University College London
Engineering Faculty
Department of Computer Science
MSc Machine Learning

Features and Representations for Multi-document Summarization

Supervised by:

Dr. Daniel Hulme, UCL

Dr. David Corney, Signal

Authored by:

Matteo Hessel

September 2015

To Chiara, you make life beautiful...

Pursuit of knowledge cannot bring alone the highest rewards,
these are only deserved when knowledge is brought among the people,
as foundation for a better life.
Dario Del Corno

Acknowledgments

First, I want to thank my academic supervisor Daniel Hulme, for his advices and encouragements. Second, thanks to my colleagues at Signal for the nice time spent together and for giving me the opportunity of working on this interesting topic, in particular David Corney and Miguel Martinez-Alvarez, for their kind supervision. Also the opportunity of giving presentations to the research team and the developers were very appreciated. Last but not least, I want to thanks my parents, my brother Tommaso and my partner Chiara, for supporting me in all my choices all these years. Credit for whatever I achieved goes to them, the mistakes are only mine.

Abstract

This project has a dual nature. From the research perspective, it attempts to develop novel ideas within the context of multidocument summarization, focusing mainly on learning good representation and features. From a more applied perspective, the project deals with the specific constraints of performing summarization in a real time environment, as the one provided by Signal's media monitoring platform.

Summarization has many dimensions: the choice whether or not a sentence may be extracted may require to analyze if it's syntactically, well formed, how it related to the topic or query we are interested in, and how it's information content relates to other sentences in the summary. We believe the best way of tackling all these issues is to adopt a supervised framework, where different features and representations (dealing with the different issues) may be combined using suitable algorithms.

In terms of features, among the novel proposals we make, it is to be noted the use of language models to generate a perplexity feature that significantly improve the performance of our system on real world data. We also evaluated a wide range of non linear regression algorithms (capable of managing noisy data and distant supervision) in order to combine the sentence features in robust relevance scoring functions.

Subsequently, a major part of our work focused on the use of deep learning models to learn distributional representations of sentences. Concerning this, we implemented different architectures, such as convolutional and LSTM recurrent networks. Both approaches were successful in learning how to compose word embeddings (such as word2vec's) into meaningful sentence representations. The application of such sentence representations to redundancy assessment proved to be extremely successful and, to the best of our knowledge, is novel to the field of summarization.

Additional, but secondary, products of our main research were: the configuration of a CPU+GPU computing environment on AWS, which may also be used by Signal for other purposes; deep learning presentations to the research team; and the evaluation of some heuristics for ordering extractive summaries. The latter is an intrinsically ill-defined problem, because no systematic evaluation is possible; however, we propose some ideas, qualitatively evaluated by manually inspecting the results.

Contents

List of Figures

Chapter 1

Introduction

In this chapter we give a brief introduction to the project, describing why it is a relevant topic and the specific research question we tried to answer with our work. Furthermore we outline the content of this thesis and the future extensions and applications. Finally, we review the role of our industrial partner and describe the computing environment we set-up to perform the experiments,.

1.1 Motivation

Automatic Summarization is a classic problem in *Natural Language Processing*: summarizing a document, or a collection of documents, means generating a shorter text that (ideally) includes all *relevant* information. The problem is non trivial because the summary should not only select the most relevant information, a challenging problem on its own, but must also avoid redundancy, satisfy grammatical constraints and avoid exceeding some bound on length. First attempts to build summarization systems go back to the *50s* and *60s*, but the problem is still open. It has also received much attention in recent years, as the proliferation of textual information caused by the Web made all technologies for making sense of unstructured data extremely valuable. In particular summarization has been shown to allow to judge the relevance of documents without reading them entirely, saving significant time when exploring large corpora. Although research is very active and new approaches are constantly introduced, there is a strong gap exists between the methods of academic research and their actual industrial application. This might be due to several reasons: first, there might be a lack of understanding of the actual needs of industry, second, evaluation methods might not be sufficiently reliable (solutions may under-perform in real world settings), finally, most research focuses on specific sub-tasks and little effort is devoted into combining the different solutions into a single coherent system. Together these facts could explain the inertia with which new models are adopted, and the fall back to standard but poor performing methods.

1.2 Aims and Research Questions

Within the wide field of automatic multidocument summarization, there are two main questions we wish to address in this thesis. One question is more applied, and concerns the applicability of methods and algorithms to the industry, the other is more research oriented, focusing on the use of deep learning methods for learning meaningful representations of words and sentences.

1 - *First, can we incorporate the corpus of research which has been developed for multidocument summarization into a system capable of addressing the needs of the industry, including its time and quality constraints? Can such system be modular enough to be tuned as a function of such constraints? How can we extend current models in order to better manage real word document collections?*

2 - *Secondly, can deep learning algorithms, by providing better word and sentence representations (embeddings), improve the quality of multidocument summarization systems? specifically, can such representation allows to better identify redundant information within selected sentences in the proposed summary? what type of deep architecture is most suitable for this challenging task?*

In this dissertation we will tackle both issues simultaneously. This should allow us to implement a system that can be used in practice by industry, filling the gap between research and application. The performance of the system will be evaluated both against classic existing data-sets (from the *DUC* and *TAC* conferences on natural language processing) and in a real word environment provided by *Signal,* our industrial partner in the research project. Besides relevant answers to the two major research questions, during this project I have also worked on some related, but not fundamental issues, including:

1 - *Configuration of a GPU computing environment on AWS, to be used not only by myself but also by Signal for other machine learning tasks.*

2 - *Development and evaluation of alternative ordering procedures for arranging the extracted sentences of a summary. This is a task which has received little attention in literature but potentially very relevant to improve understanding.*

3 - *Application of some of the techniques developed for summarization to other tasks, for example the statistical and neural language models that have been learned can be applied beyond this project (for example to clean textual document from meta-data).*

4 - *Presentations and tutorials, concerning Deep Learning and the use of GPU computing, have been given to researchers and developers of Signal.*

1.3 The Computing Environment

The choice of exploring *deep learning* methods and algorithms requires significant computational resources, both in term of memory and processing power. This because the number of parameters of deep models is such that it is important to use large amount of text for training meaningful language models, capable of good generalization: therefore it is important to be able to fit considerable amounts of such training data in main memory (to avoid the latency of accessing disk) and to exploit parallel computing for fast iteration over the corpus. In order to complete the project I therefore setup a *CPU+GPU* computing environment, using both commodity hardware and cloud services such as *AWS*. For completing the project I have employed all the following:

1. *HP* laptop with 8 cores
2. an old malfunctioning *Toshiba* laptop with 8 core and one of the first *CUDA* capable *GPGPUs*
3. *Amazon Web Services* instance with *8* cores and a cutting edge *GPGPU*

The *HP* laptop, running *Ubuntu 12.04,* is used for coding and debugging. Tests of several baseline summarization system implemented for comparison where also evaluated here. However, if run on the *8* core *CPU* of this laptop, training the deep learning models for language modeling required up to half an hour for a single epoch, making training using this machine not plausible. The Toshiba laptop, running *Ubuntu 12.04*, is an old laptop with several malfunctioning and cooling issues. The *NVIDIA GPU* installed on this machine is very old, and the *CUDA* version required for the Chainer framework does not explicitly support such *GPU* anymore. To make the current version of the *CUDA toolkit* work on such computer proved particularly complex but was finally managed. The main limitations of this machine were the limited parallelism of the processing unit, and the only *500MB* of *GPU* device *RAM* (which doesn't allow to fit all available data on the device memory). This machine was however essential in order to provide a free *GPU* environment for experimenting different deep models. Selected models were then run on the *AWS GPU* instance from remote, in order to perform extensive hyperparameter searches (using the techniques presented in Chapter 5 and the much more powerful *GPU* here available). The *AWS* instance is the *g2.2xlarge*: a *8* core machine with *16 GB* of *RAM*, connected to an *NVIDIA GPGPU* with *1536* cores. In order to run computations on *AWS* I used *100*$ credits won by participating to a *Hacker Rank* online coding challenge, together with 35$ credits provided by the *Amazon Web Service Student Educate* program, and 80$ credits took from the *100*$ voucher provided by professor Wang for the Information Retrieval and Data Mining course (the course project only required a fraction of it in order to be completed).

1.4 Industrial Partner (Signal) and Feedback on Results

This project was developed in collaboration with *Signal Media*, a technological start up based in London. Signal processes textual data from several online sources, providing its clients with a market intelligence platform giving direct access to news, blogs and social media posts regarding entities, brands or marketing campaigns. The overall aim is it allow the users to make sense of the explosion of data currently available online. The company employs several *PhD* researchers in *machine learning* and has tight collaboration with high profile academics in the field of *natural language processing.* Within the company, our supervisor was the data scientist David Corney, who's comments are listed below. The company was helpful in clarifying the requirements for a summarization system to be employed in practice by Signal:

1. *efficiency*: the summarization system must be fast. Training of some components of the system can require significant resources, but at prediction time, the system must be able to generate a summary as fast as possible.

2. *robustness*: real world data is not as clean as conference data. It is common for the text to be interleaved with captions and meta-data, the system must be capable of dealing with this.

3. *modularity*: the performance requirements, both in terms of responsiveness and quality, might change over time. Ideally the system should be modular and allow the company to select the optimal trade-off.

