le and Mikolov (2014) uses binary Huffman tree for the hierarchical softmax to quickly access more commonly used words.

2.2.2.2 Distributed Memory Model of Paragraph Vectors (PV-DM)

The same approach used to learn word vectors is expanded to learn *Paragraph Vectors*. Instead of just using word vectors as a context to predict the next word,

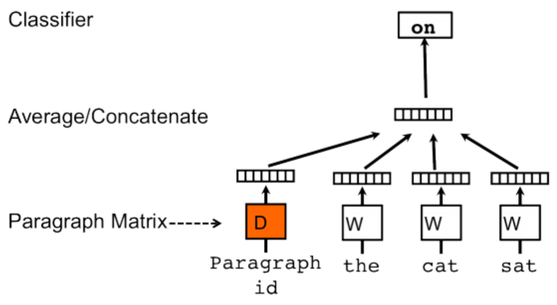

Figure 2.2: A framework for learning paragraph vectors. Le and Mikolov (2014) Comparing to figure 2.1, a column in D representing the current paragraph is included in the prediction context. The paragraph vector completes the missing information of the context by giving a representation of the paragraph topic.

a new vector representing the paragraph is added to the context to contribute in the prediction task like presented in figure 2.2.

A matrix D where each column maps to a paragraph vector is used in the framework in addition to the matrix W that stores the word vectors like in the word vectors model. The word vectors and the paragraph vectors are combined by either being concatenated or averaged to perform the task of predicting the next word. The main change is that h from equation 2.4 is now constructed from both W and D. W, D and both softmax weights U and b are trained using stochastic gradient descent with backpropagation.

The *Paragraph Vector* can be seen as a memory storing the paragraph's topic and is dealt with as an extra word in the input. Combined with word vectors that are already capturing the semantics at the training phase, it enhances the context representation and provides the prediction step with more relevant useful information. This idea inspired the name of this model Distributed Memory Model of Paragraph Vectors (PV-DM).

Contexts are fixed-length vectors that are constructed after choosing the words in a paragraph using sliding windows. The paragraph vector is shared only across the context of sliding windows inside the same paragraph but not across other paragraphs. However, W is the same for all the contexts. Which means that a vector representing a given word is the same all across the model.

Inferring a vector of an unseen paragraph is done using stochastic gradient descent like the training. The inferring process starts by adding a new column with random values representing the new paragraph to D. Using sliding windows, the model learns to predict the next word in the input. The column in D representing the input is updated with backpropagation while keeping W, U and b fixed. After a number of iterations, the vector representing the input pragraph is outputted.

Representing each paragraph and each word with vectors with respective dimensions of q and p while training on data containing N paragraphs and M words creates already $N \times p + M \times q$ parameters, which is a large number if N is large which may invoke questions related to the training efficiency. However, the fact that updates are sparse leads to efficient updates steps and so a quicker training process.

Paragraph Vectors present many advantages and relevant use cases. They can be used as paragraph features, either alone or in addition to bag-of-words models. Being a fixed length vector, they can be fed to various machine learning techniques. They are trained using an unsupervised method which means that there is no need for expensive labeled data. They can learn semantics thanks to inheriting this feature from the word vectors model where a word like ”solid” is closer to ”hard” than ”computer”. *Paragraph Vectors* are also sensitive to words order, which is a rare advantage that can be found only within n-gram models with a large n, but these models are disadvantaged by having high dimensional representations and poor generalization. This set of advantages, make of *Paragraph Vectors* a proper sentences embedding technique to apply to the task of automatic text summarization.

2.2.2.3 Distributed Bag of Words version of Paragraph Vector (PV-DBOW)

This variant model of *Paragraph Vectors* was also presented in Le and Mikolov (2014), but it is similar to the Skip-gram model that appeared originally in Mikolov et al. (2013b). The objective is to take only a paragraph vector as input and predict words randomly sampled from the paragraph. At each training iteration, a text window is sampled, a word from the sampled text is randomly chosen and stochastic gradient descent is executed. The model is presented in figure 2.3. Using this model, data storage needs are lower compared to PV-DM because we only store softmax weights for paragraphs and not for both paragraphs and words.

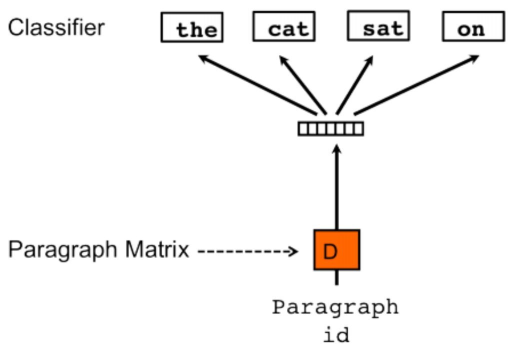

Figure 2.3: Distributed Bag of Words version of Paragraph Vector (PV-DBOW) Le and Mikolov (2014)
The *Paragraph Vector* predicts the words in a text window.

2.3 Sequence To Sequence Learning With Attention Neural Networks

The desire to bring neural networks success to the natural language processing field gave birth to a new set of approaches. The task of machine translation consists of translating an input sentence into a specific language. In other words, it takes a sequence of words as input and produces the right words sequence in the target language. Performing this task using neural networks created a new set of models capable of learning tasks involving sequences, including headline generation. Neural Machine Translation systems are end-to-end systems and most of them belong to the family of encoder-decoder (Sutskever et al. (2014); Cho et al. (2014a)). In this section, we explore successful sequence-to-sequence models in the task of neural machine translation. We start by presenting these models main components and how they work. Then we dive into the details of how these models were enhanced using the attention mechanism to establish the new state of the art in machine translation, before exploring an implementation of them.

2.3.1 The Encoder-Decoder Framework

Two variants of the Encoder-Decoder architecture that emerged almost simultaneously and achieved highly promising results with neural machine translation are Sutskever et al. (2014) and Bahdanau et al. (2014). The first presented an end-to-end approach to learning sequences and the second took it a step further with different design choices and new components such as the attention mechanism that we discuss in details in the next subsection.

Sutskever et al. (2014) primary motivation was the problem deep neural networks were facing with sequential problems despite their success in various learning tasks. A deep neural network works well when a problem can be represented with a fixed length vector like with visual object recognition. However, this is definitely not the case for sequential problems like machine translation or automatic text summarization where the input length is highly variable.

The idea is to use a first component called encoder that has a Long Short-Term Memory (LSTM) architecture Hochreiter and Schmidhuber (1997) that reads the input one timestep at a time and produces a large fixed dimensional vector representation. Then, this vector is fed to the decoder which is another LSTM Recurrent Neural Network Language Model (RNN-LM) that is conditioned on the input sequence as portrayed in figure 2.4.

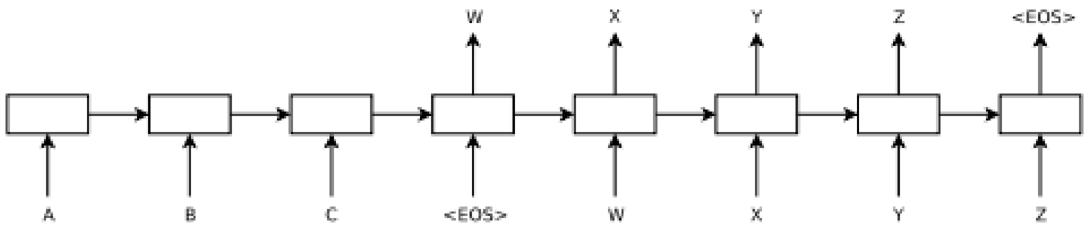

Figure 2.4: Sequence to sequence learning architecture Sutskever et al. (2014) An input sentence "ABC" is read and an output sentence "WXYZ" is produced. "<EOS >" is the end of sentence token. The model keeps outputing words till producing the "<EOS >" token.

Recurrent Neural Network (RNN) are by design the adaptation of classic feed-forward neural network to sequence learning. Given an input sequence $(x_1, x_2, ..., x_T)$, a classic RNN generates an output sequence $(y_1, y_2, ..., y_T)$ using:

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

where h_t is the hidden layer vector, W, U and b are the model weights and biases and σ are both activation functions.

One of the standard RNN drawbacks is losing long-term dependencies in the input sequence, something that is intolerable for a task like machine translation. LSTM architecture is good at learning data with long-range temporal dependencies and therefore it is used. Another difficulty is that the output sequence length can be different from the input length and things become more complicated when trying to establish non-monotonic relationships between input and output. Given an input sequence of length T, $X_T = (x_1, x_2, ..., x_T)$, the goal is to compute $p(Y_{T'}|X_T)$ where $Y_{T'} = (y_1, y_2, ..., y_{T'})$ and T' may differ from T. The LSTM encoder reads X_T step by step. When it reachs $x_T =<$ EOS $>$, which is the end-of-sentence symbol, the hidden state of the last LSTM step c is fed to LSTM decoder. c is a fixed length vector and is used as an intial state to start outputting $p(Y_{T'}|X_T)$ using the following equation:

$$p(Y_{T'}|X_T) = \prod_{t=1}^{T'} p(y_t|c, y_1, y_2, ..., y_{t-1}) \tag{2.5}$$

The LSTM decoder keeps iterating till producing the end-of-sentence token. This approach makes it possible to define a distribution over sequences of all possible lengths. Among the improvements found by Sutskever et al. (2014) are using a deep LSTM architecture with 4 layers and feeding the input sequence in a reverse order (i.e $x_T, x_{T-1}, ..., x_1$ instead of $x_1, x_2, ..., x_T$) which produced better results. However, this system was outperformed by a new architecture using an attention mechanism and jointly aligning and translating, that we cover in the following subsection.

2.3.2 Jointly Learning To Align And Translate With The Attention Mechanism

An encoder-decoder system is jointly trained to maximize the probability of the output sequence given a sentence. The problem with Cho et al. (2014a) and the previously discussed Sutskever et al. (2014) is that the sentence is always represented with a fixed-length vector before the decoding phase regardless of the input sentence length. This reduces the output translation quality with long sentences. Bahdanau et al. (2014) proposed a new model that finds where the most relevant information is concentrated in an input sentence using an attention mechanism. It generates a new word based on the context vector associated with the current time step and adaptatively computed based on the previously predicted words. This approach solved the long sentences problem.

2.3.2.1 Annotations Learning

Annotations are vectors that represent each input element in its context. We would like these vectors to be able to convey the meaning of the given element as well as its relation with elements in its surrounding. This is very important for abstractive summarization because words are not summarized one by one, but instead, one output word can summarize a group of words or an entire sentence. Learning these annotations could be achieved using a classic RNN, but the problem with this architecture is that the learned annotations will not convey any context from not-seen-yet words in the sequence, regardless of feeding the sequence to the RNN in the original order or in a reverse order. We need to get the context in both directions, forward and backward, and at the same time. The solution proposed by Bahdanau et al. (2014) is using the Bidirectional RNN architecture proposed originally by Schuster and Paliwal (1997).

The idea is to feed ”two different classic RNN” with the input sequence, one is forward and the other is backward. The forward RNN reads the input in the original order x_1 to x_{T_x}. However, the backward RNN reads the same sequence but in a reverse order x_{T_x} to x_1. At each step i, the forward RNN generates a forward hidden state $\vec{h}_i$ representing x_i in the context of the words preceding it. The backward RNN generates a backward hidden state $\overleftarrow{h}_i$ representing x_i in the context of the words following it. Both *part* annotations $\vec{h}_i$ and $\overleftarrow{h}_i$ are concatenated to form the final annotation $h_i = [\vec{h}_i; \overleftarrow{h}_i]^\intercal$ as presented in figure 2.5.

The annotations in each RNN can be learned using units that can learn long-term dependencies. Two of the most used units are long short-term memory (LSTM) Hochreiter and Schmidhuber (1997) and gated recurrent unit (GRU) Cho et al. (2014b). Bahdanau et al. (2014) model uses the GRU architecture.

