UNIVERSITA' DEGLI STUDI DI TORINO

SCUOLA DI SCIENZE DELLA NATURA

Ph.D. Program in Computer Science

XXXII cycle

Neural Network Models for Content-Aware Text Summarization

Advisor:
Prof. Luigi di Caro

Ph.D. Candidate:
Giovanni Siragusa

Contents

Abstract

The large adoption of Neural Networks in Natural Language Processing has brought to summit several tasks, such as Text Summarization. In this one, Encoder-Decoder models are used to generate the summary starting from a document (or a set of them). The Encoder reads the document(s) and creates a fixed vector representation, while the Decoder uses such vector to generate the summary.

The problem of the Encoder-Decoder Models for Text Summarization are several. First, they are not able to remember the previous generated words, and for such reason they generate in output repetitions of the same word (or phrase excerpt). Second, Encoder-Decoder Models are not able to generate words that are outside of their vocabulary, which is important in recall-oriented tasks as Text Summarization. Finally, they are not able to discern sentences (or words) that are relevant for the summary from those that are not. Many researchers investigated this problem, providing different solutions.

In this thesis, I first present a wide review of state-of-the-art methods to correct those issues, that regard: *coverage methods* to remove repetitions in the output summary; *pointer network models* to create a probability distribution over the the document words, from which one can be selected and copied in output; and *document content methods* to analyze the document content and select the relevant sentences (or words). Then, following the idea of these latter methods, I propose an Encoder-Decoder model that calculates a two-level attention: a sentence-level attention and a word-level attention. The former attention finds the sentences that are relevant for the summary, while the latter one, which is multiplied with the sentence-level attention, captures the relevant words. In this way, words coming from informative sentences are more likely to be used by the model, either to be copied in output via the pointer network or to create the document context. The proposed model obtained notable results in comparison with state-of-the art ones, showing high abstractive power in generating summaries.

Chapter 1

Introduction

The idea of automatically condensing long documents in a short form (i.e., Text Summarization), allowing for efficient and improved reading and understanding of their content, attracted particular attention in recent years. The main work on this topic was conducted by Luhn (1958), proposing a selection of relevant sentences of an input document mainly based on words frequencies. Such work paved the way for more complex systems based on classifiers [Kupiec et al., 1995; Saggion and Poibeau, 2013], graphs [Mihalcea, 2004; Litvak and Last, 2008; Dohare et al., 2018], Latent Semantic Indexing [Bhandari et al., 2008; Murray et al., 2005; Gong and Liu, 2001; Steinberger et al., 2005], clustering [Nomoto and Matsumoto, 2001] and probabilistic models [Gross et al., 2014]. Furthermore, with the growth of interest in automatic summarization and the possibility of collecting large sets of <*document*, *summary*> pairs, researchers moved from the manual evaluation of summarization systems towards automatic and semi-automatic approaches, defining a set of metrics [Lin, 2004; Nenkova and Passonneau, 2004] aiming at capturing the similarity between human-made and automatically-generated summaries.

At the same time, Neural Networks (NNs, from now on) started to dominate the field of Natural Language Processing thanks on one hand to their capability of exploit large datasets and the high accessible computational power nowadays; and on the other hand, to the ability of producing optimal results in tagging documents, or generating summaries and translations that resemble the human ones. In this context, NNs based on the *Encoder-Decoder* framework [Sutskever et al., 2014; Cho et al., 2014] have been firstly proposed, such as [Rush et al., 2015; Chopra et al., 2016; Nallapati et al., 2016; See et al., 2017; Tan et al., 2017; Paulus et al., 2018; Li et al., 2018a,b]. Generally speaking, an encoder reads the document in input and produces a compressed representation, which is then used by the decoder to generate the summary. The drawbacks of these systems concern:

- *Out-Of-Vocabulary words*: the impossibility to generate words that are outside

the vocabulary. Such problem arises because NN models require a fixed-size vocabulary;

- *Repetition of words*: when the NN generates a word in output, it does not capture which document words and sentences attended to produce it. This absence of a memory leads the network to focus on the same document words (or sentences), producing repetitions in the output;

- *Content indifferent:* when the NN reads the document, it may decide to focus on a piece of information, creating a summary that is correct but not related to the document content. For instance, if a document talks about a "new movie produced by the actor of Harry Potter", the network may focus on "the actor of Harry Potter", generating a summary on this argument instead of the "new movie". This is due by the incapability of the model to discern the relevant and informative sentences (or words) for the summary from those that are secondary for the context, treating them equally.

In this thesis, my contribution is twofold:

An up-to-date and structured review of TS techniques to contrast these drawbacks. These regard:

- Coverage methods [Mi et al., 2016; Tu et al., 2016; Sankaran et al., 2016; Chen et al., 2016b; See et al., 2017] create an artificial memory to eliminate the repetitions by keeping into account the previous generated words. Their application showed a substantial improvement in the performance of the Encoder-Decoder models;
- Pointer networks [Vinyals et al., 2015] are special NNs that create a probability distribution over the the document words, from which one can be selected and copied in output. The advantage of these networks is that they can copy rare words and those that are outside the network vocabulary (which is fixed and defined a priori). Some researchers applied them only to copy the words [Nallapati et al., 2016; Miao and Blunsom, 2016; Merity et al., 2017], while others used them to create a mixture model [Gu et al., 2016; Gulcehre et al., 2016; See et al., 2017].
- Content-based methods aim to select relevant sentences (or words) and to improve the generation of the summary. More in detail, they are divided in two research directions: methods based on templates, and methods that exploit the salient content of the document. The former one uses human-written templates containing particular slots that have to be filled with document words or phrase excerpts [Zhou and Hovy, 2004; Chen and Bansal, 2018; Cao et al., 2018; Wang et al., 2019]. The latter one, instead, contains

different approaches: mixing extractive and abstractive techniques [Hsu et al., 2018]; hierarchical encoders to capture the relevance of each document sentence [Tan et al., 2017; Li et al., 2018a]; or word-level masks to filter out irrelevant words [Zhou et al., 2017; Gehrmann et al., 2018].

A novel Neural Network-based model (relaying on the research trend of exploiting document content) to capture salient sentences, distinguishing those that are relevant for the summary from those that are not. More in detail:

- I defined a Sequence-to-Sequence model formed by a hierarchical encoder followed by a Recurrent Neural Network decoder. The encoder first reads all the words of a sentence to generate the word-level representations; then, these representations are used to create the sentence-level one;
- I defined a two-level attention mechanism to capture the informativeness of words and sentences. In detail, the sentence-level attention is based on a Pagerank algorithm, which is constructed over a graph-based NN. As in [Tan et al., 2017], the sentences are represented as vertexes, while the edges capture the connections. I weighted each edge through the cosine similarity function in order to highlight the redundancy between sentences (i.e., sentences that share a piece of information). The word-level attention is instead based on Bahdanau et al. (2014)'s one. The peculiarity is that I re-scored the word-level attention multiplying (and normalizing) it with the sentence-level one. In this way, words coming from salient sentences are more likely to be copied or focused by the model;
- I proposed an inference-time penalizing function, based on coverage methods and reinforcement learning, to remove repetitions in output and improve the summary.

The evaluation reported that the proposed model is able to generate very good summaries. Furthermore, I also discovered that the model tends to copy phrase excerpts only when it found it is necessary, preferring the use of synonyms and periphrases.

The thesis is composed of the following chapters:

Chapter 2: *The Summarization Task*. It describes the Summarization task, reporting how the task is subdivided according to the input, the output and the purpose. It also cover supervised and unsupervised methods, evaluation and existing datasets;

Chapter 3: *Background on Neural Networks*. Since the thesis is focused on Neural Networks, this chapter provides a background on this argument. It starts from the first perceptron and arrives to Recurrent Neural Networks. It also describes Word

Embedding, a necessary layer for Neural Networks oriented to Natural Language Processing (NLP);

Chapter 4: *New Frontiers on Neural Text Summarization*. This chapter contains the first contribution of my thesis. It describes the new research trends: *coverage methods*, *pointer networks* and *document content methods*.

Chapter 5: *A Sentence-level Attention for Content-Aware Summarization*. This chapter contains the second contribution of my thesis. It describes the proposed model, reporting the different sentence-level attention methods and their extensive evaluation.

Chapter 6 concludes the thesis.

The interaction between the chapters of this thesis is depicted in Figure 1.1. The chapters "*The Summarization Task*" and "*Background on Neural Networks*" form the background knowledge necessary to understand the recent research works on Neural Network-based Text Summarization, described in "*New Frontiers on Neural Text Summarization*". The former two chapters can be read independently, but I suggest to follow the order in the thesis. The latter chapter, instead, provides the fundamental onto which my model is created.

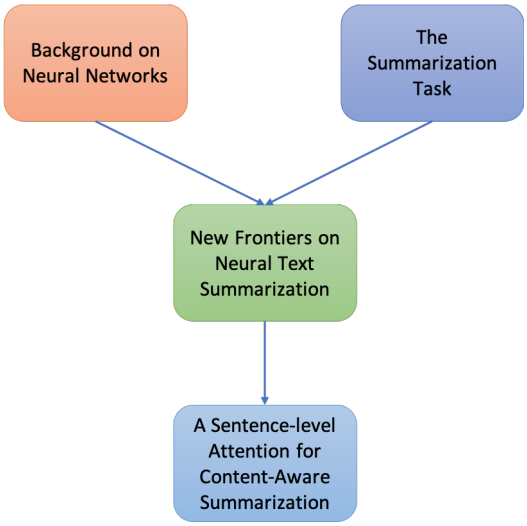


Figure 1.1: The figure reports the interaction between the chapters.

Chapter 2

The Summarization Task

Nowadays, people are surrounded by large amounts of short-to-long textual data that come under the form of documents, emails, news, blogs and so forth. Thus, condensing such data volume while preserving all relevant information, i.e. *Text Summarization* (or TS, from now on), is becoming an extremely important task in several applications.

In simple words, TS means to compress a text document in a shorter form, usually through an input compression rate, maintaining the overall expressed semantics.

In TS, systems are required to understand the text and re-organize the information to generate semantically and syntactically coherent short summaries. For this reason, crucial research questions in TS are the following:

1. How is it possible extract the most important content from a document?

2. How is it possible to compress the extracted content to generate the summary?

There exist several types of TS, that can be distinguished on the basis of the input, the output and the purpose (depicted in Figure 2.1). Based on the input, there exist *single document* and *multi-document* TS. In the former, the input is a single document to summarize, while in the latter there are several documents that can either come from a single source of information or from multiple ones. This last case is usually more complex due to the presence of redundant and segmented pieces of information.

Related to the output, it is possible to opt for *extractive* or *abstractive* TS. In extractive methods, the summary is composed by selecting original sentences from the input document(s). More in detail, sentences are extracted from the input document(s) and sorted according to a score which takes into account both the relevance of the sentence to the summary and its redundancy. Finally, the top-k scored sentences define the summary. In abstractive methods, instead, summaries are built through a generative use of words that can be both inside and outside the document(s) vocabulary.

The TS task is also divided according to its purpose: *generic*, *domain-specific* and *query-based*. While in generic TS does not make any assumption on the domain or

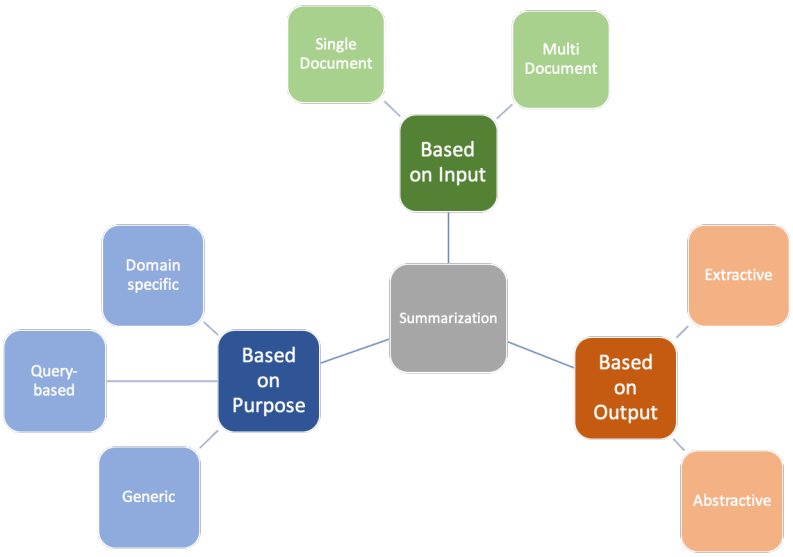


Figure 2.1: The picture shows the different types of Textual Summarization.

content of the documents, thus treating all inputs as homogeneous, in domain-specific TS the domain knowledge is employed to produce more specific summaries (e.g., TS of bio-medical documents). Finally, in query-based TS, the final goal is represented by the creation of summaries answering the input query. In other terms, documents and sentences are selected according to the query.

Another classification of TS is *informative* vs *descriptive*. In informative TS, the summary is created to inform the reader, while in the descriptive one it reports the information under an objective view. Furthermore, TS can be also *multilingual* if either the input document(s) or the output summary belong to different languages.

Finally, there are three aspects to keep in consideration when a system for TS is constructed:

Informativeness: the generated summary should contain all salient information present in the input document(s);

Non-redundancy: the generated summary should not contain multiple sentences that express the same semantic information, and in general, any type of redundancy;

Readability: since that the summary will be read by an human-being, it should be coherent and syntactically well formed.

2.1 Seminal work of Luhn

Luhn (1958) has been the first researcher working on TS, opening the research field to

the current works. He theorized that summaries can be formed using selected sentences of the input document, containing the most relevant words. According to his idea, a word is relevant if it is frequent. Thus, he counted the frequency of the words to find the relevant ones; however, he found that not all frequent words are relevant. Words such as subjects, determiners, pronouns, and so on are generally too common in texts to be relevant. From these first results, he then employed two word lists: a stopword list composed of frequent words that are not relevant, and one containing the relevant words.

He then defined a pipeline to obtain the summary: first, a document is divided into sentences; then, a pre-processing phase to remove punctuation characters and stopwords is applied to those sentences. Finally, a score for each sentence is computed, counting the number of relevant words in a window (of length 7) and dividing the result by the window length. He also posed a constraint on the window: relevant words must be separated by at most 2 irrelevant words. Once sentences are sorted according to their score, the top-k ones are picked to generate the summary. An example of Luhn's pipeline is depicted on Figure 2.2.

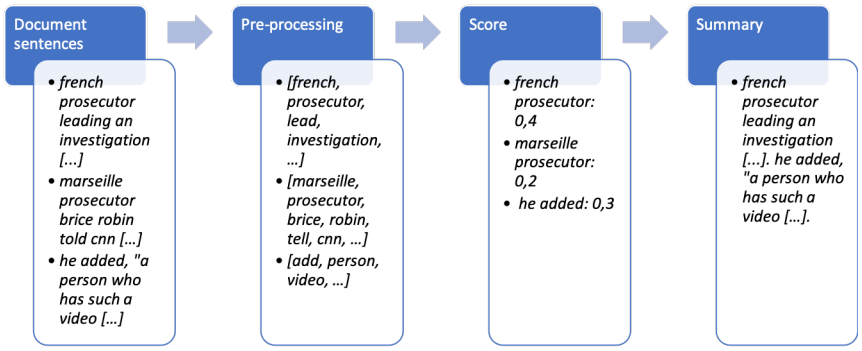


Figure 2.2: The figure shows Luhn's pipeline of a CNN (https://www.cnn.com) document excerpt.

2.2 Supervised Summarization

TS can be also divided into *supervised* and *unsupervised*, depending on the algorithms that are used to generate the summaries. To the best of my knowledge, supervised methods have better performances than unsupervised ones because each component is strictly tied (or jointly trained) to solve this specific task.

In this section, I will focus on supervised summarization, since most of the recent research works are based on Neural Networks with supervised training. First, I will describe two most used Machine Learning classifiers: Support Vector Machines and

Naive Bayes, also reporting how the training set for those classifiers can be defined. Then, I describe the textual features proposed in literature to train those models. Finally, I illustrate Neural Network-based models for Textual Summarization, reporting the first proposed ones. A deep description of Neural Networks and their application to Text Summarization can be found in Chapter 3 and Chapter 4 respectively.

2.2.1 Classifiers

In the supervised context, researchers initially employed techniques such as Support Vector Machines (SVM) [Cortes and Vapnik, 1995] and Naive Bayes classifiers [Rish et al., 2001] to capture the relevance of a sentence according to specific characteristics (e.g., presence of proper nouns, terms in the document title, etc.) and the desired output.

Support Vector Machine (SVM) tries to find the optimal hyperplane that separates the training data into two disjunctive sets A and B. It is defined as an Integer Linear Programming:

$$\begin{aligned} &min\ ||w|| \\ &\text{subject to :} \\ &\quad w^T x_i - b \geq 1 \ \text{ if } x_i \text{ belongs to A} \\ &\quad w^T x_i - b \leq 1 \ \text{ if } x_i \text{ belongs to B} \end{aligned} \tag{2.1}$$

where x represents the numerical features of i-th training point, w is a set of weights learned by the classifier, and b is constant called *bias*. As we will see in Section 3.2, the SVM has some grade of resemblance to Rosemblatt's perceptron [Rosenblatt, 1958]. Naive Bayes classifiers, instead, are a set of classifiers that use the Bayes theorem with the strong assumption that the features are independent each other. The probability to assign a class C (e.g., A or B) to the i-th training point is calculated as the product between the class prior $p(C)$ and the conditional probability $p(x_i|C)$:

$$\hat{y} = \underset{C}{\operatorname{argmax}}\, p(C) \prod_{i=1}^{n} p(x_i|C) \tag{2.2}$$

where n is the number of training points and $\hat{y}$ is the assigned class.

In some research works, the classifiers are combined with rich sentence characterizations: Daumé III and Marcu (2002) adopted the Rhetorical Structure Theory [Mann and Thompson, 1988] to compress the whole text and generate the summary; Leskovec et al. (2005) used graphs to extract relations from the document sentences and fed them in input to the classifier. These classifiers can be also used in the post-processing phase to re-rank the extracted sentences [Barzilay and Lapata, 2008].

The training of such systems requires data that may come under the form of class-labels or sentences. For instance, in extractive TS, two classes are used to label each

sentence: a sentence is labelled with 1 if it belongs to the summary (with 0 otherwise). Then, the system (the SVM or the Naive Bayes) is trained to select the sentences that will form the summary. The difficult part is to manually tag the training corpus, since someone has to label each sentence in each document. Some researchers [Nallapati et al., 2017; Cheng and Lapata, 2016] have successfully tried automatic techniques. Their idea is to select the document(s) sentences that will form the extractive summary using the human summary and Rouge-2 (see Section 2.4) score. They start from an empty summary, adding sentences if they improve the Rouge-2 score between the extractive summary (composed of the selected sentences) and the human one. At each step, only the sentence that produces the best Rouge-2 score is added to the summary, following a greedy strategy. The process is repeated until no sentence increases the score. Finally, the selected sentences are labelled with 1, while the others are labelled with 0.

2.2.2 Textual Features

In those classifiers, each sentence is represented as a vector of features. For instance, it could be a vector that has as many features as the vocabulary size[1], whereby each feature has value 1 if the word appears in the sentence, or 0 if not. Figure 2.3 visually describes this example.

	w_1	w_2	w_3	w_4	w_5	w_6		w_{20}
s_1	0	1	0	1	1	0	...	0
s_2	0	0	1	0	0	0	...	1
s_3	1	0	0	1	0	1	...	1

Figure 2.3: The figures shows a feature matrix for three sentences (s_1, s_2 and s_3). In this example, the vocabulary contains 20 words. The presence of the word w_3 in the sentence s_2 is defined with the value 1.

In general, features consist of (but they are not limited to) the following ones [Kupiec et al., 1995; Saggion and Poibeau, 2013]:

Term Frequency: One of the most used feature. Each cell contains the frequency of the corresponding term in the sentence or in the document set. However, this feature is not a good one since frequent terms could be not relevant, as pointed out by Luhn.

TF-IDF: the idea of TF-IDF is to promote relevant terms, while demoting common ones. This is made possible multiplying the Term Frequency (TF) by a regularization factor called Inverse Document Frequency (IDF). The IDF of a word w is

[1]The number of words in a vocabulary.

the logarithm of the ratio between the size of the documents set and the number of documents containing the term w. If the term is present in all the documents (e.g., if the term is a determiner), its IDF value is zero.

Clue Words: It is used to see if the sentence contains specific terms, such as those present in a manually-defined list or in the document title. This features could be either binary (e.g., 1 if the sentence contains a clue word, 0 otherwise) or numerical (e.g., the number of clue words in the sentence).

Proper Nouns: It is a binary feature that captures the presence or absence of proper nouns in the sentence. To see whether a sentence contains proper nouns, a search for words starting with a capital letter is generally used, avoiding those at the beginning of the sentence.

For instance, Kupiec et al. (1995) proposed a Naive Bayes classifier that uses features such as sentence length, presence of proper nouns, presence of thematic words, paragraph (if the sentence appears in the start of a paragraph or in the end of a paragraph) and fixed-phrases (e.g., particular unigrams and bigrams such as "this letter" or "results").

2.2.3 Neural Network-based Models

Other research works, instead, developed systems that learn how to generate the summary word-by-word, starting from those produced by humans (i.e., the sentences that compose the abstract). Such approach, called *abstractive* TS, is very common for Neural Network-based Summarization, where models are based on the *encoder-decoder framework*. The encoder reads the text in input and generates a fixed-size vector representation; then, the decoder uses the vector representation to generate the summary word-by-word. Details on this neural network models could be found in Section 3.7. To the best of my knowledge, [Rush et al., 2015; Chopra et al., 2016; Nallapati et al., 2016] conducted the first works on this research field. Rush et al. (2015) used an attention-based encoder followed by a neural language model [Bengio et al., 2003] to read the input text and generate the summary. Chopra et al. (2016) substituted the language model with a RNN-based one. Finally, Nallapati et al. (2016) adopted the Sequence-to-Sequence model [Sutskever et al., 2014] for the summarization task. Those models also use the attention approach proposed by Bahdanau et al. (2014) to select relevant portions of the input text, using them as context to generate the next word in the summary.

2.3 Unsupervised Summarization

After the research work of Luhn (1958), several unsupervised techniques have been proposed to generate a summary from the input document(s). To the best of my knowl-

edge, the most relevant ones are: Graph, Latent Semantic Indexing (LSI), Integer Linear Programming (ILP) and Lexical Chain (LC).

2.3.1 Graph-based models

In graph methods, a graph $G=<V, E>$ is used to represent the document, where the set of vertex V represents each sentence in the document, while the set of edges E describes the connections between those sentences. Such connections are computed using functions that assess the similarity between two sentences (e.g., cosine similarity). The graphs can be also oriented; in this case, edge direction respects the order of appearance of the sentences in the document: if a sentence i precedes the sentence j in the document, the edge $e_{i,j}$ will be an out-going edge for i and an in-going edge for j. An example of graph is depicted in Figure 2.4. Once the graph is constructed, ranking algorithms, such as PageRank [Brin and Page, 1998] or HITS (Hyperlinked Induced Topic Search) [Kleinberg, 1999], are used to assign a score to each vertex. In details, those algorithms computes the score of a sentence on the base of its connections. A popular sentence (i.e., a vertex with many connections) is preferred to be inserted into the summary since it contains all information present in its neighbour vertexes. The scores are then used to construct the summary, ranking the vertices in descending order and picking them until the maximum summary size is reached.

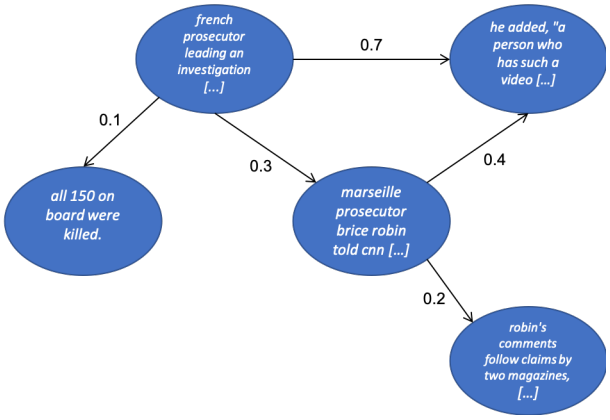


Figure 2.4: The picture shows 5 sentences and their connections. The weight on the edges describes the similarity between the two sentences.

Others, like Litvak and Last (2008), used the same mechanism to extract relevant keywords that are used to create the summary. Dohare et al. (2018), instead, fused a graph approach with Abstract Meaning Representation (AMR) [Banarescu et al., 2013]. In details, they started representing each sentence of the document as a graph; then, they found relevant nodes and key relations through heuristics. They also expanded those

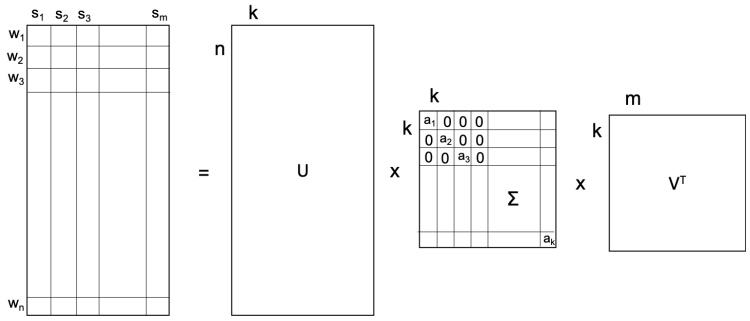


Figure 2.5: The picture shows the singular value decomposition of an $n \times m$ matrix, where n is the number of tokens and m is the number of sentences in the document. Values a_i in matrix $\mathbf{\Sigma}$ are the singular values.

graphs to capture surrounding information via Open Information Extraction [Banko et al., 2007]. Finally, they used the state-of-the-art AMR method to transform the graphs into a summary.

2.3.2 Latent Semantic Indexing-based methods

Latent Semantic Indexing (LSI), and in general Latent Semantic Analysis, are based on matrices. The idea is to construct a matrix where rows represent unique words of the document, and columns represent sentences. Each cell of the matrix can be a binary value (i.e., term presence in the sentence) or a weight (calculated with TF-IDF, for instance). Once the matrix is constructed, the LSI-based method is applied, transforming it in the product of three distinct matrices: $U\Sigma V^T$ (where T represents the transpose of the matrix) - see Figure 2.5.

Then, as proposed by Gong and Liu (2001), the matrix V^T is used to construct the summary since it expresses the importance degree of a sentence respects to the document topics. In detail, for each topic, the most informative sentence is chose using V^T. The process ends when the maximum summary length is reached.

Cagliero et al. (2019) proposed ELSA, a model that combines both LSA-based and Itemset-based summarizers, being able to consider the correlations between multiple terms and to summarize texts via LSA. First, they mined the itemsets, i.e. sets of terms that frequently co-occurs in the same sentence; then, given the concept-by-sentence matrix built using the extracted itemsets, they applied the LSA method to select the relevant sentences for the summary.

2.3.3 Integer Linear Programming-based methods

When I firstly introduced TS task, I said that a summary should be informative, non-redundant and readable. However, many of the previous presented techniques (both supervised and unsupervised) have difficulties to respect one or many of those points. For instance, a LSI-based method could select two sentences that are different at the syntactical level, but similar on the semantic one. To solve those problems, Integer Linear Programming (ILP) models are used; an ILP model is an optimization problem composed of an objective function to maximize (or minimize) and a set of constraints to hold. ILP-based methods treats TS as a *budget maximization coverage problem*, where they select the minimum number of sentences from the input document(s) that maximize a function (i.e., the informativeness of the selected sentences) under certain constraints (e.g., the maximum length of the summary).

In this context, Oliveira et al. (2016) proposed an interesting ILP model. They first extract concepts (e.g., particular keywords) present in a document. Such concepts are then weighted according to their score, which takes into account both the position of the concept in the sentence and how many sentences contain that concept. Then, they maximize the total weight of the selected concepts under the constrains that:

- the length of the selected sentences have not to exceed the maximum summary length;
- if a concept is selected, at least one sentence containing the concept have to be selected;
- if a sentence has some dependencies with other S ones, this sentence can be inserted into the summary if and only if the S sentences have been previously inserted.

They evaluated the model on DUC 2001, DUC 2002 and CNN datasets, obtaining high Rouge scores (for detail about Rouge evaluation, see Section 2.4) with respect to state-of-the-art models.

Another interesting work is the one of Nishikawa et al. (2010). The authors proposed a method to summarize opinions, i.e. the judgement that a person has towards an object or one of its aspects, such as the battery of a phone. Their idea is that the summary could be seen as a direct path from a starting sentence to an ending one. They constructed a graph where each edge connecting a pair of adjacent sentences has a weight that expresses their coherence (i.e., the discourse coherence of moving from a sentence to another one). The rest of their ILP model is similar to [Oliveira et al., 2016], where they maximize the weight of the selected concepts and the weight of adjacent sentences. In their work, they used the opinions as concepts, defined as triples of the form $<$*target*, *aspect*, *polarity*$>$, with polarity $\in [0, 1]$.

Finally, Galanis et al. (2012) created a regression model using a Support Vector Machine, called Support Vector Regression (SVR). Such model uses features such as: the sentence position, presence of named entities in the text, and a weighting schema similar to the Term Frequency - Inverse Document Frequency. The SVR is used to compute a relevance score a_i for each i-th sentence in the document. Those scores are used in the formulation of the ILP model, where they maximize the relevance of the selected sentences that form the summary while reducing the redundant information.

2.3.4 Lexical Chain-based methods

Other researchers [Saxena and Saxena, 2016; Brunn et al., 2001; Silber and McCoy, 2002; Barzilay and Elhadad, 1999], instead, tried to use Lexical Chains (LC) [Morris and Hirst, 1991]. Lexical chains are based on the idea of lexical cohesion, i.e. some relation between words. In details, the LC takes in input a set of words (extracted from the input document) and uses Wordnet [Miller, 1995] to find the relation between those words. The words that share some degree of relation (e.g., synonym relation, merynyms relation and so forth) form a chain. If a word does not form any relation with other words, it could make a chain by itself. Once the chains are constructed (a document could have one or more chains), they are scored on the basis of their length (short chains are preferred). Finally, LCs having the higher scores are used to retrieve sentences from the document to summarize. Those extracted sentences are used to construct the summary.

Figure 2.6 depicts an example of lexical chain.

Figure 2.6: The figure shows a possible lexical chain for the word "*summarization*".

2.4 Evaluation

So far, I described the different types of existing TS methods and the features used in supervised systems. In this section, I will report how the TS tools, also know as *summarizers*, are evaluated.

Generally, there exist two ways to evaluate a summarizer: manually or through automatic methods. The former is a simple (but costly) way to evaluate the summaries, since it only requires judges. In this case, judges usually express a value between 0 (void) and 10 (perfect) for each of the following aspects of the summary:

- presence of redundancy;
- syntactically accurate;
- semantically coherent;
- quality of the logical structure of the summary;
- presence of relevant information.

Then, those values can be combined with a weighted sum to obtain a single value for each summary. An average score is calculated to define the overall summarizer quality.

For the automatic evaluation, Rouge (Recall-Oriented Understudy for Gisty Evaluation) [Lin, 2004] measures are used. Rouge is based on the idea of Bleu [Papineni et al., 2002], where the automatic generated summary (called *candidate* summary) is compared with the original one (called *reference* summary). More in detail, Rouge is composed of several measures, but the most used for summarization are the following:

Rouge-1: comparison of unigrams (tokens) between the reference summaries and the candidates. The score is the ratio between the number of common unigrams and the number of those ones present in the reference summary;

Rouge-2: it is similar to Rouge-1, but compares bigrams (pairs of adjacent tokens);

Rouge-L: it takes into account the Longest Common Subsequence (LCS) between two texts. It is used to overcome the problems of Rouge-1 and Rouge-2 in case of short texts. Furthermore, it is used to see whether the candidate summary resembles the reference one.

Nowadays, after the contribution of See et al. (2017), researchers started to use Meteor[2] [Banerjee and Lavie, 2005], since it aligns the candidate with the reference using exact, stem, and paraphrase matching.

[2]http://www.cs.cmu.edu/∼alavie/METEOR/

There exists another method interposed between manually and automatic methods: Pyramid [Nenkova and Passonneau, 2004]. Pyramid is a semi-automatic method for summary evaluation. It starts defining a set of SCUs (Summarization Content Unit) that are not bigger than a clause[3] and have a weight. Such set of SCUs emerges by manual annotations. More in detail, a SCU consists in a set of contributors that express the same semantic content. The weight of each SCU is defined by the number of its contributions. Then, SCUs are partitioned into a pyramid, where the partition is based on SCUs weight. Each tier of the pyramid contains SCUs having the same weight. The number of tiers is defined by the maximum weight. For instance, if the maximum weight is 4, there will be 4 tiers. SCUs having the highest score are posed on the top of the pyramid, while those having the lowest score are posed on the base. The general idea behind tiers is that the lowest is the tier, less informative its SCUs are. Once the pyramid is constructed, SCUs are used to automatically assign a score to each summary. The score has a range in $[0, 1]$ and depends on the SCUs that are present in the text. Highest scores indicate correct and informative summaries. An optimal summary should contain the first tier SCUs and, length permitted, SCUs from the the next tiers.

2.5 Existing datasets

For Textual Summarization (TS), researchers have used several corpora through years. In this section, I digested the most famous ones, reporting (where it is possible) their size, language and summary type (i.e., how the summary was created) in Table 2.1.

corpus name	train size	eval. size	test size	language	summary type
CNN/Dailymail	287,226	13,368	11,490	English	expert
Gigaword	3,800,000	400,000	400,000	English	article headline
LCSTS	2,400,000	8,685	725	Chinese	expert
New York	589,284	32,736	32,739	English	expert
Reddit	404	24	48	English	expert
XSum	204,045	11,332	11,334	English	first sentence

Table 2.1: The table reports the number of articles present in the train, evaluation and test set for CNN/Dailymal, Gigaword, LCSTS, Reddit, XSum and New York corpora. I also include the language and the type of summary: if it was created by an human expert, if it is the first sentence of the article or if it is its headline.

The most used dataset comes from the conferences *DUC*[4] (Document Understanding Conference) and *TAC*[5] (Text Analysis Conference). DUC was held for 7 consecutive

[3] A clause is an Elemental Discourse Unit.

[4] https://duc.nist.gov

[5] https://tac.nist.gov

years, from 2000 to 2007, and it released a corpus for TS competitions from 2001 to 2007. TAC, instead, was held from 2008 to 2011, releasing a corpus for TS, textual entailment and question answering each year. The corpus from DUC, especially the one of 2004, has been used in recent work to compare systems [Rush et al., 2015; Cheng and Lapata, 2016; Nallapati et al., 2016].

Another famous corpus is CNN/Dailymail dataset, also known as CNN/DM in literature, composed of articles coming from CNN and Dailymail websites. Each article is composed of the main text and a set of bullets that represents the summary. It has been proposed by Hermann et al. (2015), where they used the dataset to evaluate the capability of neural networks to read a document with anonymized entities and answer questions. Successively, Nallapati et al. (2017) used Hermann et al.'s corpus to train and evaluate a neural network for extractive summarization. Finally, See et al. (2017) used the non-anonymized version to train their neural network for abstractive summarization. The choice of using a non-anonymized version is twofold:

1. the authors were interested to see the capability of the network to copy words from the document to the summary;

2. to avoid all the problems created by the algorithm used by Hermann et al. (2015) to anonymize the documents (see [Chen et al., 2016a] for details).

The non-anonimyzed version of CNN/Dailymail is composed of articles paired with multi-sentence summaries, consisting in 287,226 training pairs, 13,368 validation pairs and 11,490 test pairs. Each article has 781 tokens on average, while each summary has 3.75 sentences and 56 tokens on average.

Another corpus is Annotated Gigaword[6], created by John Hopkins University's Human and Technology Center. The corpus is the annotated version of Gigaword Fifth Edition, containing automatically-generated syntactic and discourse structures. The Gigaword Fifth Edition contains 4,111,240 articles with very simple and minimal markup structure. The corpus has been used by [Rush et al., 2015; Nallapati et al., 2016] to train and evaluate their systems. In their work, the models take in input the article and try to produce the headline. The authors propose to use 3,800,000 articles for training, and about 400,000 for validation and testing purpose.