This requirements were full-filled by our system which is highly modular and specifically addresses poor quality data and other issues. The most delicate requirement to be full filled was the efficiency one, however Signal platform already does some of the required processing tasks, and when integrated with the system this will reduce the computational load of our component. The modularity of our system will also be very useful in order to provide the required level of responsiveness when deployed on the platform. Beside relevant insights on the industrial needs, the company also provided us with real world data and relevant feedback on the systems we developed:

1. *Having Matteo join us as a research intern has allowed Signal to explore the area of summarization very effectively and efficiently. After initial discussions to frame the research, he has worked largely autonomously in researching and implementing several interesting methods. He has applied these methods to the commercial data that we provided (in addition to publicly available corpora), which has given us further insights into the problem.*

2. *With Matteo's help and personal involvement in the first stages, we plan next to test the integration of the results of his work into our own product later this year; as conclusion of what has been a very fruitful collaboration.*

1.5 Thesis Outline

In *Chapter 2* we present a detailed literature review describing the main results in the field of automatic summarization. The review includes details on the features and scoring functions most frequently employed, several algorithms for extraction of the summary, and also the standard evaluation metrics and tools used to validate the output of summarization systems. The methods presented include results which have been used within our system and other that have been implemented as baselines. Finally, reasons are given for those ideas that, although used in the past, have not been deemed worthy an implementation. *Chapter 3* covers deep learning methods for natural language processing. More specifically this chapter focuses on the use of deep models for learning meaningful vector representations (embeddings) of words and sentences. Many recent developments of this topic are considered, and their applicability to automatic summarization evaluated. The Chainer framework for deep learning and GPU computing is also presented. In *Chapter 4* we present the results of all our experiments. The expositions follows the development of our state of the art extractive summarization system, presenting the choice made, and the experiments with various alternative ideas. First we present a system combining sentence features in a *relevance* scoring function (redundancy is not yet considered at this stage). Several features are considered, including some of the features reviewed in Chapter 2 and several new features. In particular we consider the use of classical and neural language model in order to deal with real world data (not as clean as the documents in conference corpora). We also consider the use of non linear regressors (including Decision Trees, Random Forests, Gradient Boosting) to learn better scoring function. This is an improvement with respect to the current state of the art which led to very good results. The system is evaluated using the standard metrics used in conferences and compared to other more basic systems. In the second part of the chapter we present our experiments on the use of redundancy metrics to improve the performance of our system. The metrics include standard n-gram based functions, but also word and sentence embeddings generated using deep learning models. In the chapter we describe in details the different models and architectures used to generate such vector representations. We also present a model based approach to hyperparameter search. Again, everything is formally evaluated and compared to other systems. Some of the results in this chapter have been achieved in collaboration with Miljan Martic, another Machine Learning MSc student working on his final project in collaboration with *Signal* (it is clearly specified when this is the case). Finally, we deal with some additional issues such as the ordering of sentences within an extractive summary. In the *Appendix*, we describe how we set up a *GPU* computing environment on the cloud in order to perform our experiments.

1.6 Publication and Applications

The work done on the use of distributional representations of sentences in automatic summarization has not yet been published. However, together with Signal and our supervisor Daniel Hulme we plan to write a paper describing the results outlined in this thesis. The work done, after being wrapped in Clojure, will also be integrated in the Signal platform, providing a novel additional service.

1.7 Public Repositories

The code describing all our developments and results can be found on the github account *https://github.com/mtthss.* The code is divided in various repositories: the main code containing the summarization systems is in the */experiments-in-summarization* folder. The */neural-language-modelling* folder also contains some relevant code, although most of it has been integrated in the previous folder.

Chapter 2

Automatic Summarization

In this chapter we review relevant methods developed for automatic summarization, focusing on *extractive* methods for multidocument setting. The chapter covers the design of a summarization system's pipeline *end to end*: text features, scoring functions, extraction algorithms, post-processing and evaluation. All methods have been implemented for comparison with our system, unless specified otherwise.

2.1 Extractive Summarization

Extractive summarization is the process of building a summary by extracting from a collection of documents the most relevant sentences with respect to a given topic, suitably ordering such sentences, and ensuring that a fixed budget of J words is not exceeded. In order to do so for large collections, it is important to avoid introducing redundancy among the selected sentences. The process can be divided in two main steps: extraction of an optimal set of sentences, according to some criteria, and ordering of such sentences. The former is the most complex issue, and can be formalized as a combinatorial optimization problem. In the following we consider three classes of summarization algorithms: heuristic, unsupervised and supervised. The first category includes simple algorithms, relying on some simple intuition about what a relevant sentence might look like to rank the sentences; these methods lack of principled foundations, and are often domain dependent. Unsupervised methods also rank sentences according to some a priori principle, but rely on some more sophisticated ideas, e.g. structural properties of text; their limitation is that they have no way of explicitly dealing with the challenges of the multi-document setting. Finally, supervised methods rely on suitable training corpora in order to learn scoring functions for relevance and redundancy. Both are computed from the values of several features, and combined to define an objective function. While for previous methods the optimization step is trivial, in supervised methods such step may be the bottleneck of the process: approximate solutions are therefore required.

2.2 Evaluation of Summarization Systems

In order to build an effective automatic summarization system it is essential to have a way of evaluating its performances. This is however a big issue because defining what a good summary *is* can be challenging (as relevance, redundancy, coherence, grammaticality are all subjective concepts). In principle, in order to perform this evaluation, the ideal approach would be to make a human judge score each generated summary individually, as he could catch such subtleties naturally. However, this approach does not scale as it requires significant effort. Therefore, the most common approach is to use ad hoc corpora (e.g. those provided by the *Document Understanding Conferences*), that include large collections of documents and provide corresponding human reference summaries. This allows to evaluate single and multidocument summarization systems with respect to the *main* and the *update* tasks by comparing it to the reference summaries. The first proposals of metrics for comparing a candidate to one or more reference summaries were described in Saggion et al. [32]. Interestingly, not all metrics can be used when multiple reference summaries are available, indeed, stability and reliability of the evaluation is improved by this only if the metrics used for the comparison take advantage of the consensus among summaries, as proved empirically by Hori et al. [14] and confirmed by Nenkova and Passonneau [2]. A detailed analysis of the overall evaluation task, and of the reliability of the metrics employed, can be found in the paper by Lin [47], where the impact of using different numbers of references is also estimated.

2.2.1 The Rouge System

The *Rouge* software (*Recall-Oriented Understudy for Gisting Evaluation*) is a flexible tool for evaluating automatically generated summaries by comparing them to reference human summaries. The software implements various recall oriented metrics (focusing on different aspects):

Rouge-N Rouge-N stands for *N-gram co-occurrence statistics* and represents the n-gram recall between a summary (which has been generated automatically) and a set of reference summaries (usually human generated).

$$\frac{\sum\limits_{S \in \{Reference\}} \sum\limits_{gram_n \in S} Count_{match}(gram_n)}{\sum\limits_{S \in \{Reference\}} \sum\limits_{gram_n \in S} Count(gram_n)} \tag{2.1}$$

Note that the metric works naturally with multiple reference summaries and, as previously required in order for multiple references to improve the stability of the evaluation, the metric advantages candidates that retain information shared by more then one of the reference human summaries

Rouge-L As proposed by Saggion et al. [32], the tool also implements the normalized *Longest Common Subsequence* metric (LCS). Although initially defined at a sentence level, the metric can be extended to the summary level, defining consistently recall, precision and f-score. This metric has the advantage, with respect to n-grams, that it automatically includes the longest in sequence set of words without specifying a priori the order n. Following Cormen et al. [17], $Z = (z_1, ..., z_n)$ is a subsequence of $X = (x_1, ..., x_m)$ if there exists a strictly increasing sequence of indices $(i_1, ..., i_k)$ such that for each $j = 1, ..., k$ the relation $x_{i_j} = z_j$ holds, in such case we write $Z \in Sub(X)$. At a sentence level we have:

$$com(s_i,\, s_j) = \{c \mid c \in Sub(s_i) \,\wedge\, c \in Sub(s_i)\} \tag{2.2}$$

$$LCS(s_i,\, s_j) = max\,(len\,(com(s_i,\, s_j))) \tag{2.3}$$

$$R_{LCS}(s_i,\, s_j) = \frac{LCS(s_i,\, s_j)}{m} \tag{2.4}$$

$$P_{LCS}(s_i,\, s_j) = \frac{LCS(s_i,\, s_j)}{n} \tag{2.5}$$

Lets now first consider one reference summary $R = \{r_1, ..., r_u\}$ of m words, and lets denote with $C = \{c_{1,}, ..., c_n\}$ the candidate summary of n words, we can very easily define the same measures also at the summary level in the following way:

$$LCS_{\bigcup}(s,\, C) = len\left[\bigcup_{i=1}^{n}\left(\underset{x\in com(s,\, c_i)}{argmax}\ (len(x))\right)\right] \tag{2.6}$$

$$R_{LCS}(C, R) = \frac{\sum_{i=1}^{u} LCS_{\bigcup}(r_i,\, C)}{m} \tag{2.7}$$

$$P_{LCS}(C, R) = \frac{\sum_{i=1}^{u} LCS_{\bigcup}(r_i,\, C)}{n} \tag{2.8}$$

As usual from precision and recall also the F-score can be derived averaging the previous scores, both at sentence and summary level (as expressed by equation 9). However, at least in the DUC conferences, usually β is set to a high number so that the resulting F_{LCS} value is strongly biased towards R_{LCS}, as this accuracy metric reflects more accurately our intuition and our expectations about how a *good* summarization system should perform.

$$F_{LCS} = \frac{(1+\beta^2)R_{LCS}P_{LCS}}{R_{LCS}+\beta^2 P_{LCS}} \tag{2.9}$$

Although these equations can be applied alone, most data-sets (including the DUC conferences corpus which we have used to evaluate our system) provide instead multiple references. In such case the overall score can be computed by taking the max of the scores for each candidate-reference pair, as in the following equation.

$$Score_{multi} = \max_i \left(R_{LCS}(C, R_i)\right) \tag{2.10}$$

Rouge-W The main limit of the previously considered metric is that it doesn't differentiate depending on how a given subsequence is embedded within the parent sequence: as sub-sequences are identified by a set of non decreasing indices, with no constraint nor reward for consecutive matches. *Rouge-W* is a variant of the previous approach that takes this into account and weights longest common subsequence accordingly. The metric can be computed efficiently using a dynamic programming algorithm, as described by Lin [46].