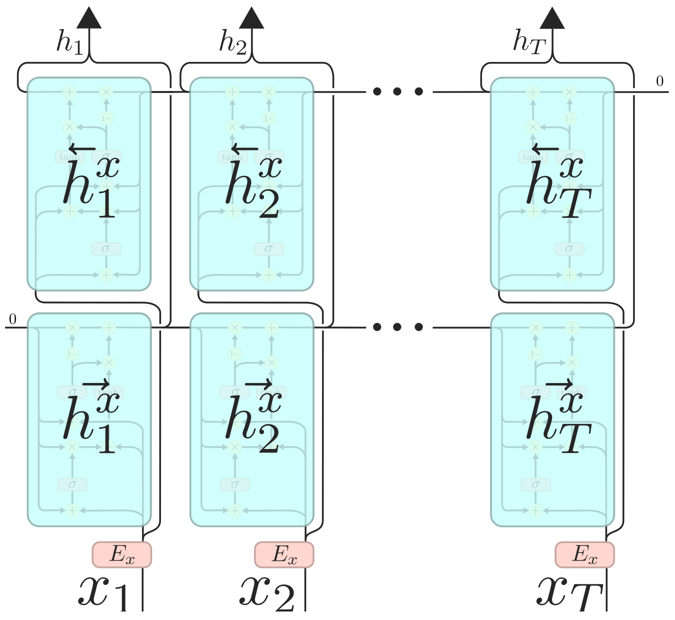

Figure 2.5: Bidirectional Recurrent Neural Network Encoder
A BiRNN of Gated Recurrent Units, taking word embedding of $X = (x_1, x_2, ..., x_{T_x})$ as input and computing their annotations.

The gated recurrent unit GRU is an essential component of this architecture. The LSTM architecture has motivated the GRU development. However, it is less complicated than an LSTM which has four gating units compared to two in a GRU.

Since appearing in Cho et al. (2014b), GRU made its way to successful applications especially with neural machine translation. In a GRU there is a hidden state h_i and two gates, a *reset gate* r_i and an *update gate* u_i. The goal is to compute at each step i the new hidden state h_i given the input at the i-th step x_i and the previous hidden state h_{i-1}. Figure 2.6 presents an illustration of a GRU.

The new hidden state is computed using this formula:

$$h_i = (1 - u_i) \odot h_{i-1} + u_i \odot \widetilde{h_i}$$

with

$$\begin{aligned}\widetilde{h_i} &= \tanh(Wx_i + U[r_i \odot h_{i-1}]]),\\ r_i &= \sigma(W_r x_i + U_r h_{i-1}),\\ u_i &= \sigma(W_u x_i + U_u h_{i-1})\end{aligned}$$

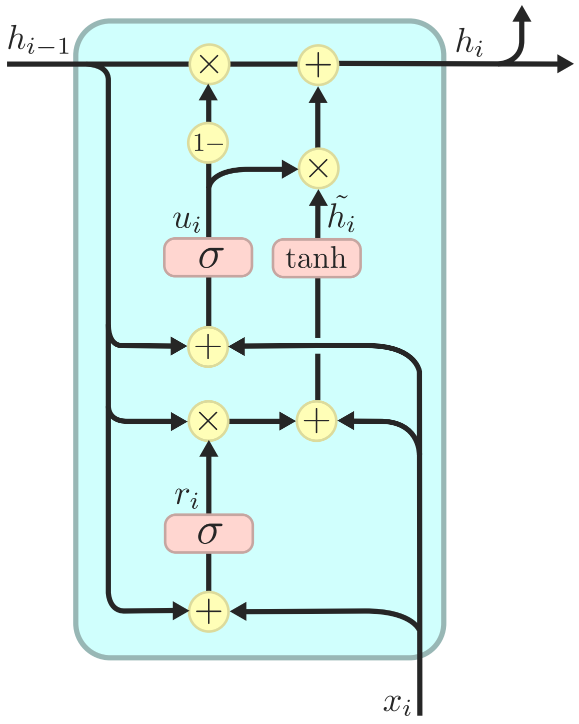

Figure 2.6: Gated Recurrent Unit
The Gated Recurrent Unit used in the encoder BiRNN

where W, U, W_r, U_r, W_u, U_u are learned parameters, σ is the logistic sigmoid function and bias terms were omitted for a better readability.

With this definition, the *reset gate* determines how much the new input affects h_i. When r_i is close to zero, h_{i-1} is almost ignored and x_i takes its way to have the biggest influence on h_i through the temporary $\widetilde{h_i}$. This mechanism gives the GRU the flexibility to switch to new representation when the new information contained in x_i is irrelevant to the previous states. The *update gate* is more about how much information to keep from the previous hidden state h_{i-1}. The goal of using GRU in the decoder or the encoder is to capture temporal dependencies. Giving each unit in the RNN its separate *update* and *reset* gate enables it to learn how to capture either long-term or short-term dependencies. The more long-term units will tend to have more active update gates, on the other side, units more sensitive to short-term dependencies their reset gates will be more active.

2.3.2.2 Attention-based Decoder

The decoder is trained to find $p(Y_{T'}|X_T)$ using an edited version of the equation 2.5:

$$p(Y_{T'}) = \prod_{t=1}^{T'} p(y_t|y_1, y_2, ..., y_{t-1}, X_T) \tag{2.6}$$

The conditional probabilities are defined as:

$$p(y_t|y_1, y_2, ..., y_{t-1}, X_T) = g(y_{t-1}, s_i, c_i) \tag{2.7}$$

where g is a nonlinear function, and at step i, c_i is the context vector and s_i is the hidden state of an RNN.

s_i is computed using the following equation:

$$s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{2.8}$$

In contrast to Sutskever et al. (2014) where there is only one context vector for the whole input sequence, in Bahdanau et al. (2014) model, each y_i has a distinct context vector c_i computed based on the sequence of annotations $(h_1, h_2, ..., h_T)$ provided by the encoder. Instead of generating the whole output sentence based on only one fixed-length vector in which the encoder must encode all the information present in the sentence, the decoder decides for every word in the target sentence which part of the source sentence deserves more attention, thanks to the following attention mechanism.

The weights α_{ij} implement an attention mechanism in the decoder by representing the importance of the annotation h_j in computing the next state s_i and generating y_i based on the previous state s_i and are computed by:

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{T} \exp e_{ik}},$$

where $e_ij = a(s_{i-1}, h_j)$ which is an *alignment model* with a a feedforward network trained with all the system components. The alignment model reflects the correlation between the inputs around position j and the output at position i by directly computing a soft alignment. This allows the backpropagation of the cost function gradient through the alignment model.

2.3.3 Nematus: A successful implementation of neural machine translation using attention mechanism

Sennrich et al. (2017) presented an open-source toolkit [1] that was used for top-performing submissions at 2016 Conference on Machine Translation (WMT16) and The International Workshop on Spoken Language Translation (IWSLT) for the task of shared translation and has been used to train systems for production environments. It implements an encoder-decoder architecture with an attention mechanism similar to the previously discussed Bahdanau et al. (2014) with many changes and enhancements. This implementation is the framework that we used for the baseline system that we used for the task of headline generation, and that will be discussed in details in a later chapter.

2.3.3.1 Dealing with Out Of Vocabulary Words

Working on natural language processing tasks using neural networks requires using fixed vocabulary sizes. This generally poses the problem of Out Of Vocabulary (OOV) words. This is the case with machine translation or abstractive summarization where, in contrast to extractive summarization, the vocabulary is open

[1] https://github.com/EdinburghNLP/nematus

and not limited only to the words present in the input text. Sennrich et al. (2015) proposed a solution to this problem, which consists in using Byte-pair encoding (BPE) to segment words. It was inspired by the compression algorithm proposed by Gage (1994). The original BPE is adapted to word segmentation, and vocabulary construction in order to avoid OOV.

The vocabulary construction starts by initializing the vocabulary with all the characters present in the training corpus and adding a special end-of-word token ("</w>" for example) to each word to be able to restore the original words after generating the tokens. The rest of the vocabulary is constructed using only merge operations of the existing symbols inside it. The number of merge operations is given as a parameter. At each iteration, the most frequent symbols pair is replaced by a new symbol added to the vocabulary. The algorithm terminates when it reaches a given number of merges. An example of the vocabulary construction algorithm execution on a demo input is given in figure 2.7.

As we can notice from the given an example, BPE starts at a character level, and the more merge operations are performed, the more whole words start to appear among the vocabulary symbols and the more the sequence of symbols forming the encoded sentence becomes shorter. These dynamics give BPE the ability to set a trade-off between the vocabulary size and the length of the sequences, two parameters that profoundly affect neural network models performances on tasks related to sequence-to-sequence learning Cho et al. (2014a).

Input Text:
vietnam</w> takes</w> measures</w> to</w> boost</w> rice</w> exports</w>

	Vocabulary	Encoded Sentence
Initialization	['a', 'c', 'b', 'e', 'i', '</w>', 'k', 'm', 'o', 'n', 'p', 's', 'r', 'u', 't', 'v', 'x']	v i e t n a m </w> t a k e s </w> m e a s u r e s </w> t o </w> b o o s t </w> r i c e </w> e x p o r t s </w>
After 1 merge operation	['a', 'c', 'b', 'e', 'p', '</w>', 'k', 'm', 'o', 'n', 'i', 's', 'r', 'u', 't', 'v', 'x', 's</w>']	v i e t n a m </w> t a k e s</w> m e a s u r e s</w> t o </w> b o o s t </w>r i c e </w> e x p o r t s</w>
After 10 merge operations	['</w>', 'vi', 'as', 'es</w>', 's</w>', 'nam', 'to', 'ri', 't</w>', 'ort', 'a', 'c', 'b', 'e', 'k', 'm', 'o', 'p', 's', 'r', 'u', 't', 'x']	vi e t n am </w> t a k es</w> m e as u r es</w> to </w> b o o s t</w> ri c e </w> e x p ort s</w>
After 34 merge operations	['takes</w>', 'measures</w>', 'exports</w>', 'boost</w>', 'rice</w>', 'vietnam</w>', 'to</w>']	vietnam</w> takes</w> measures</w> to</w> boost</w> rice</w> exports</w>

Figure 2.7: An example of the vocabulary construction using Byte-Pair Encoding (BPE) on a demo input

2.3.3.2 Decoder Initialization

According to Bahdanau et al. (2014) appendix on the model architecture, they initialize the decoder hidden state s_0 with the last backward encoder state: $s_0 = \tanh(W_s \overleftarrow{h}_1)$, where W_s are trained parameters. Nematus, uses an average annotaion instead defined by: $s_0 = \tanh(W_s \frac{\sum_{i=1}^{T} h_i}{T})$

2.3.3.3 Conditional Gated Recurrent Unit with Attention

Nematus replaced the standard RNN decoder and attention mechanism with a new conditional Gated Recurrent Unit (GRU) with attention. The new architecture has 3 components and takes as input its previous state s_{i-1}, the previous predicted word y_{i-1} and the annotations $H = (h_1, h_2, ..., h_T)$ to compute its new hidden state s_i that is later used to predict y_i:

$$s_i = \text{cGRU}_{att}(s_{i-1}, y_{i-1}, H)$$

The first component is a classic GRU architecture Cho et al. (2014b) that computes an intermediate hidden state s'_i using the following equations:

$$s'_i = \text{GRU}_1(y_{i-1}, s_{i-1}) = (1 - z'_i) \odot \tilde{s}_i' + z'_i \odot s_{i-1}$$

where

$$\begin{aligned}
\tilde{s}_i' &= \tanh(W' E_{y_{i-1}} + r'_i \odot U' s_{i-1}),\\
r'_i &= \sigma(W'_r E_{y_{i-1}} + U'_r s_{i-1}),\\
z'_i &= \sigma(W'_z E_{y_{i-1}} + U'_z s_{i-1})
\end{aligned}$$

E is the target words embedding matrix, $W', U', W'_r, U'_r, W'_z, U'_z$ are weights and σ is the sigmoid activation function.

The second component is almost the same attention mechanism as Bahdanau et al. (2014). It takes H and s'_i and computes the new context vector c_i.