Another famous corpus for Summarization is LCSTS, the Large-scale Chinese Short Text Summarization dataset [Hu et al., 2015], which has been constructed crawling Sina Weibo, a popular Chinese social network. In Sina Weibo, people or associations daily post an article with a short summary (less than 140 Chinese characters). Hu et al. (2015) started from a subset of 50 popular organization users and collected the users followed by such popular organization, filtering those ones that have less than 1 million followers. Their idea is that popular users are able to produce short, informative and grammatically

[6]https://catalog.ldc.upenn.edu/LDC2012T21

correct summaries. Then, they extracted the <*text, short summary*> pairs produced by the selected users, and divided them in three subsets: Part-I, Part-II and Part-III. Part-I contains 2,400,591 pairs that can be used for the training of a system. Part-II and Part-III contain 10,666 and 1,106 pairs respectively and they are evaluated by 5 human judges. More in detail, the authors asked to judges to give a score between 1 and 5 to each pair, according to the objectivity of the summary, its grammar and logic coherence with the original text. From the analysis of those scores, they found that pairs with a 3, 4 and 5 score are informative, concise and short. They propose to use such pairs to evaluate the systems.

Another corpus is Reddit[7], created by Ouyang et al. (2017). The authors created the corpus starting from 476 personal narratives collected by Ouyang and McKeown (2015). Each story has been annotated with an abstractive and extractive summary using Amazon Mechanical Turk. The annotation produced 1088 aligned abstractive and extractive summaries. Kedzie et al. (2018) used such corpus to train and evaluate their system, splitting it into 404 stories for trainset, 24 stories for validation set, and 48 stories for testset.

Another corpus is XSum[8], created by Narayan et al. (2018) harvesting BBC articles ranging over almost a decade (from 2010 to 2017). The corpus contains 226,711 articles that has been divided into 204,045 articles for training, 11,332 for validation and 11,334 for test. The summary of each article was created by selecting its first sentence, which is an introductory sentence (usually written by the author of the article) that digests the content of the article. Each article of the corpus has an average number of words of 431.07, divided into about 19.77 sentences; the summary, instead, has an average length of 23.26.

Finally, Paulus et al. (2018) propose to use New York annotated corpus[9] as further corpus to train and evaluate summarization systems. The corpus contains article written by the New York Times between January 1987 and June 2007. It is composed of 1,800,000 articles, of which over 650,000 articles have a summary written by library scientists. According to Paulus et al. (2018), the corpus has varied and shorter summaries, with a high level of abstraction of paraphrase. They sorted the 650,000 articles in chronological order and then used 589,284 articles for training, 32,736 documents for validation and 32,739 articles for test.

[7] www.reddit.com

[8] https://github.com/EdinburghNLP/XSum

[9] https://catalog.ldc.upenn.edu/ldc2008t19

Chapter 3

Background on Neural Networks

3.1 Introduction

Neural Networks are Machine Learning algorithms, i.e. algorithms that could be trained using real-data to solve a specific task (e.g., predict the class of a document). Neural Networks have been inspired by the structure of the brain, where the fundamental unit is the neuron (see Figure 3.1) composed of four parts:

Dendrite: It is the input channel of the neuron, which receive electro-magentical signals from other neurons;

Synapse: It is the output channel of the neuron, used to emits the elabored signal to other neurons;

Axon: It is another channel that connects the center of the neuron, called *Soma*, with the Synapse;

Soma: It is the fundamental part of the neuron. It could be seen as the CPU of a computer, since it elaborates the signals coming from the Dendrites, and decides to emit an output (i.e., to activate) or not. If the neuron activates, a signal is propagated through the Synapse; otherwise, no signal is emitted.

More in detail, the signal received by the neuron from its dendrites could *excite* it, which will generate an output if the excitement surpasses a threshold (in jargon, *activate*). The input signal could also *inhibit* the neuron, i.e. the signal prevents the activation of the neuron. This binary behaviour is at the base of our 100 billion neurons in order to understand every complex input coming from the world. For instance, a sound (that we listened) activates only some part of the brain (those ones that are designed to elaborate sounds and language), while others (e.g., the one regarding motion) will be probably inactive[1]. Such elaboration is performed hierarchically, where the raw signal

[1]This is an hypothetical representation used to explain how the brain works.

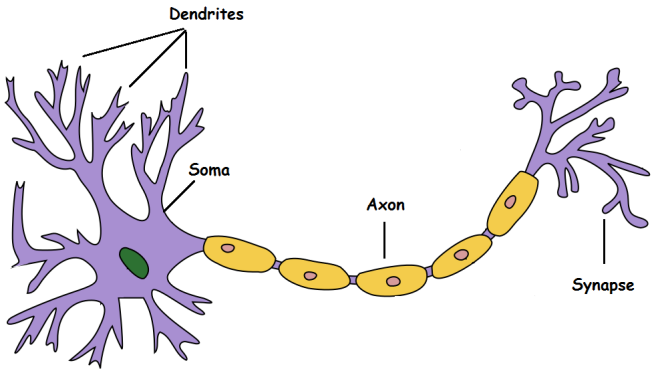


Figure 3.1: The figure represents an human brain's neuron. The image was taken from Wikipedia (https://en.wikipedia.org/wiki/Neuron)

is passed to a first layer of neurons. Some of them will fire, while other not. The output of the layer is passed to another one, and the process is repeated until the brain results in a final response.

The structure and behaviour of the neurons have been (and it is still) studied by neurologists and logicians in order to replicate it in constructed machines. To the best of my knowledge, the first work that posed the basis of modern neural network is [McCulloch and Pitts, 1943], from Warren McCulloch (neuroscientist) and Walter Pitts (logician). In their work, the authors formalize a logical system where neurons receive boolean values through the denditres. The neuron aggregates those value via a sum, and the resulting value is passed as input to a function which defines the behaviour of the neuron. Such function could represent any boolean operator, but the xor. With more details, if the coming summed value is greater than a given threshold, the function will emits (in jargon, *fire*) the value 1; otherwise, the emitted value is 0. In their article, the authors made also the following assumptions for their system (where some still hold in current neural network models):

- A neuron can emits or not a value. They call it *all-or-none* process;
- The excited synapses at a given time are independent from position and previous activities;
- The structure of the network does not change.

More details about the neuron of McCulloch and Pitts can be found in Appendix A. Their idea has been further elaborated by Rosenblatt in his perceptron [Rosenblatt, 1958]. Rosenblatt proposed some changes that are the fundamental of modern neural networks:

- The input of the neuron could be any value (belonging to $\mathbb{R}^n$), and not just a boolean one;
- The output of the neuron is a value in the set $\{-1, 1\}$;
- The input values are multiplied by a set of weight that excites or inhibits the neuron;
- A training rule to modify the weights of the perceptron.

However, both McCulloch and Pitts' neuron and Rosenblatt's perceptron are not able to solve problems involving non-separable data, or in more general to represent the *xor* function. According to Minsky and Papert (1988), many tasks involve non-linearly separable datasets. This set the research on neural networks aside until *backpropagation* was adopted. The backpropagation algorithm has no connection with how the human brain works, since it is based on the minimization of an objective function using a gradient descent method. It has been the engine of the neural network renaissance, leading to the creation of Multi-Layer Perceptron, Convolutional Neural Network [LeCun et al., 1995], and Recurrent Neural Network models [Hochreiter and Schmidhuber, 1997; Gers et al., 1999; Cho et al., 2014].

3.2 Rosenblatt's Perceptron

The perceptron model proposed by Rosenblatt (1958) is, to the best of my knowledge, one of the seminal works on Neural Networks. The architecture of the perceptron involves a set of weights that analyzes the input data to produce an output. The interesting aspect of the model is that those weights can be modified with respect to the output in order to reduce the error. Such architecture is at the base of nowadays Artificial Neural Networks.

Suppose to have a dataset composed of two classes, C_1 and C_2, and that is *linearly separable*. With linearly separable I means that, representing such points on a space, it is possible to draw an hyperplane that perfectly divides the examples into the two classes. Figure 3.2 shows a linearly separable dataset. It is a sufficient and necessary condition for the convergence of Rosenblatt's perceptron.

The Rosenblatt's perceptron is capable to learn from the dataset and classify new examples in input into the two classes. The perceptron calculates a weighted sum of the input, multiplying the values by the weights and summing the results together. The resulting value, called *local field* v, is passed as input to an hard-function ϕ, a *sign* function which assign the input to one of the two classes: C_1 if the output of the function is 1, or C_2 if it is -1. Let $(x_1, \ldots, x_n)$ be an input vector, and let $(w_1, \ldots, w_n)$ be a vector of weights. The perceptron is defined by the following equations:

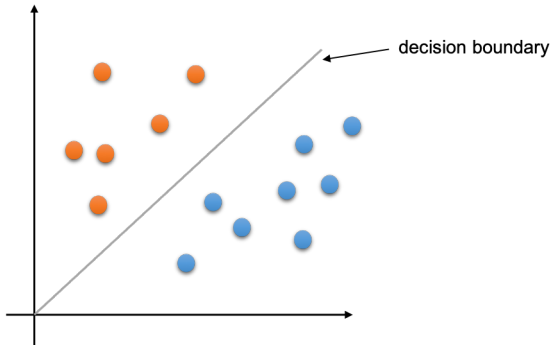


Figure 3.2: The figure show a linearly separable datasets. The grey line represents the hyperplane that perfectly separates the points of the two classes.

$$\phi(v) = \begin{cases} 1 & \text{if } v > 0 \\ -1 & \text{if } v \leq 0 \end{cases} \tag{3.1}$$

$$v = \sum_{i=1}^{n} w_n * x_n \tag{3.2}$$

A visual description of the model is depicted in Figure 3.3.

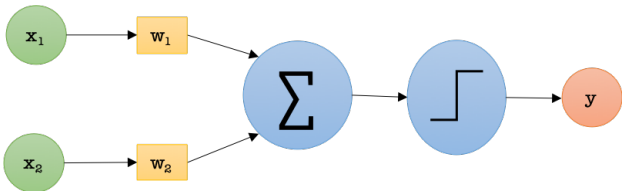


Figure 3.3: The figure shows the structure of a perceptron.

The training of the network is performed changing the weights in order that the weighted sum of a C_1 class example produces a positive local field, while the weighted sum of a C_2 class example produces a negative (or zero) local field. This means it is drawing an hyperplane that separates the two classes.

For simplicity, let $\mathbf{x}(m)$ be the input vector and $\mathbf{w}(m)$ be the weights vector at the m-th timestep. The model has to calculate a vector of weights $\mathbf{w}$ such that for each timestep m:

$$\mathbf{w}(m)^T\mathbf{x}(m) > 0 \text{ and } \mathbf{x}(m) \in C_1$$
$$\mathbf{w}(m)^T\mathbf{x}(m) \leq 0 \text{ and } \mathbf{x}(m) \in C_2$$

How is it possible to change the weights in order to make the previous statement true? Generally, each time an example is read, it falls under two cases:

case 1 : The perceptron correctly predicted the class of the example. In this case, the weights are not updated: $\mathbf{w}(m+1) = \mathbf{w}(m)$;

case 2 : The perceptron wrongly predicted the class of the example:

$$\begin{aligned} &(1) \quad \mathbf{w}(m)^T\mathbf{x}(m) > 0 \text{ and } \mathbf{x}(m) \in C_2, \text{ or} \\ &(2) \quad \mathbf{w}(m)^T\mathbf{x}(m) \leq 0 \text{ and } \mathbf{x}(m) \in C_1 \end{aligned}$$

In (1), the perceptron overestimates, so the weights have to be decreased. In (2), the perceptron underestimates, so the weights have to be increased. Rewriting $\mathbf{w}(m)^T\mathbf{x}(m)$ as $\mathbf{v}(m)$ for convenience. Those two changes can be represented as:

$$\mathbf{w}(m+1) = \mathbf{w}(m) \begin{cases} -\eta(m)\mathbf{x}(m) & \text{if } \mathbf{v}(m) > 0 \\ +\eta(m)\mathbf{x}(m) & \text{if } \mathbf{v}(m) \leq 0 \end{cases} \tag{3.3}$$

where $\eta(m)$ is a constant, called *learning rate*, that controls the update of the weights. In geometrical terms, each weight update produced by Equation 3.3 moves the hyperplane, called *decision boundary* (see Figure 3.2), in such a way that the examples of the two classes are separated.

Generally, the learning rate is in the range (0, 1]. A large learning rate (e.g., 1.0) means that all the information contained in $\mathbf{x}(m)$ is used; in this case, the update step will be large, making the network converging fast. A small learning rate (e.g., 0.05) means that the information contained in $\mathbf{x}(m)$ are rescaled to small values; in this case, the update step will be small, making the network converging slowly.

The Rosenblatt's perceptron converges if and only if the dataset is linearly separable. But, what will it happen if the dataset is not linearly separable? The perceptron will never converge, making the algorithm not useful to solve the task. Since many tasks are based on non-linearly separable datasets (as reported by Minsky and Papert (1988)), this reduced the research on Neural Networks until *backpropagation*[2] was adopted in Artificial Neural Networks.

3.3 Artificial Neural Networks

Artificial neural networks tried to replicate the structure of brain using artificial neurons (MPNs or perceptrons, for example) organized in a hierarchical manner. These networks are trained simulating the brain behaviour, where the learning comes from strengthening connections designed to solve a specific task (e.g., recognizing hand-writing). As

[2]The backpropation algorithm is described in Appendix B.

we seen in the perceptron, the learning requires a *feedback* that is used to modify the weights of the neurons, producing a specific output given a specific input. Such learning could be *supervised* or *unsupervised*. The former one requires examples of the form *<input, label>*, where the label is used to give a feedback to the network. For instance, the input could be the features of an example and the label could either be a class C_1 or C_2. In this case, the network is trained to output the correct answer given a specific input. More in detail, the network receives in input an example and predicts its class; if the class predicted is different from the label (e.g., the network predicted C_2, but the correct answer is C_1), the weights are changed via partial derivatives in order to correctly recognize the input example. In the unsupervised learning, the network will strengthen its connection in order to cluster examples with a similar structure, without using any label. An example of unsupervised neural network is the Self Organizing Map (SOM) [Kohonen, 1990]. Despite this latter case is very interesting, it is not the focus of this thesis.

Artificial Neural Networks are composed of three or more layer, and are called *feed-forward* since the information flows from the first layer to the last one. The first layer is called *input layer*, while the last one is called *output layer*. Any layer between the input and the output is called *hidden layer*. Each layer (except the input one) contains neurons that are connected with the previous one. If neurons of a layer are connected with all neurons of the next layer, we said that the network is *fully-connected*. An architecture of feed-forward network is depicted in Figure 3.4.

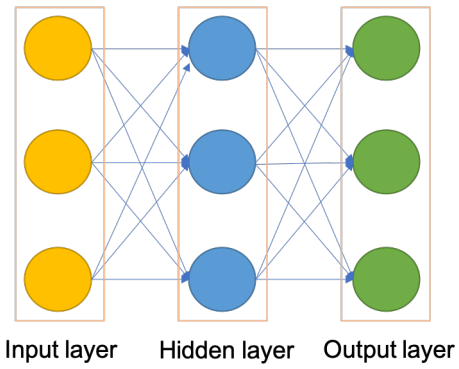


Figure 3.4: The pictures shows a fully-connected feed-forward neural network with three layers: an input layer, an hidden layer, and an output layers. Colored circles represent neurons.

Input layer simply reads the data in input and makes them usable for the next layer. In this layer, I prefer to include word embeddings [Mikolov et al., 2013c; Pennington et al., 2014], since it transforms each word in input into a numerical representation usable by the network. In detail, the word embedding is a matrix where each row represents a word in the vocabulary and each column represents a feature of a word. The idea

of word embedding comes from distributional semantic approaches, where words are represented as continue vectors. Such vectors can be pre-trained (generated by a neural language model) or learned during the training of the model. Word embeddings will be explained in Section 3.6.

Hidden layers are interposed between input and output layer. The goal of hidden layers is to extract high-level features that can be used to classify the input. For instance, hidden layers can combine pixels of an image into edges, edges into object parts, and so forth.

Output layer is used to make the prediction. One of the most used function in the output layer is the *softmax* function, which generates a probability distribution over the classes. The predicted class is the one with the highest probability. Let $\mathbf{x}$ be a vector belonging to $\mathbb{R}^{m \times 1}$, where m is the number of classes. The probability score of i-th class can be defined as:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}} \tag{3.4}$$

This function is widely used in Summarization to predict the next word in output; in those tasks, the words in the vocabulary are treated as classes.

Activation Functions

Activation Functions are widely used in Neural Network models. For instance, Rosemblatt's perceptron uses the hard-function (or step function) to generate the label. In general, the activation functions are applied to the weighted sum of the input in order to produce an output, which can be either an hidden space to extract further features or an output representation (e.g., a probability distribution over the label set). A common function is the *sigmoid*, that maps a value to the range [0, 1].

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.5}$$

The function has a characteristic S-shape and simulates the activation of a neuron. It could be also used to represent a Bernoulli distribution. The output of sigmoid function strictly depends from x:

- with positive x, the output tends to 1;
- with negative x, the output tends to 0;

Other most used functions in neural networks are:

hyperbolic tangent: a function that maps a value to the range [-1, +1]. It is one of the most used function in neural networks since it does not "stuck" the network

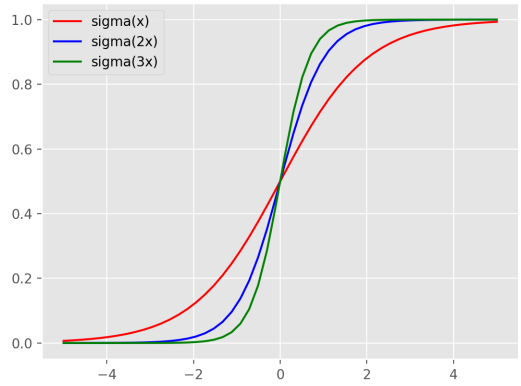


Figure 3.5: The picture shows the geometrical representation of sigmoid function in the range $[-5, 5]$.

during the training due to values close to zero.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.6}$$

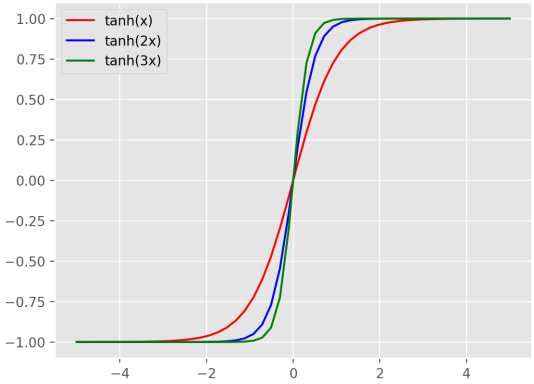


Figure 3.6: The figure shows the tanh function in the range $[-5, 5]$

Rectified Linear Unit: also called *ReLU*. It is a function that maps the input to the range [0, ∞). The idea behind ReLU is to help the model to account interaction effects, i.e., when a variable affects the prediction on the basis of another one

(e.g., weight and height).

$$ReLU(x) = max(0, x) \tag{3.7}$$

The main advantages of the ReLU function are: (i) it accelerates the training of the network compared to the previous functions (sigmoid and tanh); (ii) it involves less expensive mathematical operations than hyperbolic tangent, which derivative requires the exponential. However, it presents a drawback: larger gradients[3] could update the weights in a such way that the function will not activate (i.e., it returns 0) for any input.

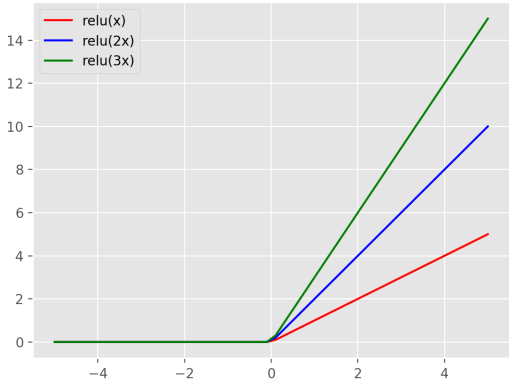


Figure 3.7: The figure shows the ReLU function in the range $[-5, 5]$

Bias

When I presented the perceptron, I said that it computes a weighted sum of the inputs values, and then it applies the activation function (e.g., sign or sigmoid). An important changes to the model is the introduction of a *bias* value. Such value, which is added to the local field, makes the neuron more sensible (or insensible) to certain inputs. An example of how the sigmoid function varies with different bias values is depicted in Figure 3.8. For instance, in case of a bias equals to 1, the network will be more insensible to negative inputs.

A compact way to represent the bias function is to treat it as a weight, which is always multiplied by the constant 1. In this way, it can be easily added to the local field of the neuron. This new architecture is depicted in Figure 3.9.

[3]The gradient is a vector whose components are the partial derivatives of the function to which it is applied.

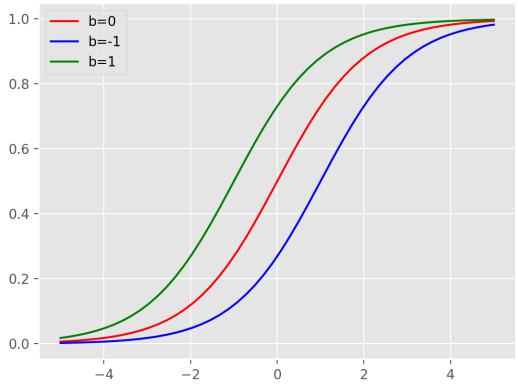


Figure 3.8: The figure shows how the sigmoid function changes when different bias values are added.

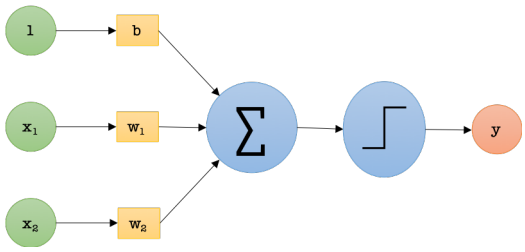


Figure 3.9: The image shows a perceptron where the bias is treated as a weight. The yellow rectangle with b represents the bias.

3.4 Computational Graph

Generally, the architecture of a Neural Network model can be represented using the *computational graph*, which describes the variables (weights) involved, how they are combined to produce an output, and in generally how the layers are organized. The purpose of the computational graph is not only to describe the network, but also to provide a structure on which the network can be implemented. The computational graph can be defined as a Directed Acylic Graph (DAG). In the computational graph, nodes indicate variables and edges represent interactions between variables. A variable can be a scalar, a matrix or a tensor. The computational graph contains also *operations*, functions of one or more variables. If a variable y depends on the application of a function on the variable x, an edge between the two variables is drawn, labelling it with the name of the function. Figure 3.10 shows an example of computational graph.

Both algebraic expressions and computational graphs operate on *symbols*, variables that have a specific value. Such symbols can be substitute with numerical value. It

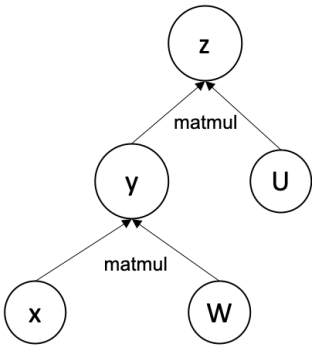


Figure 3.10: The image shows an example of computational graph.

can be compared with instantiate and initialization in programming languages: first, a variable that will contain a string (or an integer, for example) is instantiated; then, a value is assigned to the variable. I will use computational graphs to explain Recurrent Neural Networks in Section 3.7, one of the most adopted networks in the Summarization task.

3.5 Loss Functions

Artificial Neural Networks widely differ from Rosemblatt's perceptron: while this latter one only checks if its output is aligned with the expected one, artificial neural networks can access to a plethora of functions that provide an insight of how well the model is performing on the task. These functions are called in jargon *loss functions*, and are used to update the weights (biases included). In the Neural Network architecture, the loss function is located after the output prediction since it acts on this latter one. For instance, when a network receives in input the data, it processes them hierarchically through its layers, emitting the output (which can be a probability distribution over a set of classes). The output is then compared with the gold standard using the loss function, which (in this case) expresses how the network has to change the weights in order to predict the correct class.

There exist several types of loss function that depend on the task, but the most famous one is the *cross entropy*. Suppose to have a fixed model (also knows as *hypothesis*) that predicts for m classes [x_1, x_2, ..., x_m] their occurrence probabilities [y_1, y_2, ..., y_m]. Suppose to observe (in real) k_1 instances of class y_1, k_2 instances of class y_2, and so on up to k_m instances of class y_m. According to the described model, the likelihood is:

$$P[data|model] = y_1^{k_1} y_2^{k_2} \dots y_m^{k_m} \tag{3.8}$$

Taking the logarithm of Equation 3.8 and changing the sign, it is possible to obtain the log-likelihood:

$$\begin{aligned} & -\log P[data|model] \\ & = -k_1 \log y_1 - k_2 \log y_2 - \cdots - k_m \log y_m \\ & = -\sum_{i=1}^{m} k_i \log y_i \end{aligned} \tag{3.9}$$

Defining $N = k_1 + k_2 + \cdots + k_m$, the left hand side and the right hand side of Equation 3.9 can be divided by such value:

$$\begin{aligned} & -\frac{1}{N} \log P[data|model] \\ & = -\frac{k_1}{N} \log y_1 - \frac{k_2}{N} \log y_2 - \cdots - \frac{k_m}{N} \log y_m \\ & = -\frac{1}{N} \sum_{i=1}^{m} k_i \log y_i \end{aligned} \tag{3.10}$$

Setting $\hat{y}_i = \dfrac{k_i}{N}$, the equation of the cross entropy is obtained:

$$CE(\hat{y}, y) = -\sum_{i=1}^{m} \hat{y}_i \log y_i \tag{3.11}$$

where the part $\log y_i$ (in case of base 2) represents the minimal number of bits needed to encode the independent event with probability y_i, i.e. it represents the length of the encoding. In case of only two classes, Equation 3.11 is rewrote as follows:

$$CE(\hat{y}, y) = -\hat{y} \log y - (1 - \hat{y}) \log(1 - y) \tag{3.12}$$

Other functions that can be used to train a neural network are:

Negative Log Likelihood: It is used when the neural network emits a probability value for each class. The aim of this loss is to maximize the probability value of a single class.

$$L(y) = -\log p(y)$$

Hinge Loss: It is a loss used for maximum margin classification, such as the one performed in the Support Vector Machine. The idea is to draw an hyperplane that divides the instance of two classes into two distinguished groups. The following

equation shows the hinge loss for a multi-label classification, where m is generally set to 1. In the hinge loss, if the predicted class $\hat{y}$ is the same of y, the result is 0.

$$L(\hat{y}, y) = max(0, m - \hat{y}y)$$

Cosine Similarity: It is used to measure the similarity of two vectors. This loss has a range in $[-1, 1]$, where -1 means that the two vectors are different, while 1 means that the two vectors are the same. A network trained with the cosine similarity loss will modify the vectors in order to make them close.

$$L(\hat{y}, y) = \frac{\hat{y} \cdot y}{||\hat{y}|| \cdot ||y||}$$

where $|| \cdot ||$ is L2 norm of the vector.

Kullback Leibler (KL) Divergence: It measures the divergence of a probability distribution respect to another one. The KL divergence has a range in $[0, 1]$, where 0 means that the two distributions are different, while 1 means that the two distributions are the same one. The KL divergence is not a symmetric function: it only measures if the first distribution is similar to the second one. If the two distributions are swapped, the KL divergence will produce a different value. A network trained using KL divergence will modify the weights in order to make the predicted distribution similar to the target one.

$$L(\hat{y}, y) = \frac{1}{M} \sum_{i=1}^{m} D_{KL}(\hat{y}_i, y_i) = \frac{1}{M} \sum_{i=1}^{m} [y_i \cdot \log(\frac{y_i}{\hat{y}_i})]$$

3.6 Word Embedding

When I presented the neural networks, I said that they accept numerical values as inputs. As consequence, neural networks for Natural Language Processing tasks have to map vocabulary words to numerical features. But how is it possible to do that? Researchers found that those vectors can be generated using a neural network. The resulting set of vectors is called *word embedding*. In detail, a numerical vector (initialized randomly or through a uniform distribution) with fixed dimensions is created for each word in the vocabulary; then, the network is trained to solve a language model task, i.e. predicting the next word given a fixed-size context window. During the training, the network will find the features that best represent each word of the vocabulary. Furthermore, those vectors generate an n-dimensional space where words that appears in the same contexts (e.g., synonyms or city names) are close, forming clusters.

The idea of representing words as vectors comes from the distributional semantics, where words are defined according the company they keeps [Firth, 1957]. Bengio

et al. (2003) followed the core of distributional semantic approach, defining a neural network-based language model. They trained the language model to predict the next words in a sentence, given a fixed size window of previous words (they are represented via word embedding). Since the network is trained to predict the next words, it learns to distinguish them on the basis of their context in a unsupervised fashion. The features that help to differentiate a word from another are collected in the vectors. Successively, Mikolov et al. [Mikolov et al., 2013c,a,b] refined the model proposed in [Bengio et al., 2003]. The authors proposed two variants: *Continuous Bag-Of-Words* (CBOW) that is similar to the model of Bengio et al., and *SkipGram* which is trained to predict the context that surrounds a target word. They also introduced some techniques to improve the word embedding and to make the training fast, such as *hierarchical softmax* and *negative sampling*. Pennington et al. (2014) proposed a different model to create the word embedding. They combined statistical information about word co-occurrence with the word embedding. Other authors tried to include different information into word embedding. For example, Iacobacci et al. (2015) proposed *SensEmbed*, where they applied SkipGram onto disambiguated words. They used BabelNet [Navigli and Ponzetto, 2010] to find the sense of each word in the text. Bojanowski et al. (2017), instead, included character information; they called the model *FastText*.

Finally, other authors [Peters et al., 2018; Devlin et al., 2018] redefined the previous neural-based language models to obtain a deep contextualized word representation. In this context, *ELMo* and *BERT* will be presented.

3.6.1 Neural Language Model

Bengio et al. (2003) proposed a Language Model based on Neural Networks. Their model is based on a feed forward network that takes in input a phrase excerpt, and predicts the word that follows that excerpt. In details, they first constructed a training set where the input is a context window formed by m consecutive words, and the target is the $m+1$ word. They created such training set from sentences. Then, they developed a feed-forward network, depicted in Figure 3.11. The network takes in input the context window and maps the words to their corresponding word embedding; then, the hidden layer transforms those embedding into a single vector representation of size $1 \times n$, where n is the size of the hidden vector. This vector is then used to emit a probability distribution over a fixed size vocabulary. The probability distribution is generated using the softmax function. Supposing that the vocabulary size is V, the softmax function first transforms the hidden vector into a $1 \times V$ one, multiplying it with a weight matrix of size $n \times V$; then it generates the probability distribution applying the exponential and normalizing the values. Finally, the word having the highest probability is compared with the target one, and the error is backpropagated.

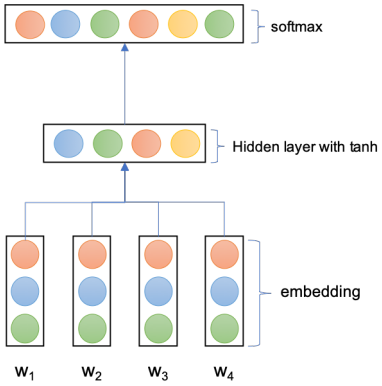


Figure 3.11: The image shows Bengio et al. [Bengio et al., 2003] language model.

3.6.2 Word2Vec: CBOW and SkipGram

Mikolov et al. [Mikolov et al., 2013c,a,b] proposed two different models to generate the word embedding, collected under the name of *word2vec*. The first one, called Continuous Bag-Of-Word (CBOW), is similar to the model defined by Bengio et al. In the description of the model, I will first start from a simplified version (with just one word in input) to explain all the layers; then, I will describe the real case, where a bag-of-word is given in input to the network. In the models, I suppose that the vocabulary size is V, the size of the hidden layer is H, and that the input is a one-hot vector, i.e. a vector where one of the V words $\{x_1, x_2, \ldots, x_V\}$ has value 1. I represent this latter vector with $\mathbf{x}$ of size $V \times 1$.

In the simplified version, I will assume that the networks receives only a word in input and predicts the following word. For instance, suppose to have the sentence "a brown fox is running"; if the word *brown* is given in input, the network will predict *fox*. In detail, the network multiplies $\mathbf{x}$ with the matrix $\mathbf{E}$, which has size $H \times V$. The matrix $\mathbf{E}$ is called *word embedding*. Since the input vector $\mathbf{x}$ has all zeros, except for one value, it will select from the matrix $\mathbf{E}$ the row that corresponds to that value. For instance, if $w_k = 1$ and $w_i = 0$ for all $i \neq k$, the product between $\mathbf{x}$ and $\mathbf{E}$ will extract the word embedding that corresponds to w_k:

$$\mathbf{v}_k = \mathbf{E}\,\mathbf{x} \tag{3.13}$$

where $\mathbf{v}_k$ is the word embedding of w_k, i.e. the k-th row of matrix $\mathbf{E}$. Differently from the model of Bengio et al., CBOW has a no activation function in the hidden layer. Once that the word embedding of a word is extracted, the network predicts the next word. The output layer of the model is the same of Bengio et al. In detail, it multiplies the vector $\mathbf{v}_k$ with the matrix W, which has size $V \times H$, in order to produce a score for each word w_j in the vocabulary. Those scores are then transformed into probability

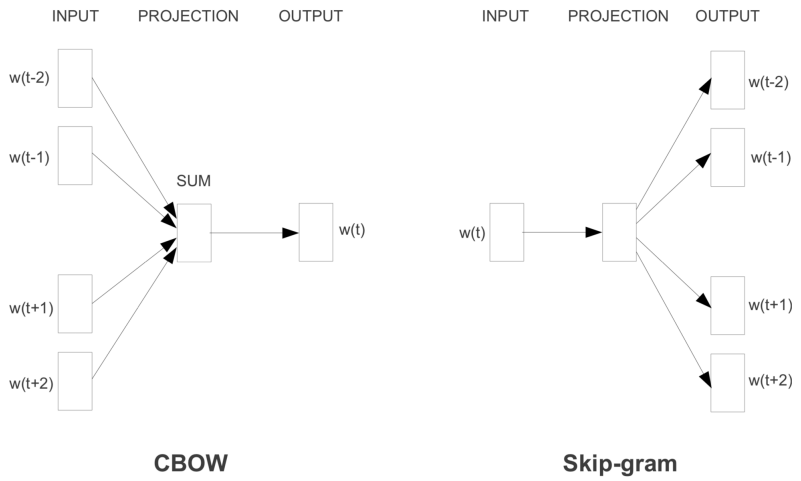


Figure 3.12: The picture shows the architecture of CBOW (on the left), and SkipGram (on the right). The image is taken from Mikolov's research paper [Mikolov et al., 2013a].

values applying the softmax function. The output layer is defined through the following equations:

$$\begin{aligned} \mathbf{u}_j &= \mathbf{W}_{k,j}\mathbf{v}_k \\ p(w = w_j) &= \frac{\exp(\mathbf{u}_j)}{\sum_{j'=1}^{V} \exp(\mathbf{u}_{j'})} \end{aligned} \tag{3.14}$$

where $\mathbf{u}_j$ is the score of the j-th word in the vocabulary, and $\mathbf{W}_{k,j}$ represents the weight that connects the word embedding of k-th word with the j-th word in output.

In the real version of CBOW, the network does not only take in input one word, but a context windows composed of C words. Each word w_k is then mapped to its word embedding $\mathbf{v}_k$. The main difference between this model and the simplified one is that each word embedding is not separately passed to the output layer, instead the network calculates an average vector. Defining the context window words $\{w_1, w_2, \ldots, w_C\}$ where each w is represented as a one-hot vectors of size C, the output of the hidden layer is defined as follows:

$$\mathbf{v} = \frac{1}{C}(\mathbf{v}_1 + \mathbf{v}_2 + \cdots + \mathbf{v}_C) \tag{3.15}$$

The average vector $\mathbf{v}$ is then multiplied by the matrix $\mathbf{W}$ to obtain a score for each vocabulary word. This model is depicted in Figure 3.12.

SkipGram is the opposite of CBOW since it swaps the output and the input: instead of predicting the next word given a context, it predicts the context given a target word.

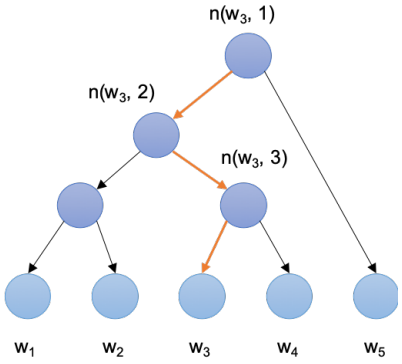


Figure 3.13: The image shows a binary tree for a vocabulary containing 5 words. The arrows in orange shows the path from the root to word w_3.