Rouge-S, Rouge-SN and Rouge-SU It is of interest to generalize the concept of n-gram also in a different direction, removing the constraint of adjacency of words. In *Rouge-S* we thus consider *skip-bigrams*, which consist of pair of words, constrained only to be arranged in sentence order. The main shortcoming of the Rouge-S performance measure is that, if any distance is allowed, spurious pairs, such as *"of the"* or *"the the"*, dominate the overall result, undermining the reliability of the measure. An extension is *Rouge-SN*, that reduces this issue limiting the distance allowed between words belonging to a given skip-bigram. Finally, *Rouge-SU*, extends the metric adding uni-gram counts to give some credit also to sentences with no word pair co-occurring in the reference.

2.2.2 Correlation with human judgment

The metrics described in the previous section allow us to compare the performances of different automatic summarization systems in a cheap and fast way. It is however natural to wonder how reliable such evaluation is. During the last decade several studies have been conducted in order to evaluate the degree of correlation between automatic and human evaluation of summaries. The results [19] show that, if a single human reference is used, the reliability of automatic evaluations is low (as even the inter-human agreement can be low). However, if multiple reference summaries are employed, as shown by Lin and Hovy [20], the results of automatic evaluation are quite reliable, although not *100%* correct. Furthermore, in several experiments, automatic and human scoring have shown also a higher degree of correlation in summarization with respect to analogous systems in machine translation. A possible explanation for this result is that the adoption of an extractive framework for summarization makes grammaticality less of an issue than it is for translation.

2.3 Trivial Heuristic Approaches

2.3.1 Leading Sentences (Lead)

The most simple approach in order to generate the summary of a single document is to greedily extract the first sentences of the document, adding sentences to the summary until the fixed word budget is fully exploited. Note that once a sentence which would make the summary exceed the budget is reached, the subsequent sentences should still be considered for extraction (as they could still fit in the budget, if shorter then the previous one). We implemented this system as first most simple baseline, to be used as comparison during the evaluation of our system.

Pros: This very trivial approach can lead to surprisingly good results when applied to single news documents, thank to the journalistic best-practice of beginning the article with a brief summary of the relevant facts (the Five Ws rule).

Cons: The good performance exhibited for single document summarization of news articles, does not necessarily hold in a more general setting. Furthermore, even in the case of news documents, this approach has a serious shortcoming: it's extension to multidocument summarization leads to just selecting the first sentence of a small random and non representative subset of the documents in the collection.

2.3.2 Frequency based ranking (Freq)

An alternative simple heuristic, initially proposed by [49], is the following: to extract the main topics of a document, we assume that the most important topics are those analyzed more in depth, and we identify the relevant sentences as those whose words are more frequent in the rest of the document. However, the idea that, if a sentences contains the most recurrent words in the text, then it covers most of the topics of the text, is obviously questionable. We have not implemented such idea as reported performances are insufficient, and there is little space for improvement.

Pros: Trivial to implement. With respect to its most naive implementation, it can be improved in a few directions: by eliminating stop-words, by using stemming in order to match more abstract concepts rather then basic words, and by using TF-IDF scores to weight in a principled way the importance of a word occurrence.

Cons: It requires empirical tuning, combining stop word elimination, stemming and TF-IDF to manage the Zipfian distribution of word occurrences and yield meaningful results. Moreover, the heuristic is not grounded in some common practice as the previous one. Finally, it's not extendable to multidocument settings: not only it doesn't avoid redundancy, but it might even boost it.

2.4 Supervised Approaches

2.4.1 Relevant Sentences (Rel)

Heuristic methods, such as the leading sentences method, can behave very poorly outside their domain. A better approach, based on the seminal paper [24], is to rank the sentence according to some relevance score, defined in a parametric way (with parameters learned from training data), and then, as for the leading sentences method, extract greedily the top scoring sentences until the summary does not exceed the given budget. It is to be noted that the leading sentences and the frequency ranking approaches could be considered as special case of this *relevant sentences* approach, in which, however, the relevance score is assumed proportional respectively to the position in the document and to the frequencies of the words in the sentence (in both cases no learning is needed as a single feature is used). This approach was the first non trivial one we did implement. In Chapter 5 we evaluate several proposals of improvement with respect to the idea here presented, and show their impact on performance (in particular we will consider more sophisticated regression methods with respect to the one currently employed and presented below).

2.4.1.1 Score Functions

The most common way of implementing the relevant sentence method is to define the score as a linear function of a certain number of sentence features, and then learn the corresponding weight in a supervised manner. In principle more complex functions of the features could be considered but this is not practical as the available data does not support learning more complex approximators. For a given sentence s_i this corresponds to defining the following score function:

$$score_{rel}(s_i) \;=\; \theta^T \boldsymbol{f}\left(s_i\right) \tag{2.11}$$

Where s_i denotes the sentence, $\boldsymbol{f}\left(s_i\right)$ the sentence features, and θ the weight vector.

Pros: Using a more complex scoring function allows to combine different sentence features and find a good trade-off between them, yielding good results in single document summarization. It is very flexible, as many other ideas and approaches can find application as features within this framework.

Cons: This approach is not optimal for multidocument summarization, as it lacks of a mechanism to avoid redundancy. Also, in early days, due to lack of training data, manual tuning was needed. This is not anymore an issue, as we describe in Section 2.5.3, yet currently only simple machine learning algorithms are used to determine θ, and there has been little work on using more sophisticated approaches to regression.

2.4.2 Maximum Margin Relevance (MMR)

The *maximum marginal relevance* approach explicitly penalizes redundancy by maximizing a linear combination of suitable relevance and redundancy scores. Redundancy of sentence pairs is computed from additional features.

$$score_{rel}(s_i) \;=\; \theta^T \boldsymbol{f}\,(s_i) \tag{2.12}$$

$$score_{red}(s_i,\, s_j) \;=\; \omega^T \boldsymbol{f}\,(s_i, s_j) \tag{2.13}$$

In literature also redundancy score, as relevance, is linear function of feature values, requiring an additional parameter vector ω to be learned.

Pros: This model deals with redundancy, as we needed, and it's sufficiently flexible to allow experiment many different ideas.

Cons: A more complex model implies more parameters to be learned; therefore more training data might be needed to learn the redundancy score function. Linear functions might not be sufficient to effectively represent relevance and redundancy.

2.4.2.1 Greedy Ranking

A first way of maximizing the combination of redundancy and relevance score was proposed in [15]. The proposed approach is to add sentences s_i to the summary, denoted by S, by greedily maximizing at each iteration the following expression which trades off sentence relevance and summary redundancy:

$$\lambda\; score_{rel}(s_i) - (1 - \lambda)\; score_{red}(s_i, S) \tag{2.14}$$

The choice of the parameter λ is crucial, but a suitable value can be easily found holding out some validation data. It has been observed that varying dynamically the value of lambda can be helpful: start with small values (0.3) to give more emphasis to novelty, and then increase it (0.7) to focus more on relevance.

Pros: Once the weights of $score_{rel}$ and $score_{red}$ have been learned, the derivation of a summary for a new collection of documents is completed very fast.

Cons: There is no guarantee of finding a global maximum of the previous expression. More specifically, the algorithm tends to favor at the beginning overly long, although only partially relevant, sentences. Due to the greedy nature of the algorithm this can subsequently prevent other short more relevant sentences from being introduced (however, suitable length penalization may solve the issue).

2.4.2.2 Knapsack problem and dynamic programming

The maximization problem we are trying to solve can be seen as an instance of the knapsack problem. We can therefore dynamical programming algorithm to solve it, reducing the issues highlighted in Section 2.4.2.1. An algorithm is presented in [60], to optimize the objective function by filling a $n \times K$ table, where element $(i,\, j)$ of the table is a high scoring summary of exactly length k containing only sentences draw from the first i. The table is filled progressively considering two previously evaluated summaries during the computation of each element.

Pros: The advantage of this approach is that it limits the errors caused by the greedy algorithm of section 4.2.1.

Cons: Significant overhead with respect to greedy approach. Not viable given our time constraints. Still no guarantee of finding a global optimum.

2.4.2.3 ILP formulation

Although the previous approaches are simpler to implement, better results can be achieved if the global optimization problem is solved, as proposed again by [60]. In order to do so we must first reformulate the problem, defining the summary S as set set of sentences that maximizes the following expression:

$$\hat{S} = \underset{S}{argmax} \left\{ \sum_{s_i \in S} \lambda \; score_{rel}(s_i) - (1-\lambda) \sum_{s_i, s_j \in S} score_{red}(s_i, s_j) \right\} \tag{2.15}$$

This expression can be cast to a Integer Linear Programming problem, defining 2 variable sets μ_i, $\mu_{i,j}$, specifying, respectively, if a given sentence is extracted and if a given pair of sentences is extracted (and this uniquely defining the summary). Denoting with α and β the weights assigned to each component:

$$(\widehat{\mu}_i, \widehat{\mu}_{i,j}) = \underset{\mu_i, \mu_{i,j}}{argmax} \left\{ \alpha \sum_{i=1}^{M} \mu_i score_{rel}(s_i) + \beta \sum_{i=1}^{M} \sum_{j=1}^{M} \mu_{i,j} score_{red}(s_i, s_j) \right\} \tag{2.16}$$

Pros: The quality of the result is, in general, significantly improved with respect to the greedy approach, as you are guaranteed to find optimal solutions.

Cons: The reported speed of such systems if too low. The approach can be made computationally less costly dropping the redundancy score, this is however not viable for us, given our focus on large collections of strictly related documents.