The goal of the attention mechanism is to generate a context vector c_i conveying information from a given annotations sequence $H = (h_1, h_2, ..., h_T)$. It takes also the intermidiate hidden state s'_t as input. The figure 2.8 provides an illustration of the model. The first step is to compute the energies e_{ij} from each annotation h_i using the alignment model:

$$e_{ij} = v_a^T \tanh(U_a s'_i + W_a h_j)$$

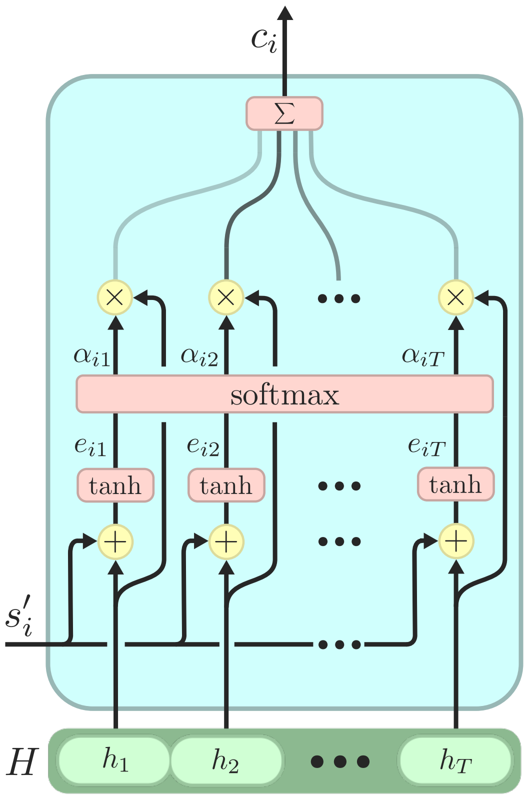

Figure 2.8: The Attention Model Computes context vector based on the relevant elements in the input sequence

where v_a, U_a, W_a are learned weights. Then, the energies are used to compute the probabilities α_{ij} defined by:

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{T} \exp e_{ik}}$$

These probabilities determine the importance of the influence that each element from the input has on the context c_i computed through this weighted sum:

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j$$

The third and last component of the conditional GRU with attention, is a second classic GRU that takes s'_i and the context vector c_i to finally produce the new hidden state s_i

$$s_i = \mathrm{GRU}_2(s'_i, c_i) = (1 - z_i) \odot \tilde{s}_i + z_i \odot s'_i$$

where

$$\begin{aligned} \tilde{s}_i &= \tanh(W c_i + r_i \odot U s'_i), \\ r_i &= \sigma(W_r c_i + U_r s'_i), \\ z_i &= \sigma(W_z c_i + U_z s'_i) \end{aligned}$$

W, U, W_r, U_r, W_z, U_z are weights and σ is the sigmoid activation function.

As we can notice, the recurrence in this architecture is not produced at the level of each GRU but is assured for the whole cGRU$_{att}$ thanks to the intermediate hidden state s'_i that assures the link between the 3 different components.

2.3.3.4 Computing Conditional Probability

Where a maxout layer Goodfellow et al. (2013) is used by Bahdanau et al. (2014) to compute the conditional probability $p(y_i|s_i, y_{i-1}, c_i)$ before applying a softmax layer, Sennrich et al. (2017) uses a feedforward hidden layer

Once the new hidden state s_i and the context vector c_i are computed, they are fed to the deep out layer in addition to a vector representing the previously generated word y_{i-1} as illustrated in the figure 2.9.

$$p(y_i|s_i, y_{i-1}, c_i) = \mathrm{softmax}(t_i W_o)$$

where

$$t_i = \tanh(s_i W_{t1} + E_{y_{i-1}} W_{t2} + c_i W_{t3})$$

where $W_{t1}, W_{t2}, W_{t3}, W_o$ are weights.

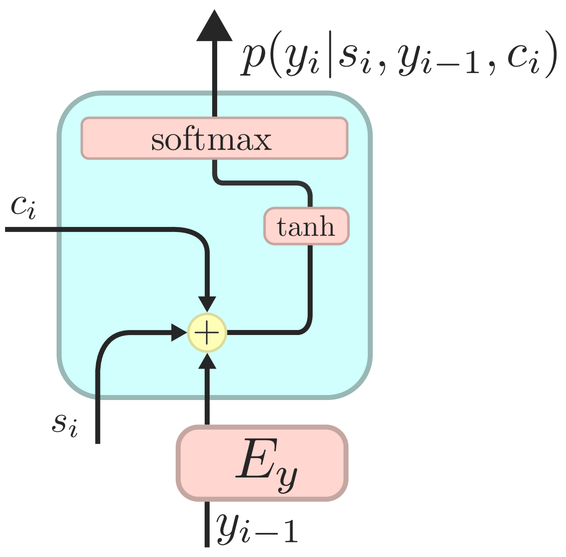

Figure 2.9: The deepout layer
Computing $p(y_i|s_i, y_{i-1}, c_i)$ given the hidden state s_i, the context vector c_i and the previously generated word y_{i-1}

2.3.3.5 Generating The Output Sequence

The output sequence generation process starts by applying the BPE encoding to the input sentence to form the input sequence X. It is then fed to the encoder to generate the annotation sequence H. $P(y_1|H)$ is computed by the decoder for all possible y_1 in the vocabulary. From this step, beam search is used to choose the words forming the full summary. Only the best k possible y_1 are kept, and the rest are pruned. Each of the y_1 candidates is fed separately to the decoder at step $i = 2$, again only the best k summaries based on their probabilities are kept, and the rest are pruned. Beam search continues by exploring at each time step the k best predictions till producing the end-of-sentence token "eos". Finally, the candidate sequence with the highest probability is returned.

Chapter 3

Related Work

In this chapter, we go through various architectures that were proposed to solve the problem of abstractive summarization using neural networks. The approaches range from using different encoding and decoding techniques, augmenting the input with a variety of features or feeding the model with extra information. These methods results will be compared to our models' results in chapter 6.

3.1 Neural Attention Model for Sentence Summarization

Rush et al. (2015) could be considered as the first attempt to generate abstractive sentences summaries using neural network language model (NNLM) with an attention mechanism. In contrast to Bahdanau et al. (2014) attention mechanism that is used in the decoder component and takes the encoder annotations as input, the attention model proposed by Rush et al. (2015) uses only feedforward networks, takes as input the source sentence with a window of the previously generated output words, and most importantly, it is located in the encoder part of the model.

The proposed architecture aims to generate a headline for a given article by taking only the article's first sentence as input.

The language model estimates the conditional probability of the next word y_{i+1} given the input sentence x, the context window of size C, the previously generated window defined by $y_c = y_{[i-C+1,\dots,i]}$ and the learned parameters θ:

As portrayed in the figure 3.1, this is the used lanquage model:

$$p(y_{i+1}|x, y_c; \theta) \propto exp(Vh + W\mathbf{enc}(x, y_c))$$
$$h = \tan(UEy_c)$$

where E is a word embedding matrix and $\theta = (E, U, V, W)$ learned parameters. The function **enc** is a black-box encoder. The paper presents the details of 3 different variants of the encoder each using completely different architectures: Bag-of-Words encoder, convolution encoder and attention-based encoder. The best performing encoder and the one on which the paper focuses the most is the attention-based encoder, presented in figure 3.1 (b).

Even though the attention-based encoder is mainly inspired by Bahdanau et al. (2014) work, it has many differences. It uses a window of generated words as context rather than a sequence of annotations and the last generated words. Moreover, it uses a weighted dot-product as alignment model instead of multi-layer perception:

$$\begin{aligned}
\text{enc}_{att} &= p^T \bar{x}, \\
p &\propto \exp(\tilde{x} P \tilde{y_c'}), \\
\tilde{x} &= FX, \\
\tilde{y_c'} &= G y_c, \\
\forall i \quad \bar{x}_i &= \sum_{q=i-Q}^{i+Q} \frac{\tilde{x}_i}{Q}
\end{aligned}$$

where F is a word embedding matrix, G is a context embedding matrix and P a weight matrix between the context and the word embedding. All 3 matrices are learned parameters.

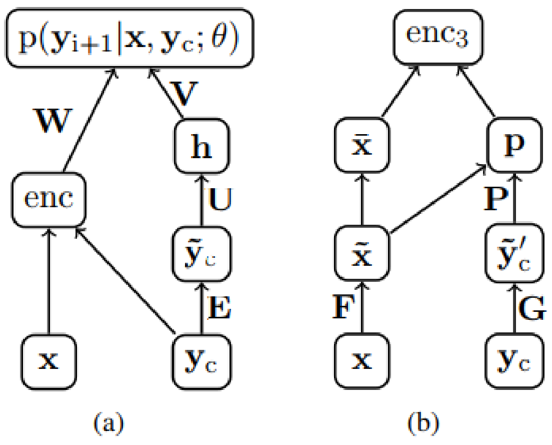

Figure 3.1: Neural language model used by Rush et al. (2015)
(a) An overview of the language model with a black-box encoder (b) The attention-based encoder architecture

3.2 Abstractive Sentence Summarization with Attentive Recurrent Neural Networks

Recurrent Attentive Summarizer (RAS) is a model that was proposed by Chopra et al. (2016). It uses an attentive convolutional encoder and a recurrent decoder. This model performed better than the state-of-the-art at its publication Rush et al. (2015) and this without any extractive features optimization and with only end-to-end training on large datasets. The proposed architecture, same as its predecessor, aims to generate a headline for a given article by taking only the article's first sentence as input.

For a given input sequence $X = (x_1, ..., x_T)$, the encoder computes a context vector c_t for each timestep that will be fed later to the decoder. Each word x_i is represented by $a_i = e_i + l_i$ where e_i is a learned embedding of the word x_i and l_i is a leaned embedding associated with the position i of the word. Let d be the dimension of a_i and let $B^1, ..., B^d$ be d matrices of dimension $q \times d$ used to convolve over the a_i. The convolution is done by computing a new vector z_i representing x_i using this equation:

$$z_i = [z_{i1}, ..., z_{id}]$$

$$\forall k, z_{ik} = \sum_{h=-\frac{q}{2}}^{\frac{q}{2}} a_{i+h} \cdot b^k_{\frac{q}{2}+h}$$

where each b^k_j is the j-th column of the matrix B^k.

Each z_i represents at the same time the word x_i and information related to its position in the input sequence thanks to the convolution window of size q. In Chopra et al. (2016) experiments, q was fixed to 5, and to compute the aggregate embedding vector z_i for words on the sequence words, padding with dummy words was applied.

Once the z_i are computed, an attention mechanism similar to Bahdanau et al. (2014) is used to compute the context vector c_t:

$$c_t = \sum_{j=1}^{M} \alpha_{j,t-1} e_j$$

where

$$\alpha_{j,t-1} = \frac{\exp(z_j \cdot s_{t-1})}{\sum_{i=1}^{M} \exp(z_i \cdot s_{t-1})}$$

s_{t-1} is the hidden state of the recurrent decoder, defined by the equation 2.8. Chopra et al. (2016) experimented with two different types of decoders. The first is an Elman RNN like described in Elman (1990), and the second, an LSTM Hochreiter and Schmidhuber (1997).

3.3 Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond

The two previously discussed models on neural abstractive summarization focused mainly on generating a headline based on the article's first sentence. Many works followed them trying either to extract linguistic features from the first sentence and feed it to the model or take more sentences from the article. For instance, Nallapati et al. (2016) proposed various models for the task of headline generation as well as the task of multi-sentence summary generation. In this subsection, we focus on the proposed models that showed promising results in the task of headline generation.

Nallapati et al. (2016) used a sequence-to-sequence with attention model like described in the previously discussed Bahdanau et al. (2014) as baseline. Then, they started enhancing mainly the encoder component using the following techniques.