Furthermore, the architecture of the network is a swapped version of CBOW, as it is reported by Figure 3.12. More in detail, SkipGram is similar to the simplified version of CBOW up to the hidden layer. For the output layer, it predicts a probability distribution over the vocabulary for each context word. The error of the network is defined as the sum of the errors of each target word.

Hierarchical Softmax

One of the biggest problem in calculating the probability distribution over the vocabulary is the bottleneck caused by softmax function. This latter one requires a lot of time to be computed, especially with very larger vocabulary. To overcome this issue, hierarchical softmax was adopted. Hierarchical softmax uses a binary tree to store all the vocabulary words. Given a vocabulary of size V, there are V leaves in the tree, and $V-1$ inner levels. For each word, there exists an unique path from the root to the leaf. Figure 3.13 shows an example of binary tree for the hierarchical softmax.

In the hierarchical softmax, words do not have a vector associated to them, i.e. one of the dimension of matrix $\mathbf{W}$ is no longer V but a single value. The vector representation, instead, is moved to to the inner nodes. The probability of emitting a word w_i is then calculated as follows:

$$p(w = w_i) = \prod_{j=1}^{L(w_i)-1} \sigma([\![n(w, j+1) = ch(n(w, j))\ \mathbf{v}'^{T}_{n(w,j)}\mathbf{h}]\!]) \tag{3.16}$$

where *ch(·)* is the left child of inner node n, $\mathbf{h}$ is the output of the hidden layer (the average vector for CBOW, and the embedding of the input word for SkipGrim), and $\mathbf{v}'_{n(w,j)}$ is a row of the output matrix $\mathbf{W}$ that corresponds to the node $n(w, j)$. The function $[\![\cdot]\!]$ is defined as follows:

$$[\![x]\!] = \begin{cases} 1 & \text{if x is true} \\ -1 & \text{otherwise} \end{cases} \tag{3.17}$$

Let me explain Equation 3.16 through an example. Suppose to calculate the probability of w_3 being an output word, given the tree of Figure 3.13. Such probability can be defined as a random walk that starts from the root and ends to the leaf in question. In detail, to calculate the probability of w_3, the probabilities of going left and right for each inner node (root included) are calculated. The probability of going left at node n is defined as:

$$p(n, left) = \sigma(\mathbf{v}'_n \mathbf{h}) \tag{3.18}$$

and the probability of going right as:

$$p(n, right) = \sigma(-\mathbf{v}'_n \mathbf{h}) = 1 - \sigma(\mathbf{v}'_n \mathbf{h}) \tag{3.19}$$

Hence, following the path from the root to the leaf, the probability of emitting the word w_3 in output is defined as follows:

$$\begin{aligned} p(w = w_3) &= p(n(w_3, 1), left) \cdot p(n(w_3, 2), right) \cdot p(n(w_3, 3), left) \\ &= \sigma(\mathbf{v}'_{n(w_3,1)} \mathbf{h})\ \sigma(-\mathbf{v}'_{n(w_3,2)} \mathbf{h})\ \sigma(\mathbf{v}'_{n(w_3,3)} \mathbf{h}) \end{aligned} \tag{3.20}$$

Negative Sampling

Differently from the hierarchical softmax, which simplifies the softmax function, negative sampling acts on the parameters level. The idea is to simplify the loss function in order to update only the vectors (output and embedding) of the target word(s), and the ones of few words that are sampled as negative examples. The *noise distribution* is used to sample the negative words, which is defined as an unigram distribution raised to the $\frac{3}{4}$th power. Given $\mathbb{W}_{neg}$ as the set of sampled negative words, the loss function is defined as follows:

$$L = -\log(p(w = w_j)) - \sum_{w_k \in \mathbb{W}_{neg}} \log(p(w = w_k)) \tag{3.21}$$

3.6.3 Finding similar words

When I first described the word embedding, I said that they represent features extracted by the Neural Network. Since they contain features, it is possible to perform several operations on them:

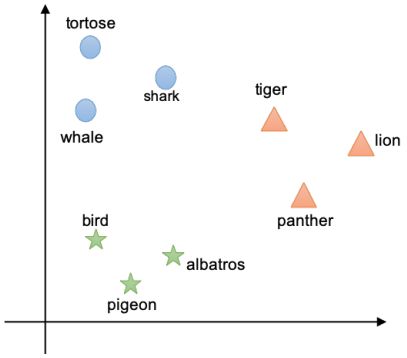


Figure 3.14: An example of word embedding involving three types of animals.

1. a projection into an n-dimensional space (with n equals to 2 or 3), where each dimension represents a feature. In this way, it is possible to see how words have been self-organized according to their features;

2. an exploration of the vicinity of a word, retrieving all similar ones that are close to the target word;

3. word analogies resolution under the form of: $a - b = c - d$, where a, b, c and d are words. In the analogy, 3 of 4 words are given; the 4th one is the wildcard that has to be found. For instance, "Italy - Rome = France - x" is an analogy where the wildcard is Paris. Such analogy could be read as "*if Rome is the capital of Italy, which is the capital of France?*".

For the first point, Principal Component Analysis (PCA) or t-distributed Stochastic Neighbor Embedding (t-SNE) [Maaten and Hinton, 2008] could be used to project the word embedding from an high-dimensional space to a 2-dimensional one. Figure 3.14 shows an example of a represented word embedding.

For points (2) and (3), the cosine similarity function is used to rank the words with respect to the target one. Given a input word w_i, the function is defined as follows:

$$cosine(E(w_i), E(w_j)) = \frac{E(w_i)^T E(w_j)}{||E(w_i)||\,||E(w_j)||} \tag{3.22}$$

where $E(\cdot)$ is a function that maps a word to its embedding vector, $||\cdot||$ is L2 norm, and w_j is another vocabulary word different from w_i. For point (3), Equation 3.22 is slightly modified in order to include all given information. Supposing that the wildcard is d, w_i will be $a - b + c$. In this case, each word d in the vocabulary that is closed to $a - b + c$ in the word embedding is ranked using the cosine function. Then, the word having the highest score (i.e., the closest one to $a - b + c$) is selected as answer. The cosine function for the Italy-France capital example is calculated as:

$$\begin{aligned} &cosine(Italy - Rome + France, d) = \\ &cosine(E(Italy) - E(Rome) + E(France), E(d)) \end{aligned} \tag{3.23}$$

3.6.4 Glove

When I first described word embedding, I said that it is based on the idea that a word could be defined according to the company it keeps [Firth, 1957]. For such reason, the previous model tried to capture those information predicting the next words (or all the words in a given context windows). Pennington et al. (2014), in their model Glove, tried to further explore this direction, including co-occurrence information extracted from a corpus.

To define their model, they first started constructing a co-occurrence matrix X, where the cell $X_{i,j}$ contains how many times the word i appears in the same context of word j. Using this matrix, it is possible to calculate the probability that a word w_i appears in the same context of a word w_j:

$$P_{i,j} = P(i|j) = \frac{X_{i,j}}{X_i} \tag{3.24}$$

where X_i is the sum of all cells of i-th row, i.e. $X_i = \sum_j X_{i,j}$. Using $P_{i,j}$, the relationship of words w_i and w_j with a context word $\hat{w}_k$ can be calculated:

$$F(w_i, w_j, \hat{w}_k) = \frac{P_{i,k}}{P_{j,k}} \tag{3.25}$$

The right-hand side of the above equation is extracted from a corpus, while the left-hand side may depend of some sort of unspecified parameters. Since Pennington et al. want to represent words as embedding vectors, they decided that F should encode the information of $\frac{P_{i,k}}{P_{j,k}}$ into the vector space. This could be done treating all the three words in the left-hand side as vectors and computing the difference between the vector that represents w_i with the vector that represents w_j. Equation 3.25 is then changed as follows:

$$F(w_i - w_j, \hat{w}_k) = \frac{P_{i,k}}{P_{j,k}} \tag{3.26}$$

It is possible to notice a problem in Equation 3.26: the arguments of function F are vectors, while the right-hand side is a scalar. This equation can be solved computing F with a neural network, but this would obfuscate the linear structure that the authors want to capture. To prevent that the vector dimensions are mixed in an undesirable way, they took the dot product of the arguments:

$$F((w_i - w_j)^T \hat{w}_k) = \frac{P_{i,k}}{P_{j,k}} \tag{3.27}$$

The next step is to include the symmetry of co-occurrence matrix X into the equation to model the possibility of swapping the two words, i.e. $X_{i,j} = X_{j,i}$. To do this in a consistently fashion, they have to make the model invariant not only to the exchange of $w_i \leftrightarrow w_j$, but also to $X \leftrightarrow X^T$. First, they created an homomorphism between the groups ($\mathbb{R}$, +) and ($\mathbb{R}_{>0}$, $\times$) as follows:

$$F((w_i - w_j)^T \hat{w}_k) = \frac{F(w_i^T \hat{w}_k)}{F(w_j^T \hat{w}_k)} \tag{3.28}$$

where function $F(\cdot)$ in the right-hand side is solved by

$$F(w_i^T \hat{w}_k) = P_{i,k} = \frac{X_{i,k}}{X_i} \tag{3.29}$$

The solution of Equation 3.28 is $F = \exp$:

$$w_i^T \hat{w}_k = \log(P_{i,k}) = \log(X_{i,k}) - \log(X_i) \tag{3.30}$$

Adding the bias of w_i and $\hat{w}_k$ into the above equation, the following equation is obtained:

$$w_i^T \hat{w}_k + b_i + \hat{b}_k = \log(X_{i,k}) - \log(X_i) \tag{3.31}$$

which will exhibit the exchange property if not for $\log(X_i)$. Then, they absorbed the term $\log(X_i)$ into b_i because the former is independent from the other terms. The resulting equation is then:

$$w_i^T \hat{w}_k + b_i + \hat{b}_k = \log(X_{i,k}) \tag{3.32}$$

Equation 3.32 represents the core engine of Glove because it is used to train the model, and consequently to generate the word embedding. The loss function follows:

$$L = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{i,k})(w_i^T \hat{w}_k + b_i + \hat{b}_k - \log(X_{i,k})) \tag{3.33}$$

where V is the vocabulary size, and $f(\cdot)$ is a function that satisfies the following properties:

- f(0) = 0;
- f(x) should be non decreasing in order to not overweight rare word co-occurrences;

- f(x) should be small for large values of x.

They found that the following function satisfies all the conditions:

$$f(x) = \begin{cases} (x/x_{max})^{\frac{3}{4}} & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases} \tag{3.34}$$

For the value of x_{max}, Pennington et al. empirically found that 100 gives the best results. Once the network is trained, the final word embedding of the i-th word of the vocabulary is obtained summing together the vector w_i and the vector $\hat{w}_i$.

3.6.5 FastText

According to Bojanowski et al. (2017), the issue of CBOW and SkipGram is that they ignore the subwords information (i.e., character n-grams) in the vector representation of a word. In their work, the authors decided to include the n-grams information in the softmax function (i.e., in the vector $\mathbf{u}_j$ - see Equation 3.14). Their idea is that distinct words have common character n-grams, and that their use could accelerate the learning of the model and improve the word embedding.

Their model first splits a word into character n-grams. For instance, given the word *rock* and an n-gram size of 3, the character n-grams are:

<ro, roc, ock, ck>

where $<$ and $>$ are two special tokens that represent the start and the end of a word. Then, the extracted n-grams are mapped to vectors and used to represent the word. Given S_k as the set of n-grams $\{g_1, g_2, \dots g_n\}$ of a input word w_k, the score $\mathbf{u}_j$ of a word w_j that appears in the same context of w_k is calculated as follows:

$$\mathbf{u}_j = \sum_{g \in S_k} \mathbf{v}_g^T \mathbf{v}'_j \tag{3.35}$$

where $\mathbf{v}'_j$ is the j-th row of the matrix $\mathbf{W}$.

3.6.6 SensEmbed

In *SensEmbed*, Iacobacci et al. (2015) fused CBOW and SkipGram models with BabelNet [Navigli and Ponzetto, 2010] senses. In detail, they first used Babelfy [Moro et al., 2014] as Word Sense Disambiguation algorithm to assign a sense to each word in the documents. For instance, given the sentence "*a fox is running on the hill*", the disambiguation algorithm will produce "*a* fox_1^n *is* $running_{36}^v$ *on the* $hill_1^n$", where in $hill_1^n$ the superscript represents the Part-Of-Speech tag (in this case noun) and the subscript

represents the first sense of the term (the id of the synset). Then, they generated the word embedding using word2vec.

In their experiments, they found that the disambiguated terms lead to a better representation of the word embedding. For instance, they found that the words closest to the financial sense of *bank* are: commercial bank, national bank, trust company, financial institution and banking.

3.6.7 ELMo

ELMo (Embeddings from Language Models) [Peters et al., 2018] is a language model that defines a deep context representation of a word, considering its syntactic role, and semantic and polysemy meaning. ELMo is composed by two Long Short Time Memory (LSTM) Neural Networks (see Section 3.7.3). The first LSTM, which reads the input sentence from left-to-right, produces a vector representation for each i-th token in sentence. Such vector contains the information of the actual token t_i and the previous context (i.e., $t_1, t_2, \ldots, t_{i-1}$ previous tokens). Then, each vector representation is used to predict the next word through a softmax layer, which produces a probability distribution over a defined vocabulary. The second LSTM reads the sequence from right-to-left and predicts the next word. The difference with the forward one is that the vector representation contains the future context of a i-th token (i.e., $t_{i+1}, t_{i+2}, \ldots$).

For the training of the network, the authors decided to jointly maximize the log likelihood of forward and backward directions. Let N be the length of a sentence (i.e., the number of tokens), the loss function of ELMo is defined as follows:

$$L = -\sum_{i=1}^{N}(\log p(t_i|t_1, \ldots, t_{i-1}; \overrightarrow{\Theta}) + \log p(t_i|t_{i+1}, \ldots, t_N; \overleftarrow{\Theta})) \tag{3.36}$$

where $\overrightarrow{\Theta}$ are the parameters of the forward LSTM and $\overleftarrow{\Theta}$ are the parameters of the backward LSTM. Figure 3.15 shows the architecture of ELMo.

3.6.8 BERT

Differently from ELMo, BERT [Devlin et al., 2018] uses a bidirectional self-attention [Vaswani et al., 2017] module to extrapolate the features to predict the next word. With more details, BERT transforms the sequence (i.e., a single sentence or two sentences packed together) in input adding two special tokens:

1. the token *[CLS]*: it is used to represent the start of the sentence. Further, the hidden state calculated on this token is used for classification tasks;

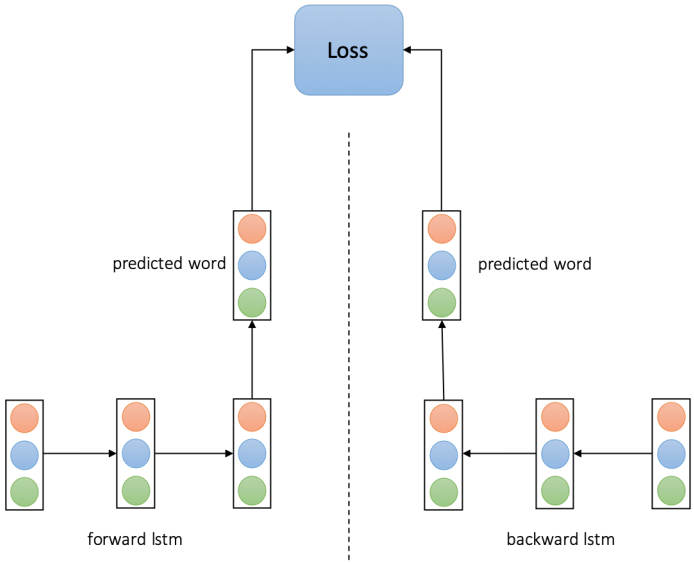


Figure 3.15: The architecture of ELMo is composed of two distinct LSTMs that generate a vocabulary distribution from which the next word is selected. The predicted words are then passed to the loss function.

2. the token *[SEP]*: it is used to separate each sentence in the sequence.

An example of BERT input follows:

[CLS] A man went to a computer store . [SEP] He bought an SSD . [SEP]

Then, each token (special ones included) are mapped to three embeddings: to a position embedding, to a token embedding and to a segment embedding. The latter embedding is used to distinguish to a which sentence a token belongs (e.g., the token t_i will have the segment embedding S_1 for the first sentence, and segment S_2 for the second one). The three embeddings representing the token are then fused together, and the self-attention layer is applied onto them. The training of the model is conducted in three steps: first, part of the input is masked using the special tag *[MASK]*; second, the model reads the context surrounding the token [MASK] and predicts a candidate that can replace the special token. Finally, the error of predicting the correct candidate is calculated and the weights are updated. Figure 3.16 shows the architecture of the model.

3.7 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a family of artificial neural networks that can process sequences of variable length. A sequence can be a sentence, a temporal data,

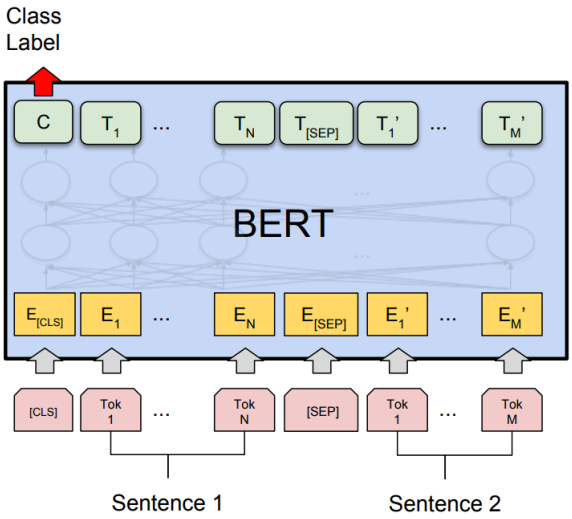


Figure 3.16: The picture, taken from [Devlin et al., 2018], shows the architecture of BERT.

or even the pixels of an image. RNNs can reuse the outputs of the previous timesteps, allowing them to be applied on sequence of different lengths. This sharing is important when a piece of information can occur in any position of the sequence and has to be evaluated according to the previous steps. Let us think about a verb in a sentence; we know that it appears after the noun phrase, but it does not have a fixed position since it depends on the length of this latter one.

More in detail, a RNN has a backward edge that allows to maintain an internal state, called *hidden state*. Such hidden state works as a *memory* for the network, allowing it to remember the previous read inputs. At each timestep, the network updates the hidden state, represented with the vector $\mathbf{h}$, using the information contained in the current input and the previous hidden state. Once the new hidden state is calculated, the network emits parts of its information in output, under the form of a vector $\mathbf{o}$. Equation 3.37 represents the RNN:

$$\begin{aligned} \mathbf{h}_t &= \tanh(\mathbf{U}\,\mathbf{x}_t + \mathbf{W}\,\mathbf{h}_{t-1} + b_h) \\ \mathbf{o}_t &= softmax(\mathbf{V}\,\mathbf{h}_t + b_o) \end{aligned} \tag{3.37}$$

where $\mathbf{U}$, $\mathbf{W}$ and $\mathbf{V}$ are the network parameters, and b_h and b_o are the biases. At timestep $t = 0$, h_0 is a zero vector. In the above equation, each member of the output is generated using the same rule of the previous output. This represents the parameter sharing.

3.7.1 Unfolding Recurrent Neural Networks

Section 3.4 described how Neural Networks can be represented formally as a computational graph. This can be done with Recurrent Neural Networks too, and it is called *unfold*. The generated graph has a repetitive structure that represents each timestep of the network, which typically corresponds to a chain of events. Figure 3.17 shows the unfolding of a RNN.

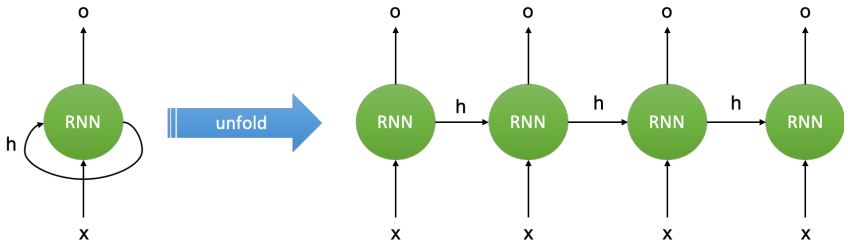


Figure 3.17: The image shows the unfolding of a Recurrent Neural Network.

To deeply understand the unfolding operation, and why the RNN has a repetitive structure, I defined the update of the memory vector $\mathbf{h}$ at a timestep t as:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t, \theta) \tag{3.38}$$

where θ contains the parameters of the network, and $\mathbf{x}_t$ is the t-th input. Equation 3.38 is recurrent (and this is why those networks are called RNNs) because the definition of $\mathbf{h}_t$ depends on the previous memory state $\mathbf{h}_{t-1}$. Thus, the graph can be unfold at time t by applying the equation $t-1$ times. For instance, if the memory state $\mathbf{h}$ is unfolded for $t = 4$, the result is:

$$\begin{aligned}
\mathbf{h}_4 &= f(\mathbf{h}_3, \mathbf{x}_3, \theta) \\
&= f(f(\mathbf{h}_2, \mathbf{x}_2, \theta), \mathbf{x}_3, \theta) \\
&= f(f(f(\mathbf{h}_1, \mathbf{x}_1, \theta), \mathbf{x}_2, \theta), \mathbf{x}_3, \theta) \\
&= f(f(f(f(\mathbf{h}_0, \mathbf{x}_0, \theta), \mathbf{x}_1, \theta), \mathbf{x}_2, \theta), \mathbf{x}_3, \theta)
\end{aligned}$$

Using the unfolded version of the Recurrent Neural Network, Equation 3.37 can be rewritten in a simplified version that highlights the connections between the hidden states of adjacent timesteps. The j-th parameter of the hidden state is calculated as:

$$\begin{aligned}
z_j^t &= \sum_{i=1}^{I} w_{ij} x_i^t + \sum_{k=1}^{H} w_{kj} h_j^{t-1} \\
h_j^t &= \varphi(z_j^t)
\end{aligned} \tag{3.39}$$

where $\varphi(\cdot)$ is the activation function (the tanh in our case), I is the number of input neurons, and H is the number of hidden neurons. Then, the j-th parameter of output layer is calculated as follows:

$$o_j^t = \sum_{l=1}^{O} w_{lj} h_j^t \tag{3.40}$$

where O is the number of output neurons.

With the unfold procedure, three types of Recurrent Neural Networks can be represented:

1. A RNN that emits an output for each timestep. This type of network is used either to tag each element of a sequence or to generate more complex features. For instance, it could be used to tag each word of a sentence with its corresponding Part-Of-Speech tag. The graph of this network is depicted in Figure 3.18;

2. A RNN that emits the output only when it reaches the last element of the sequence. This type of network is generally used to classify the entire sequence. The graph is depicted in Figure 3.19;

3. Finally, a RNN that does not emit any output. In this particular case, the last update of its memory is used as input of another network. For instance, Sequence-to-Sequence models (see Section 3.7.5) are defined using this architecture. The graph of this network is depicted in Figure 3.20.

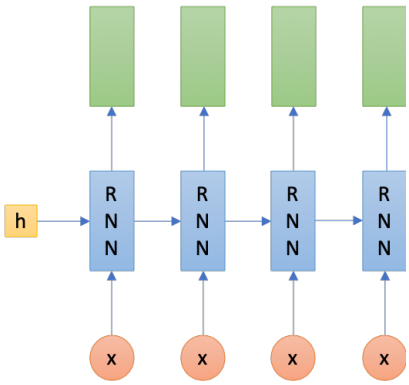


Figure 3.18: The image shows a RNN that emits an output at each timestep. The green rectangles represent the outputs, while the red circles represent the inputs.

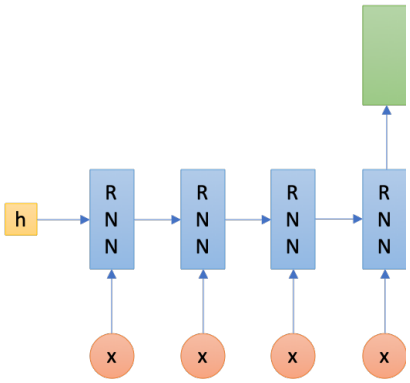


Figure 3.19: The image shows a RNN that emits an output only at the last timestep. The green rectangle represents the output of the network that contains the information of all inputs. The red circles represent the inputs.

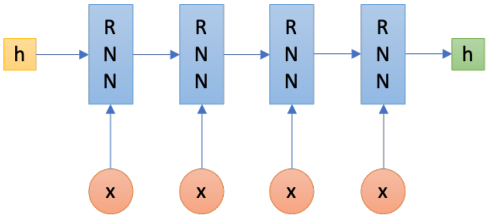


Figure 3.20: The image shows a RNN that does not emit any output, but it only updates the vector $\mathbf{h}$. The red circles represent the inputs.

3.7.2 Bidirectional Recurrent Neural Network

When I first presented the Recurrent Neural Networks, I said they read a sequence in order, from the first element to the last one. It is possible to imagine an extension of this model that includes another RNN that reads the sequence from right to left, i.e. from the last element of the sequence to the first one. The former network (i.e., the one that reads the sequence left-to-right) is called *forward RNN*, while the latter one (i.e., the right-to-left network) is called *backward RNN*. Generally, to distinguish the two type of readings, Equation 3.37 is modified as follows:

$$\begin{aligned} \overrightarrow{\mathbf{h}}_t &= \tanh(\mathbf{U}\,\mathbf{x}_t + \mathbf{W}\,\overrightarrow{\mathbf{h}}_{t-1} + b_h) \\ \overrightarrow{\mathbf{o}}_t &= softmax(\mathbf{V}\,\overrightarrow{\mathbf{h}}_t + b_o) \end{aligned} \tag{3.41}$$

for the forward RNN, and

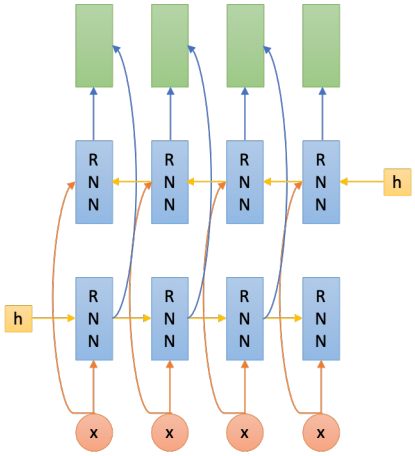


Figure 3.21: The figure shows a bidirectional RNN. The output of the forward RNN and the backward RNN are concatenated together. In the figure, the red circles represent the inputs, the blue rectangles represent each step of the network, and the green rectangles represent the output.

$$\begin{aligned} \overleftarrow{\mathbf{h}}_t &= \tanh(\mathbf{U}\,\mathbf{x}_t + \mathbf{W}\,\overleftarrow{\mathbf{h}}_{t-1} + b_h) \\ \overleftarrow{\mathbf{o}}_t &= softmax(\mathbf{V}\,\overleftarrow{\mathbf{h}}_t + b_o) \end{aligned} \tag{3.42}$$

for the backward RNN. Those equations can be represented in a compact way:

$$\begin{aligned} \overrightarrow{\mathbf{o}}_t &= \overrightarrow{RNN}(\mathbf{x}_t, \overrightarrow{\mathbf{h}}_t) \\ \overleftarrow{\mathbf{o}}_t &= \overleftarrow{RNN}(\mathbf{x}_t, \overleftarrow{\mathbf{h}}_t) \end{aligned} \tag{3.43}$$

It is possible to concatenate the outputs of the forward RNN with the outputs of the backward one. The resulting vectors will have access to the previous and future context of the sequence. Such type of architecture is called *bidirectional RNN*. Supposing to process a sequence of words, with the adoption of a bidirectional RNN the meaning of each word is defined according to the context generated reading all words that appear before and after the considered word. Figure 3.21 illustrates this neural network architecture.

3.7.3 Long-short Term Memory Networks

The Long Short Term Memory (LSTM) Network is a RNN developed by Hochreiter and Schmidhuber (1997). Such network uses two gates: an *input gate* and an *output gate*.

Those gates protect the memory content from perturbations of irrelevant inputs. The input gate is used by the network to decide when to keep or override the information in the memory; the output gate, instead, is used to select the information to emit in output. The role of the output gate is also to prevent the emission of a perturbed signal from the memory. In this way, error signals trapped in the memory cannot change. This is called by the authors *Constant Error Carousel*, because the error tends to not (excessively) vary at each step.

However, Hochreiter et al.'s LSTM has a drawback. It is not able to forget part of the information present in the memory, since it can only insert new information in it. This problem makes the network unfeasible to solve tasks where certain inputs appear repeatedly in the sequence. To fix that problem, Gers et al. (1999) introduced a forget gate that has the capability to erase part of the information stored in the memory.

The LSTM is represented in Equation 3.44:

$$\begin{aligned} i &= \sigma(\mathbf{U}_i \mathbf{x}_t + \mathbf{W}_i \mathbf{h}_{t-1} + b_i) \\ f &= \sigma(\mathbf{U}_f \mathbf{x}_t + \mathbf{W}_f \mathbf{h}_{t-1} + b_f) \\ o &= \sigma(\mathbf{U}_o \mathbf{x}_t + \mathbf{W}_o \mathbf{h}_{t-1} + b_o) \\ \tilde{\mathbf{c}}_t &= \tanh(\mathbf{U}_h \mathbf{x}_t + \mathbf{W}_h \mathbf{h}_{t-1} + b_h) \\ \mathbf{c}_t &= f \otimes \mathbf{c}_{t-1} + i \otimes \tilde{\mathbf{c}}_t \\ \mathbf{h}_t &= o \otimes \mathbf{c}_t \end{aligned} \tag{3.44}$$

where $\mathbf{c}$ is the memory of the LSTM. The symbol $\otimes$ represents the pointwise product. At timestep $t = 0$, both $\mathbf{c}_0$ and $\mathbf{h}_0$ are zero vectors.

3.7.4 Gated Recurrent Unit

Gated Recurrent Unit network, or GRU, is a type of Recurrent Neural Network developed by Cho et al. (2014). It differs from the LSTM by the number of gates used. GRU uses only two gates: the *reset* gate r, and the *update* gate z. The former gate is used to delete part of information from the previous state, keeping only the relevant ones for the current element. The latter gate is then used to combine the information coming from the past sequence with the ones extracted from the current element.

$$\begin{aligned} \mathbf{r}_t &= \sigma(\mathbf{U}\, \mathbf{x}_t + \mathbf{W}\, \mathbf{h}_{t-1} + b_r) \\ \mathbf{z}_t &= \sigma(\mathbf{U}\, \mathbf{x}_t + \mathbf{W}\, \mathbf{h}_{t-1} + b_z) \\ \tilde{\mathbf{h}}_t &= \tanh(\mathbf{U}\, \mathbf{x}_t + \mathbf{W}\, (r_t \otimes \mathbf{h}_{t-1}) + b_h) \\ \mathbf{h}_t &= \mathbf{z}_t \otimes \tilde{\mathbf{h}}_t + (1 - \mathbf{z}_t) \otimes \mathbf{h}_{t-1} \end{aligned} \tag{3.45}$$

The advantage of GRU is the reduced number of parameters (compared to LSTM), which allows the network to rapidly converge. As in the LSTM, $\mathbf{h}_0$ is a zero vector.

3.7.5 Sequence to Sequence model

Let $\mathbf{x} = [x_1, x_2, \ldots, x_M]$ be a sequence of M source tokens, and $\mathbf{w} = [w_1, w_2, \ldots, w_N]$ a sequence of N target tokens, with $N \leq M$. The Sequence to Sequence model, described in Sutskever et al. (2014), is used to transform the sequence $\mathbf{x}$ into the sequence $\mathbf{w}$. Such model belongs to the *encoder-decoder framework*, where the encoder is trained to create a vector representation of the sequence in input, and the decoder is trained to generate the output sequence using the vector representation. Generally, the encoder and the decoder are jointly trained to minimize the cross-entropy loss function.

Current networks are based on Bahdanau et al. (2014)'s model, i.e., a bidirectional LSTM Encoder and an unidirectional LSTM Decoder with attention mechanism. The Sequence-to-Sequence model is depicted in Figure 3.22.

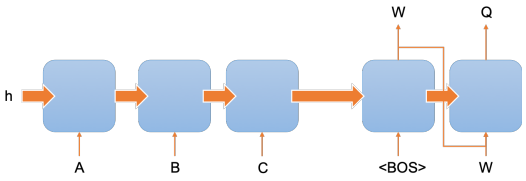


Figure 3.22: The figure represents a Sequence-to-Sequence model. Blue blocks represent states of the network, while orange arrows represent the passage of parameters from adjacent states.

Section 4.2 and Section 4.3 will present two new add-on to the model: a coverage mechanism that is used to fix some drawbacks of attention approach (e.g., focusing multiple times on the same word locations) and the pointer network, which allows to copy Out-Of-Vocabulary (OOV) words from the input sequence to the output one. Those mechanisms are widely used in Neural Networks for Abstractive Text Summarization.

Bidirectional Encoder

The encoder is composed by two LSTMs that read the input sequence $\mathbf{x}$ both in forward and in backward order, producing a sequence of encoder states $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_M]$. Each $\mathbf{h}_i$ is the result of the concatenation of i-th state of the forward LSTM with the i-th state of the backward LSTM:

$$\begin{aligned} \mathbf{h}_i &= [\overrightarrow{\mathbf{h}}_i || \overleftarrow{\mathbf{h}}_i] \\ &= [\overrightarrow{LSTM}(E(x_i), \overrightarrow{\mathbf{h}}_{i-1}) || \overleftarrow{LSTM}(E(x_i), \overleftarrow{\mathbf{h}}_{i-1})] \end{aligned} \tag{3.46}$$

where $||$ is the concatenate operator, and $E(\cdot)$ is the function that maps a word to its embed vector. For both the LSTM models, the initial state $\mathbf{h}_0$ is defined as a zero-vector. The last state calculated by the encoder is used as initial state by the decoder,

which will update it using the word emitted at the previous timestep. For timestep 1, since the decoder has not emitted any word, a special token <*BOS*> that represents the start of the sentence is used.

Decoder

The purpose of the decoder is to compute the conditional probability of the t-th word given the sequence of the previous generated ones, $y_{<t}$, and the sequence of the input tokens $\mathbf{x}$. In detail, the network computes a probability distribution over the vocabulary from which the most probable "next word" is selected.

$$p(y_t \mid y_{<t}, \mathbf{x}) = softmax(\mathbf{W}_{out}\, \tilde{\mathbf{h}}_t) \tag{3.47}$$

where $\mathbf{W}_{out}$ is a learnable parameter of the network. The vector $\tilde{\mathbf{h}}_t$ is computed extracting relevant features from the context vector $\mathbf{c}_t$, generated by the attention method, and the the current decoder state $\mathbf{s}_t$:

$$\tilde{\mathbf{h}}_t = tanh(\mathbf{W}_{\tilde{\mathbf{h}}}\, [\mathbf{c}_t \,||\, \mathbf{s}_t]) \tag{3.48}$$

where $\mathbf{W}_{\tilde{\mathbf{h}}}$ is a learnable parameters of the network.

The decoder state at timestep t is computed feeding the embedding of the previous emitted word and the previous decoder state to the LSTM:

$$\mathbf{s}_t = LSTM(E(y_{t-1}), \mathbf{s}_{t-1}) \tag{3.49}$$

Attention Approach

In the Sequence-to-Sequence network, $\mathbf{s}_t$ can only capture short-range dependencies since it has no knowledge of the whole input. To make the network able to capture long-range dependencies, the attention approach has been adopted. The attention approach dynamically composes a vector $\mathbf{c}_t$ that highlights salient passages and words of the input document. In detail, the current decoder state $\mathbf{s}_t$ and the sequence of encoded states $\mathbf{H}$ are used to calculate a score in the range $[0, 1]$ (with the constraint that the sum of the scores is equal to 1). Those scores can be interpreted as a probability distribution whereby states with an high value are the relevant ones. Then, the attention distribution and the states in $\mathbf{H}$ are combined together via a weighted sum to obtain the context vector $\mathbf{c}_t$. The following equations show how to compute $\mathbf{c}_t$ using Bahdanau et al. (2014)'s attention:

$$e_{t,j} = \mathbf{v}^T\, tanh(\mathbf{W}_a\, [\mathbf{h}_j \,||\, \mathbf{s}_t]) \tag{3.50}$$

$$a_{t,j} = \frac{exp(e_{t,j})}{\sum_{k=1}^{M} exp(e_{t,k})} \tag{3.51}$$

$$\mathbf{c}_t = \sum_{k=1}^{M} a_{t,k}\, \mathbf{h}_k \tag{3.52}$$

Bahdanau et al. (2014)'s equations are not the only way to define attention. Luong et al. (2015) defined and compared the above method with two other ones: *bilinear attention* and *global attention*. The bilinear attention uses a bilinear product to calculate the correlation between the current decoder step $\mathbf{s}_t$ and the encoder state $\mathbf{h}_j$:

$$e_{t,j} = \mathbf{s}_t^T \mathbf{W} \mathbf{h}_j \tag{3.53}$$

The global attention instead computes a dot product between $\mathbf{s}_t$ and $\mathbf{h}_j$:

$$e_{t,j} = \mathbf{s}_t^T \mathbf{h}_j \tag{3.54}$$

The authors also demonstrated that bilinear attention and Bahdanau et al.'s attention produced the best results for Machine Translation. Paulus et al. (2018) adopted the bilinear attention in their summarization model, obtaining outstanding results.