2.5 Features and Score Functions

2.5.1 Relevance Features

Relevance is obviously a subtle concept, because it depends on the context. For summarization it is usually interpreted as relevance with respect to a given topic, and its measured using the similarity between the sentence and the query associated to the topic. The query is usually either a set of keywords or a natural language sentence, in our case could be interesting to use the availability of documents that span a significant time range to enrich the query using older documents retrieved for the same feed (this would provide us more richer contextual information)

Syntactical features

- *Occurrences* [O]: number of times the non-stopped words/stems/ngrams in the sentence appear in the collection of documents to summarize [24].
- *Title-Cooccurrences* [TC]: number of words (or equivalently stems / ngrams) occurring in the sentence and simultaneously appearing in the title/headline of the document [24].
- *Position* [P]: position of the sentence in the article [24]. The first sentence in the document gets the score 1 and the last one gets a score of $\frac{1}{n}$ where n is the number of sentences in the corresponding document [1].
- *Length* [LEN]: number words in the sentence; often this feature is computed ignoring *stop* words (common non discriminative words)
- *Alert-words* [AW]: number of sentence words matching a precomputed listed of cue words such as “in conclusion”, “most importantly”, etc ... [24].
- *inFirstN* [FN]: binary feature indicating whether the sentence is located among the first N (in paper N=5) in the document it belongs to [1]. Note that usually research papers assume totally clean textual documents (in our case at the beginning there is often meta data, image captions, etc).
- *isStarter* [ST]: binary score if the sentence starts with a query term (in paper, if sentence starts with “the”, gaps up to 4 words are allowed to provide space for adjectives: “the ADJ ADJ query-term”). [1]
- *Part of Speech Tag* [POS]: grammatical role of the word inside the sentence (common grammatical categories are *Nouns*, *Verbs*, and *Adjectives*)

Semantic features

- *Tf-Idf* [TI]: weighted sum of the number of occurrences of the query words in the sentence; the raw count is offset by the inverse document frequency [64].
- *NamedEntityCount* [NEC]: number of named entities appearing in the sentence. Named entities can be identified using Stanford's NER library. NB: in Signal's data most named entities are already identified.
- *QuerySentSim* [QSS]: cosine similarity (*cos-sim*) between the query and the sentence *tf-idf* vector representation [1]. NB: maybe better with word/sentence embeddings (Section 2.5.2).
- *CentroidSentSim* [CSS]: *cos-sim* of centroid and sentence *tf-idf* vector. The centroid is computed from the 100 most frequently occurring words in the collection [1]. NB: could improve using embeddings (Section 2.5.2).
- *TopicSentSim* [TSS]: cosine similarity between the topic (identified by a natural language sentence or set of keywords) and the sentence vector representation. NB: maybe better with word/sentence embeddings (Section 2.5.2).

Unsupervised Summarization features

- *TextRank* [TR]: TextRank is an unsupervised score that measures the relevance of a sentence using the same principle used by PageRank for web retrieval: it computes relevance as the long term visit rate of a sentence in a random walk where transitions are function of cross references.

2.5.2 Redundancy Features

Concerning redundancy features, many of the similarity measures used for relevance have been used also for redundancy, but applied to *sentence-summary* pairs instead of *sentence-query* pairs. Very common are:

- *Intersection* [IS]: number of tokens cooccurring in both sentences divided by the average length of the sentences.
- *Ngram cosine similarity* [OVP]: cosine similarity between the bag of ngrams vector representation of the sentences.

Although reasonable results can be achieved with such features, redundancy identification is an area with particular space for improvement. The reason is that the features currently employed are not very effective, due to problems in dealing with synonyms and paraphrases. In Chapter 3 we present some deep learning techniques which we have used to improve redundancy measures.

2.5.3 Learn Good Score Functions

Whatever the natures of the feature which are being computed, in order to measure relevance and redundancy, in order to get good *score functions* it is essential to use a sound approach for learning the coefficients that are used to combine the feature values into appropriate sentence score functions.

2.5.3.1 Rank SVM

An interesting approach, employed by Martins et al. [50], is to use *rankSVMs* to learn a set of feature coefficients consistent with a gold-standard ranking, created sorting the sentence according to the Rouge score with respect to human reference summaries. An advantage of this approach is that *SVMs* are a sound, well tested regression algorithm, for which several open-source implementations are available.

2.5.3.2 Support Vector Regression

An alternative approach found in literature is to rely on a simple procedure for automatically scoring using reference summaries sentences taken from conference collections, and then learn the score functions using *Support Vector Regression*.

Generating training: The conferences provide us with multiple summaries for each of several groups of documents (related to different topics and associated with an explicit "query" - a sentence or a set of keywords). Thus, we need to convert a binary classification (relevant vs non-relevant) into a set of continuous score labels. Li et al. [61] proposed an automatic procedure to do so efficiently: they used a similarity measure between each sentence and the human summaries.

$$Sim(sent, sum) = \frac{\sum_{t \in sent} \sum_{t' \in sum} I[t == t']}{|sent|} \tag{2.17}$$

This extremely simple similarity measure is used to score each sentence with respect to each of the human summaries available for the given topic, and then the max among these is taken as score of the sentence. In alternative, the average can be taken, but using the max to generate the score has shown to lead to better scoring functions. Finally, more complex similarity measures could be used (e.g. cosine similarity or higher order ngrams) to improve quality.

Support Vector Regression Once a data set of scored sentence is provided, Machine learning provides a principled way of combining the feature values into a score function. [61] propose to use SVRs to learn the coefficients. This approach is sensible as support vector regression is known to provide robust models also when limited training data is available (a key issue for summarization).

2.6 Unsupervised Approaches

In literature also *unsupervised approaches* have been proposed: these methods exploit the structure and the relations among words and sentences in a document. The most popular unsupervised summarization systems are *graph* based or *coverage* based. The main advantage is that this approach does not require parameter tuning. The main shortcoming is that it does not generalize well from single document to multiple document summarization, because often the analysis is intrinsically document-wise, and cannot deal with redundancy. However, it is still worth considering a couple of unsupervised approaches, because they can provide additional structural based relevance features, to be used within the supervised framework.

2.6.1 Graph-based: TextRank (TRank)

TextRank [52] and *LexRank* [28] are two different names for the same algorithm for single document summarization, independently proposed by different research groups. The algorithms is inspired by the famous PageRank algorithm, used by Google for web-retrieval. A graph is built associating each sentence of the document to a vertex, and creating edges between pair of sentences whose similarity is above some threshold. The similarity measure can be the n-gram overlap or the cosine similarity between the tf-idf vector representations of the sentences. Given a graph, the rank of the relevance of each sentence is then the long term visit rate of each node of the graph, under a random walk whose transition probabilities given by the weights of the graph edges. This long term visit rate can be computed efficiently using the power method for determining the principal eigenvector of the transition matrix.

2.6.2 Coverage-based: Topic Analysis (Top)

Coverage-based models, seek to extract a set of sentences covering all topics present in the document collection and in the query. Interestingly this implicitly defines a trade-off between relevance and redundancy (the latter damages the amount of coverage). In order to follow this approach it is necessary to use topic analysis algorithms to assign words and sentences to topics. This has been done for example by Miller et al. [41], using *Latent Dirichlet Allocation* (or LDA), which is based on a generative probabilistic model of topics and of words given topics. LDA interprets a document collection in terms of topic-word compositions and document-topic mixtures. Reversing the generative model we can assign topics to words using Gibbs Sampling, and once topic-words compositions are found, we can form an aggregated weighted vector representation of the collection and select sentence in order to minimize the divergence (*KL* or *Jensen-Shannon* divergence) between the summary and the original collection.

2.7 Post processing

2.7.1 Sentence Compression

Extractive summarization can perform reasonably well in a wide range of circumstances, however, it is not as human summarization usually works, as the latter tends to paraphrase the content of a document rather then just extract a set of sentences from it. No successful methods have been proposed for this more abstractive summarization task in its most general form, however, very recently, a simplified form of abstractive summarization has received some attention: *Compressive Summarization*. This is most often used as a post-processing step, with important results achieved by Jing [42], using syntactical, contextual and statistical information for compression. In general, most compression methods are based on the deletion of words or part of the sentence, and therefore require parsing in order to do so preserving grammaticality (among the many possible open-source tools available for parsing the Stanford Parser is one of the most popular). Other impressive results have been achieved by Martins et al. [51], [50], who worked on joint extraction and compression. However, the proposed joint systems are not as yet efficient, and the computational overload is not suitable for our application domain. Concerning our project, sentence compression has not been implemented yet, but will be introduced as a post processing step in the follow up of the project.

2.7.2 Sentence Ordering

In order to maximize the readability and understandability of the result, the order in which the sentences of a summary are presented is potentially relevant. In single document summarization it is possible to simply preserve the ordering of the original document. However, in multidocument summarization, combining multiple partial orderings into a total ordering of sentences is non trivial. Several interesting methodologies have been proposed, but none is fully satisfying, therefore alternative ideas are developed and evaluated in Chapter 7.

***Majority Ordering*:** this approach was proposed by Barzilay et al. [59]. First, sentences from different documents, discussing the same themes, topics or entities are clustered (for example using SimFinder, developed by Hatzivassiloglou et al. [73] as a sentence clustering tool for summarization). Subsequently, a graph having the topic cluster as vertices is built, encoding in the edges weights the strength of the local orderings in the original documents: more specifically, the weight is set to be equal to the number of documents in the collection which satisfy that local ordering. Once the graph has been built, determining a global ordering is an *NP* complete problem, however, heuristic procedures can be used to solve it with reasonably good results (as shown, together with a proof of *NP* completeness, by Cohen et al. [76]).

***Chronological Ordering*:** Another approach proposed in the same paper [59] is, after performing topic clustering, to use chronological order of events to guide the arrangement of topic. This is done by associating to each topic a time-stamp, corresponding to the publication date of the first sentence contributing to the topic. Topics are then arranged according to the time-stamp, and, if two topics have the same time-stamp (and thus appeared for the first time in the same document) the document local ordering is used. This approach is less general then the previous one, being for focused on the presentation of events. Other shortcomings are that sentences providing background information might mislead the system and that the result often contains abrupt changes of topic. This last issue has been partially fixed by subsequent extension of the algorithm such as the augmented algorithm (also by Barzilay et al. [59]).