3.3.1 The Feature-rich Encoder

The idea is to capture keywords in the input using features that the word embedding may not suffice to represent them and feed them to the encoder in-addition to the traditional word embedding. The extracted features are part-of-speech tags, named entities and the previously discussed TF and IDF weights. For the vocabulary of the different discrete tag-types, new embedding matrices are used. However, for the continues values such as TF and IDF are discretized into a fixed number of bins and one-hot representation is used to code them. All these feature tags are added to pretrained Mikolov et al. (2013a) word vectors and fed to the encoder, as presented in figure 3.2

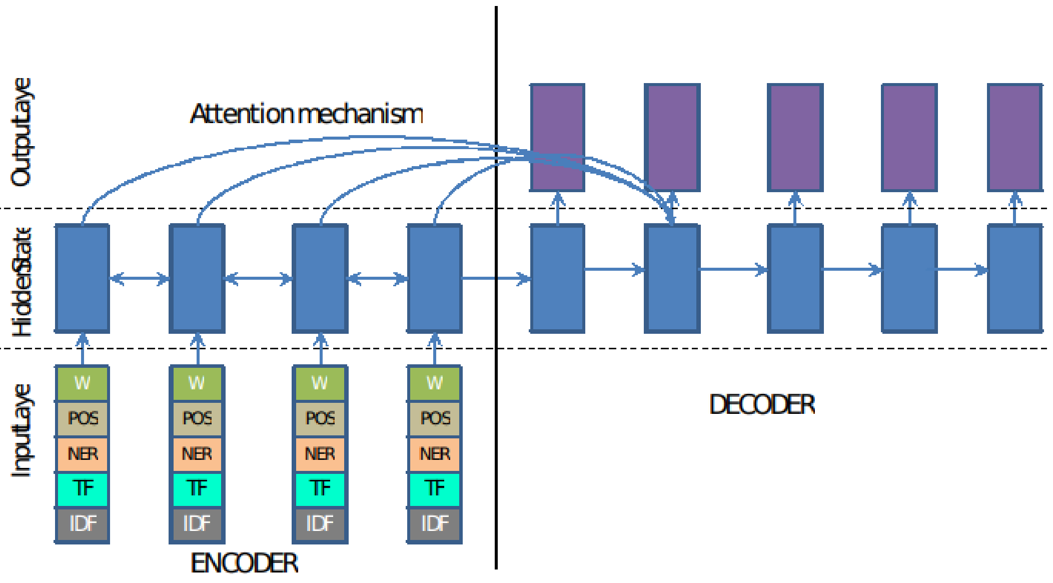

Figure 3.2: Feature-rich Encoder Nallapati et al. (2016)
Word embedding concatenated with linguistic features (POS, NER tags and discretized TF and IDF weights) and then fed to the sequence-to-sequence attentional encoder-decoder.

3.3.2 Solving Out Of Vocabulary Words Problem Using Generator-Pointer Switch

One of the significant problems in summarization is the out of vocabulary words (OOV). What should the decoder output when a word is not in his target vocabulary? This happens with rare words. And for a task like summarization, these rare words are sometimes the kind of information that the summary should contain because they can be names or highly specific words, without them, the summary loses a lot on the informativeness level. One of the standard practices is to replace these (OOV) with a special token that marks them like ”UNK”. However, seeing a ”UNK” in a summary is not a desirable result. For our model, we used the previously presented BPE encoding that was proposed by Sennrich et al. (2015). Nallapati et al. (2016) proposed another solution which is to include a pointing mechanism in the decoder that learns when a word from the known vocabulary should be generated or when it should be directly copied from a position in the input instead of outputting the ”UNK” placeholder.

At every time step, the new switch decides whether to generate a word when it is turned on or to point to the source when it is turned off. Figure 3.3 gives an overview of the used architecture. A sigmoid activation controls the switch state using the following formula:

$$P(s_i = 1) = \sigma(v^s \cdot (W_h^s + W_e^s E_{o_{i-1}} + W_c^s c_i + b^s))$$

where $P(s_i = 1)$ is the probability of the switch being ”on” at the i-th decoder timestep, $E_{o_{i-1}}$ is the embedding vector of the emission from the previous time step, c_i is the context vector outputted by the attention mechanism, and $W_h^s, W_e^s, W_c^s, b^s, v^s$ are learned parameters. The pointer's value p_i for the i-th word in the summary is determined using P_i^a, the words attention distribution over the input:

$$p_i = \text{argmax}_j(P_i^a(j))\text{for} j \in 1, ..., T, P_i^a(j) \quad = exp(v^a \cdot (W_h^a h_{i-1} + W_e^a E_{o_{i-1}} + W_c^a h_j^d + b^a))$$

where $P_i^a(j)$ denotes the probability of the decoder's i-th time-step pointing to the position j in the input and h_j^d the encoders hidden state at position j.

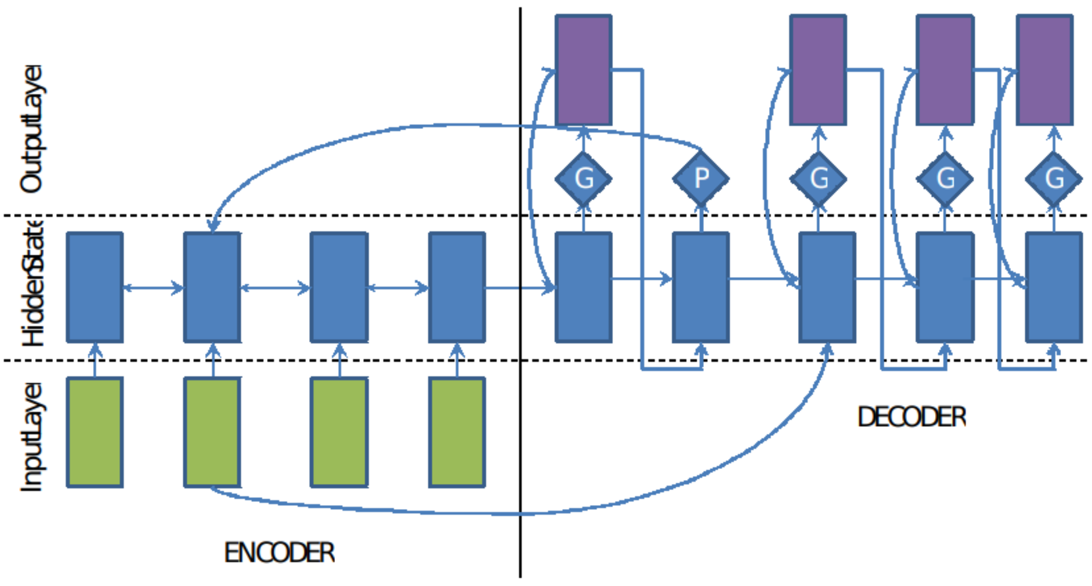

Figure 3.3: Switching generator/pointer model Nallapati et al. (2016)
When the switching gate shows ”G”, the decoder operates as originally described to generate a word from its target vocabulary. When it shows ”P”, a word at an automatically determined position in the input is directly copied to the output.

The model learns when to turn the switch on and off and in which position to point thanks to the training data that explicitly contains this information. When a word out of the decoders vocabulary occurs many times in the input document, the pointer points to its first occurrence. The training optimizes the following log-likelihood:

$$
\begin{array}{llll}
 & \log(P(Y|X)) & = \sum_i (U(i) + D(i)) \\
\text{where} & \forall i, \quad U(i) & = g_i \log(P(y_i|y_1, ..., y_{i-1}, X) P(s_i)), \\
 & \forall i, \quad D(i) & = (1 - g_i) \log(P(p(i)|y_1, ..., y_{i-1}, X)(1 - P(s_i))), \\
\text{and} & \forall i, \quad g_i & = \begin{cases} 0 & \text{if} \quad x_i \quad \text{is out of the decoder's vocabulary} \\ 1 & \text{otherwise} \end{cases}
\end{array}
$$

When testing, the switch state is determined using the estimated probability $P(s_i)$.

3.4 Faithful to the Original: Fact Aware Neural Abstractive Summarization

When many previous works mainly focus on generating informative abstractive summaries, Cao et al. (2017) focused however on the faithfulness of the summaries. They reported that nearly 30% of summaries outputted by state-of-the-art sequence-to-sequence systems convey false facts. To solve this problem, they got the idea of extracting the facts that are present in the article's first sentence and feed them to the model in addition to the words of the same sentence. To do so, they use the Open Information Extraction (OpenIE) Banko et al. (2007) to get the facts. Since OpenIE is not able to always extract facts like in imperative sentences, a dependency parser is also used to enhance the extracted facts. All the extracted facts are then concatenated using a special separator,"|||" for instance, and are fed to a BiGRU encoder as described in Cho et al. (2014b).

To deal at the same time with the input sentence and the input facts, Cao et al. (2017) proposed a dual-attention network shown in the figure 3.4. Two separate encoders are used. The first deals with the input words sequence and the second one with the facts sequence. Each encoder's annotations are to a separate attention mechanism to generate context vectors. The two context vectors are then combined into one vector either by concatenation or by a weighted sum where multi-layer perceptrons learn the weights. Finally, the conditional probability of the next word in estimated using a softmax layer, the same way as in Bahdanau et al. (2014).

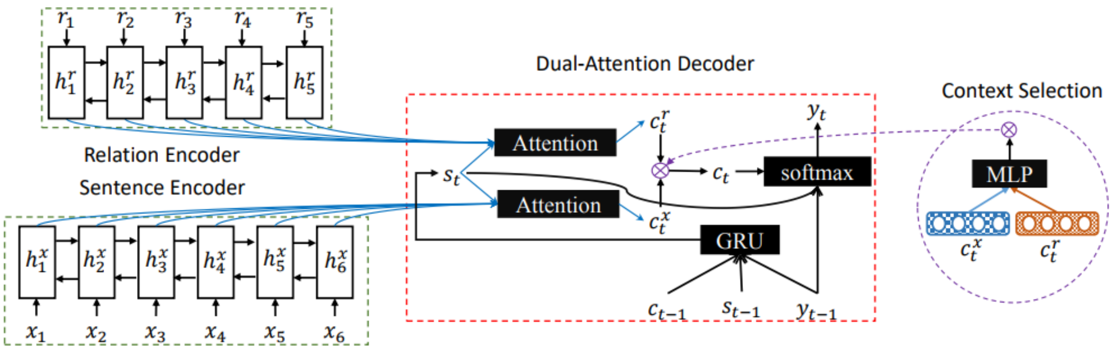

Figure 3.4: Dual-attention model for facts aware neural summarization Cao et al. (2017)

Two separate encoder-attention-mechanisms are used to deal with each sequence. The first receives words of a sentence and the second gets extracted facts. Attention contexts are then combined to generate the next word.

Chapter 4

Abstractive Headline Generation Based On Multiple Sentences

One of the most significant problems of the existing abstractive headline generation approaches is that they take only the article's first sentence as input, and so miss any critical information that may appear later in the article. Also, from a use scenario point-of-view of such a system, a headline is at best as informative as the article's first sentence. Despite being a good starting point, it still needs a lot of improvements in the hope of being able to use the system in more challenging environments and use cases. Among the goals of this thesis is generating the headlines while taking in consideration additional information that may appear anywhere in the article. To try to achieve this goal, we tested various possible approaches, and we go through them in this chapter.

Given that trying to feed every word in the article to the baseline system leads to extremely long input sequences which require more computing capacity and extended training time, we explored other methods that still train in a reasonable time with the available computation power. The approaches we tested involved encoding every sentence in the article using the Paragraph Vector model and using it as an additional input to enhance the baseline sequence-to-sequence with attention architecture. This approach provides us with two potential input sequences: the words of the article's first sentence and the sentences' vectors. The challenge was adapting the baseline architecture to receive more than one sequence as input.

We start the chapter by presenting a simple architecture we used and which is based on concatenating the input sequences and feeding them to the baseline architecture. The second part of the chapter is dedicated to presenting a new architecture we implemented based on using a dual-attention mechanism.

4.1 Concatenated Sequence of Articles Sentences Vectors And Word Vectors

Using the Paragraph Vectors presented in section 2.2.2 we pretrain a model on generating sentence vectors. The same model contains word vectors that are shared between all the inferred sentence vectors. Given an article composed of T_s sentences, we infer the T_s corresponding sentence vectors $(s_1, ..., s_{T_x})$. At the same time, we get the word vectors for every word in the articles first sentence forming the sequence $(w_1, ..., w_{T_w})$. These two sequences are concatenated as presented in the figure 4.1 and are fed to the sequence-to-sequence model presented in section 2.3.3 with only one difference: we remove the word embedding layer from the input sequence since we are already feeding the model embedding vectors. However, we keep the word embedding layer for the output, and we do not force it to use the same pretrained word embedding as the input to give the summarizer more flexibility at learning the desired task.