Chapter 4

New Frontiers on Neural Text Summarization

4.1 Introduction

Recurrent Neural Networks (and Neural Networks in general) allowed to improve the task of Text Summarization, from understanding the document content to generating more abstractive summaries. But they also posed new challenges to researchers. On my opinion, Neural Networks suffer from three drawbacks:

Repetitions in output: those networks have no memory about the previous generated words, thus they tend to repeat words (and sometimes phrase excerpts) in output, disrupting the quality of the generated summary. This has been reported by Ranzato et al. (2016) and Kiyono et al. (2017), and several researchers have started to propose coverage methods to avoid repetitions in the output. The idea of those works is to capture the previous generated words and use them to guide the network;

Out-Of-Vocabulary words: One of the problem of Neural Networks, as other Machine Learning algorithms, is the fixed number of classes that can be predicted. In other words, it is not possible to add new classes to a trained model. Since in NLP tasks those classes could be words, this means that the model cannot generate new ones. This problem takes the name of *Out-Of-Vocabulary words* because those words are not present in the vocabulary of the model, i.e. the classes to predict. It is also accentuated in Text Summarization, since the task is a Recall-oriented one which aims to have all the relevant words of the document in the output summary. Thus, how can this problem be bypassed? Researchers defined a family of Neural Networks, called *Pointer Networks*, that have the ability to copy words from the input to the output. The application of those networks to the Summarization task

resolved this issue.

Content indifferent: Neural Networks are not capable of deeply understand the content of a document, i.e. they are not able to distinguish salient sentences from those that are secondary for the context, treating them equally. Such problem creates summaries that are correct at syntactic and semantic level, but not related to the document topic. In this context, researchers tried different methods to capture the salience via masks, gates, and attention. Others used template-based methods.

In this chapter, I describe the methods present in literature to solve those three drawbacks. To the best of my knowledge, they form a wide and complete view of the topic.

The remain of this chapter is formed as follows: Section 4.2 presents the coverage problem, while Section 4.3 covers the Pointer Network models. Finally, Section 4.4 describes the recent works on content selection.

4.2 Coverage Methods

The adoption of the attention approach has brought some drawbacks, i.e., duplicated words and sentences in the output. Researchers referred to this unwanted behaviour with two different names:

Lack of coverage *In Machine Translation*. Since the model has no memory about previous attention distributions, it tends to focus multiple times on the same word or source text locations. Researchers have found that a generated output may contain up to 8 repetitions of the same word or sentence [Sankaran et al., 2016];

Odd-generation *In Text Summarization*. Kiyono et al. (2017) reported the same problem.

According to Ranzato et al. (2016), this problem is accentuated by two factors:

- exposure bias: the model is only exposed to train data distribution, and not to its predictions; in other words, during training we feed the decoder with the previous target word, while in testing we feed it with the previous output emitted by the decoder;

- no sequence-level loss: the training is performed computing the loss at word-level, checking if the predicted word corresponds to the target one, while evaluation is performed at sequence level. Thus, the network generates correct single words, but the sentence in its whole may be erroneous.

To solve this problem, *coverage methods* proposed by Koehn (2009) for Statistical Machine Translation have been adapted to Sequence-to-Sequence models. The original

method used a vector of the same size of the input sequence. The vector is initialized with zeros to represent that words are not covered, $\mathbf{C} = \{0_{x_1}, 0_{x_2}, \ldots, 0_{x_n}\}$. Then, it is updated at each decoder step to keep trace of the translated words, changing the coverage value of a translated word into 1. For instance, suppose that x_2 is translated; the coverage vector is updated as follows: $\mathbf{C} = \{0_{x_1}, 1_{x_2}, \ldots, 0_{x_n}\}$. The translation ends when the vector contains only 1s, $\mathbf{C} = \{1_{x_1}, 1_{x_2}, \ldots, 1_{x_n}\}$. However, the original method cannot be applied to Neural Networks because it is difficult to know which input words has been translated by the decoder. Furthermore, the case is more complex for Text Summarization, where encoder words may be used multiple times. A solution is to use the attention distribution, but it could contain erroneous scores. Thus, *How could the coverage vector be represented? How could the coverage vector be updated?* Mi et al. (2016) and Tu et al. (2016) proposed to represent the coverage vector as a *memory* that is updated and read by the model. The idea is to associate a vector, belonging to $\mathbb{R}$, to each word that describes (for the network) if it has been covered. Let $\mathbf{cov}_{t,j}$ be the coverage vector for j-th word at timestep t. It is used to guide the attention approach to explore different parts of the input sequence and generates correct translations or summaries. Equation 3.50 is changed as follows:

$$e_{t,j} = \mathbf{v}^T \, tanh(\mathbf{W}_a \, [\mathbf{h}_j \,||\, \mathbf{s}_t \,||\, \mathbf{cov}_{t,j}]) \tag{4.1}$$

In this section, I report the different coverage methods proposed for Neural Network models. Those methods can be logically divided into:

based on sum of attention scores: I collocated here all those models that use the sum of all previous generated attention distribution to compute the coverage vector. It is possible to find the *language coverage* proposed by Tu et al. (2016), where the sum of previous attention scores is used to capture the coverage degree of a word; the method proposed by See et al. (2017), which is a simplified version of Tu et al. coverage; the method proposed by Chen et al. (2016b), where the sum of the previous attention scores is subtracted from the attention energy e to force the network to explore other words; and the model proposed by Sankaran et al. (2016) where the sum of pre-normalized attention scores is divided by the current one in order to re-balance the weights according to the previous attended words.

based on RNN: I collocated all those models that use a Recurrent Neural Network to update the coverage vector of a token. Here, there are the work of Mi et al. (2016) and Tu et al. (2016) where the coverage vector is used as an internal state, updated every time the attention distribution is generated in output.

based on embeddings: I collocated those models that represent coverage as embedded vectors, learned by the network. Such embedded vectors are update at each decoder step to account previous attended words. The work of Mi et al. (2016) belongs to this topic.

Figure 4.1 depicts the logical division.

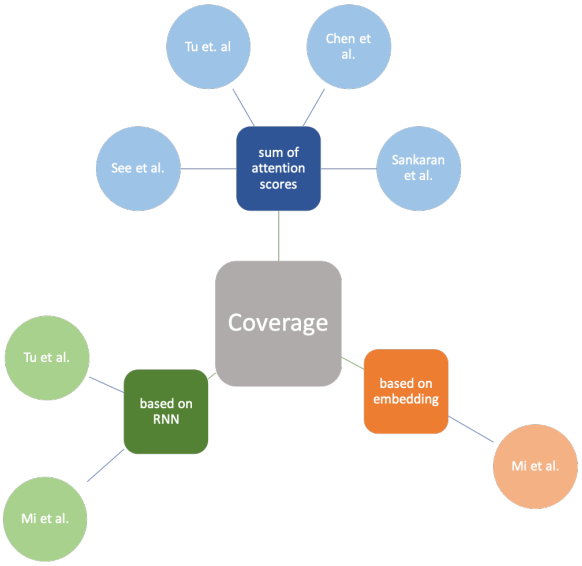


Figure 4.1: The figure resumes all coverage works, dividing them by their typology.

4.2.1 Tu et al.'s

Tu et al. (2016) propose two methods to compute the coverage vector: *language coverage* and *neural-based coverage*. The difference between the two methods regards how the coverage vector is updated.

In language coverage, described in Equation 4.2, the coverage vector of x_j is a scalar that contains the sum of previous word attention scores, divided by a weight Φ_j that indicates the number of target words x_j is expected to generate.

$$cov_{t,j} = \frac{1}{\Phi_j} \sum_{t'=1}^{t-1} a_{t',j} \tag{4.2}$$

In detail, Φ_j is the *fertility* of x_j (simplified and adapted from the statistical original one) and it is calculated as follows:

$$\Phi_j = N\, \sigma(\mathbf{W}_f\, \mathbf{h}_j) \tag{4.3}$$

where N, sets to 2, is a constant that denotes the maximum number of target words that a source word can generate, and $\sigma(\cdot)$ is the sigmoid function. The idea is to change the value of N according to the source word. At timestep 1, $cov_{t,j}$ is initialized to 0.

Differently from the language coverage, the neural-based coverage uses a Gated Recurrent Unit (GRU) [Cho et al., 2014] to update the coverage vector of $\mathbf{x}_j$ word:

$$cov_{t,j} = GRU(cov_{t-1,j}, a_{t-1,j}, \mathbf{h}_j, \mathbf{s}_t) \tag{4.4}$$

In this model, $cov_{t,j}$ is treated as the hidden state of the GRU network, which is used to capture long dependencies, removing old information from $cov_{t-1,j}$ and adding the new ones provided by $a_{t-1,j}$, $\mathbf{h}_j$ and $\mathbf{s}_t$.

Tu et al. (2016) also experimented with an auxiliary loss function (for the language coverage) that penalizes the discrepancies between the sum of attention scores of a word and its expected fertility:

$$auxloss = \sum_{j=1}^{M}(\Phi_j - \sum_{t=1}^{N} a_{t,j})^2 \tag{4.5}$$

Their empirical studies showed that the auxiliary loss worsen the translations.

4.2.2 Mi et al.'s

Mi et al. (2016) propose a different coverage vector. Since each source word has a different fertility (the number of target words that a source one could generate), they decided to create a coverage embedding that is updated at each decoder timestep. They present two methods to update the coverage embedding of a word: through *subtraction* or through a GRU. In subtraction, the coverage embed is updated subtracting the embed of the previous generated word, weighted by the attention score:

$$cov_{t,j} = cov_{t-1,j} - a_{t-1,j}\, \mathbf{W}_s\, y_{t-1} \tag{4.6}$$

where $\mathbf{W}_s$ is learnable parameter that maps the word embed to the size of the coverage embed.

The idea of Equation 4.6 is that the embedding of y_{t-1} implicitly carries the words used to predict it. Furthermore, since each source word can contribute differently to the generation of y_{t-1}, the authors weighted the value by the attention score of x_j, which works as a sort of gate.

The GRU update, instead, is similar to the one proposed by Tu et al.:

$$cov_{t,j} = GRU(cov_{t-1,j}, a_{t-1,j}, y_{t-1}) \tag{4.7}$$

The difference is that Tu et al. initialized the coverage vector with zeros, while in Mi et al. it is learned by the network.

The authors also used an L2 regularization on the coverage vector, forcing it to be close to zero:

$$auxloss = \sum_{t=1}^{N} \sum_{j=1}^{M} ||cov_{t,j}|| \tag{4.8}$$

4.2.3 See et al.'s

See et al. (2017) propose a coverage model for Neural Abstractive Summarization that resembles Tu's language model. In details, they removed the parameter Φ from Equation 4.2, defining the coverage of word x_j as the sum of its previous attention values:

$$cov_{t,j} = \sum_{t'=1}^{t-1} a_{t',j} \tag{4.9}$$

The authors also introduced an auxiliary loss to penalize overlapping values between the coverage vector and the attention distribution at each decoder timestep t:

$$auxloss = \sum_{t=1}^{N} min(cov_{t,.}, a_{t,.}) \tag{4.10}$$

where $cov_{t,.}$ is the coverage vector of all input words and $a_{t,.}$ is the attention distribution. Penalizing overlapping values forces the network to focus on different encoder words.

Differently from the other coverage models, they introduced two changes:

- they didn't applied the coverage vector at the first decoder step (since the vector contains only zeros);
- they applied the auxiliary loss on the trained model.

4.2.4 Chen et al.'s (distraction method)

Distraction method is another coverage method for Neural Abstractive Summarization proposed by Chen et al. (2016b). The name *distraction* born from the fact that the authors try to distract the network by subtracting the history of the previous attention distributions and context vectors. In this way, the network is forced to explore different encoded words. Their coverage method is composed of two parts: *distraction in training* and *distraction in decoding*. In *distraction in training*, the authors construct an history vector for the attention distribution and the context vector, that simply are the sum of the their values at previous decoder timesteps. Then, they subtract the history vectors in the calculation of attention distribution and context vector for the current timestep:

$$
\begin{aligned}
\mathbf{c}\_hist_t &= \sum_{t'=1}^{t-1} \mathbf{c}_{t'} \\
attn\_hist_t &= \sum_{t'=1}^{t-1} a_{t',\cdot} \\
e_{t,j} &= v^T \, tanh(\mathbf{W}_a \, [\mathbf{h}_j \, || \, \mathbf{s}_t] - \mathbf{U}_a \, attn\_hist_{t,j}) \\
\mathbf{c}_t &= tanh(\mathbf{V}_c \, \mathbf{c}_{t'} - \mathbf{U}_c \, \mathbf{c}\_hist_t)
\end{aligned}
\tag{4.11}
$$

where $\mathbf{c}_{t'}$ is calculated by Equation 3.52. In *distraction in decoding*, the authors introduce three auxiliary loss functions, that are weighted by three different λ values, to distract the decoder. First, the loss function calculates a score comparing the context vector (or the attention distribution, or the decoder state) at the current timestep with the previous ones; then, they select the best score among the calculated ones:

$$
\begin{aligned}
d_{a,t} &= \min_{1 \leq i \leq t-1} KL(a_t, a_i) \\
d_{c,t} &= \max_{1 \leq i \leq t-1} cosine(c_t, c_i) \\
d_{s,t} &= \max_{1 \leq i \leq t-1} cosine(s_t, s_i)
\end{aligned}
\tag{4.12}
$$

where $d_{a,t}$ is the auxiliary loss for the attention distribution that uses Kullback-Leibler (KL) divergence to compare the probability distributions[1], $d_{c,t}$ is the auxiliary loss for the context vector and $d_{s,t}$ is the auxiliary loss for the decoder state. The total loss is then rewrote as follows:

$$
loss = \sum_{t=1}^{T} loss_t + \lambda_1 d_{a,t} + \lambda_2 d_{c,t} + \lambda_3 d_{s,t} \tag{4.13}
$$

4.2.5 Sankaran et al.'s (temporal attention)

Temporal attention method [Sankaran et al., 2016] is different from the other coverage models presented so far. It could be seen as a temporal network that memorizes previous decisions and uses them to explore different parts of the input text. In temporal attention, the sum of the previous attention scores is memorized and used to model the scores e. Let t be the current timestep, the sum of the previous attention distributions $b_{t,j}$ of word x_j, called *history*, is defined as follows:

[1] Lower value is better.

$$b_{t,j} = \sum_{t'=1}^{t-1} \exp(e_{t',j}) \tag{4.14}$$

The history b_t is then used to modify the score of $e_{t,j}$:

$$e_{t,j} = \frac{\exp(e_{t,j})}{b_{t,j}} \tag{4.15}$$

The remaining steps are the same of Section 3.7.5. Temporal attention substantially differs from the previous coverage methods for the following reasons:

- It uses no gates to update the coverage vector, as the ones proposed in [See et al., 2017; Chen et al., 2016b];
- It does not add further learnable parameters to the network;
- It does not require an auxiliary loss function.

Finally, temporal attention can be applied to both Neural Network-based Machine Translation [Sankaran et al., 2016] and Text Summarization [Paulus et al., 2018].

4.2.6 Comparison between methods

So far, I described different coverage methods that has been created for Neural Machine Translation and Text Summarization. In this section, I will summarize those methods through the help of a table, reporting (where it is possible) the results that they obtained on the same dataset.

Table 4.1 reports each method, its authors, if it introduces further parameters (i.e., if the context vector is used to compute the attention score - see Equation 4.1), and if it requires an auxiliary loss function. With dimension $d = 1$, I mean that the coverage value of word x_j is represented using a scalar.

I also reported the results of Tu et al.'s methods, Mi et al.'s methods and Sankaran et al.'s method on machine translation NIST datasets MT05, MT06, and MT08. Table 4.2 shows the Bleu score obtained by the methods on that dataset[2].

From the table, it is possible to see that *LVNMT+GRU+subtraction+loss* achieves the best BLEU score (36.80). However, GRU coverage and subtraction coverage introduce a lot of parameters to the network, making it subjects to overfit, especially in case of small datasets. In this case, temporal attention seems the best choice, since it does not introduce further parameters.

I have not been able to compare the coverage methods for the Neural Abstractive Summarization task because the models that use them have been trained and tested on different datasets.

[2]The scores are taken from the authors' research articles.

cov. method	authors	vect. size d	params?	auxloss?
language coverage	Tu et al.	$d = 1$	yes	no
neural-based coverage	Tu et al.	$d = 1$ or $d = 10$	yes	no
subtraction	Mi et al.	$d = 100$	yes	yes
GRU update	Mi et al.	$d = 100$	yes	yes
See's coverage	See et al.	$d = 1$	yes	yes
distraction	Chen et al.	$d = 1$	yes	yes
temporal attention	Sankaran et al.	$d = 1$	no	no

Table 4.1: The table summarizes the methods presented in Section 4.2. In the table, *cov. method* describes the coverage method name, *vect. size* d represents the size of the coverage vector, *params* describes if the method introduces further parameters to the model, and *auxloss* describes if the coverage model uses an auxiliary loss function.

Model	MT05	MT06	MT08-news	MT08-web
GroundHog	30.61	31.12	-	-
+ language with fertility [Tu et al., 2016]	32.36	32.31	-	-
+ neural-based [Tu et al., 2016]	32.73	32.47	-	-
LVNMT	-	34.53	28.86	26.78
+ temporal attention [Sankaran et al., 2016]	-	36.34	31.49	27.29
+ GRU [Mi et al., 2016]	-	35.59	30.18	27.48
+ subtraction [Mi et al., 2016]	-	35.90	30.49	27.63
+ GRU + subtraction + loss [Mi et al., 2016]	-	36.80	31.83	28.28

Table 4.2: The table reports the Bleu score on NIST datasets MT05, MT06 and MT08 applying the coverage methods. High value is better.

4.3 Copying Out-Of-Vocabulary Words

In Section 3.7.5, I said that the network computes a probability distribution over the vocabulary, using a softmax function, to emit in output a word. This approach suffers of two drawbacks:

1. the softmax function is the bottleneck of the network, since it is computed on a fixed size vocabulary and it requires a lot of time to be executed;

2. even with large vocabularies, some words will be excluded. This is in part caused by a delexicalization process, where only the most N frequent words are used,

discarding the others (e.g., very rare words), and in part by the constant creation of new terms. For instance, people tend to created new nouns every day.

How is it possible to use words that are outside of our vocabulary? The idea is to use a network to point words in the input and copy them to the output. Such network, called *pointer network* [Vinyals et al., 2015], computes a probability distribution over the input that is used to select the word (or words) having the highest probability. More in detail, the authors used the attention approach to compute a probability distribution over the input elements. Then, they selected the one having the highest probability. Figure 4.2 depicts an example of a pointer network over a small text.

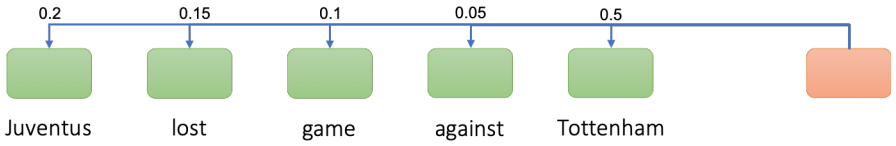


Figure 4.2: The figure shows an example of pointer network over the sentence "*Juventus lost game against Tottenham*". The red rectangle is the decoder state which selects a word from the sentence. In this example, the pointer network will select the word "*Tottenham*".

The pointer network has been widely used in Neural Machine Translation [Gulcehre et al., 2016], Text Summarization [Miao and Blunsom, 2016; Zeng et al., 2017; Nallapati et al., 2016; Gu et al., 2016; See et al., 2017; Paulus et al., 2018], Code Suggestion [Bhoopchand et al., 2017] and Language Model [Merity et al., 2017]. The described pointer networks can be easily switched one another since they are all variants of the original one. For instance, Chen and Bansal (2018) adopted a variant of the sparse pointer network proposed in [Bhoopchand et al., 2017] to select sentences, while Paulus et al. (2018) adopted, with small changes, the pointer sentinel defined by Merity et al. (2017). In this section, I describe several pointer networks that have been created, from the the original model of Vinyals et al. (2015) to the recent ones. Those models widely differ one another: some models simply use the attention distribution to select a word to copy, while others create a mixture model combining the vocabulary distribution with the pointer network distribution. Table 4.3 digests all the presented methods. However, they are just a small subset of all research works conducted on this topic, but, to the best of my knowledge, they form a widely background that allows the reader to comprehend the latest ones.

Authors	Year	Description
Vinyals et al.	2015	Attention distribution used to select an input element to copy.
Nallapati et al.	2016	The network uses a switch to decide when to copy and when to generate. The pointer network is simply an attention distribution over the input.
Gu et al.	2016	Probability of a word defined combining the vocabulary distribution with the attention distribution. There is no gate that controls the merge of the two distributions, but a shared normalization factor Z.
Miao and Blunsom	2016	Attention distribution over the input used to select the word to copy in output.
Gulcehre et al.	2016	They define the probability of a word combining the vocabulary distribution with the attention distribution. Both distributions are weighted by a score.
Zeng et al.	2017	Attention distribution used to select the word to copy. If the copied word is outside of the vocabulary, its embedding is created using the corresponding encoder state.
Bhoopchand et al.	2017	They used a sparse Pointer Network to copy only certain words from the input.
Merity et al.	2017	Mixture model between the vocabulary distribution and the Pointer Network distribution. The contribution of each distribution is controlled by a gate.
See et al.	2017	Similar to [Merity et al., 2017], they create a mixture model combining the vocabulary distribution with the attention distribution.
Paulus et al.	2018	They propose a model similar to [See et al., 2017; Merity et al., 2017]. The main difference is that they do not sum the attention scores of all identical words.

Table 4.3: The table summarizes the several Pointer Network models described in this section.

4.3.1 Merity et al.'s

Merity et al. (2017) integrated the pointer network in a Recurrent Neural Network (RNN) language model to copy a word of the sequence that is outside of the vocabulary. In their model, the RNN reads N-1 words in a sequence and predicts the N-th word, using the state $\mathbf{h}_{N-1}$. Those latter states, represented with $\mathbf{h}_i$, are then saved in order to compute the probability distribution. In detail, the current RNN state $\mathbf{h}_{N-1}$ is used to compute the query q

$$\mathbf{q} = \tanh\left(\mathbf{W}\ \mathbf{h}_{N-1} + b\right) \tag{4.16}$$

which is then multiplied with the previous RNN state $\mathbf{h}_i$ to select a word in the sequence

$$\mathbf{a} = softmax([\mathbf{q}^T\mathbf{h}_i||\mathbf{q}^T\mathbf{s}]) \tag{4.17}$$

In Equation 4.17, $\mathbf{q}^T\mathbf{s}$ is used to create a mixture model combining the vocabulary distribution with the pointer network distribution, where $\mathbf{s}$ is the *sentinel*, a vector which values are learned by the model. For the vector, their model takes the name of *pointer sentinel*. Since the last value of the softmax (the sentinel) is extracted to compute the mixture model, the other ones are scaled using the following formula:

$$\hat{a}_j = \frac{1}{1 - a_j}\mathbf{q}^T\mathbf{s} \tag{4.18}$$

Finally, given the extracted value as g (a scalar), the probability of a word y_i in output can be computed as follows:

$$p(y_i|\mathbf{y}_{<i}, \mathbf{x}) = g \cdot p_{vocab}(y_i|\mathbf{y}_{<i}, \mathbf{x}) + (1 - g) \cdot \sum_{j \in I(y_i)} \hat{a}_j \tag{4.19}$$

where $\mathbf{x}$ is the input sequence and $I(\cdot)$ is the function that returns the index of all words equal to y_i.

4.3.2 Zeng et al.'s

Zeng et al. proposed an interesting idea in their paper [Zeng et al., 2017]. They say that people tend to read again the document to summarize: the first time is to understand the content of the document, while the second time is to extract relevant information. Thus, they implemented a re-read mechanism. First, they read a sequence of words $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ using a RNN (LSTM or GRU), producing a set of encoded states $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n]$. Then, such states are read by another RNN that produces a sequence $\mathbf{H}' = [\mathbf{h}'_1, \mathbf{h}'_2, \ldots, \mathbf{h}'_n]$. The last state of the second encoded sequence is used as input for the decoder. They also extended the model with a pointer network that

uses the attention distribution to copy an input word into the summary. Since many words copied by the pointer network are not present in the vocabulary, they defined an innovative way to create an embedding vector for such words. Supposing that x_i is the copied word, and that it is not present in the vocabulary, the embedding of x_i is created as follows:

$$E(x_i) = \tanh\left(\mathbf{W}\ \mathbf{h}'_i + b\right) \tag{4.20}$$

where $E(\cdot)$ is the function that maps a word to its embedded vector.

4.3.3 Bhoopchand et. al's

Bhoopchand et al. (2017) implemented a neural network to propose code suggestions based on the existing RNN language models. Their model, an attention-based RNN language model, reads a snipped of code and suggest the next sequence of tokens. It uses a sparse pointer network to copy words representing function identifiers, class identifiers, and so on from the input to the output. Their method is called sparse because they obtain a pseudo-sparse distribution removing some words. Given $\mathbf{m}$ as a vector of ids for the identifiers and the attention score vector $\mathbf{e}$ computed by the network, they produce the attention distribution through the following formula:

$$\begin{aligned} s_i &= \begin{cases} e_j & \text{if } m_j = i \\ -C & \text{otherwise} \end{cases} \\ \mathbf{a} &= softmax(\mathbf{s}) \end{aligned} \tag{4.21}$$

where C is a large constant. The obtained distribution $\mathbf{a}$ is used to select the word to copy. Finally, similar to Merity et al. (2017), they created a mixture model summing the vocabulary distribution with the pointer network distribution. The contribution of the two distributions is regulated by a parameter λ, which is computed by the network. A variation of this model can be found in Section 4.4.

4.3.4 Miao et al.'s

Miao and Blunsom (2016) propose an auto-encoder network for the task of sentence compression, where the goal is to compress a sentence removing irrelevant words from it. Their network is composed of three parts: an encoder, a BiLSTM that encodes the sentence in input; a compressor, a LSTM that creates the compressed version of the sentence using the pointer network; and a decoder, a LSTM that takes in input the compressed sentence and produces the input sentence. In detail, the pointer network computes an attention vector α over the input using the encoder states and the current compressor state. Then, the next word is selected from α.

Since the auto-encoder is not enough to generate a coherent compressed sentence, they decided to introduce a forced-attention model, which simply decides either to copy a word from the input or to generate one from the vocabulary. The improvement is obtained sharing the pointer-network of the auto-encoder with the one of the forced-attention, where the latter one is trained on a small dataset of labeled sentence-compression pairs. Finally, the network is trained using reinforcement learning due to the non-derivability of the model.

4.3.5 Gu et al.'s

Gu et al. (2016) propose a Sequence to Sequence model for Text Summarization that uses the pointer network to copy words that are outside of the vocabulary. Their idea is to use a probability value (a sort of switch) to choose either to copy a word from the document in input or to generate one picking it from the vocabulary. Initially, they define two sets that are used to compute the probabilities: the set of words in the vocabulary V, which is extended with the token *UNK* for the words that are outside the vocabulary; and the set of unique words in the document X. They also define the set of words outside the vocabulary with $\overline{V}$. Then, given the previous generated sequence $y_{<i}$ and the input sequence x, the probability of generating the word y_i is calculated as:

$$p(y_i|\mathbf{y}_{<i}, \mathbf{x}) = p(y_i, g|\mathbf{y}_{<i}, \mathbf{x}) + p(y_i, c|\mathbf{y}_{<i}, \mathbf{x}) \tag{4.22}$$

where $p(y_i, g|\cdot)$ is the probability of selecting a word from V:

$$p(y_i, g|\cdot) = \begin{cases} \frac{1}{Z} e^{\psi_g(y_i)} & \text{if } y_i \in V \\ 0 & \text{if } y_i \in X \cap \overline{V} \\ \frac{1}{Z} e^{\psi_g(unk)} & \text{if } y_i \notin V \cup X \end{cases} \tag{4.23}$$

and $p(y_i, c|\cdot)$ is the probability to copy a word from the document:

$$p(y_i, c|\cdot) = \begin{cases} \frac{1}{Z} \sum_{j \in I(y_i)} e^{\psi_c(x_j)} & \text{if } y_i \in X \\ 0 & \text{otherwise} \end{cases} \tag{4.24}$$

and Z is the normalization term shared between the two models. The model resembles the one proposed in [Merity et al., 2017], since both combines the two probabilities: generation and copying. The main difference is that Merity et al. control the contribution of both probability distributions directly via a gate g, while Gu et al. use the normalization term Z.

Both the function ψ_c and ψ_g are computed by the network, which uses the encoder state $\mathbf{h}$ and the decoder state $\mathbf{s}$:

$$\begin{aligned}\psi_g(y_i) &= \mathbf{v}_i^T \, \mathbf{W}_g \, \mathbf{s}_i \\ \psi_c(y_i = x_j) &= \sigma(\mathbf{h}_j^T \, \mathbf{W}_g)\mathbf{s}_i\end{aligned} \tag{4.25}$$

where $\mathbf{v}_i$ is the one-hot vector for the i-th word in the vocabulary.

4.3.6 Gulcehre et al.'s

Gulcehre et al. (2016) propose a Sequence to Sequence model for Text Summarization and machine translation that uses a pointer network to copy words that are outside of the vocabulary. In detail, their model uses two type of softmax: a shortlist softmax that is computed over the vocabulary, and the location softmax that is the pointer network. In the following equations, I use p_s for the shortlist softmax and p_l for the location softmax. Their model calculates the probability of a output sequence $\mathbf{y}$ as follows:

$$\begin{aligned}p(\mathbf{y}, \mathbf{z}|\mathbf{x}) = &\prod_{i \in T} p_s(y_i, z_i, |\mathbf{y}_{<i}, \mathbf{x}) \times \\ &\prod_{i \in T} p_l(y_i, z_i|\mathbf{y}_{<i}, \mathbf{x})\end{aligned} \tag{4.26}$$

where $\mathbf{x}$ is the encoded input and T is the set of timesteps. The two probabilities are computed as follows:

$$\begin{aligned}p_s(y_i, z_i = 1|\mathbf{y}_{<i}, \mathbf{x}) = &p_s(y_i|z_i = 1, \mathbf{y}_{<i}, \mathbf{x}) \times \\ &p(z_i = 1|\mathbf{y}_{<i}, \mathbf{x})\end{aligned} \tag{4.27}$$

$$\begin{aligned}p_l(y_i, z_i = 0|\mathbf{y}_{<i}, \mathbf{x}) = &p_l(y_i|z_i = 0, \mathbf{y}_{<i}, \mathbf{x}) \times \\ &p(z_i = 0|\mathbf{y}_{<i}, \mathbf{x})\end{aligned} \tag{4.28}$$

The probability $p(z_i|\cdot)$ is called *switch network* and it is used to switch between the shortlist softmax and the location softmax. Such switch network is different from the one proposed by Nallapati et al. (2016), since it does not force the network to just copy or generate. It is similar to the ones proposed by Merity et al. (2017) and See et al. (2017), where the switch is used to weight the two distribution. The probability of the switch network is calculated as follows:

$$\begin{aligned}p(z_i = 1|\mathbf{y}_{<i}, \mathbf{x}) &= \sigma(f(\mathbf{x}, \mathbf{h}_{i-1})) \\ p(z_i = 0|\mathbf{y}_{<i}, \mathbf{x}) &= 1 - \sigma(f(\mathbf{x}, \mathbf{h}_{i-1}))\end{aligned} \tag{4.29}$$

where $f(\cdot)$ is a multi-layer perceptron, $\mathbf{x}$ is the input sequence, and $\mathbf{h}_{i-1}$ is the previous output of the multi-layer perceptron.

4.3.7 Nallapati et al.'s

Nallapati et al. (2016) defined a decoder extended with a pointer network to copy words from input to the output. The decoder uses a switch to decide either to select a word from the vocabulary or to copy a word from the input. If the switch is turned on, the decoder produces a word in the normal fashion (selecting it from the vocabulary); otherwise, it uses the attention distribution over word positions in the document. The activation of the switcher is defined as follows:

$$s = \sigma(\mathbf{v} \cdot (\mathbf{W}\mathbf{h}_i + \mathbf{U}\mathbf{y}_{i-1} + \mathbf{V}\mathbf{c}_i + b)) \tag{4.30}$$

where $\mathbf{h}_i$ is the current decoder state, $\mathbf{y}_{i-1}$ contains the embedding of the previous emitted word, and $\mathbf{c}_i$ is the context vector.

In case that the switch is not active, the network selects the word with the highest probability from the attention distribution:

$$\begin{aligned} e_{i,j} &= exp(\mathbf{v} \cdot (\mathbf{W}\mathbf{h}_{i-1} + \mathbf{U}\mathbf{y}_{i-1} + \mathbf{V}\mathbf{h}_i^e + b)) \\ p_i &= \max_j \ e_{i,j} \end{aligned} \tag{4.31}$$

where $\mathbf{h}^e$ is the encoder hidden state, and p_i is the probability to copy the i-th word.

4.3.8 See et al.'s

Differently from previous research works, See et al. (2017) propose a very simple but efficient pointer network. Their idea is to treat the attention distribution α, used to construct the context vector, as the output of the pointer network. Then, they create a mixture model summing together the vocabulary distribution and the attention distribution. In detail, the probability of the output word y_i is computed as in [Merity et al., 2017]:

$$\begin{aligned} p(y_i|\mathbf{y}_{<i}, \mathbf{x}) =& g \cdot p_{vocab}(y_i|\mathbf{y}_{<i}, \mathbf{x}) \\ &+ (1-g) \cdot \sum_{j \in I(y_i)} a_j \end{aligned} \tag{4.32}$$

The difference from Merity et al's work is that the value g is not extracted from the distribution, but it is computed separately using the context vector $\mathbf{c}_i$, the current decoder state $\mathbf{h}_i$, and the previous decoder output $\mathbf{y}_{i-1}$:

$$g = \sigma(\mathbf{W}\mathbf{c}_i + \mathbf{V}\mathbf{h}_i + \mathbf{U}\mathbf{y}_{i-1} + b) \tag{4.33}$$

4.3.9 Paulus et al.'s

Paulus et al. (2018) propose a simple pointer network to copy an input word. As in [See et al., 2017], they combine the vocabulary distribution with the attention distribution. However, they do not sum the scores of the same word, but they treat all words as "different ones". Thus, the final distribution of a word y_i is composed as follows:

$$p(y_i|\mathbf{y}_{<i}, \mathbf{x}) = g \cdot p_{vocab}(y_i|\mathbf{y}_{<i}, \mathbf{x}) + (1 - g) \cdot a_i \tag{4.34}$$

where $\mathbf{a}$ is the attention vector, and the score g is computed using the current decoder state $\mathbf{h}_i$, the current context vector $\mathbf{c}_i$ and the novel decoder context vector $\mathbf{c}_i^d$:

$$g = \sigma(\mathbf{W}\mathbf{h}_i + \mathbf{V}\mathbf{c}_i + \mathbf{U}\mathbf{c}_i^d + b) \tag{4.35}$$

The decoder context vector is the weighted sum of all decoder states up to timestep $i - 1$:

$$\begin{aligned} \mathbf{e}_{i,j} &= \mathbf{h}_j \mathbf{W} \mathbf{h}_i \\ \beta_i &= softmax(\mathbf{e}_i) \\ \mathbf{c}_i^d &= \sum_{j=1}^{i-1} \beta_j h_j \end{aligned} \tag{4.36}$$

The idea is to provide more information about the previous states to the network and avoid repetitions.

4.3.10 Comparison between models

In this section, I described the several pointer networks present in literature, starting from the first one created by Vinyals et al. (2015); these are used to copy in output documents words that are outside the NN's vocabulary.