***Local Coherence*:** Barzilay and Lapata [58] more recently proposed a more general approach to information ordering then the two previous ones, based on chronological order. The idea behind this approach is to try to enforce global coherence by ensuring at least a form of local coherence, measured by the *continuity* of adjacency sentences. In the paper, continuity is defined as function of the entities occurring in the sentences. In order to measure it, Barzilay and Lapata propose to create an entity grid, showing the sentences vs the entities cited in the text. This matrix structure can be filled by assigning to element *(i,j)* one value among *Subject*, *Object*, *Other*, and *None*. Given this matrix we can then compute for any candidate summary the number of transitions of each type in adjacent sentences (*e.g* from Subject to Object, from Subject to None, etc ...). This provides us with a simple representation of any candidate summary. The next step is to create a corpus of pairwise rankings composed by a document and one of its random permutations. Finally an SVM ranking model can be trained on this data-set of pairwise rankings, to learn to predict which is the most coherent text. The problem with this approach is that only allows to evaluate two orderings relatively to each others, so finding the optimal ordering would still require an exponential number of evaluation.

***Hidden Markov Models*:** A last approach, theoretically very appealing, was proposed by Barzilay and Lee [57]. Their proposal is to train a *Hidden Markov Model* in order to learn what is the common information flow in the documents, and then rearrange sentences in order to maximize the likelihood of the observed summary under the given generative model. This probabilistic model has several advantages, as it provides solid theoretical grounding and efficient inference algorithms are available for this particular class of graphical models. Furthermore, the method has proved quite effective for domain specific summarization. However, the models is not applicable in general, as presentation flows depend on the topic/theme.

Chapter 3

Deep Learning for NLP

In this chapter we give a brief overview of the main results in deep learning, analyzing the main advantages and limitations of such approach. Then we give more details concerning its application to natural language processing. More specifically, we present distributional representations of words and sentences and see how the representations generated by deep models relate to classical derivations.

3.1 Deep Learning

Deep learning is a branch of machine learning that relies on complex *layered* or *hierarchical* architectures to learn high level abstract *representations* of data. This field has received considerable attention since *2006*, when several publications - e.g. Hinton et al. [34], Bengio et al. [8] - proved the potential of this class of algorithms. Among the deep models which have been used more extensively in recent years, there are several classical neural networks architectures. This is due to the fact that the training of these models is particularly suitable for intensive parallelization using large clusters or *graphical processing units*. Furthermore, the availability of large data-sets for most mainstream machine learning tasks makes the size of their parameter spaces less daunting than it was in the past. These facts, together with smarter ways of initializing the parameters values (e.g. Erhan et al. [25]) and preventing overfitting (e.g. Nitish et al. [56]), makes it now possible for deep neural networks to provide state of the art solutions to a wide range of challenging problems. *Machine vision* in particular has been among the fields where deep learning as achieved the most impressive results, with deep convolutional neural networks claiming state of the art results on most classical data-sets (for example the MNIST handwritten digits data-set - Wan et al. [74] - and the ImageNet object recognition data-set - Simonyan and Zisserman [66]). Also in the field of *speech recognition* neural networks are among the best performing algorithms, either alone (such as in Mohamed et al. [55]) or in combination with other models such as *HMM*s (as in Seide et al. [63]). The nature of this field is however suitable for exploring a wider range of deep architectures, including convolutional neural nets and autoencoders for feature extraction, and recurrent neural nets for dealing with the sequential nature of the problem. *Natural language processing* tasks have been addressed using deep learning architectures with a few years of delay with respect to the previously cited fields, as *NLP* was traditionally relying on handcrafted features and shallow classification algorithms. However, in only a few years the applications of deep learning methods to *NLP* have flourished: in particular recurrent neural networks were used to provide state of the art results in sentiment mining (Socher et al. *[69]*), parsing (Socher et al. [67]), and language modeling (Mikolov et al. [53]). Convolutional neural networks have also been widely employed in *NLP*, initially for *word embedding* (e.g. Collobert et al. [16]), and lately also for *sentence embedding (*Kalchbrenner [43]) and the use of such embeddings in machine translation (Kalchbrenner *[44]*). Word and sentence embedding is the form of deep learning which we have been mostly interested in for this project, as it provided us with dense vector representations of sentences, which we used to detect redundancy more effectively with respect to classical sparse bag of words representations.

3.2 Why deep learning?

Observing the successful adoption of deep learning models in such a wide range of fields it is natural to wonder the reason that make deep architecture suitable for certain tasks. In order to understand this, an analysis of some theoretical results on the representation power of deep networks with different number of layers is appropriate. When neural networks first became popular, research on the theoretical properties of such models provided several results strongly suggesting that using more than a few layers was not needed.

For example, Hochreiter [37] showed in his thesis that the traditional difficulties in training by back-propagation neural networks with multiple layers were due to a fundamental problem, which affects both feed-forward and recurrent networks: cumulative error signals back-propagated through the network either explode or, most often, decay exponentially with the number of layers. Further studies of the stability of such signals from a dynamical systems perspective (e.g. Bengio et al. [6]) confirmed that vanishing gradients are very difficult to avoid if the system is reasonably robust to noise. Finally, a famous theorem by Cybenko [21], known as the *universal approximation theorem*, showed that, at least for feed forward neural networks, a single hidden layer with finite number of neurons is sufficient to approximate with arbitrary precision any continuous function on a compact subset of R^n. All this evidence suggesting against deep learning had a huge impact on neural network research for many years, with shallow models being largely preferred to deep architectures.

However, it is now understood that these results were partially misleading. For example, the vanishing gradient problem can be partially addressed in various ways. First, the availability of GPU capable of massive parallelism is, alone, enough to train networks with a few more layers than the classic architecture with a single hidden layer (although the issue still holds for very deep networks). Second, smart ways of initializing and pretraining the layers of the network also can lead to significant improvements with respect to naive back-propagation [8] . Third, non gradient algorithms can be used (e.g. hessian free methods). Fourth, special architectures, such as recurrent nets with LSTM units are not affected by the issue (Hochreiter [38]). Also concerning Cybenko's theorem, although it states that a shallow network can approximate any continuous function, it does not say anything about how easy it is to *learn* that function, nor how many neurons are required for the single layer of the network. Investigations on the computational complexity of representing circuits using neural networks actually show that functions represented by deep architectures with few nodes might require an exponential number of nodes to be approximated by a more shallow model (Bengio [7]). Given that using more nodes implies more parameters to be learned, this is a strong indication that deep architectures can be a better choice for many supervised learning task (where some function must be Lent from data). Besides such theoretical insight, empirical investigations on the result of training deep architectures have shown that there are also more intuitive reasons that can explain the success of deep learning architectures: multiple layers allow the construction of hierarchical representations where higher level concepts are represented as a function of features extracted at lower levels in the network. In machine vision, for example, it is possible to visualize very effectively the type of inputs the various nodes react to (see Erhan, Bengio et al. [26]). Doing so, it is easy to see how, at lower layers, edges and simple shapes are recognized, while, at higher levels, there are specific nodes that activate when more abstract concepts (such as *cat* or *man*) are identified in an image. This ability of learning effective representations of data goes far beyond machine vision, and we will see it is the aspect of deep learning we are more interested in. In *NLP* it is however less obvious to understand the nature of the representations, but in the next sections we will give some intuition about their interpretation. In order to do so we will start with comparing discrete representations, classical continuous representations and semantic embeddings. Then we will describe the alternative approaches used to generate semantic embeddings (this will include a review of matrix factorization algorithms and their comparison to neural embeddings). Finally we will present how deep learning algorithms combine word representations into meaningful sentence embeddings.

3.3 Discrete vs continuous representations

In natural language processing, *embedding* is the process of mapping words or sentences to continuous dense low dimensional vectors (with at most a few hundred elements), in such a way to encode not only the identity of the words and sentence but their *semantic* value. This is very different from the discrete representation of word, sentences and documents known as *bag of words*. In such classical representation, a word is one-hot encoded as a high dimensional unitary sparse vector (with a single *1* at the index uniquely assigned to the given word, and as many elements as the words in the vocabulary), and sentences are just the sum of the word's vectors (with each index of the sentence vector containing the count of occurrences of the corresponding word). At a word level, the limit is that there is no semantic al relation among words: all words are orthogonal to all others, no matter how similar the meaning. At a sentence level, there are further limitations: first the order of words in the sentence is totally lost, second the representation of two sentences gives the same weight to each word. The loss of information due to the first issue is considerable: also humans would find very difficult to perform even simple natural language understanding tasks. Furthermore, in combination with the zipfian distribution of words in natural language, the second issue makes the similarity of two sentences depend mostly on common non discriminative words. A first improvement with respect to this basic discrete representation was proposed in the early days of information retrieval: use continuous representations and make the value of an index on the vector depend on the probability of that word occurring in language (the most famous way of doing so relies on the so-called tf-idf weighting [62]). This continuous representation is certainly an improvement but does not solve the most important issues: the fact that all word are orthogonal, and the fact that order of words in a sentence is lost. The vector embeddings which we analyze in this Chapter do a step more, and aim at encoding semantic aspects in their vector representation. This allows them to preserve better the actual meaning of words and sentences and has been proved to allow significant performance improvements in several natural language processing tasks. Deep Learning methods have been shown to be particularly suitable at generating good embeddings, especially at a sentence level. For example, in paraphrase detection the state of the art on the *MSRP* corpus is currently claimed by a system using recursive autoencoders to build semantic representations of sentences (Socher, Huang et al. [68]). Given that for multidocument summarization it is crucial to identify redundancy in the information content at a sentence level, in this project we have therefore explored several approaches to building semantic al embeddings for words and sentences using deep neural networks. In this chapter we present the literature review in the field, for both word and sentence embeddings, considering alternative approaches to learn such representations.