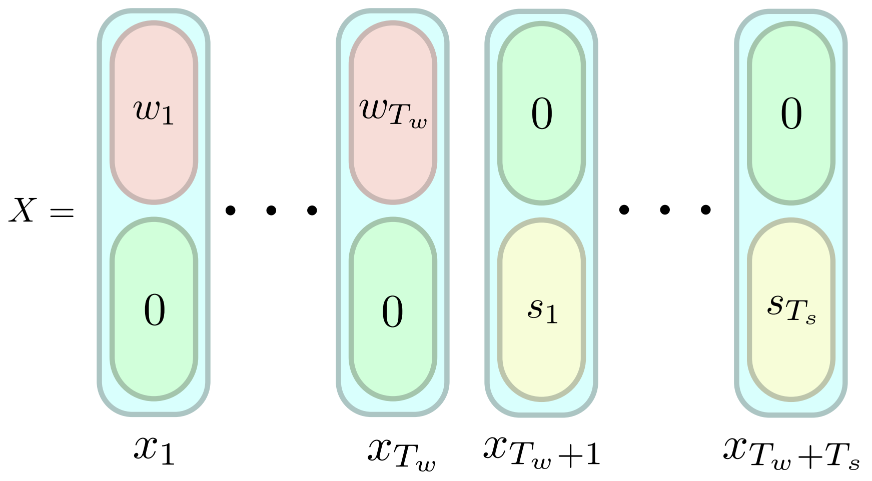

Figure 4.1: Concatenating word vectors and sentence vectors
Vector embeddings of every word in the first sentence in the article are concatenated with the sentence vectors before feeding them to the sequence-to-sequence with attention model.

We expect from this way of combining the words and sentences embedding that the neural network will be able to learn the difference between the dimensions occupied by the words vectors space and the dimensions reserved for the sentence vectors. This should give the existing attention mechanism the ability to better weight at each time step the sentences and the words in the input relevant to generate the next word in the summary. We hope that this simultaneous access to both the words sequence and sentence sequence will improve the baseline system performance.

In another experiment, instead of using the word vectors that are generated thanks to training a model on inferring sentence vectors, we extract pretrained word embeddings from the same baseline model presented in section 2.3.3 after training it on the task of headline generation. We use the model's learned word embedding layers at the input and use it as our new pretrained vectors for words representation. The word vectors of the article's first sentence and the sentence vectors of each sentence in the article are fed to the sequence-to-sequence model with attention after being combined as presented in the figure 4.1. This approach enables us to experiment with a second way of representing the words and so more chances to improve the results or to compare them with existing ones and better understand the dynamics of the system.

4.2 Sequences To Sequence Learning Using Conditional Dual-Attention

In this section, we dive into the main contribution of this thesis. It is a new model adapted to learn the transformation between two sequences as input and one target sequence. The architecture is heavily inspired from Bahdanau et al. (2014) and Sennrich et al. (2017) and reuses some ideas from Cao et al. (2017). The section starts by presenting the task and taking an overview of the different model components and how they interact with each other. After that, each component's architecture is deeply described.

4.2.1 The Task

This model was developed for the task of abstractive automatic text summarization and more precisely abstractive headline generation. However, it can be reused or extended for any task involving sequence generation based on an input consisting of more than one sequence. Let us start by formally presenting the task and fixing the notations for this entire section.

Given an input article, the goal of the headline generation task is to produce a condensed one sentence summary. Let the input article consist of T_y sentences, where the article's first sentence is a sequence of T_x words from a vocabulary of size V.

Let $X = (x_1, x_2, ..., x_{T_x}) \in [1 \mathinner{..} V]^{T_x}$ denote the article's first sentence words sequence, and $Y = (y_1, y_2, ..., y_{T_y}) \in [\mathbb{R}^{d_y}]^{T_y}$ a sequence of vectors of dimension d_y each representing one of the article's sentences and inferred using the Mikolov et al. (2013b) approach presented in subsection 2.2.2.

The model aims to find a target summary sequence $Z = (z_1, z_2, ..., z_{T_z}) \in [1 \mathinner{..} V]^{T_z}$. The sequence should maximize the conditional probability of Z given X and Y: $\arg\max_Z p(Z|X, Y)$. A parametric model is trained to maximize the conditional probability of article-headline pairs using a corpus containing training pairs.

4.2.2 The Model Overview

An overview of the proposed model is presented in figure 4.2. It takes X and Y as input and is consisted of 3 main components: two remarkably similar encoders for X and Y, the only difference is an extra learned word embedding layer for the words in X, and the conditional dual-attention decoder that takes the learned annotations of the input sequences and its previous hidden state to estimate the conditional probability $p(z_t|X, Y, Z_{t-1})$ where $Z_{t-1} = (z_1, z_2, ..., z_{t-1})$.

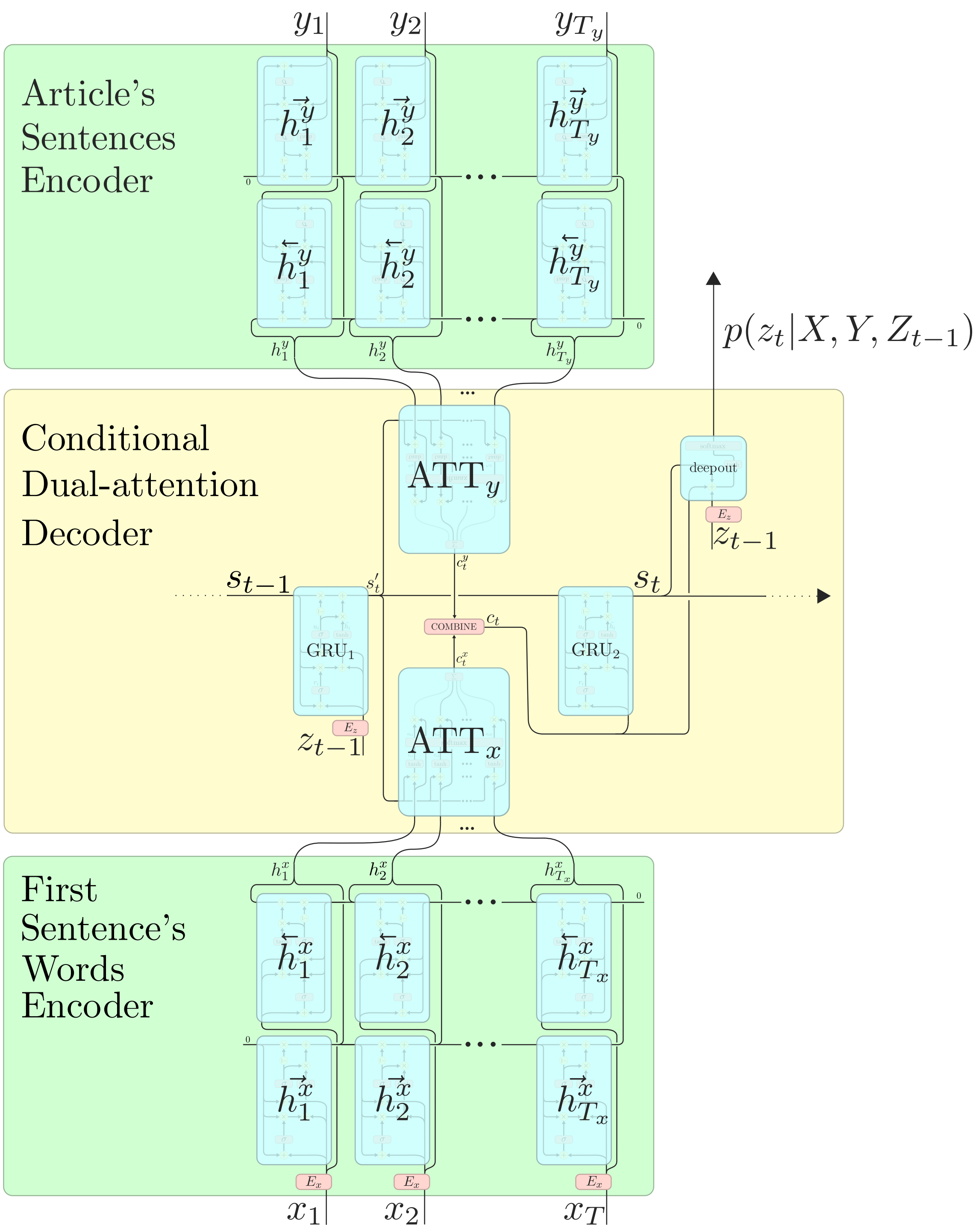

Figure 4.2: Sequences to sequence learning with conditional dual-Attention model
An illustration of the different model components estimating the t-th output word conditional probability $p(z_t|X, Y, Z_{t-1})$ where $Z_{t-1} = (z_1, z_2, ..., z_{t-1})$

4.2.3 Inputs

The input consists of two sequences, the words of the article's first sentence X and the article sentences Y.

4.2.3.1 The article's first sentence

X is the sequence of the words of the articles first sentence. Each word is mapped to an integer representing its position in a fixed vocabulary of size V.

After encoding each word in the article's first sentence to its corresponding symbols in a learned BPE vocabulary like presented in the section 2.3.3.1, we get X by mapping each symbol to its encoding integer id in the vocabulary. Each word embedding vector is jointly leaned by the model when training for the task of headline summarization.

4.2.3.2 The articles sentences

A distributed memory model of Paragraph Vectors as presented in the subsection 2.2.2 is pretrained using sentences from the training corpus. The model is trained on separate sentences and not on paragraphs or whole articles to keep consistency between the training and the inferring phase. This pretrained *"sentence vector"* model is then used to infer a float value vector of dimension d_y for every sentence in the input article. The sequence of the inferred vectors is Y.

4.2.4 The Encoder

The new model as illustrated in the figure 4.2, contains two separate encoders that are very similar. Each encoder's task is to transform the input sequence into a sequence of annotations of the same size that are fed later to the attention mechanism in the decoder. These encoders have the same architecture presented in figure 2.5 with one difference. The only difference is in the encoder responsible for learning annotations for the sentence vectors sequence Y, where the embedding layer is removed because the model is directly fed with embedding vectors. Y is already a sequence of vectors and no further embedding is needed before feeding it to the gated recurrent units in the encoder.

4.2.5 The Decoder

The decoder is in charge of estimating the conditional probability $p(z_t|X, Y, Z_{t-1})$ at each time step t. It is also a RNN, but with many small components with an architecture partially similar to the architecture proposed by Sennrich et al. (2017) and presented in the subsection 2.3.3. Here we present the main differences between the existing architectures. It takes as input the annotations $H_x = (h^x_1, h^x_2, ..., h^x_{T_x})$ and $H_y = (h^y_1, h^y_2, ..., h^y_{T_y})$ of respectively the sequences X and Y that are the output of the previously presented encoders, its own previous hidden state s_{t-1} and the previous predicted target word z_{t-1}.

4.2.5.1 The Initial Hidden State

The choice of the decoder's initial hidden state s_0 is very important for the whole model performance. A wrong choice at this step affects the predictions of the first word and possibly all the following words in the summary, thus the overall model performance. Bahdanau et al. (2014) choose to compute it using only the first backward encoder state $\overleftarrow{h}_1$ by defining $s_0 = \tanh(W_{\text{init}}\overleftarrow{h}_1)$ where W_{init} is a learned matrix. However, Sennrich et al. (2017) took in consideration all the annotations outputted by the decoder to compute s_i using an average annotation $s_0 = \tanh\left(W_{\text{init}} \frac{\sum_{i=1}^{T_x} h_i}{T_x}\right)$ where W_{init} is a learned matrix.

For our model, the problem is a bit more complicated because we do not have annotations of only one sequence, but of two different sequences. This situation is similar to the one that will be discussed shortly when presenting the dual-attention mechanism. For now, let us say that we have a function named COMBINE that takes as input two vectors of the same size containing float values, one related to X and the other related to Y and outputs one single float vector. The COMBINE function changes according to the implementation of the model. To compute s_0 for our dual-attention decoder, we start by calculating the means of the annotations outputted by the encoders for both input sequences:

$$h^x_{mean} = \frac{\sum_{i=1}^{T_x} h^x_i}{T_x},$$
$$h^y_{mean} = \frac{\sum_{i=1}^{T_y} h^y_i}{T_y}$$

Then, we compute h_{mean} using COMBINE:

$$h_{mean} = \text{COMBINE}(h^x_{mean}, h^y_{mean})$$

Next, we compute s_0:

$$s_0 = \tanh(W_{\text{init}} h_{mean})$$

where W_{init} is a learned matrix.