Now, I compare them, reporting their performance on the same dataset for Text Summarization. The problem, however, is that some models have been trained and tested on different ones (e.g., Merity et al. (2017)'s model is trained on the Penn TreeBank), while others have been designed for different tasks (see the model of Bhoopchand et al. (2017) for code generation). Thus, I decided to report the Rouge scores of only those that have been evaluated either on Gigaword or CNN/Dailymail dataset. For the former one, I compare Miao and Blunsom (2016)'s model and Zeng et al. (2017)'s model, while for the latter dataset, the models of Nallapati et al. (2016), See et al. (2017), and Paulus et al. (2018). To perform a proper comparison, I only consider their baseline one (i.e., the model that only uses the pointer network).

Table 4.4 reports the results on the CNN/Dailymail dataset. From the table, it is possible to see that Paulus et al.'s model has the best R-1 and R-L scores. This is due by the use of the bilinear attention, which better captures the relation between the decoder state and an encoder one.

Model	R-1	R-2	R-L
Nallpati et al.'s model	35.46	13.3	32.65
See et al.'s model	36.4	**15.66**	33.42
Paulus et al.'s model	**37.86**	14.69	**34.99**

Table 4.4: The table reports the Rouge scores on the CNN/Dailymail dataset. High value is better.

Table 4.5 reports the results on the Gigaword dataset, where the model of Miao et al. obtained the best results thanks to its extractive ability and the training via Reinforcement Learning.

Model	R-1	R-2	R-L
Miao et al.'s model	**34.17**	**15.94**	**31.92**
Zeng et al.'s model	27.96	12.65	26.18

Table 4.5: The table reports the Rouge scores on the Gigaword dataset. High value is better.

4.4 Exploiting Document Content

Neural Abstractive Summarization is a constantly changing field, where many new research directions are appearing, while others are converging onto a single one. I identified two very interesting research directions: 1) Retrieve, Rank and Rewrite Summarization (R^3S from now on) systems [Chen and Bansal, 2018; Cao et al., 2018; Wang et al., 2019] and 2) Content Selection methods [Zhou et al., 2017; Gehrmann et al., 2018; Hsu et al., 2018; Tan et al., 2017; Li et al., 2018a]. Both are emerged in the last two years. In the former one, which I address with the name of R^3S *systems*, the systems are trained to select a sentence from the input document or an external resource (retrieve and rank part) that is salient (i.e., that expresses the content of a sentence in a very concise manner). The selected sentences are then rewrote (substituting, adding and deleting words) to generate the summary. In the latter one, researchers have defined layers, masks, and network structures to uncover salient sentences, while removing redundant information. Other interesting works integrated further information into Sequence-to-Sequence models (e.g., external knowledge or topic information) or improved the capability of the

model to create a better document representation that contains all salient information for the summary.

4.4.1 Retrieve, Rank and Rewrite systems

In Section 4.2, I described how Sequence-to-Sequence (S2S) models tend to focus multiple times on the same words, producing summaries that contains up to 8 repetitions of the same word. This impact on the readability and informativeness of the generated summaries. The problem is caused by the generation of long summaries because the S2S model tends to lose the control. Fortunately, the adoption of coverage methods have solved, and in some extreme case mitigated, such behaviour.

Other researchers [Cao et al., 2018; Chen and Bansal, 2018], however, have explored the problem under a different point-of-view. Their idea is that an human-written sentence is fluent, non-redundant and well structured, compared to the ones generated by the model. They pioneered a new set of of systems, that I named with Retrieve, Rank and Rewrite Summarization (R^3S) systems inspired by the title of Cao et al.'s paper, where the model is trained to select a human-written sentence and rewrite it using the high abstractive power of the S2S model in order to generate the summary. Successively, Wang et al. (2019) improved this architecture.

Cao et al.'s model

Cao et al. (2018) try to solve the problem regarding the generation of long summaries using template-based summarization (e.g., the templates used in [Zhou and Hovy, 2004]). A template is a general manually-written summary which can be automatically filled with information extracted from the document. For instance,

> [REGION] shares [open/closed] [NUMBER] percent [lower/higher]

is a template that could be rewritten as "UK shares open 7 percent higher". They used Lucene[3] to store the templates, and retrieve those ones that are similar to a given sentence x. Then, both the sentence x and the retrieved templates $\mathbf{t} = [t_1, t_2, \ldots, t_n]$ are encoded using a Bidirection GRU. Those encoded representations are used to assign a score, through the function $s(\cdot)$, to the retrieved templates:

$$s(t, x) = \sigma(\mathbf{h}_t^T \mathbf{W} \mathbf{h}_x + b) \tag{4.37}$$

where $\mathbf{h}_t$ is the encoded representation of the template, $\mathbf{h}_x$ is the encoded representation of the sentence, and $\sigma(\cdot)$ is the sigmoid function. The template with the highest score is then selected and concatenated with the encoded representation of the sentence

[3] https://lucene.apache.org/

x to form the hidden representation $\mathbf{h}_c$. Such representation is used in the decoder to create the summary, rewriting the template with the document information.

Cao et al. compared their model, called Re^3Sum, against Rush et al. (2015)'s ABS model and Chopra et al. (2016)'s ABS model, on the Gigaword Dataset. Table 4.6 shows the Rouge scores.

Model	Rouge-1	Rouge-2	Rouge-L
ABS [Rush et al., 2015]	29.55	11.32	26.42
ABS+ [Rush et al., 2015]	29.78	11.89	26.97
RAS-Elman [Chopra et al., 2016]	33.78	15.97	31.15
Re^3Sum [Cao et al., 2018]	**37.04**	**19.03**	**34.46**

Table 4.6: The table shows the comparison of Chopra et al.'model, Rush et al.'s model with Cao et al.'s model on Gigaword corpus.

Chen et al.'s model

Differently from Cao et al., Chen and Bansal (2018) selected a set of sentences from the input document and rewrote them to create the summary. More in detail, their model is composed of a hierarchical encoder followed by a decoder. The hierarchical encoder first reads the words of a sentence using a Convolutional Neural Network [LeCun et al., 1995] to create the sentence representation. Then, it processed all the sentence representations via a bidirectional LSTM. The decoder, instead, selects a document sentence using a sparse pointer-network [Bhoopchand et al., 2017], where C is set to infinity. The constant is used to skip all those sentences that the model has already selected. Finally, the selected sentence is rewritten by the decoder.

Chen et al. compared their model against See et al. (2017)'s Pointer Generator model, Paulus et al. (2018)'s model, and Nallapati et al. (2017)'s models on CNN/Dailymail corpus, obtaining state-of-the-art results. Table 4.7 reports the comparison.

Model	Rouge-1	Rouge-2	Rouge-L
Nallapati et al.†	39.6	16.2	35.3
lead-3 [Nallapati et al., 2017]†	39.02	15.7	35.5
Paulus et al.†	38.30	14.81	35.49
Paulus et al.(RL)†	39.87	15.82	36.90
Pointer Generator [See et al., 2017]	39.53	17.28	36.38
Chen and Bansal(RL)	**40.88**	**17.80**	**38.54**

Table 4.7: The table shows the comparison between Chen et al.'s model with Nallapati et al.'s models, See et al.'s model and Paulus et al.'s model. In the table, RL stands for Reinforcement Learning training method, while † means that the model cannot be strictly compared since it is trained and tested on the anonymized version of CNN/Dailymail corpus.

Wang et al.'s model

Wang et al. (2019)'s model for template summarization uses a bidirectional selective encoder to extract relevant passages of the input article and fill the template. It starts ranking all the templates using a neural network called *Fast Rerank Module*. More in detail, the Fast Rerank Module first extracts a set of features from the article and the template using a Gated Linear Unit (GLU) [Dauphin et al., 2017]. Then, it constructs a similarity matrix comparing each feature of the article with those of the templates. The constructed matrix is then passed in input to a max pooling layer followed by a Multi-Layer Perceptron (MLP) to select a template. The selected template is given in input to a Bidirectional LSTM to calculate the hidden states $\mathbf{h}^t$, which are used to filter the hidden states $\mathbf{h}^a$ of the article. In other words, a gate g_i is calculated for each i-th word as follows:

$$\mathbf{g}_i = \sigma(\mathbf{W}_a \mathbf{h}_i^a + \mathbf{W}_t \mathbf{h}_i^t + b_g). \tag{4.38}$$

Then, the gate of each word is multiplied to the corresponding article hidden state:

$$\mathbf{h}_i^g = \mathbf{h}_i^a \otimes g_i \tag{4.39}$$

where $\otimes$ is the pointwise product. The authors also calculated a confidence degree d, which captures how much the article is credible. Such degree is used to add some $\mathbf{h}^a$ information (a residual) to $\mathbf{h}^g$ states as follows:

$$\mathbf{z}_i^a = d\mathbf{h}_i^g + (1-d)\mathbf{h}_i^a \tag{4.40}$$

where the score d is calculated as:

$$d_i = \sigma((\mathbf{h}_i^a)^T \mathbf{W}_d \mathbf{h}_i^t + b_d) \tag{4.41}$$

The final states $\mathbf{z}^a$ are used to calculate the energies (i.e., e scores) of the attention. They called their model **BiSET**.

Table 4.8 reports the results obtained by Wang et al.'s model on Gigaword corpus.

Model	Rouge-1	Rouge-2	Rouge-L
ABS [Rush et al., 2015]	29.55	11.32	26.42
ABS+ [Rush et al., 2015]	29.78	11.89	26.97
RAS-Elman [Chopra et al., 2016]	33.78	15.97	31.15
Re^3Sum [Cao et al., 2018]	37.04	19.03	34.46
BiSET [Wang et al., 2019]	**39.11**	**19.78**	**36.87**

Table 4.8: The table reports the results obtained by Wang et al.'s model on Gigaword corpus.

4.4.2 Content Selection

When people summarize a text, they first read the document to highlight sentences that are salient (i.e., being relevant to construct the summary) and novelty (i.e., non-redundant). Those mechanisms, modelled in extractive summarization, have not been adopted in Text Summarization with Neural Networks, where the networks are based on the Sequence-to-Sequence model (see Section 3.7.5). This means that the networks give the same relevance to all words (and abstractly, to all sentences), without discerning the relevant ones. For instance, if a document talks about both a big snow storm and a man that started to sell snow, the network may decide to focus on just one piece of information, creating a summary that is grammatically and semantically correct, but it is not related to the document.

To improve the summary generation, researchers have started to experiment with the different layers of the Sequence-to-Sequence model, adding gates, masks, sentence scores, or a combination of all. Those methods can be separated onto two levels:

word-level: mechanisms that obscure or filter redundant and irrelevant information. Zhou et al. (2017) defined a gate to filtered out irrelevant information from the encoded states, while Gehrmann et al. (2018) proposed a mask to do not consider certain words.

sentence-level: Hsu et al. (2018) decided to use extractive techniques to assign a score to each sentence. Scores take in consideration the content, the novelty (i.e., if the sentence is not redundant) and the importance of the sentence. Then, the score of a sentence is multiplied with the attention scores of all words appearing in that sentence. This multiplication can be seen as a soft mask, compared with the one of Gehrmann et al. (2018), that elicits relevant words and passage from the

document. Tan et al. (2017) adopted a graph-based sentence-level attention to capture such information. The difference between Hsu et al.'s idea and Tan et al.'s idea is that the model can re-score the sentences according to the previous used information.

Finally, Li et al. (2018a) decided to incorporate both word-level and sentence-level methods into a hierarchical decoder model [Tan et al., 2017]. The following sections will explain deeply all the proposed works.

I reported in Table 4.9 the Rouge scores for all the models, except for Zhou et al.'s one which has been trained and tested on Gigaword dataset.

Model	Rouge-1	Rouge-2	Rouge-L
Nallapati et al.†	39.6	16.2	35.3
lead-3 [Nallapati et al., 2017]†	39.02	15.7	35.5
Paulus et al.†	38.30	14.81	35.49
Paulus et al.(RL)†	39.87	15.82	36.90
Pointer Generator [See et al., 2017]	39.53	17.28	36.38
Gehrmann et al.	41.22	**18.68**	**38.34**
Hsu et al.	40.68	17.97	37.13
Tan et al.	38.1	13.9	34.0
Li et al.	**41.54**	18.18	36.47

Table 4.9: The table reports Rouge scores for all described models (except for Zhou et al.'s one). In the table, RL stands for Reinforcement Learning training method, while † means that the model cannot be strictly compared since it is trained and tested on the anonymized version of CNN/Dailymail corpus.

Zhou et al.'s Selective Gate

Zhou et al. (2017) proposed a gate to filter irrelevant and redundant information from the encoder states. First they read the input sequence $\mathbf{x} = [x_1, \ldots, x_n]$ using a Bidirectional GRU[4], producing a sequence of states $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_n]$. Each i-th state is the concatenation of i-th forward state and backward state:

$$\mathbf{h}_i = [\overrightarrow{\mathbf{h}_i}||\overleftarrow{\mathbf{h}_i}] \tag{4.42}$$

Then, they construct the document representation s using the first backward state and the last forward state:

[4]The tokens of the input sequence are mapped to their word embedding before being processed by the GRU.

$$\mathbf{s} = [\overrightarrow{\mathbf{h}_n}||\overleftarrow{\mathbf{h}_1}] \tag{4.43}$$

The document representation is used to filter non-salient and redundant information from the encoder states, simply measuring the relation between the encoder state $\mathbf{h}_i$ and the document $\mathbf{s}$:

$$\begin{aligned} \mathbf{sGate}_i &= \sigma(\mathbf{W}_s\,\mathbf{h}_i + \mathbf{U}_s\,\mathbf{s} + b) \\ \mathbf{h}'_i &= \mathbf{sGate}_i \otimes \mathbf{h}_i \end{aligned} \tag{4.44}$$

where $\otimes$ represents the pointwise multiplication. Those filtered states are then used to compute both the attention and the context vector $\mathbf{c}$.

Gehrmann et al.'s attention mask

As I said in the introduction of this section, people tend to focus (i.e., pose attention) on relevant passages of the document to create the summary. Gehrmann et al. (2018), inspired by this human-mechanism, defined an hard mask that is applied on the attention distribution. The mask creates a sparse distribution modifying the values as follows:

$$p(\tilde{a}_i|\mathbf{y}_{<i}, \mathbf{x}) = \begin{cases} p(a_i|\mathbf{y}_{<i}, \mathbf{x}) & \text{if } q_i > \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{4.45}$$

where q_i is the probability that the i-th word is selected, calculated as:

$$q_i = \sigma(\mathbf{W}_q\,\mathbf{h}_i + b_q) \tag{4.46}$$

where $\mathbf{h}$ is an encoder state and ϵ is an hyperparameter of the model. After that the new distribution scores are calculated, they are re-normalized to obtain the distribution.

Hsu et al. model

Hsu et al. (2018) fused Extractive and Abstractive Text Summarizzation into one model. In their model, extractive techniques are used to give a score to each sentence, i.e., how much a sentence is relevant for the summary. Then, the score of a sentence is multiplied with the attention scores of the words that appear in that sentence. In this way, the attention distribution is re-scored, focusing only on those words that are informative for the summary. Their idea is that "words coming from less attended sentences are less likely to be generated", i.e. words coming from sentences that are not salient for the summary are not selected (or focused). Given the word-attention scores $\alpha = [\alpha_1, \ldots, \alpha_n]$ and the sentence scores $\beta = [\beta_1, \ldots, \beta_m]$, the final word-attention scores are calculated as follows:

$$\delta_i = \frac{\alpha_i * \beta_{s(i)}}{\sum_j \alpha_j * \beta_{s(j)}} \tag{4.47}$$

where $s(\cdot)$ is a function that returns the sentence index where the i-th word appears. The product ensures that δ will be high only when both α_i and $\beta_s(i)$ are high. However, the drawback of their approach is that the scores remain fixed at each decoder timestep, not capturing the information that have already been used. This drawback is fixed in [Li et al., 2018a].

Tan et al.'s model

Tan et al. (2017) propose a different model compared with the previous ones. Their model is a hierarchical Sequence-to-Sequence model with a graph based attention. With the term hierarchical, I mean a model that first generates a sentence representation reading all the words present in that sentence (e.g., using a recurrent neural network), and then it processes all the sentences with another neural network (e.g., another recurrent network) to generate the document representation. Generally, such model is only applied to the encoder, but they extended it also to the decoder. This means that they first generate a summary-sentence representation, and then they use this latter one to generate the words of the sentence. In this case, the sentence-decoder takes in input the document representation as initial state, while the word-decoder takes in input the current sentence-decoder state as initial state. At each timestep, the sentence-decoder is updated using the last word-decoder state. The sentence-decoder stops when it generates the special token $<$*EOD*$>$ (End of Summary). They also used a double attention: a sentence-level attention and a word-level attention. The sentence-level attention is used to capture the relevance of each sentence, i.e., if a sentence is informative. Furthermore, since a sentence can have a piece of information contained in other sentences, they decided to use a graph-based attention. The graph-based attention computes the attention score of a sentence using the current sentence-decoder state, the sentence-encoder state (for which the score has to be calculated), and all other sentences. In detail, their idea is that a sentence is relevant if it is heavily linked with many sentences. The sentence-level attention is calculated as follows:

$$\mathbf{f}(t+1) = \lambda \mathbf{W}\mathbf{D}^{-1}\mathbf{f}(t) + (1-\lambda)\mathbf{y} \tag{4.48}$$

where $\mathbf{f} = [f_1, \ldots, f_m]$ is the set of attention scores of the sentences, t expresses the current iteration, $\mathbf{D}$ is a diagonal matrix with its (i,i)-elements equal to the sum of the i-th column of the adjacent matrix $\mathbf{W}$. λ is a damping factor and it is passed as hyperparameter. Finally, $\mathbf{y}$ is calculated as follows:

$$y_i = \begin{cases} \frac{1}{|T|} & \text{if } i \in T \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \tag{4.49}$$

where T is a topic. They treated the current sentence-decoder state as the topic. Thus, $\mathbf{y}$ is always a one-hot vector.

Li et al.'s model

Li et al. (2018a) define a hierarchical Sequence-to-Sequence model based on the ideas of Zhou et al. (2017) and Hsu et al. (2018) for sentence filtering and scoring, respectively. In detail, their model reads the words of a sentence using a Bidirection GRU to generate the sentence representation. Then, all sentence representations are read by another Bidirectional GRU to compose the document representation. Such document representation is used to compute the selective gate for each sentence as in [Zhou et al., 2017].

The sentence-decoder state is initialized using the document representation, and it is update using the last state of the previous generated sentence and the sentence-level attention α. The latter parameter is passed in input for the coverage mechanism. Once the weights are calculated, the sentence-level context is computed as:

$$\mathbf{c}_s = \sum_i \alpha_i \mathbf{h}'_i \tag{4.50}$$

where $\mathbf{h}'_i$ is the sentence representation after the application of the selective gate. They use the sentence-level context to initialize the word-level decoder for the current sentence. The word-level decoder is the same of the other models, where the current word-decoder state and the word-level context are used to generate the vocabulary distribution. However, in order to model the importance of each sentence, they multiply the word-level attention scores of a sentence with the sentence scores as in [Hsu et al., 2018].

4.4.3 Other Works on Text Summarization

Other researchers tried to solve the Summarization task from different point-of-views. Some research works [Liu and Lapata, 2019; Song et al., 2019; Kryściński et al., 2018] used a pre-trained language model to improve the summary generation. Their idea is that the language model carries both fluency and domain style since it is able to deeply understand the meaning of each token. In detail, Liu and Lapata (2019) and Song et al. (2019) used BERT-based models [Devlin et al., 2018] to solve the Summarization task. The former authors adapted BERT for Extractive Summarization. They modified the final layer of BERT to compute a score for each sentence which expresses if it has

to be selected for the summary. The latter authors, instead, developed a Sequence-to-Sequence model based on BERT. Such model, as BERT, removes random words from the input to improve the generalization and the word embedding. Furthermore, thanks to the decoder, it can be applied to any language generation task, such as Abstractive Summarization. Kryściński et al. (2018), instead, proposed to extend the Sequence-to-Sequence model proposed by Paulus et al. (2018) with a language model that integrates external knowledge. In detail, they used a three layer stacked LSTM, trained on CNN/Dailymail corpus, to create the language model. Then, the output of the stacked LSTM is used, together with the current decoder state and the context vector, to predict the next word. Their idea is to allow the decoder to focus on attention and extraction, while the language model is responsible to generate the summary.

Other authors, like Narayan et al. (2018), preferred to use the topic model to tied the model output to the document. They proposed both the task of *extreme summarization*, where the goal is to create a very short sentence that contains all relevant information of the input document, and a topic-based model to accomplish such task. Their model is an encoder-decoder one, enriched with document-topic distribution and word-topic distribution. More in detail, given t_D as the topic distribution of the document and t_w as the topic distribution of the words, the encoder and decoder word inputs are defined as follows:

$$\begin{aligned} e_{w_e} &= [E(w_e) \,||\, P(w_e) \,||\, t_{w_e} \otimes t_D] \\ e_{w_d} &= [E(w_d) \,||\, P(w_d) \,||\, t_D] \end{aligned} \tag{4.51}$$

where the $E(\cdot)$ is the function that returns the embedding of a word, $P(\cdot)$ is the function that returns the position embedding of a word (i.e., where the word appears in the sentence). w_e and w_d are the encoder input word and decoder input word, respectively. The concatenation of the word embedding with the document-topic distribution forces the model to generate a summary that has the same theme of the document.

Finally, Ma et al. (2018) and Li et al. (2017) used autoencoders to capture the latent structure of the summaries. Ma et al. proposed to use an autoencoder to improve the ability of the Sequence-to-Sequence model of creating a fixed-size document representation. First, they defined an autoencoder to create a representation z_s of an input summary. Such autoencoder is trained using the abstractive summaries present in the trainset. Then, they trained the Sequence-to-Sequence model, passing both the document sentences and the target summary. During the training, they forced the model to minimize the distance between the document representation z_d (created by the sequence-to-sequence) and the representation z_s. Such distance is measured as follows:

$$d(z_s, z_d) = \frac{\lambda}{N_h} ||z_d - z_s||_2 \tag{4.52}$$

where λ is an hyperparameter of the model to balance the distance loss and N_h is the number of hidden neurons, used to limit the magnitude of the distance function. Li et al., instead, found that the summaries follow a latent structure composed of "*Who Action What*". They fused the decoder with a variational autoencoder to capture such structure from the data and use it to improve the quality of the summaries.

Chapter 5

A Sentence-level Attention for Content-Aware Summarization

In the previous chapters I described different aspects of the Summarization task, moving from its definition to the recent models based on Neural Networks. I started describing the task in Chapter 2, highlighting the different facets about how it can be solved (e.g., with Neural Networks, with graph-based methods, with lexical chains-based methods, and so forth). Then, I moved on describing Neural Networks, and specifically those ones that can be used for Summarization in Section 3.7. However, those models suffer from three drawbacks: (i) repetitions in the output summary; (ii) the impossibility of generating output words that are outside of the Neural Network vocabulary; (iii) the impossibility to distinguish the relevant and informative sentences from the other ones, treating all of them equally. Following the recent research works conducted to contrast this latter issue, I propose a method based on a graph to capture the relevance of a sentence (via a score) according to two factors:

1. the summary that the network has already generated up to the current timestep, and

2. the information contained in the other sentences.

Then, the attention score of a word is multiplied by the score of the sentence where that word appears. In this way, I normalize the word-level scores according to the relevance of the sentences in the document.

5.1 Introduction

The main drawback of Neural Network-based models for Summarization is their incapability of capturing the salience of each sentence. They treat all words and sentences

as equals, not distinguishing those that are relevant for the summary from those that are used as semantic glue to connect the different parts of the document. For instance, if a document talks about both a big snow storm and a man that started to sell snow, the network may decide to focus on just one piece of information (e.g., the snow storm), creating a summary that is grammatically and semantically correct, but not related to the desired target output (e.g., the man who sells the snowballs).

To solve this drawback, I decided to assign a score to each sentence that represents its saliency. A high score means the sentence is relevant for the summary, while a low score means the sentence is not useful for it. These sentence-level scores are than multiplied by the word-level ones to focus only on words that appear in salient sentences. However, a document could have redundant sentences, i.e. sentences that have a similar content; thus, analyzing those ones in a stand-alone fashion is not the correct way to proceed. The score of a sentence has to be calculated not only considering itself and the decoder state, but also the information contained in the other ones.

I decided to apply a graph-based method to calculate the scores. Such method, as reported in Section 2.3, creates a graph $G=<V, E>$ where the set of vertex V represents each sentence in the document, while the set of edges E describes the connections between those sentences. These connections are defined via a similarity function (e.g., cosine similarity), which score is reported as the weight of the edge. Then, an algorithm (e.g., HITS or Pagerank) is applied on the resulting graph to obtain a score for each vertex analyzing its connections. I reported this mechanism in the model using a graph-based Neural Network [Veličković et al., 2017; Tan et al., 2017; Klicpera et al., 2018; Yao et al., 2019], which (generally) requires an adjacency matrix and a feature matrix (the sentence representations) to be applied.

My proposed model is composed of a hierarchical encoder to compute both word-level and sentence-level representations, followed by a Recurrent Neural Network decoder. In detail, the hierarchical encoder can be seen as two Bidirectional Recurrent Neural Networks (RNNs from now on); the one at word-level, called *word-encoder*, is a bidirectional RNN that reads each word of a sentence and emits its representation, while the one at sentence-level, called *sentence-encoder*, uses the last states of the word-encoder to compute the sentence representations. I then used this latter ones to calculate the sentence-level scores, using the graph-based method, at each decoder timestep.

In this chapter, my contribution is twofold:

1. A sentence-level attention based on a graph. The sentence-level attention is then multiplied with the word-level one to rescore these latter values;

2. An inference-time score to remove repetitions and attention errors, based on the coverage vector and reinforcement learning.

This chapter is structured as follows: Section 5.2 describes with an example how the model should work; Section 5.3 describes the proposed model, reporting the proposed

sentence-level attention methods; Section 5.4 describes the training of those models and the results obtained. Finally, the chapter concludes with Section 5.5.

5.2 How the Model Should Work: A Practical Example

In this section, I am going to explain how the model should work. I will start reporting a CNN/Dailymail article and its topics. Then, I will explain which topic the model has to use to generate the summary and the words regarding it. Finally, I will highlight how the proposed model will select those words and use them to generate the summary.

Let us start with a document present in the CNN/Dailymail corpus regarding a man that started to sell snow online:

> A Massachusetts man has found a way to profit from the several feet of snow in his yard: He's shipping it to people in warmer climates for the bargain price of $89 for six pounds.
>
> Kyle Waring, of Manchester-by-the-Sea, just outside of Boston, got the idea while shoveling snow earlier this winter and launched ShipSnowYo.com.
>
> At first he shipped 16.9-ounce snow-filled bottles for $19.99, but he found the snow melted by the time it arrived at its destination.
>
> So he came up with a new plan, selling six pounds at a time and shipping the snow via overnight delivery.
>
> The ShipSnowYo website specifically appeals to buyers in warmer states.
>
> 'Wouldn't you love to make snowballs and touch snow for the first time?!' it reads. 'Now you can!
>
> 'This is historic snow. Boston Snow. . .
>
> The storms so far on the northeast have broken historic records and continues to release it's anger on the city of Boston.
>
> Over 70 inches of snow have been dumped on the Greater Boston area in the last 17 days.

For simplicity, I separated each sentence of the article in different lines. The article presents two topics: the first one regards that a man in Massachusetts started to sell snow via ShipSnowYo.com website. This is the main topic onto which the reference summary is created, and the one that the model should use to create the final summary. The second topic regards the Boston storm, which broken historic records; it gives more details to the reader, describing the context of the article and the reasons behind the snow sell. This topic is not relevant for the model and should be avoided in the creation of the summary. However, the model could copy via the pointer network some words to enrich the generated summary, providing more context to this latter one.

A problem of Neural Networks oriented to the Summarization task is that they tend to focus to irrelevant information. For instance, in the above article a simple Sequence-to-Sequence model could focus on the last part of the article (the snow storm) - producing a summary on this topic - which despite being correct, it is not related to the human one. To overcome this issue, many researcher added layers and masks to avoid these irrelevant information. In this thesis, I structured the model to find the relevant sentences and use them to create the summary. More in detail, the proposed model first reads each sentence to capture its meaning via a vector representation using a Recurrent Neural Network. Then, it combines those sentences to have a general view of the article.

At each timpestep, the sentence representations, together with the general view of the article and the current generated summary, are used to calculate a score for each sentence that highlights its relevance to the summary. These scores are produced using the Pagerank algorithm, which is capable to handle both the relevance and the redundancy of the sentences. For instance, in the above document the algorithm will assign an high score (close to 1) to the first 6 sentences since they are related to the same topic and are relevant to the summary. The remaining 3 sentences, instead, will receive a lower score because they are not strictly connected to the previous ones and they are not relevant for the summary. However, in some particular cases, the Pagerank algorithm could select an irrelevant sentences if it contains particular words that should be included into the summary. Finally, the scores and the weights of the attention layer are multiplied together to select the relevant passages and the words to copy in output.

Since the scores affect the sentence representation, they have an impact to the pointer network too. In this case, the pointer network will select the relevant terms only from the first 6 sentences. For instance, it may select: "*Kyle Waring*" or "*ShipSnowYo.com*"; these terms are not present in the model's vocabulary.

Let me further explain how the model will generate the summary through an example. For simplicity, I will skip intermediate passages to focus on how the model selects the sentences to continue the summary. Let me assume that the model has already generated the summary "*Kyle Waring*". Let me suppose that the next excerpts that the model could generate are "launched ShipSnowYo.com" or "is shipping". In this case, the model will give an high score to the first and second sentences because it requires

them to continue the summary. Assuming that the model generated "is shipping", now it requires to generate what Kyle is shipping. Thus, it will give an high score to the fourth sentence to focus on "six pounds", "snow" and "via overnight delivery" (this one will be copied using the pointer network). Combining them together, the summary is: "*Kyle Waring is shipping six pounds of snow via overnight delivery*". At this point, the only word missing is the price. Since it is present in the first sentence, the model will give an high score to this one. The final word "$89" will be copied to the summary via the pointer network.

5.3 Model

Let $\mathbf{x} = [x_1, x_2, \ldots, x_M]$ be a sequence of M source tokens (the tokens of the input document), and $\mathbf{w} = [w_1, w_2, \ldots, w_N]$ a sequence of N target tokens (the tokens of the summary), with $N \leq M$. Neural Network-based encoder-decoder models, also knows as Sequence-to-Sequence models [Sutskever et al., 2014] (*Seq2Seq* from now on), are used to transform the sequence $\mathbf{x}$ into the sequence $\mathbf{w}$. Generally, the encoder and the decoder are jointly trained to minimize the cross-entropy loss function.

My proposed model is similar to the one of Nallapati et al. (2016). It consists of a hierarchical Bidirectional LSTM Encoder and an attention-based LSTM Decoder. In detail, differently from the classic encoder, the hierarchical one is composed of two encoders, each one formed by a Bidirectional LSTM. For simplicity, I will call the first encoder *word-encoder*, and the second one *sentence-encoder*. First, the word-encoder generates a sentence representation reading all M words of a sentence; then, the sentence representations are read by the sentence-encoder to generate the document representation. Let $\mathbf{x_1} = [x_{1,1}, x_{1,2}, \ldots, x_{1,M}]$ be the tokens that compose the first sentence of the document, and $\mathbf{D} = [\mathbf{x}_1, \ldots, \mathbf{x}_K]$ the set of K sentences that compose the document. At each step i, the word-encoder is fed with the input tokens $x_{1,i}$ and produces two encoded states: $\overrightarrow{\mathbf{h}}_{1,i}$ and $\overleftarrow{\mathbf{h}}_{1,i}$, where the first one is generated by reading the sentence words from left-to-right, while the latter one by reading the sentence words right-to-left. Those two states are distinguished through the jargon terms *forward state* and *backward state*. Equations 5.1 and 5.2 represent the forward and backward states construction:

$$\overrightarrow{\mathbf{h}}_{i,j} = \overrightarrow{LSTM}_W(E(x_{i,j}), \overrightarrow{\mathbf{h}}_{i,j-1}) \tag{5.1}$$

$$\overleftarrow{\mathbf{h}}_{i,j} = \overrightarrow{LSTM}_W(E(x_{i,j}), \overleftarrow{\mathbf{h}}_{i,j-1}) \tag{5.2}$$

where $\overrightarrow{LSTM}_W$ represents the word-level forward LSTM, $\overleftarrow{LSTM}_W$ the backward one, and $E(\cdot)$ is a function that returns the word-embedding [Mikolov et al., 2013c; Pennington et al., 2014] of an input word. Once the encoded states of a sentence are generated, the last forward state $\overrightarrow{\mathbf{h}}_{i,M}$ and the last backward state $\overleftarrow{\mathbf{h}}_{i,1}$ are concatenated

together to generate the sentence representation $\mathbf{h}_i$. Such operation is represented in Equation 5.3:

$$\mathbf{h}_i = [\overrightarrow{\mathbf{h}}_{i,M} || \overleftarrow{\mathbf{h}}_{i,1}] \tag{5.3}$$

where $||$ is the concatenation operator. Then, the sentence representations $[\mathbf{h}_1, \ldots, \mathbf{h}_K]$ are passed to the sentence-encoder which will produce a forward state $\overrightarrow{\mathbf{d}}_i$ and backward state $\overleftarrow{\mathbf{d}}_i$. The document representation $\mathbf{d}$ is constructed as for the sentence representation, concatenating the last forward state with the last backward one:

$$\mathbf{d} = [\overrightarrow{\mathbf{d}}_K || \overleftarrow{\mathbf{d}}_1] \tag{5.4}$$

Similar to the encoder, on each step t the decoder (a single unidirectional LSTM) receives in input the embedding of the previous word[1], the decoder state $\mathbf{s}_t$ and the context vector $\mathbf{c}_t$, and uses them to emit a word in output (the network emits a probability distribution over the vocabulary that is used to select the next word). In my model, the decoder state is initialized with the last state of the sentence-level encoder which captures all the information present in the document. The context vector $\mathbf{c}_t$ is calculated as in [Bahdanau et al., 2014]:

$$\mathbf{c}_t = \sum_{k=1}^{M} \hat{a}_k^t \, \mathbf{h}_k \tag{5.5}$$

where the score $\hat{a}_k$ represents the attention score, calculated through the attention approach, and $\mathbf{h}_k$ is the representation of the t-th word.

In my model, the attention scores $\hat{\mathbf{a}}$ are computed differently from the other models that can be found in the literature. Inspired by the works of Hsu et al. (2018), Nallapati et al. (2016) and Tan et al. (2017), I decided to combine the word-level attention with the sentence-level scores to select only those words coming from sentences that are relevant for the summary. Given $\mathbf{b}$ as the sentence-level scores and $\mathbf{a}$ as the word-level scores, I computed the attention scores $\hat{\mathbf{a}}$ as follows:

$$\hat{a}_i = \frac{a_i * b_{s(i)}}{\sum_{k=1}^{M} a_k * b_{s(k)}} \tag{5.6}$$

where $\mathbf{a}$ scores are computed by Equation 5.7 (the symbol T in $\mathbf{v}^T$ is the transpose operator). The $\mathbf{b}$ scores are calculated using a graph-based Neural Network, which keeps into account both the current decoder state $\mathbf{s}_t$ and the sentence representations. The graph-based network is described in Section 5.3.1. The model is depicted in Figure 5.1.

[1] During training it is the embedding of the target word w_{t-1}, while in testing it is the embedding of the previous word emitted by the decoder.