3.4 Word embeddings

Generating vector embeddings capable of mapping a large vocabulary into a finite dimensional vector space, while preserving relevant semantic al feature is a complex task, and a central one in *computational semantics*. This idea also has a long history (see for example Hinton et al. [35]), although only recently has become widely applied. There are many different approaches that can be taken in order to learn such representations. The various models proposed with this aim differ partially in the kind of semantic relations that they can learn, and also in the evaluation metrics they respond best. However, all approaches rely (explicitly or implicitly) on a common underlying idea, known as the *distributional semantics* hypothesis. This hypothesis states that the *meaning* of a word is defined by its *context*, and therefore *linguistic items with similar distributions have similar meanings* (Harris [33]). Within this general principle, there are two main classes of methods to learn meaningful word embeddings:

- A first group of models, which we shall denote as *global*, generates word embeddings by counting and modeling statistical co-occurrences of words.

- In *local* methods, instead, only a small window around each token is considered and the word embeddings are usually parameters of some complex model predicting some language feature.

The advantage of the former class of representations (which includes *LSA* and *GloVe*) is that it can leverage efficiently all the available knowledge in order to learn the vector embeddings. However global methods are often computationally expensive. Instead, local approaches, which include the famous *word2vec* (described in Mikolov, Chen et al. [71]), compensate the limitation of relying only on local information by allowing efficient and highly parallel training algorithms, based on various forms of gradient optimization. Furthermore, methods belonging to this class can exhibit a rich vector space structure, with geometric relations encoding meaningful semantic relations. For example in *word2vec* embeddings, linear combinations of vectors to exhibit consistent semantic interpretations such as $king$ - man + $woman$ = $queen$. (see Mikolov, Sutskever et al. [72] for further examples). This feature is potentially very important to go beyond word representations, using compositionality to learn sentence embeddings, which is what we are most interested for summarization. Although in our project we will focus on windows based approaches, in the next sections we will give, however, an overview of both. For a more comprehensive comparison we refer to Baroni, Dinu et al. [3].

3.4.1 Global methods

This class of algorithms for deriving dense continuous representation of words relies on global statistical information about a corpus. This information is usually encoded in co-occurrence matrices, and, most of the times, the algorithm is based on deriving low rank approximated factorization of such matrices.

LSA: *Latent Semantic Analysis* is one of the most popular approaches to semantic vector representations of words. It was originally introduced for information retrieval by Deerwester et al. [22]. The algorithm uses singular value decomposition to factorize the term–document *co-occurrence matrix* of a large corpus of textual data. Then *LSA* can be used to explicitly learn k-dimension word vectors by selecting the elements corresponding to the k largest singular values from the word vector representation in the new basis. This is a principled way of extracting a low dimensional representation of words which minimize the reconstruction error with respect to the original one and which perform reasonably good on word-analogy challenges. However, as for most global methods, this process is computationally very intensive, as the time complexity of singular value decomposition is cubic in the size of the term-document matrix.

LDA: *Latent Dirichlet Allocation* [11] is a three level hierarchical generative graphical model for corpora of documents. The model was originally developed for information retrieval, but it can be easily used to derive k-dimensional vector representation of words. *LDA's* model interprets each document of a collection as a mixture over latent topics, and each topic is characterized by a distribution over words in a vocabulary. The word vectors can therefore be generated training such topic model and using the conditional probabilities $p(w|T)$ to fill a word–topic matrix, whose rows constitute the word embeddings. As for many graphical models, exact inference is intractable, however efficient approximation algorithms (including *EM*) are available. For what concerns our project, the main limitation of *LDA* is that the emphasis is on modeling topics, not word meanings: therefore resulting word vectors are not always sensible as points in a k-dimensional space, as their distance does not relate to how similar their meaning is, but rather how similar is their occurrence rate in the various topics.

GloVe The previous models efficiently exploit all statistical information available, exhibit interesting properties and have been used successfully for many tasks in the past. However, the space in which the vocabulary is embedded does not exhibit a very rich structure and geometrical relation have often little semantic meaning (this is is potentially limiting when we seek to compose word vectors in suitable sentence embeddings). *GloVe* [40] is the only global statistical method that can compete with local methods on this aspect, and also performs better than the previous on word analogy tasks. It is a *log-bilinear* model with a weighted least-squares objective, based on the observation that ratios of word-word co-occurrence probabilities have the potential for encoding semantics better then simple co-occurrences (which do not manage effectively common (non discriminative) words). The problem with GloVe vectors is that it requires to store massive matrices in memory and online training is not available, for this reason, although the quality of the word embeddings generated with this model is very good, we have decide to focused on local methods.

3.4.2 Local methods

We will now consider a different class of algorithms for learning word vectors. These algorithms iterate over the words of large corpus and consider, at each step, only a small window around every given word. These techniques have the advantage with respect to the previous global methods that can be trained on bigger corpora, with billions of words and millions of entries in the vocabulary. This allows to make up of the loss of global information, yielding overall to better results. The proposed techniques also exhibit a rich vector space structure, suitable to support compositionality [72].

Feed-forward Neural Net Language Model (NNLM) The first probabilistic feed-forward neural language model for learning distributed representations of words was proposed by Bengio et al. [77]. Training is pursued by sliding a small window of size n over the lines of a corpus, attempting to predict the next word using a feed-forward neural model. The network takes as input the index in a vocabulary of size v for each of the n words in the window. The indices of the window's words are then mapped to continuous vectors of size k (whose elements are also parameters of the model), forming a projection layer to whom a non linear function is applied. The projection layer is subsequently mapped to a hidden layer of size h and topped with a soft-max. The resulting output of the network is a distribution of probability over all entries of the vocabulary, specifying how likely it is for each entry to follow the given window. The structure is differentiable *end to end* and the word vectors parameters are therefore learned by gradient descent. The complexity per training example of a naive implementation is [71]:

$$C = n \times k + n \times k \times h + h \times v \tag{3.1}$$

This expression is dominated by $h \times v$ and, in practice, training the system requires some smart solutions to reduce the impact of this term (e.g. using non-normalized models, hierarchical versions of soft-max, or binary tree representations of the vocabulary). The model architecture is shown at Figure 1.

Continuous Bag of Words Mikolov et al. [71] [72] presented in 2013 two *log-linear* models that attempt at learning distributed representations of words reducing the significant computational complexity of the previous model. The first model is similar to the *NNLM* but without the non linear hidden layer and with the projection layer shared across words. The resulting architecture can be seen as a *continuous bag of words* (*CBOW*) because the original order of the words is lost. The network attempts to predict a given word in a text by taking as input n surrounding words. Training complexity is considerably reduced with respect to the previous model:

$$C = n \times k + k \times log_2(v) \tag{3.2}$$

The model is again trained by applying stochastic gradient descent, as for the previous model, however the complexity is reduced so much that word vectors with a dimensionality of an order of magnitude higher can be trained. Training can also be easily done on the CPU without requiring specialized hardware such as GPUs. The model provides very good results on the Semantic-Syntactic Word Relationship test [71], although performs significantly better on the syntactic part. The model architecture is shown at Figure 2a.

Continuous Skip-gram Model In the same paper, Mikolov et al. also propose a second log-linear model, named the *continuous skip-gram* model. In such model each word in the training corpus is fed as input to a log-linear classifier with continuous projection layer aiming at predicting the words within a certain symmetric window (of size w) around the current token. In order to compensate for more distant words being only weakly related, during training we give more weight to nearest ones by oversampling them with respect to the others. The complexity of the architecture is:

$$C = w \times (k + k \times log_2(v)) \tag{3.3}$$

The lower complexity implies that is possible to train the word representations on bigger corpora and this gives this model a significant advantage (in literature, corpora up to several hundred billion words have been successfully trained). The resulting vectors achieved the state of the art on the Semantic-Syntactic Word Relationship test [71], significantly improving the results of both *NNLM* and *CBOW*. More interestingly for us, given our interest in sentence embeddings, geometrical relation in the embedded space can encode also very subtle semantic relations among the words, and this allows for compositionality. Beside the previously cited example of *king - man + woman* approximating very well the embedding corresponding to *queen,* other examples include the relation between countries and capitals, between an adjective and its comparative form (e.g. big and bigger), or between an element and its symbol on the periodic table. The model architecture is shown at Figure 2b.