4.2.5.2 The Conditional Dual-attention Mechanism

At this step, we have the annotations of both input sequences $H_x = (h^x_1, h^x_2, ..., h^x_{T_x})$ and $H_y = (h^y_1, h^y_2, ..., h^y_{T_y})$. In previous works (Sennrich et al. (2017); Bahdanau et al. (2014)), the annotations of one sequence are computed, and the attention mechanism is applied to only one sequence of annotations. In this case, we have two sequences, and we would like to have a model able to learn to extract from the two sequences the right information to generate the next word in the summary. We take the same architecture with the conditional GRU as in the subsection 2.3.3 and add an attention mechanism for the second annotations sequence as well as a fifth component to combine two generated context vectors. The individual components are not recurrent, but the whole system is recurrent via the hidden state s_t.

After generating the two context vector c^x_t and c^y_t at time step t for both input sequences X and Y, a context vector c_t is computed as a combination of both context vectors using COMBINE:

$$c_t = \text{COMBINE}(c^x_t, c^y_t)$$

We have already seen the function COMBINE a first time in the previous paragraph 4.2.5.1, but we did not detail what it is and how it can be implemented. The

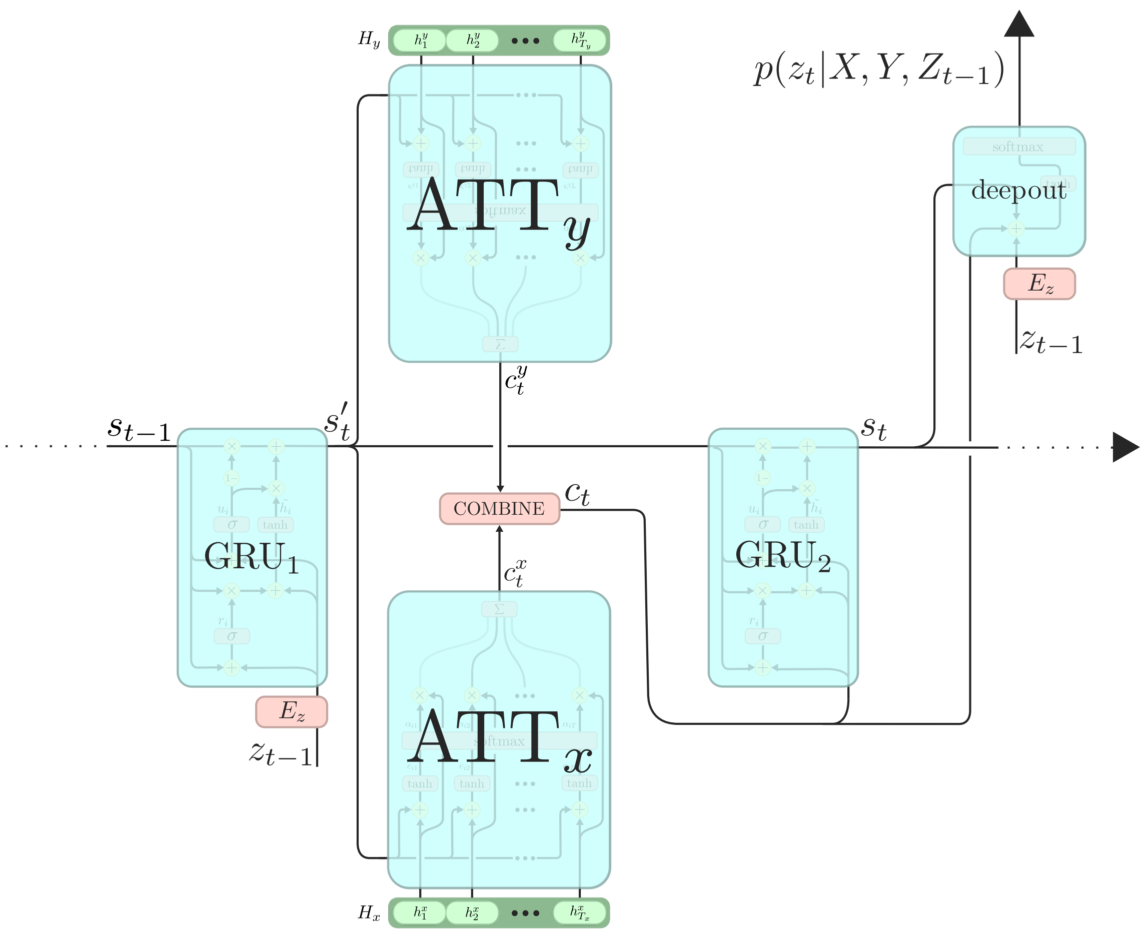

Figure 4.3: Conditional Dual-Attention Decoder
Conditional dual-attention decoder with 2 GRUs and 2 attention mechanisms and a deepout layer. It takes the annotations H_x and H_y, the previous hidden state s_{t-1} and the previously generated word z_{t-1} to estimate the next conditional probability $p(z_t|X,Y,Z_{t-1})$

COMBINE takes as input two vectors of the same size containing float values, one related to X and the other related to Y and outputs one single float vector.

In our different experimental implementation of the model, we tried many possible variances of the COMBINE function:

$$\begin{aligned}
\text{Mean:} \quad & \mathbb{R}^d, \mathbb{R}^d \to \mathbb{R}^d \\
& c_t^x, c_t^y \mapsto \tfrac{1}{2}(c_t^x + c_t^y) \\
\text{Concatenate:} \quad & \mathbb{R}^{d_x}, \mathbb{R}^{d_y} \to \mathbb{R}^{d_x+d_y} \\
& c_t^x, c_t^y \mapsto [c_t^x; c_t^y] \\
\text{MLP:} \quad & \mathbb{R}^d, \mathbb{R}^d \to \mathbb{R}^d \\
& c_t^x, c_t^y \mapsto p_t \odot c_t^x + (1-p_t)c_t^y \\
& \text{with } p_t = \tanh(W_p \odot [c_t^x; c_t^y] + b_p), \\
& \text{where } W_p \text{ and } b_p \text{ are learned parameters.}
\end{aligned} \tag{4.1}$$

Chapter 5

Experiments Environment & Implementations

5.1 Datasets

We train and evaluate all our proposed models for headline generation using two different data sets: Gigaword and DUC-2004

5.1.1 Gigaword

The annotated version of the Gigaword corpus (Graff et al. (2003);Napoles et al. (2012)) is the main data set we use for our experiment. We use it in a similar way as described in Rush et al. (2015) with editing some preprocessing steps in the script that they made available [1].

The Gigaword corpus contains nearly 10 million articles from seven news sources. Most of the annotations in the original data set are ignored such as parse trees, dependency tress, named entities and in-document coreference chains. Only tokenization and sentences segmentation are used to form the article-headline pairs. We start by deleting all the unused tags and by keeping only the article headline and the full article with a separation between each sentence in contrast with the original script that only creates pairs of the headline and the article's first sentence. Also, each digit is replaced with a '#'. At step two, the data set is compiled into a train set, development set and test set. For the splitting we use the same sets as Rush et al. (2015). For step three, each data entry is filtered if one of the following conditions it true:

- The headline is blank or the articles first sentence does not contain a period.
- The headline contains words from a blacklist set of spurious words
- The headline is shorter or longer than a specific length parameters
- There are no common words between the headline and the article's first sentence

[1] https://github.com/facebookarchive/NAMAS (retrieved on Feb 05, 2018)

After filtering the data set, we get 4,167,425 headline-article pairs for the training and two subsets for development and test with respective sizes of 214,348 and 201,133 pairs. From the development and test, we randomly take 10,000 pairs from each subset to use it with our models.

At this point, Rush et al. (2015) performs a series of operations including lowercasing, replacing words seen less than 5 times with 'UNK'. We skip this step because, for our various models, either we use BPE Sennrich et al. (2015) for out-of-vocabulary words or we the rare words are eliminated while training the Paragraph Vectors.

5.1.2 DUC 2004

One of the most used evaluation datasets for summarization and headline generation is the shared task of the Document Understanding Conference DUC-2004 Over et al. (2007). As Rush et al. (2015) and Nallapati et al. (2016), we train our models using the Gigaword dataset and use DUC-2004 only for evaluation. It contains 500 articles issued by The New York Times and Associated Press paired with four reference summaries written by humans and capped at 75 characters length. Even though the references are summaries of the article and not headlines, they are still used in the literature to evaluate models for the task of headline generation.

5.2 Implementation Details

We ran many experiments using various model architectures based on different hypothesis. In this section, we present the details of the experiments setups and implementations.

5.2.1 Sequence-to-sequence baseline system

As baseline system, we take the exact architecture already presented in section 2.3.3 which is proposed by Sennrich et al. (2017) and published by the authors [2]. We train the model with the articles first sentence as source sequence and the article's headline as target sequence on pairs from the Gigaword dataset. The model is trained end-to-end. We use word embeddings of size 500, a BPE vocabulary size of 10000, ADADELTA Zeiler (2012) as an optimizer with a learn rate of 0.0001 and batch size of 80. We trained the model for 10 epochs. This model results are referred to as **baseline** in the results chapter.

5.2.2 Models With Sentence Vectors

5.2.2.1 Training Paragraph Vectors

To train sentence vectors using the Pragraph Vectors model from Mikolov et al. (2013b) and presented in section 2.2.2, we use the implementation included in the

[2]https://github.com/EdinburghNLP/nematus (retrieved on Feb 05, 2018)

Gensim Python library [3]. The implementation requires storing vector representations of all the documents used to train it. To economize space and RAM usage, we do not train the model on all the 46,153,425 sentences that we have in our preprocessed Gigaword data set. Instead, a random sentence from each article of the 4,167,425 present in our preprocessed set is chosen, lowercased, tokenized and fed to the Paragraph Vector model. Every token seen less than 10 times is ignored. This retains a vocabulary of 101,320 unique tokens, 19% of the original 516,719 unique tokens present in the training input. It resulted in dropping only 867,533 tokens from the 122,327,376 total tokens in the input, which is 0.7%. The model is then trained for 60 epochs to generate sentence vectors of size 500. The trained model is used to feed various sequence-to-sequence models in our experiments.

5.2.2.2 Concatenated Sequence Implementation

We trained two models using the architecture presented in section 4.1. The first, we denote it **sent2vec+word2vec** in our results chapter, it is fed using word embedding issued from the already trained Paragraph Vector. And the second, we call it **sent2vec+wemb**. It uses the word embeddings from the trained sequence-to-sequence baseline system. All of the training specific parameters are the same as the baseline system.

5.2.2.3 Conditional Dual-Attention Implementation

To evaluate the proposed Conditional Dual-Attention proposed architecture, we trained 3 models each with one of the variants of the COMBINE function presented in the section 4.2.5.2 with the sequence of the sentences' vectors and the sequence of the first sentence words as input sequences. Our models' variant names are as follow: **dual_att_mean** using Mean, **dual_att_concat** using Concatenate and **dual_att_mlp** using MLP as defined in the equation 4.1.

[3]https://github.com/RaRe-Technologies/gensim (retrieved on Feb 05, 2018)

Chapter 6

Evaluation & Results

In this chapter, we evaluate our new models through quantitative and qualitative analysis. Our models' performance is compared to the state-of-the-art of abstractive headline generation using an automatic evaluation metric on both the DUC 2004 task and an evaluation set from the Gigaword dataset. We also present the outcomes of a survey we carried out to evaluate the readability and relevance of the produced summaries by human participants.

6.1 Quantitative Analysis

6.1.1 Automatic Evaluation Metric

Summaries quality evaluation and comparison is not an easy problem to approach even by humans. For any input article, there is no best summary, and it is also common to find two summaries with a lot of differences but succeeding at conveying the main information present in the original text and at fulfilling all the expectations of a summary. This difficulty is valid too when it is the case of tools of automatic evaluation. In this thesis, we use the ROUGE metric to automatically evaluate the performance of the studied models and compare them to other reported performances in the literature. We choose this metric because it is still the standard metric used to evaluate summaries qualities since its introduction by Lin (2004).

ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation, and it computes for a generated summary a score based on how well the summary overlaps with a set of golden truth summary references that are typically human generated. Lin (2004) introduced four variants of ROUGE measures. Three of them were adopted as the official evaluation metrics in the DUC 2004 summarization tasks. Since then, these metrics become the reference to evaluate automatic summarization systems and still used till the time of writing these lines, even though advances related to this task and recent work meet limits of this metric Paulus et al. (2017).