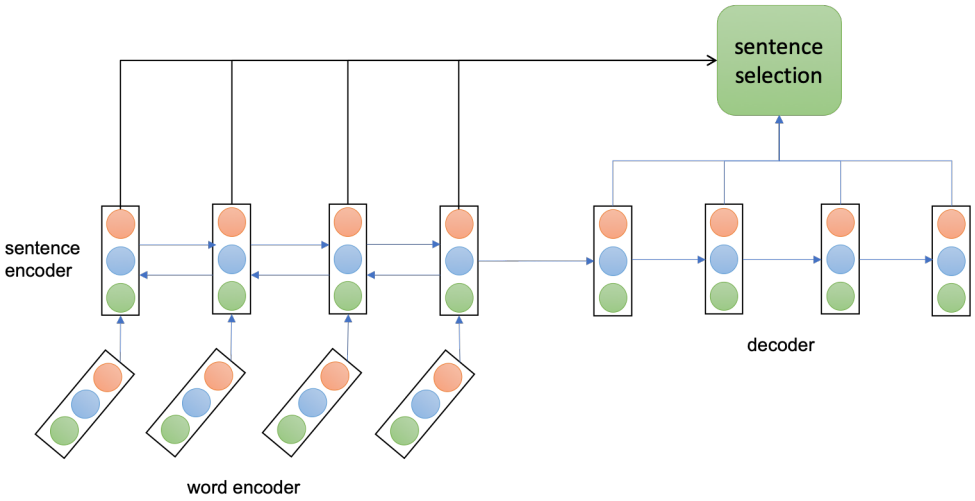


Figure 5.1: The figure shows the proposed model. The *sentence-selection* method is described in Section 5.3.1.

$$
\begin{aligned}
e_j &= \mathbf{v}^T \tanh(\mathbf{W}_e \left[\mathbf{s}_t || \mathbf{h}_j\right] + b_e) \\
a_i^t &= \frac{\exp(e_j^t)}{\sum_{k=1}^{M} \exp(e_j^t)}
\end{aligned}
\tag{5.7}
$$

For the output, I adopted the mixture model proposed by See et al. (2017), which combines the probability of a vocabulary word with its attention score to deal with the Out-Of-Vocabulary (OOV) problem and rare words. Thus, the probability of emitting a word y_i at step t is computed as follows:

$$
\begin{aligned}
\mathbf{o}_t &= \mathbf{W}_c \left[\mathbf{c}_t || \mathbf{s}_t\right] + b_c \\
P(y_i^t) =& \beta \; softmax(\mathbf{W}_o \, \mathbf{o}_t + b_o) \\
&+ (1 - \beta) \sum_{w=y_t} \hat{a}_w^t
\end{aligned}
\tag{5.8}
$$

where $\mathbf{W}_o$, $\mathbf{W}_c$, b_o and b_c are learnable parameters and y_{t-1} is the previous emitted word. β is a value within the range $[0, 1]$ used to mix the two distributions and it is calculated through Equation 5.9, where $\mathbf{W}_b$ and b_b are learnable parameters.

$$
\beta = sigmoid(\mathbf{W}_b \left[E(y_{t-1}) || \mathbf{c}_t || \mathbf{s}_t\right] + b_b) \tag{5.9}
$$

In the Equation 5.8, β is used to control how much information coming from the attention distribution will entry in the vocabulary distribution. If β is close to 1, the

model generates the next word only using the vocabulary (i.e., selecting the word with the highest probability over the vocabulary); if β is close to 0, then the model uses the attention distribution over the input document. This latter case is used by the model to copy OOV words.

I trained the model to minimize the sum of the negative log-likelihood of the sequence of target words w^*:

$$loss = \frac{1}{N}\sum_{t=1}^{N} -log\ P(w_t^*) \tag{5.10}$$

During the evaluation of the model, I found that it tends to generate summaries that have repeated words (or sentences). This problem is common in the task of Neural-based Abstractive Summarization, as I reported in Section 4.2. I thus decided to apply some tricks both in training and inference step to contrast this issue. In the training step, I applied the coverage method defined by See et al. (2017); a short description of the method is reported in Section 5.3.2. At inference time, I decided to modify the scoring function of the beam search:

$$s(x, y) = \frac{logp(y)}{|y|} \tag{5.11}$$

in order to include the length penalty $ln(y)$ defined by Gehrmann et al. (2018), which encourages the network to generate long and rich summaries. The length normalization is defined as:

$$ln(y) = \frac{(5 + |y|)^\alpha}{(5 + 1)^\alpha} \tag{5.12}$$

where α is an hyperparameter that controls the length of the summary. High α values lead to long summaries. The score function is then rewrote as:

$$s(x, y) = \frac{logp(y)}{ln(y)} \tag{5.13}$$

I also set a minimum summary length based on the training data. Additionally, I followed [Paulus et al., 2018] restricting the beam search to never repeat the same trigram in the summary.

5.3.1 Sentence-level Scores

In the introduction, I described that current Neural Network-based models for the Abstractive Summarization task are not able to distinguish between sentences that are informative and salient for the summary from those that are not. Hsu et al. (2018) and Tan et al. (2017) tried to solve this issue adopting a two-level attention: first, they computed

a score for each sentence and each word; then, a word score is multiplied by the score of the sentence where it appears. This multiplication allows to have an high word-level attention score only for those words that appear in sentences having an high sentence-level score (i.e., that are relevant for the summary). In this way, it ensures that salient words are more likely to be copied in output through the pointer network (see Chapter 4.3), and that the context vector poses its focus only onto certain (relevant) words.

I propose three methods based on graph to compute the sentence-level scores. All methods compare each sentence with the current decoder state in order to find informative sentences according to the summary generated so far. However, if a sentence is only compared with the decoder state, the informativeness of the other ones is not captured. For such reason, I decided to adopt a graph method since it permits to keep into account all those aspects. For instance, suppose that the score for the sentence d_1 has to be calculated. If d_1 is compared with the decoder state d, only the relevance of d_1 with respect to the generated summary is captured. Now, suppose that the score for the sentence d_3 has to be calculated too, and that d_3 contains the same facts of d_1 and adds more ones. If d_3 is compared with d, only its relevance with respect to the summary is captured, and not with d_1 too. In other words, it is not possible to know (unless all the sentences are manually compared) if a sentence is more informative than another one. With the adoption of the graph attention, a sentence is not only compared with the decoder state, but with the other ones that are similar. Such comparison allows to assign an high score to informative and relevant sentences while pushing the score of the redundant ones down. Figure 5.2 depicts an example of graph score calculation.

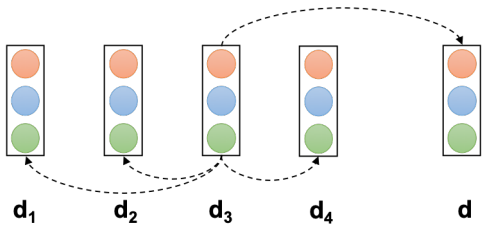


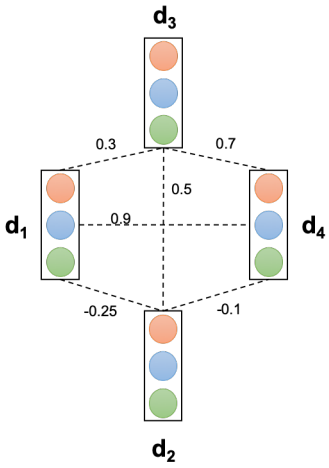
Figure 5.2: A graphical example of the score calculation of sentence $\mathbf{d}_3$. Dotted arrows represent the relations between the current sentence and the other representations.

I propose three sentence-level attention methods based on graph, each one calculating the scores in a different way. More in detail, differently from Tan et al. (2017) that defined the matrix A via a bilinear product of the sentences (to which they added the decoder state), I decided to adopt the Pagerank method for the Summarization task. Following Yao et al. (2019), where the adjacent matrix is filled using Pointwise Mutual Information or TF-IDF, I decided to use the cosine similarity function. Given two vectors $\overrightarrow{a}$ and $\overrightarrow{b}$, the cosine similarity function returns a value in the range $[-1, 1]$, where -1 indicates that the two vectors are opposite and 1 that are the same one. Further, the

cosine similarity ensures that the diagonal of A contains only values equal to 1 (because the vector is compared with itself), adding self-loops to the graph while normalizing the values (generally, the graph network requires to add the identity matrix to A for normalization). In detail, A is defined as follows:

$$A[i, j] = cosine(\mathbf{d}_i, \mathbf{d}_j) \tag{5.14}$$

where $\mathbf{d}_i$ and $\mathbf{d}_j$ are two sentence representations (for simplicity, I threat the current decoder state as a sentence). Figure 5.3 shows an example of a weighted graph with its adjacent matrix.



	d1	d2	d3	d4
d1	1.0	-0.25	0.3	0.9
d2	-0.25	1.0	0.5	-0.1
d3	0.3	0.5	1.0	0.7
d4	0.9	-0.1	0.7	1.0

Figure 5.3: The image shows a graph composed of four sentence representations (on the left) and its corresponding adjacent matrix (on the right).

I called the three methods: *cosine-pagerank*, *cosine-graph* and *pagerank-scores*.

Cosine-Pagerank Attention Method

This method is similar to the one proposed by Tan et al. (2017) to compute the Pagerank scores. Following their idea, I defined it as follows:

$$\mathbf{b} = softmax((1 - \lambda)(I - \lambda A\, D^{-1})^{-1}\mathbf{y}) \tag{5.15}$$

where I is the identity matrix, A is the adjacent matrix of the graph, D is the diagonal matrix where the i-th diagonal element is equal to the sum of the i-th columns of the matrix A, and y is a one-hot vector of size $K + 1$. It has all values equal to zero except on the one that represents the decoder state. The scalar λ is the dumping factor.

Cosine-Graph Attention Method

The problem of Equation 5.15 is that it does not use a weight matrix to modify the scores. The missing of that learnable parameter does not permit the neural network to adapt to the inputs, increasing the accuracy of the model and speeding up the convergence. For this reason, I decided to use a classic graph neural network, which is defined as follows:

$$\mathbf{b} = softmax(A\, D^{-1}\, F\, W) \tag{5.16}$$

where F is a matrix that contains all the sentences representation plus the current decoder state, and W is the learnable parameter of the model. As for *cosine-pagerank*, the matrix A is defined through Equation 5.14.

Pagerank Score Method

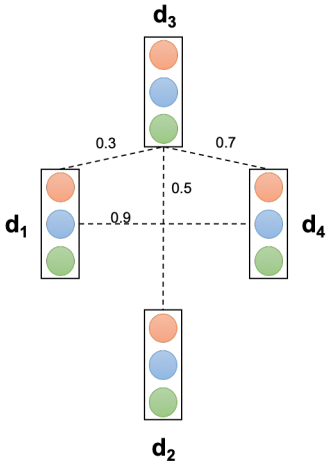
The problem of *cosine-pagerank* and *cosine-graph* is the probability distribution calculated over the sentences. The probability distribution could focus onto a single sentence, while (partially) ignoring the others that could be relevant to generate the next word. However, the model could require more than one sentence to understand a passage in the text and create the context vector, or a specific sentence in order to copy a word to the output. Hence, imposing a probability distribution over the sentences could lead to sub-optimal summaries. I thus decided to assign a score comprises in the range [0, 1] to each sentence. In this way, the model will decide which ones are important for the summary, whether it requires them for the context vector or the pointer network. I modified Equation 5.15 as follows:

$$\mathbf{b} = \sigma((1-\lambda)(I - \lambda\tilde{A}\, D^{-1})^{-1}\mathbf{y}) \tag{5.17}$$

where $\tilde{A}$ is defined applying the Rectified Linear Unit (ReLU) function to the adjacent matrix A, and $\sigma(\cdot)$ is the sigmoid function. The ReLU function removes all the negative values from the matrix (or vector) to which is applied, leaving only those that are equal or greater than 0. Applying it to the adjacent matrix A means that all those edges connecting sentences that are dissimilar are removed, leaving only the ones between sentences that share a common piece of information. In this way, the model selects the best sentence for each cluster of similar ones. Figure 5.4 shows the adjacent matrix of the Figure 5.3 after the ReLU.

5.3.2 Coverage Method

To contrast the word and sentence repetitions, I decided to add a coverage vector to Equation 5.7 to force the model to attend other words during the summary generation.



	d1	d2	d3	d4
d1	1.0	0	0.3	0.9
d2	0	1.0	0.5	0
d3	0.3	0.5	1.0	0.7
d4	0.9	0	0.7	1.0

Figure 5.4: The image shows a graph composed of four sentence representations (on the left) and its corresponding adjacent matrix (on the right). To increase the visual impact, I remove the edges with score 0.

I decided to use a coverage vector instead of the temporal attention due to the pointer network of Equation 5.8. During my experiments, I found that any method that directly modifies the attention scores has a terrible impact on the learning of the model. In detail, the model is not able to calculate a proper attention distribution, and as consequence to copy words in the output.

I defined the coverage vector $\mathbf{cov}_t$ as the sum of the attention distributions:

$$\mathbf{cov}_t = \sum_{t=1}^{N} \hat{\mathbf{a}}_t \tag{5.18}$$

The coverage vector, as reported in [See et al., 2017], is a non-normalized distribution over the document words that represents the degree of coverage that those words have received from the attention method. Highest the coverage score of a word is, more that word has been attended. Note that I did not set the maximum score of each cell to 1 because a word could be attended multiple times in the Summarization task.

I then modified the Equation 5.7 to include the coverage vector:

$$\begin{aligned} e_j^t &= \mathbf{v}^T \tanh(\mathbf{W}_e \left[\mathbf{s}_t || \mathbf{h}_j || \mathbf{cov}_t\right] + b_e) \\ a_i^t &= \frac{\exp(e_j^t)}{\sum_{k=1}^{M} \exp(e_j^t)} \end{aligned} \tag{5.19}$$

Since at the first decoder timestep the model has not generated an attention distribution yet, I defined the coverage vector as a zeros one.

Finally, I used the auxiliary loss proposed by See et al. (2017) to penalize the overlapping between the attention distribution and the coverage vector:

$$covloss = \sum_{i=1}^{N}\sum_{j=1}^{M} min(a_j^i, cov_j^i) \tag{5.20}$$

During my experiments with the coverage loss, I noticed that it tends to produce semantically wrong summaries. In my opinion, since the coverage loss has an impact on the attention, it indirectly impacts to the sentence-level attention and the sentence representations too, invalidating part of the training made by the model. I decided to use such loss at inference time in order to guide the model on selecting those words that minimize the overlap between the attention and the coverage vector. Thus, I have to modify Equation 5.13 to include the covloss. The problem, however, is how to rewrite the equation to reward the model when it generates a word that minimize such overlap, and penalizing it otherwise. I combined the covloss with the reinforcement learning q-function because this latter one has the wanted behaviour. In detail, the q-learning function allows the model to adapt to the environment, selecting the best action (in this case, the best word to generate) at each timestep. I have to specify that the proposed function is not a q-learning one, even if it is based on that; it does not calculate a score for each state (timestep) and action (generated word) as for those of reinforcement learning. I defined the score function, called *covscore*, as follows:

$$covscore_t = covscore_{t-1} + \frac{1}{t}(covloss_t - covscore_{t-1}) \tag{5.21}$$

which is added to Equation 5.13:

$$s(x, y) = \frac{logp(y)}{ln(y)} + covscore \tag{5.22}$$

5.4 Evaluation

I defined 6 models combining the three different sentence-level attention methods, the coverage vector, the auxiliary loss and the covscore function. In detail, I trained and tested the following models:

Seq2Seq + cosine-graph attention: It is the hierarchical model that uses the cosine-graph attention to compute the sentence-level scores;

Seq2Seq + cosine-pagerank attention: The hierarchical sequence-to-sequence model with the pagerank attention;

Seq2Seq + pagerank scores: the model computes the sentence-level scores through the sigmoid function. It is the model that gave the best results in the evaluation. For such reason, I used the coverage vector, the covloss and the covscore on it;

Seq2Seq + pagerank scores + coverage: It uses the coverage vector to reduce duplicated tokens in the generated summary;

Seq2Seq + pagerank scores + coverage + covloss: Previous model with the use of the coverage loss;

Seq2Seq + pagerank scores + coverage + covscore: Instead of using the coverage loss, it uses the covscore function at inference time.

I trained and tested them on *CNN/DailyMail* dataset [Hermann et al., 2015], which contains online articles paired with multi-sentence summaries. I used the script supplied by Nallapati et al. (2016) to obtain the dataset, which consists in 287,226 training pairs, 13,368 validation pairs and 11,490 test pairs. Each article has 781 tokens on average, while each summary has 3.75 sentences and 56 tokens on average. Differently from Nallapati et al., I used an *non-anonymized* version of the dataset[2].

The models have 256 dimensional hidden layer and 128 dimensional embedding layer. Following See et al. (2017), I used a small vocabulary comprises of 50k tokens for both source and target. I set the dumping factor λ to 0.9. All the weights are initialized with a normal distribution with mean 0 and standard deviation 0.1. The models are trained using AdaGrad [Duchi et al., 2011] with a learning rate of 0.15. I also used gradient clipping with a gradient norm of 2.0 and dropout [Srivastava et al., 2014] with probability 0.2 (keep probability of 0.8) to improve model generalization. I used the loss on the validation set for early stopping[3]. The training was performed on a single GPU RTX 2080Ti, with a batch size of 8. At testing time, I used a beam size of 5 and an α value of 1.4 for the length penalty. I set the maximum number of encoder input sentences to 8. I truncated the length of each sentence to 50 tokens, and the length of the summary to 100 tokens in order to speed up the convergence of the models. In detail, I started the training with an article length of 200 tokens and a summary length of 50 tokens, and I progressively increased those values.

I trained the models for about 1,200,000 iterations. The training of each model took about 5 days of computation.

5.4.1 Attention Analysis

Before looking to overall results obtained through Rouge scores, I would like to focus on the attention scores calculated by the models to understand how they learned.

[2]Please, see [Chen et al., 2016a] for problem regarding the anonymized version.

[3]See Appendix C for more details about dropout and early stopping.

During my experiments, I noticed that the sentence-level scores have a large impact on the word-level ones, focusing only onto the relevant words of the sentence. This is evident in the plots of the attention scores, where the model interactions come to light. In these plots, dark colors represent high attention scores (i.e., the model focused on that sentence or word), while bright colors represent low attention scores (i.e., the model lightly consider a sentence or a word).

In this section, I will analyze the plots produced by "*Seq2Seq + cosine-graph attention*", "*Seq2Seq + cosine-pagerank attention*" and "*Seq2Seq + pagerank scores*".

Cosine-Pagerank Attention

Figure 5.5 shows the sentence-level distribution for *Cosine-Pagerank Attention* method. From the picture, it is possible to see that the model was able to select the relevant sentence at each timestep. The only case where it selected more than one sentence is in the first three timesteps.

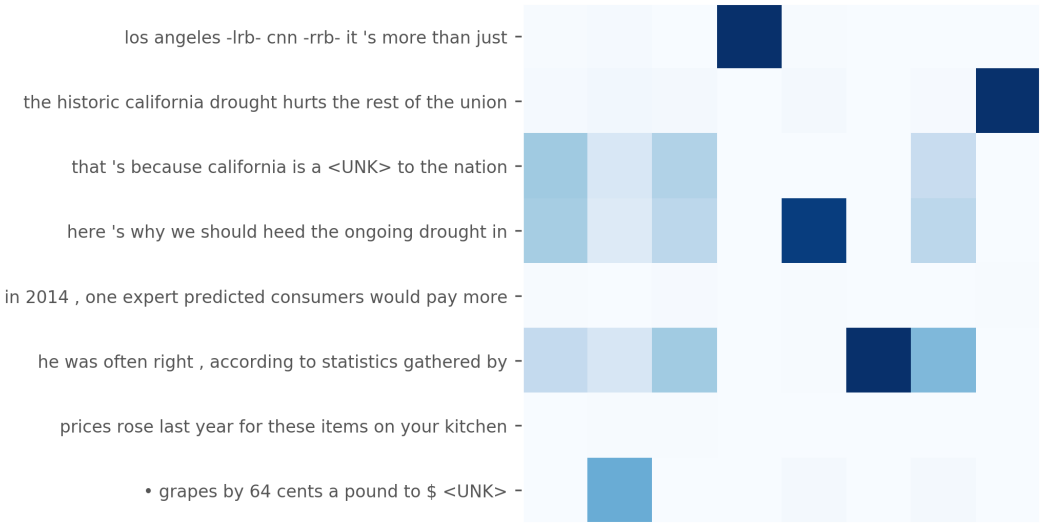


Figure 5.5: The figure shows the sentence-level distribution calculated using Cosine-Pagerank Attention method.

Looking to the word-level attention distribution depicted in Figure 5.6, it is possible to notice that the sentence-scores did not have a large impact on the word level. It smoothed them, leaving only a value associated to *California*.

An other example of sentence-level and word-level scores are depicted in Figure 5.7 and Figure 5.8 respectively. In the former figure, the model focused to multiple sentences (with different attention degree). This means it found those sentences relevant to construct the summary. However, such probability distribution did not improve the

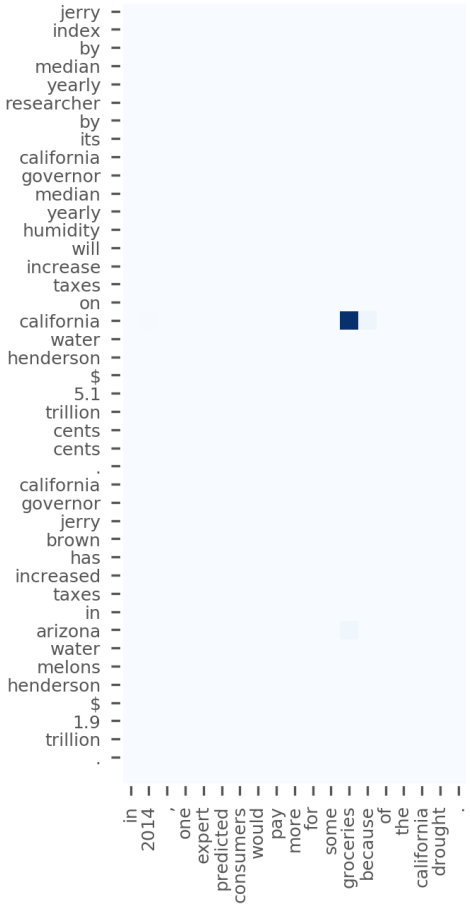


Figure 5.6: The figure shows the word-level distribution when it is combined with Cosine-Pagerank Attention method. In detail, the image represents the distribution on the fourth sentence. On the left column, the summary words; on the bottom, the sentence words.

word-level attention distribution. As reported by Figure 5.8, the model focused to the same words at different timesteps.

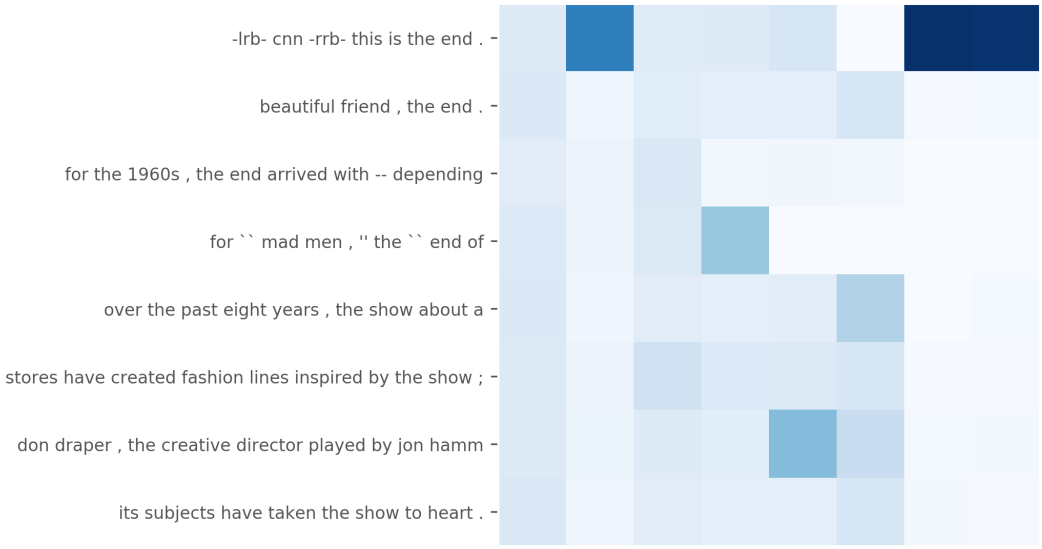


Figure 5.7: The figure shows the sentence-level distribution calculated using Cosine-Pagerank Attention method.

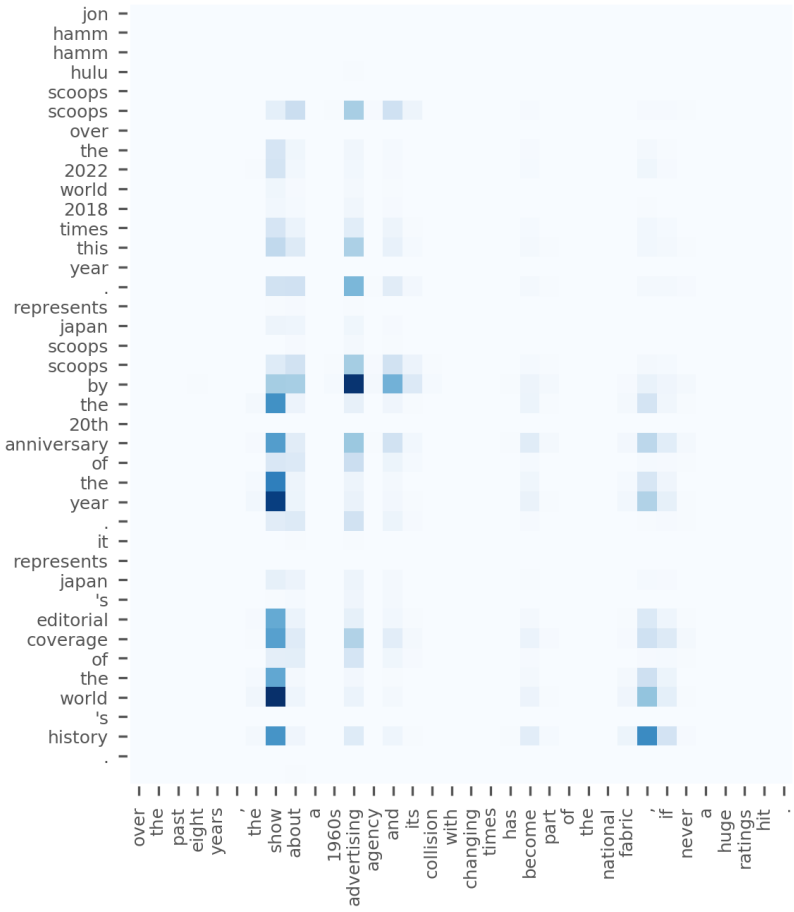


Figure 5.8: The figure shows the word-level distribution when it is combined with Cosine-Pagerank Attention method. In detail, the image represents the distribution on the fourth sentence. On the left column, the summary words; on the bottom, the sentence words.

Cosine-Graph Attention

Figure 5.9 shows the sentence-level distribution for *Cosine-Graph Attention* method. In the figure, there are 4 sentences that received an attention score at each timestep. Furthermore, the 4th, the 5th and the last sentences are the ones that received high scores.

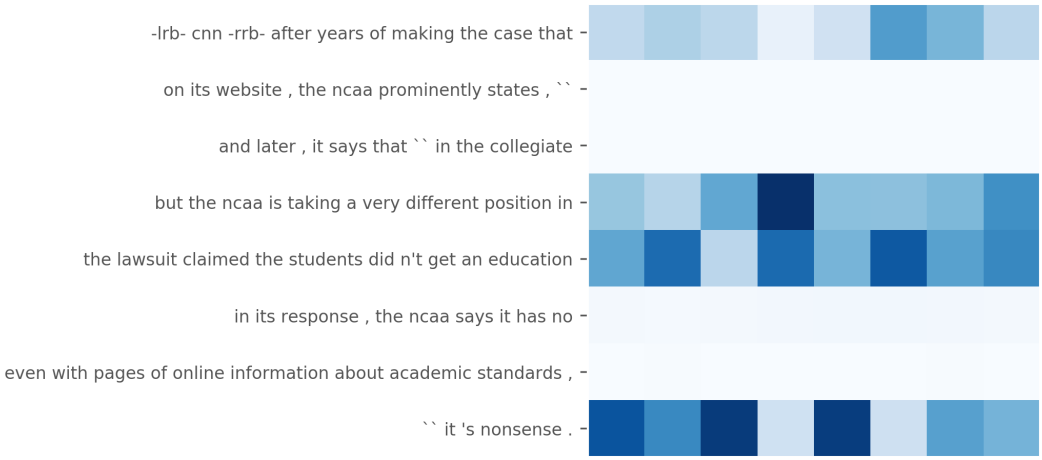


Figure 5.9: The figure shows the sentence-level distribution calculated using Cosine-Graph Attention method.

Such behaviour had an impact on the word-level attention distribution, where the model focused only on certain words, as depicted in Figure 5.10. More in detail, the model, despite the repetitions in the summary, focused on *ncaa*, *different*, *filed* and *athletes*. It is also possible to notice that the model focused on this latter word at almost each timestep.

I report another example of sentence-level and word-level attention distribution in Figures 5.11 and 5.12, respectively. The former figure is similar to Figure 5.9, where the model attended the same sentences. Figure 5.12 shows another example of word-level attention distribution, where the model focused on the same words at each timestep.

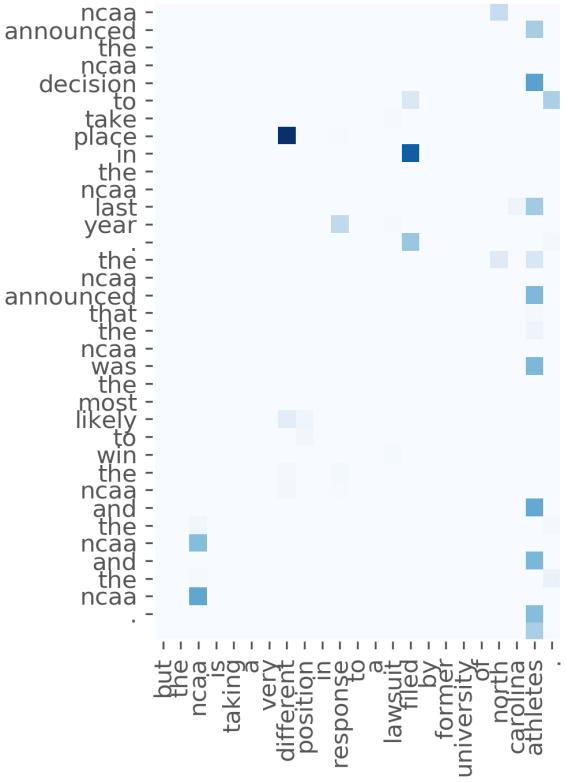


Figure 5.10: The figure shows the word-level distribution when it is combined with Cosine-Graph Attention method. In detail, the image represents the distribution on the third sentence. On the left column, the summary words; on the bottom, the sentence words.

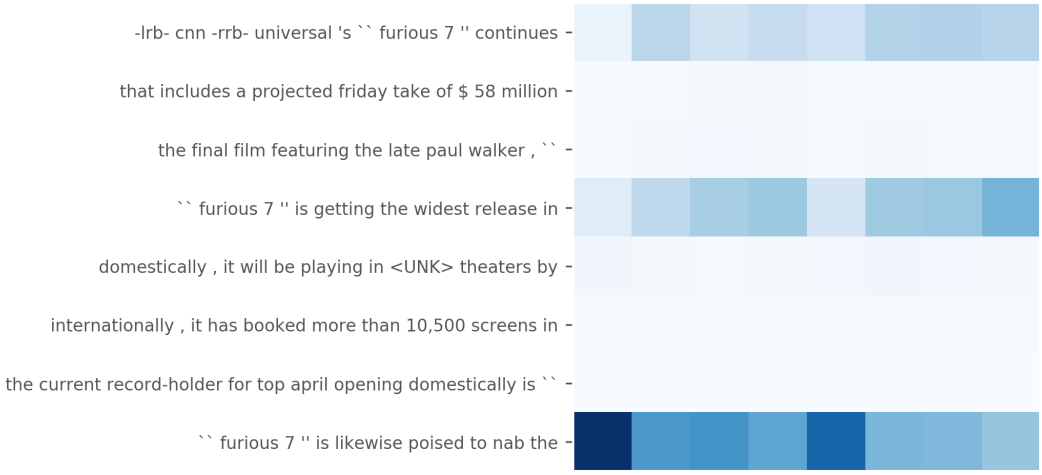


Figure 5.11: The figure shows the sentence-level distribution calculated using Cosine-Graph Attention method.

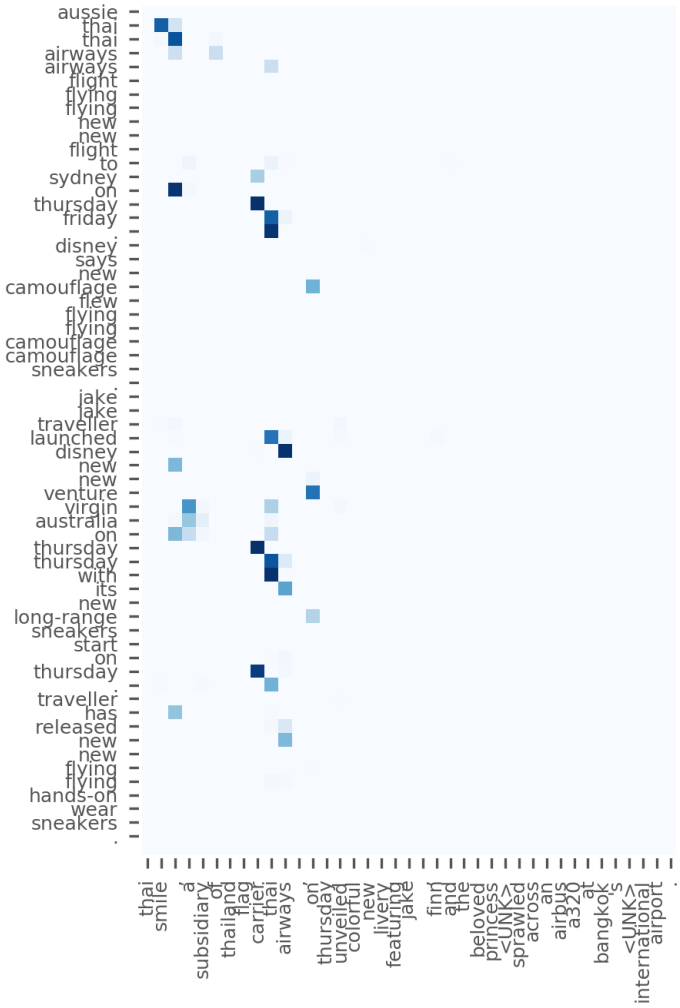


Figure 5.12: The figure shows a word-level distribution when it is combined with Cosine-Graph Attention method. The network was not able to focus on specific words, leading to multiple repetitions. On the left column, the summary words; on the bottom, the sentence words.

Pagerank Score Method

The previous plots reported that the model was not able to focus on words that are relevant for the summary. On one hand, the Cosine-Pagerank Attention was able to select the relevant sentence at each timestep, but this distribution terribly impacted on the word-level attention. On the other hand, the Cosine-Graph Attention was not able to select the relevant sentence, but it attended the relevant words. It generally focused on different sentences at each timestep, assigning the same score to those ones.

The problem of the previous methods is that they tried to select only one sentence at each timestep, but using a single sentence could not be sufficient to find salient words (or phrase excerpts) for the summary. The model could need two or more sentences to understand a passage. Furthermore, since Equation 5.8 includes all the attention scores of a word into the vocabulary distribution, the sentence-level attention could smooth, or in the worst case delete, those values.

For this reason, I substituted the softmax function in the sentence-level attention with a sigmoid one. This change had a positive impact on both sentence-level and word-level attention scores. As illustrated by the sentence-level scores of Figure 5.13 and Figure 5.14, the model was able to select the relevant sentence at each timestep (dark colors). It also gave a score to the other sentences according to their relevance for the summary.

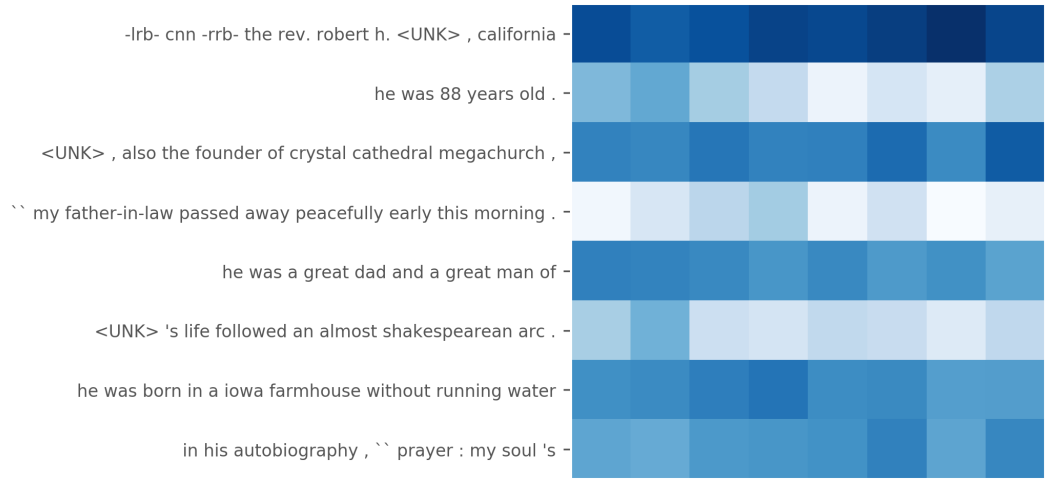


Figure 5.13: The figure shows the sentence-level distribution calculated using Pagerank Score method.