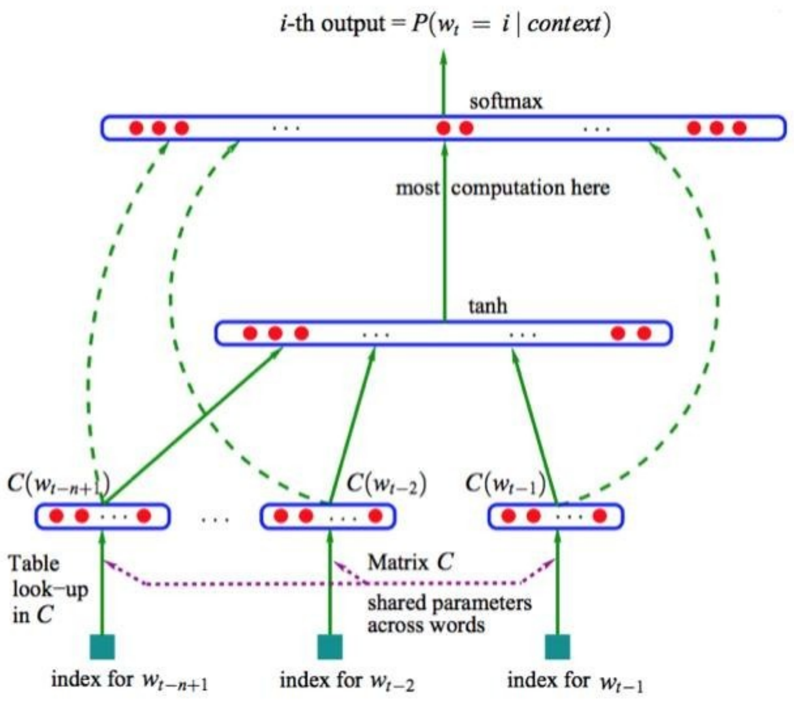
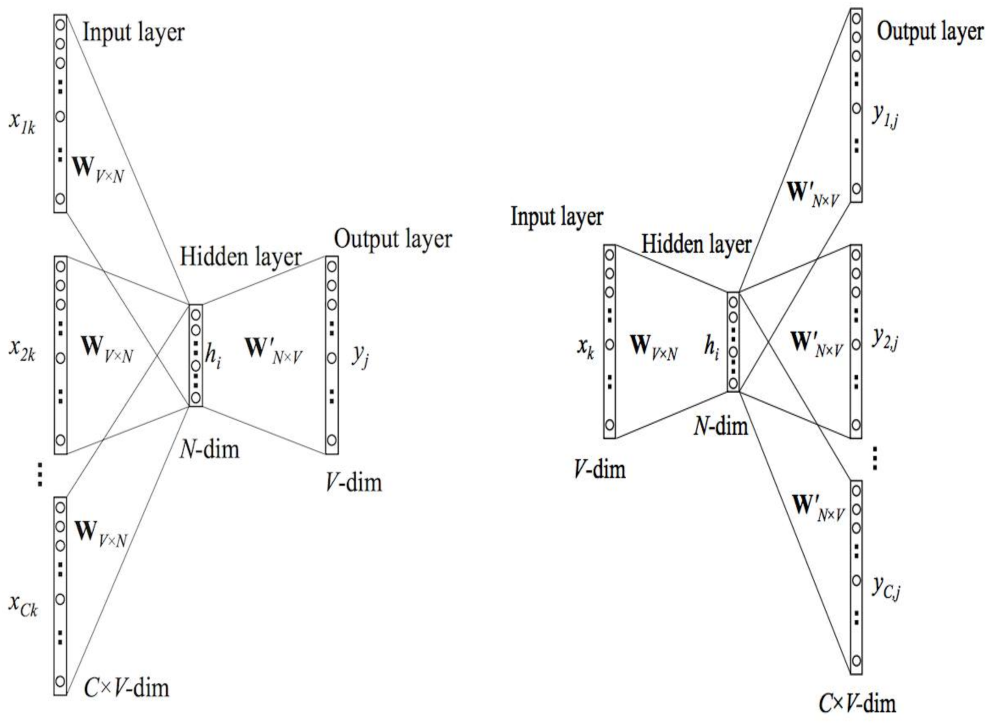


Figure 3.1: The Feed-forward *NNLM* architecture



(a) The continuous bag of words model (b) The continuous skip-gram model

Figure 3.2: The Architectures of Mikolov's *word2vec* models

3.5 Sentence embeddings

If word embedding is a topic which has been already explored in great depth, this is much less true for sentence level embedding. Learning a vector representation capable of encoding the meaning of an entire sentence is indeed a challenging task. First, while word embeddings are learned by filling a *look-up* table which maps each entry of the vocabulary to a vector, this is not a plausible approach for sentence representations (as the number of possible sentences is potentially unlimited). A more sensible approach is to use the learned word vectors as building blocks, and learn how to combine them in a meaningful way. This approach has also principled foundation, reflecting the *compositional* nature of natural language. If, for example, we choose the word2vec embeddings as building blocks in the process of generating sentence embeddings, a first very trivial approach would be to simply sum all vectors corresponding to the words in a sentence and take the result as representation of the entire sentence. Given the geometrical relations described for *w2v* vectors, this can indeed produce reasonable representations for the meaning of short sentences, and it is sufficient for tackling a wide range of tasks. For example, in February, together with Miljan Martic, Hector Mourino and Bojan Vujatovic, I used this approach to sentence level embedding in a Kaggle competition on sentiment mining. This approach allowed us to classify ourselves *14th* out of *861* competitors, a result that goes even beyond expectations. Although it might be sufficient for some tasks, the quality of the resulting embeddings is however still poor, and the chosen approach performs especially badly when sentences are long. One reason is that many type of linguistic relations are not well expressed by simple addition of the word embeddings. In order to understand this, we can consider, for example, the vector embeddings corresponding to *not, good and bad.* Semantically, we would want both the expressions $\boldsymbol{not}+\boldsymbol{good}=\boldsymbol{bad}$ and $\boldsymbol{not}+\boldsymbol{bad}=\boldsymbol{good}$ to hold simultaneously: these vector equations are however mutually inconsistent. Furthermore, another reason is that this approach totally ignores the order in which words are arranged in the sentence. If more complex relations are to be found, the order of words certainly becomes relevant, and a commutative compositional operator is unlikely to perform well. In order to rely on more complex compositions for deriving the sentence embeddings there are two paths we can follow. One path is to try to *explicitly* encode the various forms of compositionality, and use parsing to determine the order in which such rules should be applied. Alternatively, we can train a deep model to learn automatically the best way of composing words in a sentence. Both methods present significant challenges, and we now present the main results achieved by either. In general, however, the sentence embeddings generated with the second approach have had more widespread adoption, leading to impressive results in various application domains (reviewed in the next sections).

3.5.1 Explicit compositionality

Explicit compositionality is the definition of explicit rules for composing word vectors into sentence vectors. The simple idea of simply summing the word vectors can be considered a basic form of it. More complicated rules could however be defined. Just as it is for *not* (in the example of the previous section), many other linguistic elements in a sentence should actually be interpreted as operators or functions, rather then points in the vector space. This option has been first investigated by Mitchel and Lapata [54], who proposed several additive and multiplicative models for semantic composition of word embeddings. Further work by Erk and Pado *[27]* has shown that the composition of noun and verb can indeed effectively resolved using a combination of linear models and parsing. Finally, Baroni et al. [4] have shown that *adjective* composition too can be effectively represented as *linear functions*, as they change the meaning of a word by mapping the corresponding vector to some different location. One appealing fact of this approach is that a complex problem is reduced to a set of simple regression problems on suitably defined corpora. For example, linear functions for adjective-noun composition are represented using matrix and learned by *least square regression* on the corpus presented in [4]. This approach, combined with *parsing* has been shown to generate meaningful compositional embeddings, that can benefit specific NLP tasks such as paraphrases detection [27].

3.5.2 Deep models for sentence embedding

The alternative approach, which we have selected for our thesis is use a deep model to process a sentence into a semantically valid vector encoding. The variety of architectures which have been proposed for deep networks make this class of models very flexible, and the use of *GPU* computing, in combination with cloud computing services such as AWS, makes the challenge of training such complex models manageable. The main architectures which have been used for this are *convolutional neural networks* and *recurrent neural networks.* We now give a brief overview of the main results in generating sentence embedding with each of these architectures. This will also allow the reader to get an intuition of the variety of purposes that this class of representations is suitable for.

Convolutional neural networks Convolutional neural network have been first introduced in machine vision in order to make pattern matching in images invariant to rigid translations. This is achieved by using *convolutional* layers, in which small *filters* slide along the figure according to some *tile* pattern, in combination with *pooling* layers, in which only the values corresponding to strongly responding *tiles* are extracted. This constructs a translation invariant representation that preserves relative, although non absolute positions of the tiles. It can be seen that the use of the term *convolution* in the context of neural networks is different but related to its use in other fields of mathematics. There are different patterns that can be used for sliding the filters along a matrix (containing pixels or other features). The process may also include sliding over the borders if the picture has padding around it: this leads to the difference between *narrow* and *wide* types of convolution (the former does slide beyond the borders of the matrix). It has been shown that the convolutional layers transforming matrix data according to the same process are potentially very useful also for natural language processing. Indeed if, following Kalchbrenner et al. [44], the vector embeddings corresponding to the words of a sentence are arranged in order into a matrix, a uni-dimensional convolutional filter can slide across the words along each of the dimensions of the vector representations, learning to respond to specific patterns. Understanding the meaning of these patterns is not always obvious, however the sliding windows of convolution have a strong analogy with n-grams, and therefore these patterns may be intuitively understood as a generalization of n-grams. In the case of convolution, however, the extraction process operates along the multiple continuous semantic dimensions of the embedding vector space, and can therefore more effectively manage synonyms and other similar issues. Furthermore, the filters may learn to respond to patterns of different *orders* (number of words involved) and may also learn patterns that connect non consecutive words (as long as the word are within the size of the filter). Analogously to what happened in machine vision, also in NLP convolution and pooling can thus provide us with a sentence representation which is invariant of the absolute position of these continuous generalized n-grams, while simultaneously preserving the relative position of such sentence features (which instead is completely lost in standard discrete representations). Kalchbrenner's convolutional architecture built according to these idea is shown in Figure 4. The issues of feed-forward neural networks with varying length input is here managed by using *padding* to map all sentences to a fixed large dimension. Considering the distribution of sentence lengths in English language using a fixed length of 48, as we did in our experiments, is sufficient to deal will more than 99% of the sentences (and those which are longer are in general not a good choice for a summary anyway). This approach, although does not achieve as good results on next word prediction has been shown to generate very high quality sentence embeddings, providing state of the art results in a variety of very high level tasks such as machine translation [43] and sentiment mining *[44]*. Particularly impressive is the result achieved in machine translation, as the sentence representation generated according to the previously described process is sufficiently complete to be used by a second model to generate the corresponding translation in a different language. In this project we have implemented an architecture built according to similar principles (further details are given in Chapter 5), and used the corresponding embeddings to detect redundancy.

.