In our reporting, we use the ROUGE-1, ROUGE-2 and ROUGE-L measures. The first two are computed based on respectively unigrams and bigrams overlaps and are special cases of the more general ROUGE-N measure defined for n-grams. ROUGE-L is based on the *longest common substring* LCS and has the advantage of computing in-sequence matching instead of limiting the computations to the consecutive matches within n-grams. For each measure there is a *recall* R score and a *precision* P score that are then more balanced using the F_1 score defined by:

$$F_1 = 2 \times \frac{P \times R}{P + R}$$

We suppose given a set of reference summaries RS and a system generated summary G. We compute the ROUGE-N measure using the following more general formula:

$$\begin{aligned}
\textit{ROUGE-N}_{recall} &= \frac{\sum_{S \in RS} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in RS} \sum_{gram_n \in S} Count(gram_n)} \\
\textit{ROUGE-N}_{precision} &= \frac{\sum_{S \in RS} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{gram_n \in G} Count(gram_n)}
\end{aligned}$$

where $Count_{match}(gram_n)$ computes the number of the n-grams present in both the generated summary and the reference, and $Count(gram_n)$ is the number of n-grams in the reference.

For a reference summary S from RS, ROUGE-L score is computed as follows:

$$\begin{aligned}
\textit{ROUGE-L}_{recall} &= \frac{LCS(S,G)}{length(S)} \\
\textit{ROUGE-L}_{precision} &= \frac{LCS(S,G)}{length(G)}
\end{aligned}$$

6.1.2 Results Comparison

6.1.2.1 DUC 2004 Task

The table 6.1 summarizes the results reported in the papers presented in Chapter 3.

- ABS and ABS+ Rush et al. (2015)
- Features_s2s Nallapati et al. (2016)
- RAS-Elman Chopra et al. (2016)

Recall-only ROUGE measures are reported because the DUC2004 task ignores the system outputs after 75 characters and shorter summaries are not awarded any bonus. Thus, many of the systems force the output to be exactly 75 characters. Under these conditions, reporting the precision and F1 measures does not give more information about the system performance. For our models' implementations detailed in section 5.2, we report the results in the table 6.2. We do not use the strategy of forcing a fixed length and give the model more freedom about the output generation.

None of our studied models performs better than the related models on the recall scores. This is maybe related to the fact that we did not optimize any of our models for this metric in contrast for example to the tuned model proposed by

Model	ROUGE-1*	ROUGE-2*	ROUGE-L*
ABS	26.55	7.06	20.12
ABS+	28.18	8.49	23.81
Features_s2s	28.35	**9.46**	**24.59**
RAS-Elman	**28.97**	8.26	24.06

*Recall only

Table 6.1: Various recall-only ROUGE measures on the DUC 2004 task as reported in the papers presented in Chapter 3

Model	ROUGE-1			ROUGE-2			ROUGE-L		
	R	P	F1	R	P	F1	R	P	F1
baseline	21.41	**35.75**	26.26	6.99	**12.54**	8.76	19.48	**32.65**	23.92
sent2vec+word2vec	23.59	29.88	25.95	7.76	9.98	8.57	21.23	26.96	23.37
sent2vec+wemb	**24.87**	31.51	**27.18**	**8.40**	10.98	**9.25**	**22.32**	28.33	**24.40**
dual_att_mean	21.55	32.70	25.52	7.07	11.17	8.47	19.52	29.67	23.13
dual_att_concat	21.26	31.11	24.83	6.92	10.62	8.20	19.04	28.00	22.27
dual_att_mlp	21.13	30.52	24.52	6.65	9.89	7.77	19.12	27.64	22.19

Table 6.2: Our models' results on the DUC 2004 task

Rush et al. (2015). Also, a recall based measure gives more advantage to extractive summaries so it is theoretically possible to have abstractive summaries with relatively better quality but that score worse on the ROUGE measures. Thus, in addition to the automatic evaluation method, we carried out a survey to have more feedback about the actual summaries quality.

Among our models, the **baseline** model is the best performing on the precision measure. This could be explained by the fact that this model generates summaries with the second shortest average length of 55.31 characters per summaries, which is 22% shorter than the human-generated references with an average summary length of 71 characters (see table 6.3). The best performing model on both the recall and the F1 measure is the **sent2vec+wemb** as presented in the table 6.2. The same model produces the second longest summaries with an average length of 59.29 characters.

Longer summaries are generally related to an increase in the recall score and a decrease in the precision score. The **sent2vec+wemb** produces summaries 7.2% longer than **baseline** and scores 16.16% better on recall, 11.86% lower on precision and 3.5% better on F1. However, longer summaries do not automatically lead to better recall and worse precision scores. For instance, the model **dual_att_mean**, despite being the model with the longest average summary, it is only the third-best scoring on recall and takes the second position on precision.

6.1.2.2 Gigaword Test Set

Tables 6.4 and 6.5 report respectively the results of the models presented in the related work chapter and of our models. The new entry compared to the results of the DUC2004 task previously discussed is **FTSum**$_g$ Cao et al. (2017). All the models were trained on the same Gigaword splits that were shared by Rush et al. (2015). However, to evaluate them, only a random subset is taken from the test set

Source	Average summaries length
Human-generated references	71.00
baseline	55.31
sent2vec+word2vec	58.82
sent2vec+wemb	59.29
dual_att_mean	**61.02**
dual_att_concat	*52.81*
dual_att_mlp	57.25

Table 6.3: Average summaries lengths for the DUC2004 task

split. We don't have the exact pairs on which the other models were evaluated, so we randomly sampled 10000 pairs from the test split and evaluated our models on them.

In contrast to the DUC2004 results, all our models perform better than the related models. However, we find the same results pattern with the **baseline** system leading the precision scores and the **sent2vec+wemb** leading both the recall and F1 scores. Our models' good performance on the Gigaword test with 17.31% improvement over the best model from the related works on the ROUGE-1 F1 measure and its relatively poor performance on the DUC2004 tasks are signs that maybe our models are overfitting on the Gigaword dataset and failing to generalize to the DUC2004 task.

As noticed with the DUC2004 task, all our dual-attention architectures are underperforming our other models relatively to the ROUGE metrics. This confirms the need for a more qualitative analysis of our models to better judge them.

Model	ROUGE-1*	ROUGE-2*	ROUGE-L*
ABS	29.55	11.32	26.42
ABS+	29.76	11.88	26.96
Features_s2s	32.67	15.59	30.64
RAS-Elman	33.78	15.97	31.15
FTSum$_g$	**37.27**	**17.65**	**34.24**

*F1 Score

Table 6.4: Various F1 ROUGE measures on a random subset from the Gigaword test set as reported in the papers presented in Chapter 3

Model	ROUGE-1			ROUGE-2			ROUGE-L		
	R	P	F1	R	P	F1	R	P	F1
baseline	38.99	**47.29**	41.77	19.41	**23.72**	20.80	36.76	**44.58**	39.38
sent2vec+word2vec	43.35	43.62	42.55	21.60	21.81	21.22	40.51	40.82	39.80
sent2vec+wemb	**45.12**	44.39	**43.72**	**22.93**	22.51	**22.16**	**42.07**	41.43	**40.80**
dual_att_mean	39.70	45.31	41.37	19.60	22.49	20.43	37.31	42.60	38.89
dual_att_concat	40.11	45.01	41.49	19.73	22.15	20.38	37.68	42.31	38.99
dual_att_mlp	40.05	45.07	41.46	19.73	22.12	20.37	37.66	42.40	39.00

Table 6.5: Our models' results on a random subset from the Gigaword test set

6.1.3 Sentence Vectors Influence On The Generated Summaries

Among the natural questions that could be asked given the automatic evaluation results presented in the previous subsections is: do the used *Sentence Vectors* have any positive influence on the generated output? Given the fact that none of the dual-attention models performed better than the baseline system, is the model learning to simply ignore any given *Sentence Vectors*, so it converges to the already better-performing baseline system? To answer the previous question, we made a series of experiments on the **dual_att_concat** trained model with always the original article's first sentence as input and varying the used sentence vectors. The following table reports the obtained results:

Experiment	Original ROUGE-1 F1	New ROUGE-1 F1	Δ
Random vectors	25.19	12.25	-51.37%
The first sentence vector duplicated many times	25.19	20.98	-16.71%
Using all null vectors	25.19	21.28	-15.52%

Table 6.6: Sentence vectors influence on the **dual_att_concat** trained model performance

Feeding the **dual_att_concat** trained model with the articles first sentence and sentence vectors filled with random values decreases the ROUGE-1 F1 measure by **51.37%**. This considerable decrease of the system performance proves that the dual attention architecture is not learning to *ignore* the sentence vectors. In the second experiment, only the first sentence vector is duplicated as many as the number of the articles sentences and is fed to the system in addition to the first sentence separate words. After this experiment, the ROUGE-1 F1 measure decreased by **16.71%**. We can deduce from that, that feeding the system with vectors adequately representing every sentence in the article leads to better results. The last experiment consists of feeding the system with the words of the articles first sentences and null vectors. The ROUGE-1 F1 measure decreased only by 15.52%. This result balances the idea that we may have after the results of the first experiment that the sentence vectors are as important as the first sentence word for our **dual_att_concat** architecture. Here we see that when the sentence vectors are null, and so do not give any misleading information compared to the random vectors, the system performance is less affected and the article's first sentence remains the biggest contributor to the whole system performance.

6.1.4 Percentage Of Copied Words

One of this thesis goals is to build abstractive summarization neural networks that can get information that may be present anywhere in the article and not limit it to the article's first sentence content. One way to evaluate this aspect is by counting the percentage of the words in the generated summary that where copied from the article sentences except the first one. Even though this evaluation approach makes more sense on an extractive summarization task rather than an abstractive one like we have here, its results are still able to tell us how far are the new models from reaching the thesis goal.

Based on the results reported in the table 6.7, we see that the models based on the dual-attention architecture not only tend to have a more abstractive behavior by having fewer words copied directly from the input, but also, tend to rely more on information coming from other article's sentences in addition to the first one, on both the Gigaword and the DUC2004 evaluation sets. The summaries generated by **dual_att_concat** scores the best on the DUC2004 set with 11.75% of its summaries coming from the article's sentences but not the first sentence, and the **dual_att_mlp** leads the Gigaword evaluation set with a score of 9.75%. These are respectively **54.61%** and **49.31%** improvements compared to the **baseline** system, which despite not having access to the whole articles, manages through its abstractive character to reproduce 7.60% and 6.53% of unseen words in the article's first sentence on respectively the DUC2004 and the Gigaword evaluation sets.

Model	DUC 2004			Gigaword		
	% ext. 1st sent.	% ext. article	Δ % art. - % 1st sent.	% ext. 1st sent.	% ext. article	Δ % art. - % 1st sent.
baseline	69.56	*77.16*	*7.60*	80.48	87.01	6.53
sent2vec+word2vec	70.01	81.25	11.24	75.89	84.00	8.10
sent2vec+wemb	**75.98**	**84.35**	8.37	**81.76**	**88.05**	*6.29*
dual_att_mean	69.28	80.00	10.72	74.56	83.51	8.95
dual_att_concat	*67.62*	79.36	**11.75**	74.67	*83.43*	8.75
dual_att_mlp	69.49	80.21	10.72	*73.71*	83.46	**9.75**

Table 6.7: Average percentage of copied words from the article's first sentence and the whole article

6.1.5 Improving Recall Score

When observing the results of our models on the DUC2004 task reported in 6.2, we notice that for all our models, the precision scores are clearly higher than the recall measures. For instance, the ROUGE-1 precision measure of the **dual_att_concat** is 46.33% higher than the recall measure. To try to improve the relatively low recall scores, we carried three experiments on the model **dual_att_concat** by altering the way the summaries are generated after the model predicts the probabilities of all the words in the vocabulary.

Recall measures are based on dividing the number of intersection between the system output and the reference by the length of the reference as described earlier in this chapter. We noticed from the results given in Rush et al. (2015) that taking a prefix of length 75 characters of the articles' first sentence already scores a better recall than our model **dual_att_concat**. As a first experiment, we decided to increase the probability of every word present in the input article's first sentence by 10% just before running every iteration of the beam search. The results are reported in the table 6.8. As we succeeded in improving the recall score after this experiment by 0.74%, surprisingly, the precision slightly improved too. This led to a slight improvement of the F1 score by 0.45%.