The scores also allowed the model to copy words from the document to the summary, as it possible to see from the word-level attention of Figure 5.15 and Figure 5.16.

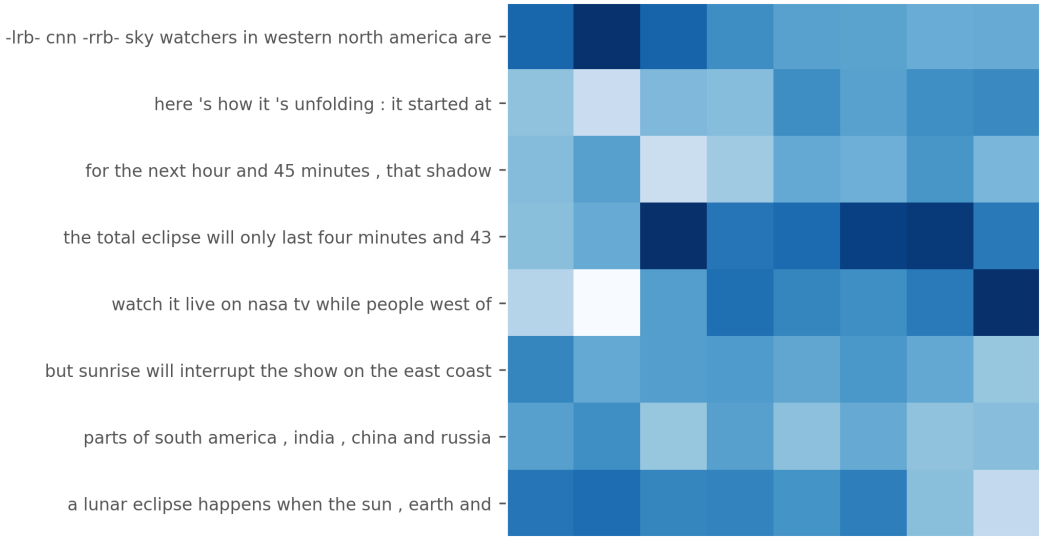


Figure 5.14: The figure shows the sentence-level distribution calculated using Pagerank Score method.

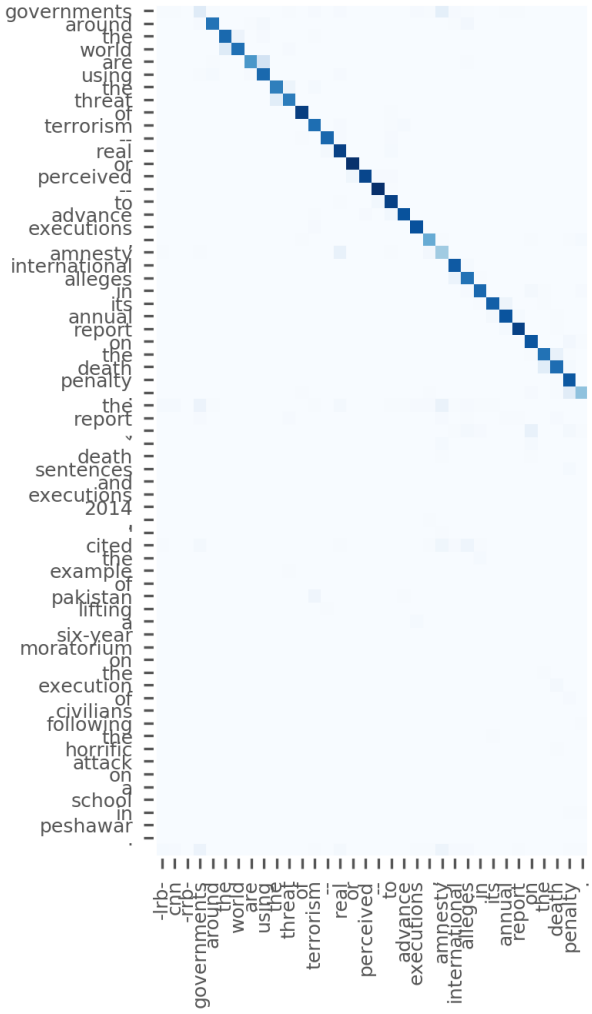


Figure 5.15: The figure shows the word-level distribution when it is combined with Pagerank Score method. On the left column, the summary words; on the bottom, the sentence words.

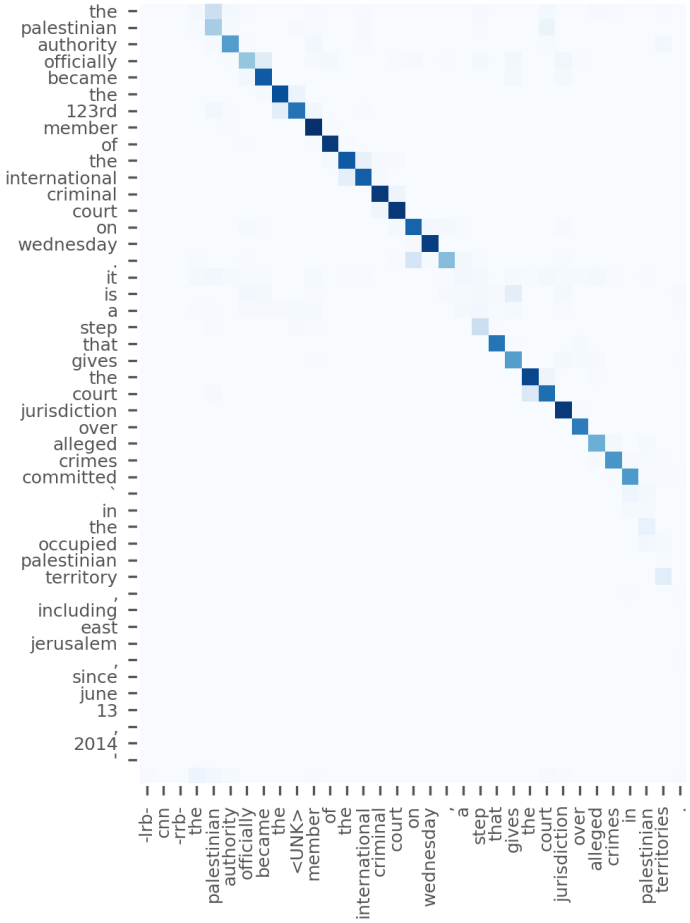


Figure 5.16: The figure shows the word-level distribution when it is combined with Pagerank Score method. On the left column, the summary words; on the bottom, the sentence words. From the figure, it is possible to see that the pointer network substituted the special token $<UNK>$ (that represents a token outside of the network vocabulary) with *123rd*.

5.4.2 Results

In this section, I will report Rouge scores (Rouge-1, Rouge-2 and Rouge-L) [Lin, 2004] of my models. I compared them with 5 state-of-the-art NN-based models to understand if they are able to learn the mapping function from the document to the summary. The state-of-the-art models are the following ones:

See et al. (2017)'s pointer generation model. It is a Sequence-to-Sequence model that combines the vocabulary distribution with the attention distribution to select Out-Of-Vocabulary or rare words.

Nallapati et al. (2016)'s hierarchical model. It is a hierarchical encoder followed by a RNN decoder. They combined the word attention distribution with the sentence one. Differently from my models, they used a simple attention method for the sentences.

Nallapati et al. (2016)'s pointer model. This model is similar to the hierarchical one, but with the difference that it uses a vanilla encoder and no sentence-level attention distribution. It also uses a pointer network to copy words from the input text to the summary.

Hsu et al. (2018)'s model. In their article, Hsu et al. propose a model that combines extractive with abstractive summarization. In detail, they first use a NN-based model to assign a score to each sentence. The score expresses if the sentence should be used to generate the summary or not. Then, they use a Sequence-to-Sequence model to generate the summary. In this latter one, the word-level attention scores and the sentence-level attention scores are combined together. Differently from my models where the sentence scores change according to the decoder state, their ones are fixed.

Tan et al. (2017)'s abstractive model. The model proposed by Tan et al. is a hierarchical encoder followed by a hierarchical decoder. To initialize the sentence-level decoder, they use a graph-based attention over the document sentences. In detail, the sentence-level attention is used to create a sentence context, which is fed in input to the word-level decoder as initial state.

Once I trained the models, I calculated their average summary length. The idea is that if the average summary length of a model is short, its Rouge scores will be low because only few tokens will be in common. With a long summary, the Rouge scores will be high because there is an high chance that a lot of tokens are present in the reference one. Table 5.1 reports the average summary length for the proposed models. It is possible to see that "*Seq2Seq + cosine-pagerank attention*" is the model with the shortest average summary length. The model closest to the average length of

the reference summaries is "*Seq2Seq + pagerank scores + coverage + covscore*", which has an average length of 55 tokens (1 token less than the reference one).

Model	Avg. summary length
Seq2Seq + cosine-graph attention	44.40 tokens
Seq2Seq + cosine-pagerank attention	43.70 tokens
Seq2Seq + pagerank scores	52.36 tokens
Seq2Seq + pagerank scores + coverage	54.22 tokens
Seq2Seq + pagerank scores + coverage + covloss	44.28 tokens
Seq2Seq + pagerank scores + coverage + covscore	55 tokens
Reference summary	56 tokens

Table 5.1: The table reports the average summary length for the models and the reference summary.

Table 5.2 reports the results obtained from all the models. Comparing the average summary lengths of Table 5.1 with the Rouge scores of Table 5.2, it is possible to notice that the models with the longest summaries obtained the best results. The models "*Seq2Seq + pagerank scores + coverage*" and "*Seq2Seq + pagerank scores + coverage + covscore*" obtained highest R-1 and R-2 than Nallapati et al.'s ones. I suspect that my models have a low R-L due to their high abstraction power: since the generated summaries contain tokens that are not present in the reference ones, the Longest Common Sequence is short.

On the other hand, both "*Seq2Seq + cosine-pagerank attention*" and "*Seq2Seq + cosine-graph attention*" have very low Rouge F1 scores. Those scores are inline with the attention plotting, i.e. that the two models neither learned to recognize the best sentence nor to focus on the relevant words and passages. In my opinion, the problem of those models is that the sentence-level attention had a disastrous impact on the word-level one, modifying those scores and invalidating the learning. I also suspect that keeping the scores of dissimilarity sentences in the matrix $\mathbf{A}$ do not help the model to discern the relevant sentences; on the other hand, the ReLU function helps to identify the sentence clusters and pick the relevant ones. Further, both "*cosine-pagerank attention*" and "*cosine-graph attention*"generated summaries with words not related to the document topic.

I noticed that the auxiliary loss did not improve the Rouge scores. Since the auxiliary loss modifies the attention scores, it also impacts on the sentence-level ones and shakes the sentence representations, invalidating part of the training. This is supported from the fact that the covloss incremented the loss on the validation set from 3.5 to 3.8, instead of decreasing it.

Giving those results, I can conclude that the proposed graph-based sentence-level attention generally performs better than the sentence-level attention proposed by Nal-

lapati et al. (2016). It allows to generate more abstractive summaries as reported in Section 5.4.2, at the cost of sacrificing recall. More in detail, since Rouge only evaluates the presence of tokens of the generated summary in the reference one, it indirectly penalizes models that use synonyms and periphrasis. Indeed, my best model has a very high precision (41.76 for R-1, 16.27 for R-2 and 27.16 for R-L) but a very low recall (33.22 for R-1, 13.03 for R-2 and 21.50 for R-L).

This is also the reason why my models have lower Rouge scores than the ones of See et al.. Those latter models are more oriented on copy words from the input document than generating novel ones, as it possible to notice comparing their novelty depicted in Figure 5.19, which is close to 0, with mine in Figure 5.18.

Finally, my models under-perform respect to Hsu et al. (2018) and Tan et al. (Tan et al.) ones. This is an expected result since their ones have a more complex architecture and use more complex features.
The model of Hsu et al. is composed of two trained models: an extractive model to select the sentences that are useful for the summary and an abstractive one to digest them; mine, instead, are jointly trained to compute a score on each sentence and generate the summary. It does not know which sentence is better for the summary.
The model of Tan et al. uses both a sentence-level attention and a word-level attention, but in a different way. Their model uses an hierarchical decoder to generate the summary. First, it generates a decoder state for each sentence of the summary using the sentence-level attention; then, it generates the words of a sentence using the word-level attention and the pointer-network. Compared to my models that only combines both attentions and require less resources to be trained, their one is resource intensive because the hierarchical decoder adds a further million parameters (neurons), which have a positive impact on the results.

Model	R-1	R-2	R-L
See et al.'s pointer generation	29.7	9.21	23.24
See et al.'s pointer generation with coverage	36.44	15.66	33.42
Nallapati et al.'s pointer model†	32.1	11.7	29.2
Nallapati et al.'s hierarchical model†	31.8	11.6	28.7
Hsu et al.'s model	**40.68**	**17.97**	**37.13**
Tan et al.'s model	38.1	13.9	34.0
Seq2Seq + cosine-pagerank attention	15.03	2.70	11.03
Seq2Seq + cosine-graph attention	15.92	2.84	11.51
Seq2Seq + pagerank scores	26.77	9.67	19.04
Seq2Seq + pagerank scores + coverage	34.50	13.47	22.26
Seq2Seq + pagerank scores + coverage + covscore	**34.67**	**13.61**	**22.36**
Seq2Seq + pagerank scores + coverage + covloss	25.14	7.56	16.78

Table 5.2: The table reports the Rouge-1 (R-1), Rouge-2 (R-2) and Rouge-L (R-L) F1 scores. The symbols † means that the model cannot be directly compared to mine because it has been trained and tested on the anonymized version of the dataset.

I report an example of summaries generated by my models in Table 5.3, where it is possible to note that "*Seq2Seq+pagerank scores*" generated a summary rich of details. Furthermore, the model copied the first sentence of the source article in the summary. The model "*Seq2Seq+cosine-graph attention*" duplicated the same sentence multiple times in the summary, while the one generated by "*Seq2Seq+cosine-pagerank attention*" is difficult to read due to the repetitions. The model also substituted "*amnesty international*" with "*national geographic*". The adoption of the coverage vector removed a small portion of the first sentence from the summary of the "*Seq2Seq+pagerank scores*" model, leaving the rest intact. On the other hand, the use of the coverage loss substantially modified the summary, focusing on the use of terrorism and the number of executions. Finally, the covscore function kept the generated summary of the coverage-based model, adding a sentence to the end of it.

Source Document governments around the world are using the threat of terrorism – real or perceived – to advance executions, amnesty international alleges in its annual report on the death penalty. “the dark trend of governments using the death penalty in a futile attempt to tackle real or imaginary threats to state security and public safety was stark last year,” said salil shetty, amnesty’s secretary general in a release. “it is shameful that so many states around the world are essentially playing with people’s lives – putting people to death for ‘terrorism’ or to quell internal instability on the ill-conceived premise of deterrence.” [...]
Human Summary amnesty ’s annual death penalty report catalogs encouraging signs, but setbacks in numbers of those sentenced to death. organization claims that governments around the world are using the threat of terrorism to advance executions. the number of executions worldwide has gone down by almost 22 compared with 2013, but death sentences up by 28.
Seq2Seq + cosine-pagerank attention report published by the national security agency in the national geographic report. the report was compiled by the national institute of national security. the report was compiled by the national institute of national security.
Seq2Seq + cosine-graph attention a number of executions in the united states has risen by 17 per cent in the past decade. it was the first time in a history of violence in the united states. it was the first time in a history of violence in the united states.
Seq2Seq + pagerank scores governments around the world are using the threat of terrorism – real or perceived – to advance executions, amnesty international alleges in its annual report on the death penalty. the report, ‘death sentences and executions 2014,’ cited the example of pakistan lifting a six-year moratorium on the execution of civilians following the horrific attack on a school in peshawar.
Seq2Seq + pagerank scores + coverage amnesty international alleges in its annual report on the death penalty. the report, “death sentences and executions 2014,” cited the example of pakistan lifting a six-year moratorium on the execution of civilians following the horrific attack on a school in december.
Seq2Seq + pagerank scores + coverage + coverage loss governments around the world are using the threat of terrorism – real or perceived – to advance executions, amnesty international says. the report, ‘death sentences and executions 2014’ cited the example of pakistan lifting a six-year moratorium on the execution of civilians.
Seq2Seq + pagerank scores + coverage + covscore amnesty international alleges in its annual report on the death penalty. the report, “death sentences and executions 2014,” cited the example of pakistan lifting a six-year moratorium on the execution of civilians following the horrific attack on a school in december . amnesty has gone down by almost 22% on the previous year.

Table 5.3: The table shows the summaries generated by my models, and the original one created by the human.

Table 5.4 reports another example. From the table, it emerges that "*Seq2Seq + pagerank scores*" produced again a good summary. There is only a small error in the summary, caused by the phrase excerpt "*somalia's internationally recognized government that had been under pressure from al-shabaab*". The summary generated by the other two models, instead, contains several errors and repetitions. In "*Seq2Seq+cosine-graph attention*", the model repeated the sentence "*al-shabaab has been linked with a string of deadly attacks in somalia*", substituting the subject with *kenyan*. In "*Seq2Seq + cosine-pagerank attention*", there is the repetition of the first and second sentences. The adoption of the coverage vector corrected the errors made by "*Seq2Seq + pagarank scores*" model, while the use of coverage loss lightly improved the summary. The use of the covscore function modified part of the summary, including more details about the "beliefs" of the terrorist group.

Source Document the terrorist group al-shabaab has claimed an attack on garissa university college in eastern kenya, in which many people have been killed and still more taken hostage. the attack is another step in the ongoing escalation of the terrorist group's activities, and a clear indicator that the security situation in east africa is deteriorating fast. somalia-based al-shabaab has been behind a string of recent attacks in kenya, the most well-known of them being the massacre at the westgate shopping centre in nairobi in 2013. [...]
Human Summary terrorist group al-shabaab has attacked a kenyan college, killing and taking hostages. it is a clear indicator the security situation in east africa is deteriorating, says stefan wolff. more than military action aloe is needed to combat terrorism in the region, he says.
Seq2Seq + cosine-pagerank attention al-shabaab has been linked with attacks in somalia and syria. the group has been linked to a string of attacks in somalia. the group has now been linked with a move to somalia.
Seq2Seq + cosine-graph attention al-shabaab has been linked with a string of deadly attacks in somalia. the kenyan government has been linked with a string of attacks. al-shabaab has claimed responsibility for the attack in kenya.
Seq2Seq + pagerank scores al-shabaab has been behind a string of attacks in kenya, the most well-known of them being the massacre at westgate shopping centre in part of somalia's internationally recognized government that had been under pressure from al-shabaab. the attack is another step in the ongoing escalation of the terrorist group's activities, and a clear indicator that the security situation in east africa is deteriorating fast.
Seq2Seq + pagerank scores + coverage al-shabaab has been behind a string of recent attacks in kenya. the attack is another step in the ongoing escalation of the terrorist group's activities, and a clear indicator that the security situation in east africa is deteriorating fast.
Seq2Seq + pagerank scores + coverage + coverage loss al-shabaab has been behind a string of recent attacks in kenya. it has yet to recover almost quarter of a century later this year. the attack is another step in the ongoing escalation of terrorist group's activities, and a clear indicator that security situation in east africa is deteriorating.
Seq2Seq + pagerank scores + coverage + covscore al-shabaab has been behind a string of recent attacks in kenya. the attack is another step in the ongoing escalation of the terrorist group's activities. al-shabaab is predominantly driven by the same radical interpretation of the koran as al-qaeda and isis (also known as islamic state).

Table 5.4: Another example of the summaries generated by my models. In this example, "*Seq2Seq + pagerank scores*" generated a phrase excerpt that is not related to the summary.

Table 5.5 reports some interesting summaries generated by the models. Both "*Seq2Seq + cosine-pagerank attention*" and "*Seq2Seq + cosine-graph attention*" generated a summary that is not related to input document. The models are not able to generalize, and combined the term *died* - related to the document - with *nancy regan* or *jfk*. "*Seq2Seq + cosine-pagerank attention*", instead, repeated the same sentence multiple times. "*Seq2Seq + pagerank scores*" is the only model that was able to generate a syntactically and semantically correct summary. The only error of the model is at the start of the first sentence, where it generated a phrase excerpt that is erroneous with respect to the remaining text. Again, the use of the coverage vector slightly improved the summary. However, the model started to associate Anna Frank with the phrase excerpt *"her husband"* which is semantically incorrect. The use of the coverage loss corrected some errors, but it added the sentence "*she was separated from their mother and sent away to work as slave labor at the camp in germany*" that is not relevant for the summary. Finally, the use of the covscore function corrected the errors of the "*Seq2Seq + pagerank scores + coverage*" model, producing a more concise summary.

Source Document seventy years ago, anne frank died of typhus in a nazi concentration camp at the age of 15. just two weeks after her supposed death on march 31, 1945, the bergen-belsen concentration camp where she had been imprisoned was liberated – timing that showed how close the jewish diarist had been to surviving the holocaust. but new research released by the anne frank house shows that anne and her older sister, margot frank, died at least a month earlier than previously thought. [...]
Human Summary read more about anne frank's cousin, a keeper of her legacy. museum: anne frank died earlier than previously believed. researchers re-examined archives and testimonies of survivors. anne and older sister margot frank are believed to have died in february 1945.
Seq2Seq + cosine-pagerank attention nancy reagan died at the age of 81, aged 88. she was found dead at the age of 81, aged 88. she was found dead at the age of 81.
Seq2Seq + cosine-graph attention jfk died peacefully at the age of 81. she was found dead at the scene of the death of her late grandmother. she was found dead in the house of her late grandmother 's house.
Seq2Seq + pagerank scores anne frank's final entry that same year, anne frank died of typhus in a nazi concentration camp at the age of 15. but new research released by the anne frank house shows that anne and her older sister margot frank died at least a month earlier than previously thought.
Seq2Seq + pagerank scores + coverage anne frank's final entry that same year, anne and her older sister, died at least a month earlier than previously thought. anne and her husband were separated from their mother and sent away to work as slave labor at the age of 15.
Seq2Seq + pagerank scores + coverage + coverage loss seventy years ago, anne frank died of typhus in a nazi concentration camp at the age of 15. she was separated from their mother and sent away to work as slave labor at the camp in germany.
Seq2Seq + pagerank scores + coverage + covscore anne frank died of typhus in a nazi concentration camp at the age of 15. she died at least a month earlier than previously thought. she had been imprisoned since 1945, and died in 1945.

Table 5.5: The table shows another example of the summaries generated by my models. Differently from the previous one, this reports some errors of the models.

Figure 5.17 reports the average percentage of duplicate n-grams. The measure gives

an insight both on the reasons behind the low Rouge F1 scores and on the impact of the coverage vector; in detail, a model with duplicate tokens in the summary could have low Rouge F1 scores because those repetitions prevent the generation of relevant terms while stretching the summary until it satisfies the minimum length. I computed the average percentage of duplicate n-grams as follows: for each method, I counted the number of repeated n-grams (from 1-grams to 4-grams) in the generated summaries, and I divided it by the length of the summary. From the figure, it is possible to notice that "*Seq2Seq + cosine-pagerank attention*" has the highest number of repeated 1-grams, followed by "*Seq2Seq + pagerank scores*". This latter model has also the highest number of repeated 2-grams, 3-grams and 4-grams. The application of the coverage vector on this model reduced the repetitions. Such value is further reduced with the application of the coverage loss, but it did not improve the Rouge scores as reported by Table 5.2. Finally, the application of the covscore function produced summaries with an average n-gram repetition close to the reference ones.

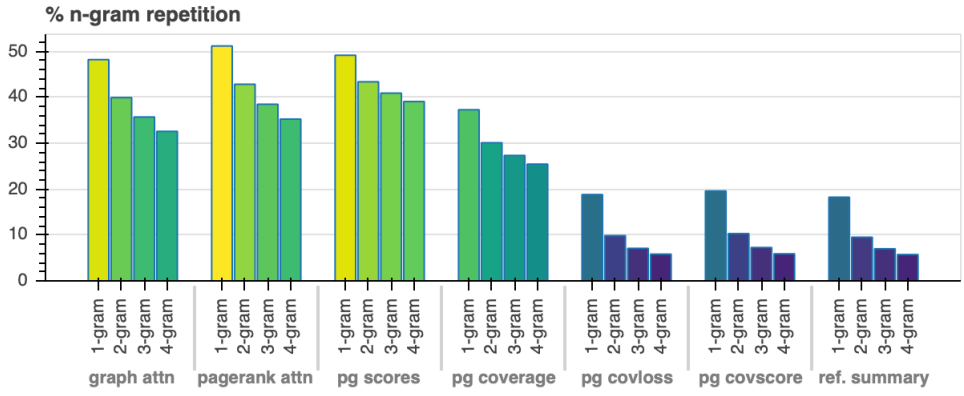


Figure 5.17: The figure shows the average percentage of repeated n-grams (from 1-gram to 4-gram) for each sentence-level attention method and the reference summary.

Ablation Study

In the description of the model, I suggested the use of the length penalty $lp(\cdot)$ to improve the summary. Such function controls the length of the summary through a parameter α in the range $(0, \infty)$; an high value forces the beam search to generate long summaries, while a low one forces to have short summaries. But this function really improves the Rouge scores of the model with respect to Equation 5.11? Is it better to use an high α value or a low one?

In this section, I perform an ablation study to answer those questions. For the study, I will compare 4 models based on "*Seq2Seq + pagerank scores + coverage*" with different settings on the score function. In details, the settings are:

no-lp: this model does not use the length penalty on the beam search, but only the score function of Equation 5.11. I will use this model as a baseline to analyze the improvement of the length penalty.

0.6 lp: It uses the length penalty with α sets to 0.6. I set a low value to check if forcing the beam search to generate short summary could improve their informativeness.

1.0 lp: I set α to 1.0 to analyze if the exponent has some sort of impact on the lp function.

1.4 lp: I set α to 1.4 to verify if long summaries are more informative than short ones. Since the model is forced to generate long sentences (and in general, more sentences), it has to use all the words it finds relevant.

Table 5.6 reports the average length of the generated summaries. The value α had an impact on the average length: with α set to 0.6, the beam search produced summaries with an average length of about 47.91 tokens, while with the value 1.4 it produced summaries with an average length of about 55 tokens.

Model	setting	Avg. summary length
Seq2Seq + pagerank scores + coverage	no-lp	54.12 tokens
Seq2Seq + pagerank scores + coverage	0.6 lp	47.91 tokens
Seq2Seq + pagerank scores + coverage	1.0 lp	54.21 tokens
Seq2Seq + pagerank scores + coverage	1.4 lp	55.18 tokens
Reference Summaries	-	56 tokens

Table 5.6: The table reports the average summary length for the different settings.

Table 5.7 reports the Rouge scores of the 4 tested models, where it is possible to notice that the use of the length penalty substantially improved the quality of the summaries, increasing the scores of about 3 points for R-1, 2 points for R-2 and 1 point for R-L. The second interesting aspect is that the length penalty boosts the performance only when α has a value different from 1.0. For this latter case, there is no significant difference from the *no-lp* setting. Finally, the highest Rouge scores are obtained with the *1.4 lp* setting, meaning that the model was able to include all the salient words in the summary.

Model	setting	R-1	R-2	R-L
Seq2Seq + pagerank scores + coverage	no-lp	29.14	11	20.15
Seq2Seq + pagerank scores + coverage	0.6 lp	33.52	13.05	21.98
Seq2Seq + pagerank scores + coverage	1.0 lp	29.23	11.14	20.23
Seq2Seq + pagerank scores + coverage	1.4 lp	34.50	13.47	22.26

Table 5.7: The table reports the Rouge-1 (R-1), Rouge-2 (R-2) and Rouge-L (R-L) F1 scores for the different beam search settings.

Analyzing the Rouge scores in comparison to the average summary length, it is interesting to notice that the length penalty not only impacted on the length of the summaries, but also on their quality. Setting α to 0.6 led to short (about 47 tokens) and informative summaries, while increasing it to 1.4 had only a small impact on the Rouge scores, despite it produced the longest ones. The only borderline case is α set to 1, where the results are the same of the *no-lp* setting.

Abstractiveness of the models

One important point to evaluate is the capability of the models to generate abstractive summaries. This implies complex operations that a model has to be capable to do, such as using synonyms and making periphrasis, which could decrease the likelihood of matching the reference (gold) summaries. To evaluate the abstractiveness of the models, I decided to calculate the average percentage of novel n-grams, which measures the capability of the model to generate n-grams that are not present in the reference summary. A low percentage of this measure means that the model is more conservative and tends to use words that appear in the document, while an high percentage means that the model is abstractive and tends to generate summaries that contain synonyms and periphrasis.

Figure 5.18 depicts the percentage of novel n-grams present in the generated summaries. I computed these percentages in the following way: for each n-gram (1-gram to 4-gram), I counted how many (unique) n-grams of the generated summary are not present in the reference one. Then, I divided such count by the total number of (unique) n-grams in the generated summary. The figure shows that "*Seq2Seq + cosine-pagerank attention*" and "*Seq2Seq + cosine-graph attention*" have the highest number of novel n-grams, surpassing 60% for the 1-grams and being close to 90% for the 4-grams. In my opinion, those two methods have a lot of novel n-grams because they tend to generate words that are not related to the input document, and thus not present in the reference summary (see Table 5.5 for an example). "*Seq2Seq + cosine-pagerank score*", instead, has less novel n-grams percentage than the other two methods because its summaries are strictly related to the input document. Further, the number of novel n-grams remains unchanged with the application of the coverage vector, meaning that the model used all

words that it considered relevant for the summary. Finally, the coverage loss increased the number of novel n-grams. Such increment, however, is due to generation errors of the model, i.e. the model tends to generate summaries that contain words not related to the topic of the document. The application of the covscore slightly increased the number of novel n-grams.

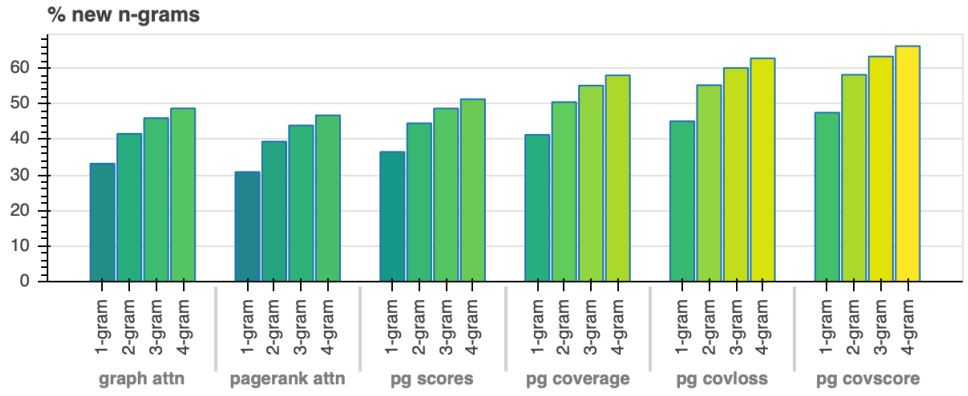


Figure 5.18: The figure shows the average percentage of novel n-grams (from 1-ngram to 4-gram) for each sentence-level attention method.

Comparing the novelty of the models to the one of See et al. (2017) reported in Figure 5.19, it is possible to notice that their pointer network has a n-gram novelty close to 0 (with a peak of about 15% on 4-gram), while mine varies from 55% to 80%. The ability of their model to copy as much as possible from the input document, while using the vocabulary distribution only to connect the phrase excerpts, allowed them to obtain high Rouge scores at the cost of reducing synonyms and periphrases, i.e. n-grams that are not present in the reference summary. Further, those copied phrase excerpts boost the recall because all the relevant words of the document are in the generated summary. On the other hand, my models rely on the vocabulary distribution, copying from the input document only when they found it is necessary. As a consequence, they have a very high novelty that leads to low recall.

I also computed the average β value of Equation 5.8 to check if the models are more oriented towards the vocabulary or the pointer network. Table 5.8 reports the average β value for each model. The models "*Seq2Seq + cosine-pagerank attention*" and "*Seq2Seq + cosine-graph attention*" have a low average β value, denoting that they largely use the pointer network to copy words (compared to the other models). They also have a standard deviation of 0.15, which expresses how the models oscillates between the use of the vocabulary and the pointer network. The β value increases with the adoption of the *pagerank scores* method and its variations, reaching the peak with the "*Seq2Seq + pagerank scores + coverage + coverage loss*" model. This latter one

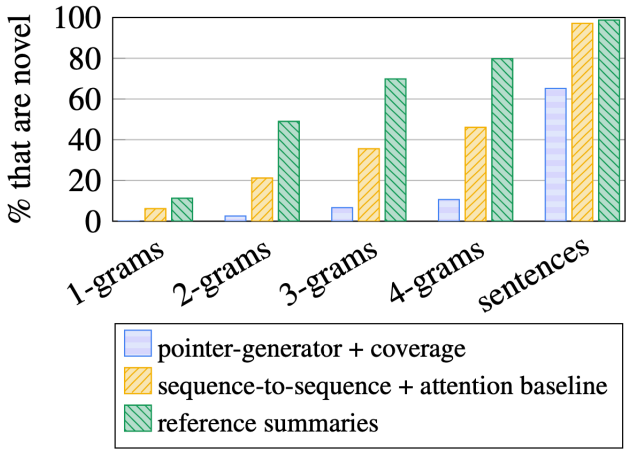


Figure 5.19: The figure shows the novelty of See et al.'s model [See et al., 2017]. The image is taken from their article.

has an average β value of 0.95, meaning that the model seldom uses the pointer network to copy relevant words.

Model	Avg. β value
Seq2Seq + cosine-pagerank attention	0.84 ± 0.15
Seq2Seq + cosine-graph attention	0.84 ± 0.15
Seq2Seq + pagerank scores	0.92 ± 0.04
Seq2Seq + pagerank scores + coverage	0.93 ± 0.03
Seq2Seq + pagerank scores + coverage + covscore	0.93 ± 0.03
Seq2Seq + pagerank scores + coverage + coverage loss	0.95 ± 0.03

Table 5.8: The table reports the average β value for each model. High β values denote that the model tends to use of the vocabulary distribution in the generation of the next word, while low β values denote that the model tends to copy words from the input document through the pointer network.

Finally, I computed the average number of summary sentences that have a phrase excerpt copied from the input document. I obtained such percentage comparing each summary sentence with each document one via Longest Common Subsequence (LCS from now on). If the output of the LCS is greater than 15 tokens[4] (i.e., the summary sentence contains more then 15 tokens that appear in the document one), I marked the

[4]I defined this threshold empirically, analyzing the LCS outputs for the different models.

sentence as copied. Then, I normalized the number of copied sentences on the number of summary sentences.

Figure 5.20 reports the results of this evaluation. Both "*Seq2Seq + pagerank scores*" and "*Seq2Seq + pagerank scores + coverage*" have a percentage close to 50, i.e. half of their summary sentences contain phrase excerpts of the input document. The high percentage of the former model is due to the phrase repetitions in the output summary (i.e., repeating the same sentence multiple times); indeed, it is one of the three models with the highest number of repetitions, as reported by Figure 5.17. The adoption of the coverage vector slightly reduced such percentage, but it remains high due to repetitions in the summary.

Comparing Figure 5.20 with Figure 5.17, it is possible to notice that both results are aligned: if the number of repetition is high, then the number of sentences with copied phrase excerpts is high too. In fact, the percentage decreased with the adoption of the coverage loss and the covscore, that penalize repetitions in the output summary. In the former one, the percentage is lower than 40, while it slightly decreases for the latter one, being 45% . However, I have to say that those percentages will never be close to zero because the model will always copy words from the document, either via the pointer network or the vocabulary distribution, when it found a relevant phrase excerpt for the summary. The only contradictory cases are "*Seq2Seq + cosine-pagerank attention*" and "*Seq2Seq + cosine-graph attention*", caused by both the inability of these models to generate proper summaries and the high number of repetitions in the output.