Motivated by the results of the first experiments, we tried increasing the words present in the first sentence probabilities by 100% expecting to see an even better

Experiment	R-1 Recall (20.90)		R-1 Precision (30.10)		R-1 F1 (24.28)	
	New	Δ	New	Δ	New	Δ
Increase first sentence words probs. by 10%	21.06	+0.74%	30.17	+0.24%	24.39	+0.45%
Increase first sentence words probs. by 100%	21.02	+0.56%	29.68	-1.39%	24.18	-0.39%
Force minimum 75-character headlines	24.62	+15.10%	26.30	-14.42%	25.04	+3.07%

Table 6.8: Improving the **dual_att_concat** trained model recall score
Original values are reported between (). Δ is the relative improvement compared to the original score

improvement on the recall score. As presented in the table 6.8, for this time both the precision and the F1 measures decreased by respectively 1.39% and 0.39% and even the recall improvement was lower than the first experiment. We concluded from these two experiments that this way of manipulating the words probabilities is harmful to the model performance because we are directly altering values that after all the model was trained to output them differently.

Given the results in table 6.3 about the average length of the outputted summary and a possible correlation between better recall scores in 6.2 and the average summary length, we carried the following third experiment. We forced the **dual_att_concat** model to output summaries with a minimum length of 75 characters. As reported in the table 6.8, the ROUGE-1 recall clearly improved by 15.10%. However, the summary readability suffered a lot as we can see in the table 6.9. By forcing the system not to end the sentence before the 75 characters mark, the model was obliged to keep outputting unnecessary words till reaching the target. This puts in question, one more time, the limitations of the recall measure in evaluating summaries. As expected, forcing the system to output more words reduced the precision score by 14.41%. However, the F1 measure still improved by 3.07%. This proves that the best scoring model on recall or the F1 measure is not always the most readable and relevant summarizer.

6.2 Qualitative Analysis

6.2.1 Survey Setup

To perform a more qualitative analysis of our models' output and surpass some of the limits imposed by the automatic evaluation metrics, we asked humans to evaluate the relevancy and readability of the generated summaries. The survey's primary goal is to collect human feedback to compare the output of 3 different models including the baseline. 25 random samples were extracted from the 500 evaluation articles in the DUC 2004 dataset and 25 more from the 10000 Gigaword evaluation subset. For each sample, the participant is presented with the original article and 4 different potential headlines in a random order. The presented headlines are:

- The **reference** headline

Article's first 5 sentences	Cambodian leader Hun Sen on Friday rejected opposition parties' demands for talks outside the country, accusing them of trying to "internationalize" the political crisis. Government and opposition parties have asked King Norodom Sihanouk to host a summit meeting after a series of post-election negotiations between the two opposition groups and Hun Sen's party to form a new government failed. Opposition leaders Prince Norodom Ranariddh and Sam Rainsy, citing Hun Sen's threats to arrest opposition figures after two alleged attempts on his life, said they could not negotiate freely in Cambodia and called for talks at Sihanouk's residence in Beijing. Hun Sen, however, rejected that. "I would like to make it clear that all meetings related to Cambodian affairs must be conducted in the Kingdom of Cambodia" Hun Sen told reporters after a Cabinet meeting on Friday.
Reference	Cambodian government rejects opposition's call for talks abroad
dual_att_concat output forced to at least 75 chars	Cambodia's Hun Sen rejects opposition demands for 'internationalizing' crisis
dual_att_concat original output	Cambodia's Hun Sen rejects opposition demands for talks

Table 6.9: An example of a headaline generated by the model **dual_att_concat** for an article from the DUC2004 task

- The headline generated by the **basline** model
- The headline generated by the **sent2vec+wemb** model
- The headline generated by the **dual_att_concat** model

Each of the 10 participants is asked to give each potential headline a relevancy and readability score ranging from 1 to 10 with 1 being the worst and 10 being the best evaluation. Using the collected data with this survey protocol, we can give each model an average relevancy and readability score and better analyze our models' performance compared to the human reference.

6.2.2 Survey Results

The results on the samples extracted from the DUC 2004 evaluation set are reported in the table 6.10.

Unsurprisingly, the human reference scores are really close to 10 out of 10 for both readability and relevance. This could be used as an indicator that validates the quality of the participations in the survey since the participants did not know that there is a headline generated by humans among the candidate headlines.

Concerning our 3 models, **basline**, **sent2vec+wemb** and **dual_att_concat**, the human evaluation results go in the same direction as the automatic evaluation results using the different ROUGE measures. The **baseline** model is performing clearly better than our other models from a readability point of view on the DUC 2004 task. However, the **sent2vec+wemb** is the best on relevance with only a slight advantage in front of the **baseline**. We can say that even though our models succeeded in the thesis goal to include information from the article that is beyond the first sentences content, they could not beat the **baseline** system on the DUC 2004 task.

Model	Readability	Relevance
Human reference	9.84	9.64
baseline	**8.8**	7.56
sent2vec+wemb	7.76	**7.68**
dual_att_concat	7.64	6.48

Table 6.10: Human evaluation on summaries from the DUC2004 task

The human evaluation on the Gigaword test set brought into light more information thanks the results summarized in the table 6.11. The first thing to notice is the relatively low relevance score for the human reference of 8.52. This score could be justified by the fact that, in contrast with the high scoring human references from the DUC 2004 task where humans were asked to summarize the articles, what we use as a reference in the Gigaword test are the real headlines that were chosen by the journalists. Sometimes, these headlines do not serve to best summarize the article but try for example to catch the reader's attention or trigger his curiosity to read the article.

For the first time in our different evaluation series, the **baseline** system is completely behind our other models. **dual_att_concat** achieves the best readability and **sent2vec+wemb** the best relevance score. The same scores are also clearly

Model	Readability	Relevance
Human reference	9.08	8.52
baseline	8.64	8.12
sent2vec+wemb	8.8	**8.36**
dual_att_concat	**8.84**	7.8

Table 6.11: Human evaluation on summaries from the Gigaword test set

better than the models' performances on the DUC 2004 task. For instance, the **dual_att_concat** readability score is 15.71% better. These results confirm our models' good performances on the Gigaword dataset previously reported by the different ROUGE measures.

Chapter 7

Conclusion & Future Work

The ever-increasing volume of content shared every day keeps pushing content creators to create concise and short content more than ever before so their message can make it to their audiences. Automatic text summarization is among the best candidates to serve this need. Manually curating content and generating summaries is costly and automatizing this task efficiently would be more than welcome for a variety of use cases. These are some of the motivations behind the active research in this field. The blocking point is that existing systems are still a bit far from hitting the markets with reliable commercial solutions. Something that other transduction tasks like machine translation achieved many years ago. The big success of the neural machine translation approaches during the last years motivated a new trend of research works trying to bring this success to the world of automatic text summarization. And this thesis is one of them.

Abstractive summarization approaches using neural networks started by focusing on ”simpler” tasks like headline generation instead of long summaries generation due to the complexity of the task and various current limitations such as massive memory requirements, extended training time or even architectures inefficiency with long output sequences. To simplify the task even more and make some progress in a field that did not know as much relative success as the field of the extractive summarization, most of the proposed models in the recent years limit their input the articles' first sentence. Thus, ignoring any other information in the article outside the first sentence. Experimenting and trying to study how to lift this limitation was one of the primary goals of this work.

In this thesis, we started by carrying out background research about the requirements of achieving our goal. We went through the background of the automatic summarization for a better overview of the task. Always in chapter 2, basic approaches for words representation of transduction tasks through the successful methods of neural machine translation were presented. After having a better understanding of the current state-of-the-art and the current challenges in chapter 3, chapter 4 was the showcase of the various models that were designed for this thesis. We tried to adapt successful ideas and merge them to develop better neural summarizers, like combining Paragraph Vectors with a new dual-attention architecture.

In chapter 5, we provided an overview of how we implemented the tested ideas in this thesis, before diving into an in-depth qualitative and quantitative analysis of these models efficiency on our target task of abstractive headline generation.

Despite having results lower than the state-of-the-art and our new models barely outperformed our baseline, our systems successfully achieved the goal for which they were designed. The new models can generate headlines including pieces of information beyond the articles' first sentence. In front of the limitations of the standard methods used the evaluate summarizers which are the different ROUGE measures, we carried out a manual human evaluation of some of our models. Among the outcomes of this study, on the Gigaword test set, our new models' headlines achieve a highly comparable readability to the headlines initially proposed by the journalists and score an extremely close relevance score. More than that, in some cases, participants rated an automatically generated headline better than the original one for its relevance.

Given the discussed results, there still a lot of improvements and future work that could be a continuity of this thesis work. Here we used only one algorithm of sentence vector representation which is based on the Paragraph Vector. We think that it would be interesting to test the same architectures with other existing sentence representation approaches. We believe that better results are achievable if we find a way to improve the quality of the used sentences representations.

A second possible research direction is to try to adapt the current conditional dual-attention architecture to support more than two input sequences and thus feed the article sentences separately to the model. For instance, limit the number of the input sequences to the article's five sentences because of the large memory and computation requirements of a such model.

Finally, we think that the currently proposed models deserve more experiments to better assess their efficiency. Here we trained all our model on the Gigaword dataset. The human evaluation showed that the reference headlines in the dataset are not always the best possible short summaries of their corresponding articles. With more summarization oriented datasets being released, training the same architectures with various sentence representation algorithms on multiple datasets with a human evaluation may lead to better models.

Bibliography

Baayen, R. H. (2001). *Word frequency distributions*, volume 18. Springer Science & Business Media. 5

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*. 11, 12, 13, 15, 17, 18, 20, 22, 25, 29, 31, 32

Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., and Etzioni, O. (2007). Open information extraction from the web. In *IJCAI*, volume 7, pages 2670–2676. 25

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155. 7

Cao, Z., Wei, F., Li, W., and Li, S. (2017). Faithful to the original: Fact aware neural abstractive summarization. *arXiv preprint arXiv:1711.04434*. ix, 25, 26, 29, 39

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*. 11, 12, 16

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*. 13, 14, 17, 25

Chopra, S., Auli, M., and Rush, A. M. (2016). Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98. 21, 22, 38

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM. 7

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211. 22

Gage, P. (1994). A new algorithm for data compression. *The C Users Journal*, 12(2):23–38. 16

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. *arXiv preprint arXiv:1302.4389*. 18

Graff, D., Kong, J., Chen, K., and Maeda, K. (2003). English gigaword. *Linguistic Data Consortium, Philadelphia*, 4:1. 34

Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162. 6

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780. 11, 13, 22

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196. ix, 2, 6, 7, 8, 9, 10

Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*. 37

Luhn, H. P. (1958). The automatic creation of literature abstracts. *IBM Journal of research and development*, 2(2):159–165. 1, 3, 4

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*. 8, 23

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119. 8, 10, 29, 35

Mnih, A. and Hinton, G. E. (2009). A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088. 8

Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. 8

Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B., et al. (2016). Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*. ix, 22, 23, 24, 35, 38

Napoles, C., Gormley, M., and Van Durme, B. (2012). Annotated gigaword. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*, pages 95–100. Association for Computational Linguistics. 34

Nenkova, A., Vanderwende, L., and McKeown, K. (2006). A compositional context sensitive multi-document summarizer: exploring the factors that influence summarization. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 573–580. ACM. 5

Over, P., Dang, H., and Harman, D. (2007). Duc in context. *Information Processing & Management*, 43(6):1506–1520. 35

Paulus, R., Xiong, C., and Socher, R. (2017). A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*. 37

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1. 8

Rush, A. M., Chopra, S., and Weston, J. (2015). A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*. ix, 20, 21, 34, 35, 38, 39, 42

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681. 13

Sennrich, R., Firat, O., Cho, K., Birch, A., Haddow, B., Hitschler, J., Junczys-Dowmunt, M., Läubli, S., Barone, A. V. M., Mokry, J., et al. (2017). Nematus: a toolkit for neural machine translation. *arXiv preprint arXiv:1703.04357*. 2, 15, 18, 29, 31, 32, 35

Sennrich, R., Haddow, B., and Birch, A. (2015). Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*. 16, 23, 35

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112. ix, 11, 12, 15

Turian, J., Ratinov, L., and Bengio, Y. (2010). Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics. 8

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*. 35