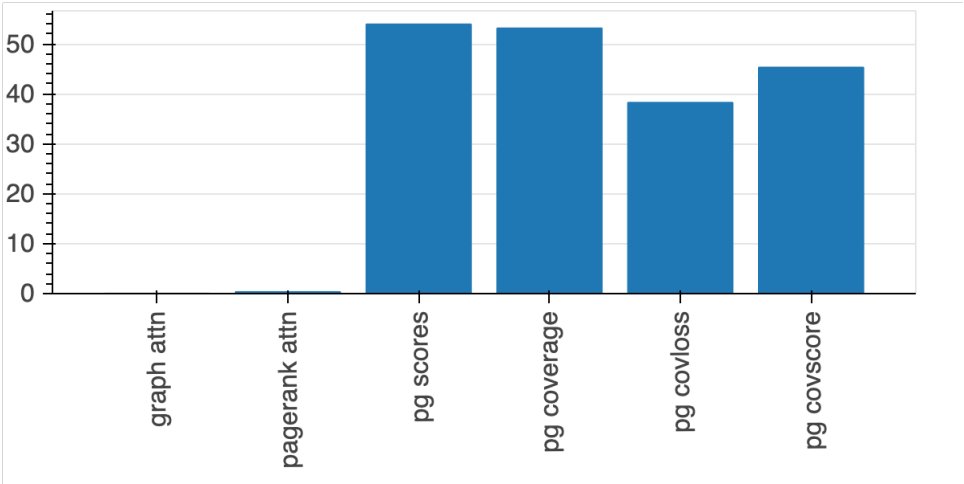


Figure 5.20: The table shows the average number of summary sentence that have a phrase excerpt copied from the input document.

5.5 Final Considerations

In this chapter, I presented a Neural Network-based model for Summarization. The model uses an hierarchical encoder to obtain both word and sentence representations. Then, it computes a score for each sentence that expresses its importance with respect to the other ones. Those scores are used to adjust the word-level attention ones, ensuring that the network will focus only on the important words. Further, since I used a pointer network model, the network will only copy in the summary those words coming from the informative sentences.

I proposed three methods to calculate the sentence-level attention scores, each one based on a graph-based NN [Veličković et al., 2017; Tan et al., 2017; Klicpera et al., 2018; Yao et al., 2019]. Such network allows to keep into account both the decoder state and the other sentences that are similar to the consider one. Following Tan et al. (2017), I based two methods on Pagerank, while the remaining one is a classic graph-based NN. For all these methods, I defined the adjacent matrix through cosine similarity to capture similar sentences and to select the best one. I called the three attention methods: *cosine-pagerank attention*, *cosine-graph attention* and *pagerank scores*.

From the analysis of the attention plots, I noticed that the methods *cosine-pagerank attention* and *cosine-graph attention* were not able to compute a correct word-level attention, negatively impacting on the model. This impact was confirmed by Table 5.2, where the two methods obtained very low Rouge scores [Lin, 2004]. I think that both methods did not work well due to: (i) the negative weights in the adjacency matrix $\mathbf{A}$: the presence of those weights distracted the method in finding the relevant sentences, worsting the performance; and (ii) the softmax function: in some cases, this function could assign all the probability mass to a single sentence (see Figure 5.5 for a real case), deleting the other ones. This issue had a bad impact on the word-level attention, as it is possible to see in Figure 5.6. The *cosine-graph attention* method was able to contrast part of the issues thanks to the learnable parameter, slightly increasing the Rouge scores.

The *pagerank scores* method, instead, was able to compute a correct sentence-level and word-level attention. For the former one, the method was able to select the best sentences at each timestep. It also gave a small score to sentences that are not salient for the summary to copy some words in output via the pointer network (since this one is based on the attention scores). These scores had a positive impact on the word-level attention, allowing the model with the *pagerank scores* method to focus on the relevant words. I also noticed that the model copied some phrase excerpts of the document in the generated summary.

The application of the coverage vector to this model boosted the Rouge scores to 34.67 for R-1, 13.61 for R-2, and 22.36 for R-L, performing better than the models of Nallapati et al. (2016). However, my models have lower scores (about 2 Rouge points) than See et al. (2017)'s models; I found that this is due to their high abstractive power, i.e. the ability of using synonyms and periphrases. In detail, since Rouge is a metric

that only evaluate the presence of tokens of the generated summary in the reference one, it indirectly penalizes abstractive models. Indeed, my best model has a very high precision (41.76 for R-1, 16.27 for R-2 and 27.16 for R-L), but a low recall (33.22 for R-1, 13.03 for R-2 and 21.50 for R-L) because it is not able to return all those terms that are present in the reference summaries. The abstractive power of my models has been confirmed by both the novelty scores (see Figure 5.18) and the β values (see Table 5.8).

Chapter 6

Conclusion

In this thesis, I started describing the Summarization task in Chapter 2, where I reported the different subtasks that have emerged after the seminal work of Luhn (1958). I divided those subtasks in supervised and unsupervised ones. Supervised tasks are based either on Support Vector Machines (or Naive Bayes) or Neural Networks. In the former case, features like TF-IDF and Part-Of-Speech tags are used to assign a value to each sentence according to its relevance for the summary; high score sentences are then selected to form the summary. In the latter case, an Encoder-Decoder Neural Network model is used to generate the summary. Such model reads the input document through the encoder, producing a document representation; then, the representation is used to generate the summary word-by-word.

I described the Neural Network models in Chapter 3, covering the existing architectures, as Recurrent Neural Networks models (in Section 3.7) and their Encoder-Decoder variation. I then provided a wide review of recent research trends on Neural Network-based Text Summarization in Chapter 4. I divided it in three macro-blocks: coverage methods in Section 4.2, pointer network models in Section 4.3 and document content methods in Section 4.4. The first two sections illustrate the solutions to two important Encoder-Decoder model drawbacks: word repetitions in the output and the impossibility of generating Out-Of-Vocabulary (OOV) words, respectively. Coverage methods create a sort of memory that keeps into account the previous generated words. In this way, the network does not focus on the same document words (or sentences), producing repetitions in the output. Pointer networks [Vinyals et al., 2015], instead, create a probability distribution over the document words, from which one can be selected and copied in output. Some researchers applied them only to copy rare words or OOV words [Nallapati et al., 2016; Miao and Blunsom, 2016; Merity et al., 2017], while others used them to create a mixture model [Gu et al., 2016; Gulcehre et al., 2016; See et al., 2017]. Finally, the latter section, document content methods, describes a set of methods used to analyze the document content. More in detail, they are divided in two research directions: methods based on templates, and methods that exploit the salient content of the

document. The former is based on template-based Text Summarization, where human-written templates are used to generate the summary. Those templates contain particular slots that have to be filled with document words or phrase excerpts [Zhou and Hovy, 2004; Chen and Bansal, 2018; Cao et al., 2018; Wang et al., 2019]. The latter one, instead, contains different approaches: mixing extractive and abstractive techniques to extract the relevant sentences and use them to create the summary [Hsu et al., 2018]; hierarchical encoders to capture the relevance of each document sentence in order to either initialize the decoder or re-score the word-level attention [Tan et al., 2017; Li et al., 2018a]; or word-level masks to filter out irrelevant words [Zhou et al., 2017; Gehrmann et al., 2018].

Finally, I described the proposed Neural Network-based model in Chapter 5. It is composed of an hierarchical encoder followed by a decoder that combines sentence-level attention scores with the word-level ones. The idea is to overcome the problem of the model of distinguishing relevant and informative sentences for the summary from those that are not. More in detail, the combination of those two-level representations highlights which sentences and words are relevant for the summary such that words coming from salient sentences are more likely to be copied in output, and that the model poses its focus only onto the relevant text passages. I proposed three sentence-level attention methods, each one based on a graph-based Neural Network because it captures the relevance of a sentence according to both the current decoder state (i.e., the summary generated up so far) and the other sentences. For all sentence-level attention methods, I initialized the adjacency matrix through cosine similarity.

From the evaluation, I found that only one attention model produced very good results, being able to capture the sentence saliency. In detail, I based this model on Pagerank, but with a sigmoid activation function. Such function allowed the Neural Network to select salient sentences, whereby it needed them either for the context vector or to copy words with the pointer network. The model obtained very high Rouge scores, being close to the model proposed by See et al. (2017) (about 2 Rouge points low). Such difference is due to the high abstractive power of the proposed model; it tends to largely use the vocabulary distribution and seldom the pointer network, as opposed to See et al.'s model. The evaluation part showed that the model has both an high average β values (i.e., it generates the next word using the vocabulary) and novelty (i.e., the capability of the model of generating words not present in the reference summary). The remaining two attention methods, instead, produced very good sentence-level scores, but their impact on the word-level ones was disastrous; they assigned all the attention to a single sentence, zeroing the word-level scores for the other ones. Furthermore, such behaviour prevented the model to copy rare words in the output summary. For those reasons, they obtained very low Rouge scores.

As future works, I am interested to improve the recall of the model, which is its weakness. In detail, it has a very high Precision, but a very low Recall since it is not

able to retrieve all the relevant words for the summary. To increment the Recall, I think that the use of topics generated by an LDA model [Blei et al., 2003] could be useful since they capture the semantic content and the domain style of the document, tying the summary to it. I will also improve the graph-based sentence-level attention, which showed very good results. I will follow Hsu et al. (2018)'s idea, training the sentence-level attention (through a loss function) to recognize the best sentences for the summary. However, the method proposed by Hsu et al. cannot be directly adopted; I have to define a loss that can adapt to the generated summary, identifying the best sentences and penalizing the model in case that it does not select them. Finally, I will continue to study how to improve the *covscore* function, since it was able to slightly increase the performance of the model while reducing redundancy and attention errors.

Appendix A

McCulloch and Pitts Neuron

The McCulloch and Pitts Neuron (MPN from now on) is a neuron that accepts a set of boolean inputs $[x_1, x_2, \ldots, x_n]$ (I will describe them with $\mathbf{x}$ from now on) through its input channels (dendrites), and emits a value in the set of $\{0, 1\}$. The value 0 means that the MPN has not fired, while the value 1 means it has fired.

The MPN could be represented as a combination of two functions f and g:

- g aggregates the inputs via a sum operation. These inputs could be excitatory or inhibitory. Excitatory inputs could make the MPN fires when they are combined together; inhibitory inputs, instead, could alone prevent the MPN to fire.
- f emits in output either 0 or 1. In details, the function f simply applies a transformation (e.g., a sign function) on the output of function g and checks if the output is greater or equal than a given threshold θ. If it is, the MPN will fire (i.e., it emits 1); otherwise, the MPN will not fire (i.e., it emits 0).

Function g and f are represented in Equation A.1.

$$\begin{aligned} g(\mathbf{x}) &= \sum_{i=1}^{n} x_i \\ f(g(\mathbf{x})) &= \begin{cases} 1 & \text{if } sign(g(\mathbf{x})) \geq \theta \\ 0 & \text{if } sign(g(\mathbf{x})) < \theta \end{cases} \end{aligned} \tag{A.1}$$

I will now make an example to explain how the MPN works and the difference between excitatory and inhibitory inputs. Suppose to have a MPN that decides for a person if it is convenient to buy a t-shirt or not. Its features could be:

1. *likeColor*: it describes if the person likes or not the color of the t-shirt;

2. *doesNotFit*: it describes if the person can wear or not the t-shirt (e.g., it is too small or too big);

3. *isCotton*: it describes if the fabric is cotton;

4. *likeDesign*: it describes if the person likes the design.

Inputs (1), (3) and (4) are excitatory because they could affect the decision when their aggregation is considered. For instance, if inputs (3) and (4) are 1, the person can buy the shirt because it is made of cotton and they like the design. The only inhibitory input is (2) because it could affect alone the decision. For instance, if all inputs are 1, only *doesNotFit* is taken into consideration, since the person cannot buy a t-shirt that they cannot wear.

Since the MPN accepts and emits boolean values, it can represent the boolean functions *AND* and *OR*. For those functions, f can be defined as a function that compares the result of the function g with the given threshold θ. Figure A.1 shows the MPN for the *AND* function, while Figure A.2 shows the MPN for the *OR* function. In all the figures, values x_1 and x_2 represent the inputs, while y represents the output. The value represented inside the neuron is the threshold θ.

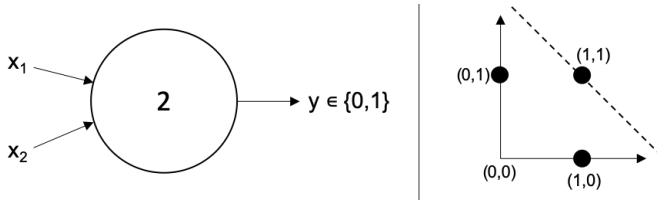


Figure A.1: The image represents a MPN for the *AND* function (on the left), and its geometrical representation (on the right), i.e. $x_1 + x_2 \geq \theta = 2$.

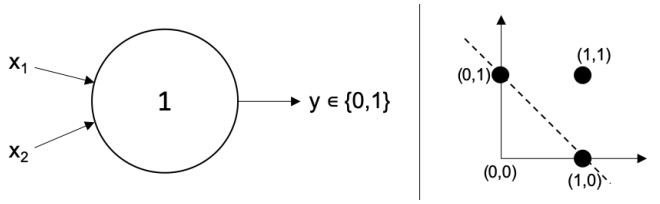


Figure A.2: The image represents a MPN for the *OR* function (on the left), and its geometrical representation (on the right), i.e. $x_1 + x_2 \geq \theta = 1$.

Appendix B

Training the Artificial Neural Network

In Section 3.2, I said that the proposed algorithm cannot be applied to *xor* problems (or non-linearly separable dataset). The problem is that many tasks can be defined as *xor* problems. To overwhelm this issue, backpropagation algorithm was adopted.

The training of current Neural Network models is composed of two steps: *forward* step and *backward* step. In the former one, the network processes the data hierarchically through its layers, emitting an output which can be a number (in case of regression) or a probability distribution over a set of classes. The output is then compared with a gold standard using an objective function (in jargon, *loss function*). Then, in the latter step, the network uses the result of the objective function to modify its weights, which are update in reverse order (from output to input) via partial derivatives.

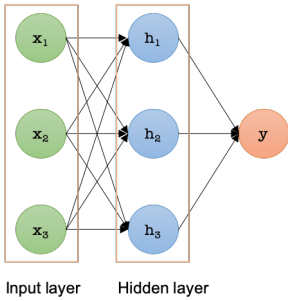


Figure B.1: The picture shows a neural network with one hidden layer. The input and hidden layers have 3 neurons, while output layer has one neuron (red circle).

For the description of forward step and backward step, I will use the feed-forward network depicted in Figure B.1. Such network is defined by the following equations:

$$\begin{aligned} \mathbf{h} &= sigmoid(\mathbf{W}\,\mathbf{x}) \\ y &= sigmoid(\mathbf{U}\,\mathbf{h}) \end{aligned} \tag{B.1}$$

where $\mathbf{x} = [x_1, x_2, x_3]$, and $\mathbf{h} = [h_1, h_2, h_3]$. In this model, I will adopt the mean squared error as loss function:

$$L(y, g) = \frac{1}{2}(g - y)^2 \tag{B.2}$$

to train the network to distinguish between two classes C_1 and C_2, where C_1 is represented with 1 and C_2 with 0.

B.1 Forward pass

In the forward pass, the network receives in input a set of training examples, called *batch*, which are pairs formed of the input data x and the golden label g:

$$\{(x_1, g_1), (x_2, g_2), (x_3, g_3), \ldots, (x_n, g_n)\}$$

For instance, the training examples could be composed of <*image, label*> pairs or <*sentence, summary*> pairs. Those input values will activate the layers of the network, which will emit a value (i.e., the class) for each example. The predicted class of each example is then compared with its golden label g using the loss function of Equation B.2. Such equation will highlight how much the predicted class is far from the correct one. Then, the sum of the errors of the batch examples is passed to the second step, the backward step, in order to update the weights of the network. In detail, the network will change the weights in order to minimize the loss function.

B.1.1 Feed Forward: a numerical example

I will make a numerical example to explain how the forward pass works. For the example, I will use the model depicted in Figure B.1, with the following input $\mathbf{x}$:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -0.16 \\ -0.21 \\ -0.96 \end{bmatrix}$$

the weight of the hidden layer $\mathbf{W}$:

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix} = \begin{bmatrix} 0.2 & -0.28 & -2.0 \\ 0.56 & 0.34 & -1.25 \\ 1.22 & -0.39 & 2.4 \end{bmatrix}$$

and the weight of the output layer $\mathbf{U}$:

$$\mathbf{U} = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix} = \begin{bmatrix} -0.60 & 2.2 & 7.4 \end{bmatrix}$$

The output of the first layer, that I called $\mathbf{h}$, is calculated as follows:

$$\begin{aligned}
\mathbf{h} &= sigmoid\Big(\mathbf{W}\,\mathbf{x}\Big) \\
&= sigmoid\Bigg(\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\Bigg) \\
&= sigmoid\Bigg(\begin{bmatrix} 0.2 \cdot -0.16 + -0.28 \cdot -0.21 + -2.0 \cdot -0.96 \\ 0.56 \cdot -0.16 + 0.34 \cdot -0.21 + -1.25 \cdot -0.96 \\ 1.22 \cdot -0.16 + -0.39 \cdot -0.21 + 2.4 \cdot -0.96 \end{bmatrix}\Bigg) \\
&= sigmoid\Bigg(\begin{bmatrix} -1.89 \\ -1.36 \\ 2.19 \end{bmatrix}\Bigg) = \begin{bmatrix} 0.87 \\ 0.79 \\ 0.1 \end{bmatrix}
\end{aligned}$$

The output value $\mathbf{y}$ is calculated in the same way of the hidden layer:

$$\begin{aligned}
\mathbf{y} &= sigmoid(\mathbf{U}\,\mathbf{h}) \\
&= sigmoid\Big(\begin{bmatrix} -0.60 \cdot 0.87 + 2.2 \cdot 0.79 + 7.4 \cdot 0.1 \end{bmatrix}\Big) \\
&= \begin{bmatrix} 0.12 \end{bmatrix}
\end{aligned}$$

Supposing that the ground truth label of the example is C_1, the loss function is computed as the difference between of y and the gold class value 1 to maximize the output score:

$$L(y, g) = \frac{1}{2}(1 - 0.12)^2 = 0.38$$

B.2 Backward pass

In the previous section, I presented the loss function L which is used to compute the error of the network, i.e. how distant is the output of the network respect to the desired one. The function is also used to train the network, modifying the weights (bias included) in order to reduce the error of the network. In this section, I present how the network uses L and how the weights change with respect to the error. The operation

of changing the weights according to the objective function is called *backward pass* because it traverses the network from the output layer to the input one, i.e. in reverse order. The algorithm that implements the backward pass is called *backpropagation*, or simply *backprob*.

So far, I said that a neural network uses the weights to respond to input values, activating neurons and producing output values. Thus, there exists a correlation between the weights and the error function. The error can be represented with respect to a single weight (or bias) w as depicted in Figure B.2.

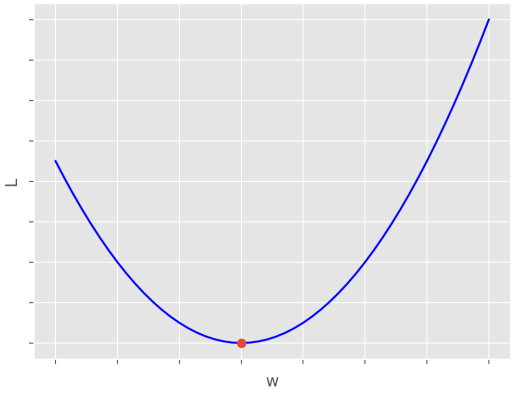


Figure B.2: The images shows an hypothetical loss function with respect to a weight w. The red dot represents the optimal weight value, which leads to the minimum error.

Since the error depends from a weight w, its partial derivative can be calculated to obtain the direction towards w is updated; this procedure is called *gradient descent*. Figure B.3 shows an example of gradient descend. The yellow point represents the new value of the weight with respect to the error. The process can be repeated until the red point is reached.

The above description can be transposed in mathematical terms. According to the notion of the derivative, there are two possible directions toward the weight can be updated:

- if $\frac{\partial L}{\partial w} > 0$, then the network is overestimating, i.e. L augments when the weight is augmented. In this case, the weight is decreased;
- if $\frac{\partial L}{\partial w} < 0$, then the network is underestimating, i.e. L augments when the weight is reduced. In this case, the weight is increased.

Those two cases can be compacted into a single equation:

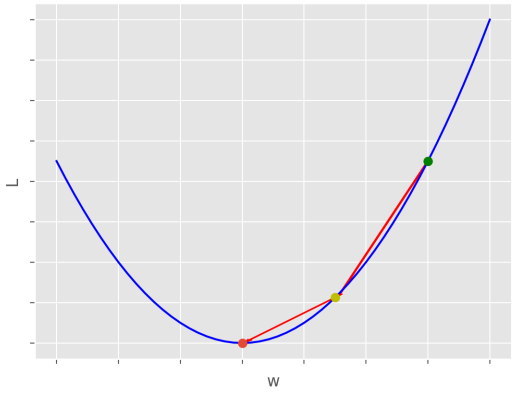


Figure B.3: The images shows the gradient descent from a starting point (in green) to the minimum one (in red). The red rows show the gradient direction.

$$w_{t+1} = w_t - \frac{\partial L}{\partial w} \tag{B.3}$$

where the minus will increase the weight if the partial derivative is negative, or decrease the weight if the partial derivative is positive.

Appendix C

Generalization

When I described the artificial neural networks, I said that they are trained to minimize the error on the training set. Such training could capture the intrinsic bias which occurs in the instances of the training set, impacting on the test set results. The issue of whether the training set performance has an impact on the test set (and in which measure) is called *generalization*, and it is fundamental in Machine Learning. A neural network that generalizes can give a proper answer on data that they have never seen before. For instance, suppose that a neural network to recognize cats in pictures is trained, and that a picture of a cat with a green fur is provided to the network. If the neural network was trained on a large cat dataset, it should recognize that the picture shows a cat. In general, the largest is the training set the better is the generalization. There are methods to improve the generalization of the network (also knows as *regularizes*) that have been proposed during the years. In this thesis, I used two regularizes: *early stopping* and *dropout*.

C.1 Early Stopping

Early stopping uses part of the training set, which forms the *validation set*, to evaluate the stopping criteria. A criterion could be the comparison of the error on the training set with respect to the error on the validation set. If the error on validation set is lower than the one in the training set, the training is stopped. Another criterion, generally used in combination with the first one, is to check if the error on the validation set does not decrease significantly for a fixed number of iterations (or epochs). If the error does not decrease, it means that the network cannot further generalize (and the training it stopped).

Some reader can think about to substitute the validation set with the test set, since the network has to be evaluated on this latter one. The test set should not be used for evaluation until the training is completed. The reason is twofold: on one hand, it is a

indirect training on the test set; on the other hand, it is not possible to evaluate if the network is able or not to generalize on unseen data.

However, the early stopping presents several drawbacks. A problem is that part of the training set has to be used to create the validation set; this could reduce the performance and the generalization of the network, especially if the training set is small. Another problem is that the validation set could be distant from the test set, not representing an accurate predictor of the performance of the network. Finally, there is no rule to define the optimal size of the validation set. Some researchers prefer to use the 5% of the training set, while others the 10%.

C.2 Dropout

Dropout [Srivastava et al., 2014] is another technique to generalize neural networks. Differently from the early stopping, it does not operate on the error function, but directly on the layers. In detail, dropout is a mask composed of binary values (zeros and ones) that is multiplied (pointwise) to the layer input. The mask is constructed sampling the values from a Bernoulli distribution, where each cell of the mask has a probability p to be 1, and probability $1 - p$ to be 0. The probability p is chosen by the user (it is an hyperparameter of the network).

The idea of dropout is to shutdown some connections (those ones where the mask contains 0), and consequently the neurons, simulating as if there were several simplified versions of the same neural network. Each version, composed of only those neurons where the signal is propagated, is then exposed to a batch of training examples and it will learn the features that are useful to distinguish them. The idea is that a network trained using dropout is able to learn a wide range of features, since only a part of neurons are active for each batch. Suppose to have a layer of a Multi-Layer Perceptron defined as follows:

$$\mathbf{y}^{(l)} = \tanh(\mathbf{W}^{(l)}\, \mathbf{x}^{(l-1)} + \mathbf{b}^{(l)}) \tag{C.1}$$

where l represents the l-th layer of the network, $\mathbf{y}^{(l)}$ the output of the layer, and $\mathbf{x}^{(l-1)}$ the input of the layer. The dropout on this layer is defined through the following equations:

$$\begin{aligned} r_j &\sim Bernoulli(p) \\ \hat{\mathbf{x}} &= \mathbf{r} \otimes \mathbf{x}^{(l-1)} \\ \mathbf{y}^{(l)} &= \tanh(\mathbf{W}^{(l)}\, \hat{\mathbf{x}}^{(l-1)} + \mathbf{b}^{(l)}) \end{aligned} \tag{C.2}$$

where $\otimes$ represents the pointwise multiplication. At test time, since dropout is still applied to l-th layer, $\mathbf{W}^{(l)}$ is rescaled by p, i.e. $p\mathbf{W}^{(l)}$. In this way, all information of

$\mathbf{x}^{(l-1)}$ will be used. An example of dropout on a Multi-Layer Perceptron is depicted in Figure C.1.

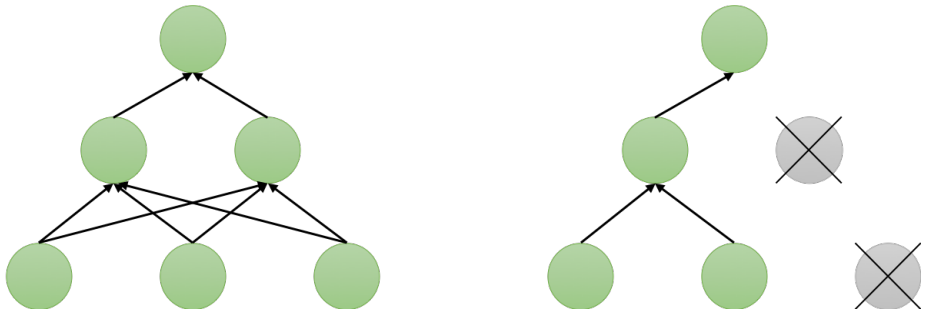


Figure C.1: The figure shows: on the left, a Multi-Layer perceptron without dropout; on the right, the same network with dropout. The cross represents a shutdown neuron.

Bibliography

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. (2013). Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186.

Banerjee, S. and Lavie, A. (2005). Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72.

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.

Barzilay, R. and Elhadad, M. (1999). Using lexical chains for text summarization. *Advances in automatic text summarization*, pages 111–121.

Barzilay, R. and Lapata, M. (2008). Modeling local coherence: An entity-based approach. *Computational Linguistics*, 34(1):1–34.

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

Bhandari, H., Shimbo, M., Ito, T., and Matsumoto, Y. (2008). Generic text summarization using probabilistic latent semantic indexing. In *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-I*.

Bhoopchand, A., Rocktäschel, T., Barr, E., and Riedel, S. (2017). Learning python code suggestion with a sparse pointer network. In *International Conference on Learning Representations (ICLR)*.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117.

Brunn, M., Chali, Y., and Pinchak, C. J. (2001). Text summarization using lexical chains. In *Proc. of Document Understanding Conference*. Citeseer.

Cagliero, L., Garza, P., and Baralis, E. (2019). Elsa: A multilingual document summarization algorithm based on frequent itemsets and latent semantic analysis. *ACM Transactions on Information Systems (TOIS)*, 37(2):1–33.

Cao, Z., Li, W., Li, S., and Wei, F. (2018). Retrieve, rerank and rewrite: Soft template based neural summarization. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 152–161.

Chen, D., Bolton, J., and Manning, C. D. (2016a). A thorough examination of the CNN/Daily Mail reading comprehension task. In *Association for Computational Linguistics (ACL)*.

Chen, Q., Zhu, X., Ling, Z., Wei, S., and Jiang, H. (2016b). Distraction-based neural networks for document summarization. pages 2754–2760.

Chen, Y.-C. and Bansal, M. (2018). Fast abstractive summarization with reinforce-selected sentence rewriting. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 675–686.

Cheng, J. and Lapata, M. (2016). Neural summarization by extracting sentences and words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 484–494.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

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.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Daumé III, H. and Marcu, D. (2002). A noisy-channel model for document compression. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 449–456. Association for Computational Linguistics.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 933–941. JMLR. org.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dohare, S., Gupta, V., and Karnick, H. (2018). Unsupervised semantic abstractive summarization. In *Proceedings of ACL 2018, Student Research Workshop*, pages 74–83.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*.

Galanis, D., Lampouras, G., and Androutsopoulos, I. (2012). Extractive multi-document summarization with integer linear programming and support vector regression. *Proceedings of COLING 2012*, pages 911–926.

Gehrmann, S., Deng, Y., and Rush, A. M. (2018). Bottom-up abstractive summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4098–4109.

Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: Continual prediction with lstm.

Gong, Y. and Liu, X. (2001). Generic text summarization using relevance measure and latent semantic analysis. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 19–25. ACM.

Gross, O., Doucet, A., and Toivonen, H. (2014). Document summarization based on word associations. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 1023–1026. ACM.

Gu, J., Lu, Z., Li, H., and Li, V. O. (2016). Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1631–1640.

Gulcehre, C., Ahn, S., Nallapati, R., Zhou, B., and Bengio, Y. (2016). Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 140–149.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701.

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

Hsu, W.-T., Lin, C.-K., Lee, M.-Y., Min, K., Tang, J., and Sun, M. (2018). A unified model for extractive and abstractive summarization using inconsistency loss. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 132–141.

Hu, B., Chen, Q., and Zhu, F. (2015). Lcsts: A large scale chinese short text summarization dataset. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1967–1972.

Iacobacci, I., Pilehvar, M. T., and Navigli, R. (2015). Sensembed: Learning sense embeddings for word and relational similarity. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 95–105.

Kedzie, C., McKeown, K., and Daume III, H. (2018). Content selection in deep learning models of summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1818–1828.

Kiyono, S., Takase, S., Suzuki, J., Okazaki, N., Inui, K., and Nagata, M. (2017). Source-side prediction for neural headline generation. *arXiv preprint arXiv:1712.08302*.

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632.

Klicpera, J., Bojchevski, A., and Günnemann, S. (2018). Predict then propagate: Graph neural networks meet personalized pagerank.

Koehn, P. (2009). *Statistical machine translation*. Cambridge University Press.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480.

Kryściński, W., Paulus, R., Xiong, C., and Socher, R. (2018). Improving abstraction in text summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1808–1817.

Kupiec, J., Pedersen, J., and Chen, F. (1995). A trainable document summarizer. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 68–73. ACM.

LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.

Leskovec, J., Milic-Frayling, N., and Grobelnik, M. (2005). Impact of linguistic analysis on the semantic graph coverage and learning of document extracts. In *AAAI*, pages 1069–1074.

Li, P., Lam, W., Bing, L., and Wang, Z. (2017). Deep recurrent generative decoder for abstractive text summarization. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2091–2100.

Li, W., Xiao, X., Lyu, Y., and Wang, Y. (2018a). Improving neural abstractive document summarization with explicit information selection modeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1787–1796.

Li, W., Xiao, X., Lyu, Y., and Wang, Y. (2018b). Improving neural abstractive document summarization with structural regularization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4078–4087.

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

Litvak, M. and Last, M. (2008). Graph-based keyword extraction for single-document summarization. In *Proceedings of the workshop on Multi-source Multilingual Information Extraction and Summarization*, pages 17–24. Association for Computational Linguistics.

Liu, Y. and Lapata, M. (2019). Text summarization with pretrained encoders. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, page 37303740.

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

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Ma, S., Sun, X., Lin, J., and Wang, H. (2018). Autoencoder as assistant supervisor: Improving text representation for chinese social media text summarization. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 725–731.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Mann, W. C. and Thompson, S. A. (1988). Rhetorical structure theory: Toward a functional theory of text organization. *Text-interdisciplinary Journal for the Study of Discourse*, 8(3):243–281.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2017). Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Mi, H., Sankaran, B., Wang, Z., and Ittycheriah, A. (2016). Coverage embedding models for neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 955–960.

Miao, Y. and Blunsom, P. (2016). Language as a latent variable: Discrete generative models for sentence compression. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 319–328.

Mihalcea, R. (2004). Graph-based ranking algorithms for sentence extraction, applied to text summarization. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 20. Association for Computational Linguistics.

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

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.

Mikolov, T., Yih, W.-t., and Zweig, G. (2013c). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751.

Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Minsky, M. and Papert, S. A. (1988). *Perceptrons: An introduction to computational geometry*. MIT press.

Moro, A., Cecconi, F., and Navigli, R. (2014). Multilingual word sense disambiguation and entity linking for everybody. In *International Semantic Web Conference (Posters & Demos)*, pages 25–28.

Morris, J. and Hirst, G. (1991). Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Computational linguistics*, 17(1):21–48.

Murray, G., Renals, S., and Carletta, J. (2005). Extractive summarization of meeting recordings.

Nallapati, R., Zhai, F., and Zhou, B. (2017). Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In *AAAI*, pages 3075–3081.

Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B., et al. (2016). Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning (CoNLL)*, pages 280–290.

Narayan, S., Cohen, S. B., and Lapata, M. (2018). Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807.

Navigli, R. and Ponzetto, S. P. (2010). Babelnet: Building a very large multilingual semantic network. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 216–225. Association for Computational Linguistics.

Nenkova, A. and Passonneau, R. (2004). Evaluating content selection in summarization: The pyramid method. In *Proceedings of the human language technology conference of the north american chapter of the association for computational linguistics: Hlt-naacl 2004*.

Nishikawa, H., Hasegawa, T., Matsuo, Y., and Kikui, G. (2010). Opinion summarization with integer linear programming formulation for sentence extraction and ordering. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 910–918. Association for Computational Linguistics.

Nomoto, T. and Matsumoto, Y. (2001). A new approach to unsupervised text summarization. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 26–34. ACM.

Oliveira, H., Lima, R., Lins, R. D., Freitas, F., Riss, M., and Simske, S. J. (2016). A concept-based integer linear programming approach for single-document summarization. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 403–408. IEEE.

Ouyang, J., Chang, S., and McKeown, K. (2017). Crowd-sourced iterative annotation for narrative summarization corpora. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, volume 2, pages 46–51.

Ouyang, J. and McKeown, K. (2015). Modeling reportable events as turning points in narrative. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2149–2158.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Paulus, R., Xiong, C., and Socher, R. (2018). A deep reinforced model for abstractive summarization.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. (2016). Sequence level training with recurrent neural networks. In *International Conference on Learning Representations*.

Rish, I. et al. (2001). An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Rush, A. M., Chopra, S., and Weston, J. (2015). A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389.

Saggion, H. and Poibeau, T. (2013). Automatic text summarization: Past, present and future. In *Multi-source, multilingual information extraction and summarization*, pages 3–21. Springer.

Sankaran, B., Mi, H., Al-Onaizan, Y., and Ittycheriah, A. (2016). Temporal attention model for neural machine translation. *arXiv preprint arXiv:1608.02927*.

Saxena, S. and Saxena, A. (2016). An efficient method based on lexical chains for automatic text summarization. *International Journal of Computer Applications*, 144(1):47–52.

See, A., Liu, P. J., and Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 1073–1083.

Silber, H. G. and McCoy, K. F. (2002). Efficiently computed lexical chains as an intermediate representation for automatic text summarization. *Computational Linguistics*, 28(4):487–496.

Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. (2019). Mass: Masked sequence to sequence pre-training for language generation. In *Proceedings of the 36th International Conference on Machine Learning*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Steinberger, J., Kabadjov, M. A., Poesio, M., and Sanchez-Graillet, O. (2005). Improving lsa-based summarization with anaphora resolution. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 1–8. Association for Computational Linguistics.

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.

Tan, J., Wan, X., and Xiao, J. (2017). Abstractive document summarization with a graph-based attentional neural model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1171–1181.

Tu, Z., Lu, Z., Liu, Y., Liu, X., and Li, H. (2016). Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 76–85.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.

Wang, K., Quan, X., and Wang, R. (2019). Biset: Bi-directional selective encoding with template for abstractive summarization. In *57th Annual Meeting of the Association for Computational Linguistics*, pages 2153–2162.

Yao, L., Mao, C., and Luo, Y. (2019). Graph convolutional networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7370–7377.

Zeng, W., Luo, W., Fidler, S., and Urtasun, R. (2017). Efficient summarization with read-again and copy mechanism. In *International Conference on Learning Representations (ICLR)*.

Zhou, L. and Hovy, E. (2004). Template-filtered headline summarization. *Text Summarization Branches Out*.

Zhou, Q., Yang, N., Wei, F., and Zhou, M. (2017). Selective encoding for abstractive sentence summarization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 1095–1104.