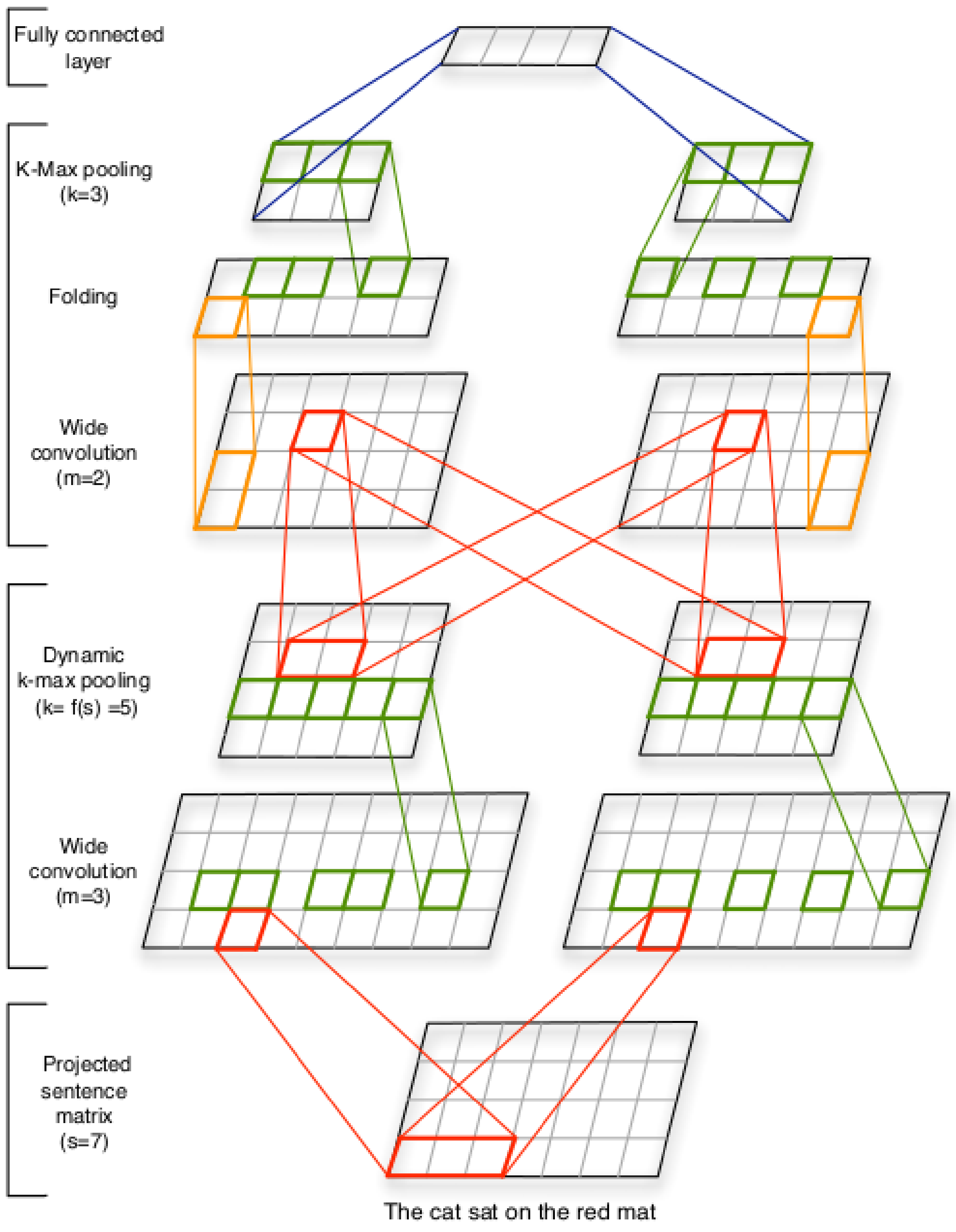

Fully connected layer
K-Max pooling (k=3)
Folding
Wide convolution (m=2)
Dynamic k-max pooling (k= f(s) =5)
Wide convolution (m=3)
Projected sentence matrix (s=7)
The cat sat on the red mat

Figure 3.3: Kalchbrenner's [44] convolutional network architecture

Recurrent networks Recurrent networks are another class of neural network models which has also been employed in a wide range of natural language processing tasks. Such networks can deal more naturally with all kind of sequential data (such as time series, speech recognition and, also, natural language processing). In particular such networks have been shown to perform well in language modeling (i.e. in predicting the next word of a sentence given the previous). Indeed, Mikolov et al. [53] have shown that even *simple recurrent network* based language models can significantly improve the perplexity on several standard corpora with respect to classic statistical language models (n-based and with various forms of smoothing). More impressively, they can even achieve this when trained with less data [53]. In order to perform well on such task, the network needs to update its internal state in order to keep track of the meaning of the sentence until the current moment, therefore, at the end of the sentence, the internal state of the network could potentially be used as sentence embedding. However, several experiments have shown that the internal representation is less sensible to distant word in the sentences: i.e. it tends to forget the past too easily [53]. Given that we are only marginally interested in language modeling (see Section 4.5), and that we are mainly interested in generating meaningful representations, we therefore used a more sophisticated variant of RNNs, that relies on the so called *long-short-term-memory* units [39]. These LSTM units are composed by a special memory cell, which is capable of memorizing the output of a given neuron (the *conditional input*), and whose behavior is controlled by the output of three other neurons (the *input*, *forget* and *output* gates). The structure of the unit is shown in Figure 3.4 and can be easily understood: the *input gate* decides whether the content of the memory cell must be replaced by the current state of the *conditional input*, the *output gate* controls if the current content of the memory cell is fed or not to the subsequent layers of the network, the *forget gate* may, or may not, reset to zero the content of the memory cell. Together, the components of this unit allow to preserve relevant information without any modification for arbitrary lengths of time. The addition of these special units to RNNs have led to state of the art results in many challenging tasks, such as handwriting [30] or phoneme [31] recognition. Concerning sentence embedding, which we are interested in, Sutskever et al. [70] showed that very good representations can be generated by such architectures, sufficiently good to be used for machine translation (by using a second recurrent network to decode the sentence embedding into a different language). However, a general problem with recurrent neural network, which is not solved by LSTM, is that training performed using *back-propagation-through-time* [75] and *parameter tying* can be extremely slow. The reason is that BPTT works by unfolding a recurrent neural network through time as shown in Figure 3.5. Although theoretically simple, the algorithm has severe issues due to the fact that the resulting error function is particularly complicated to optimize, as it presents frequent very poor local optima [10].

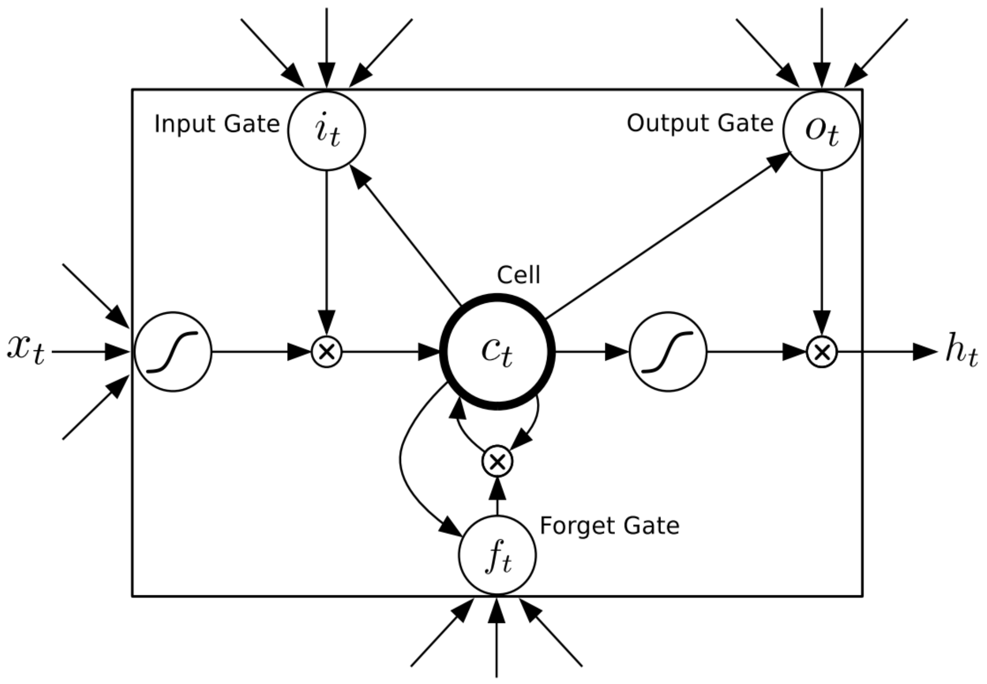


Figure 3.4: Structure of a LSTM unit

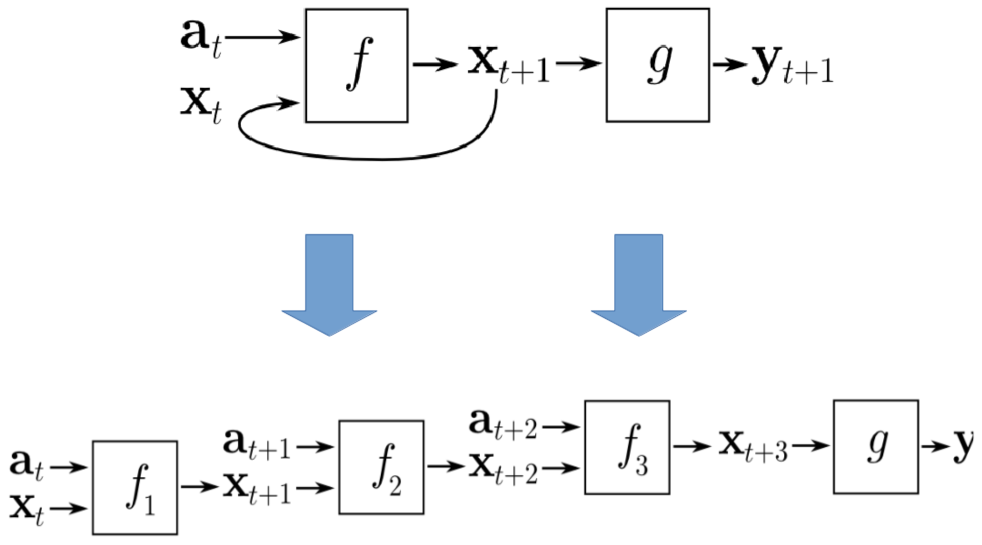
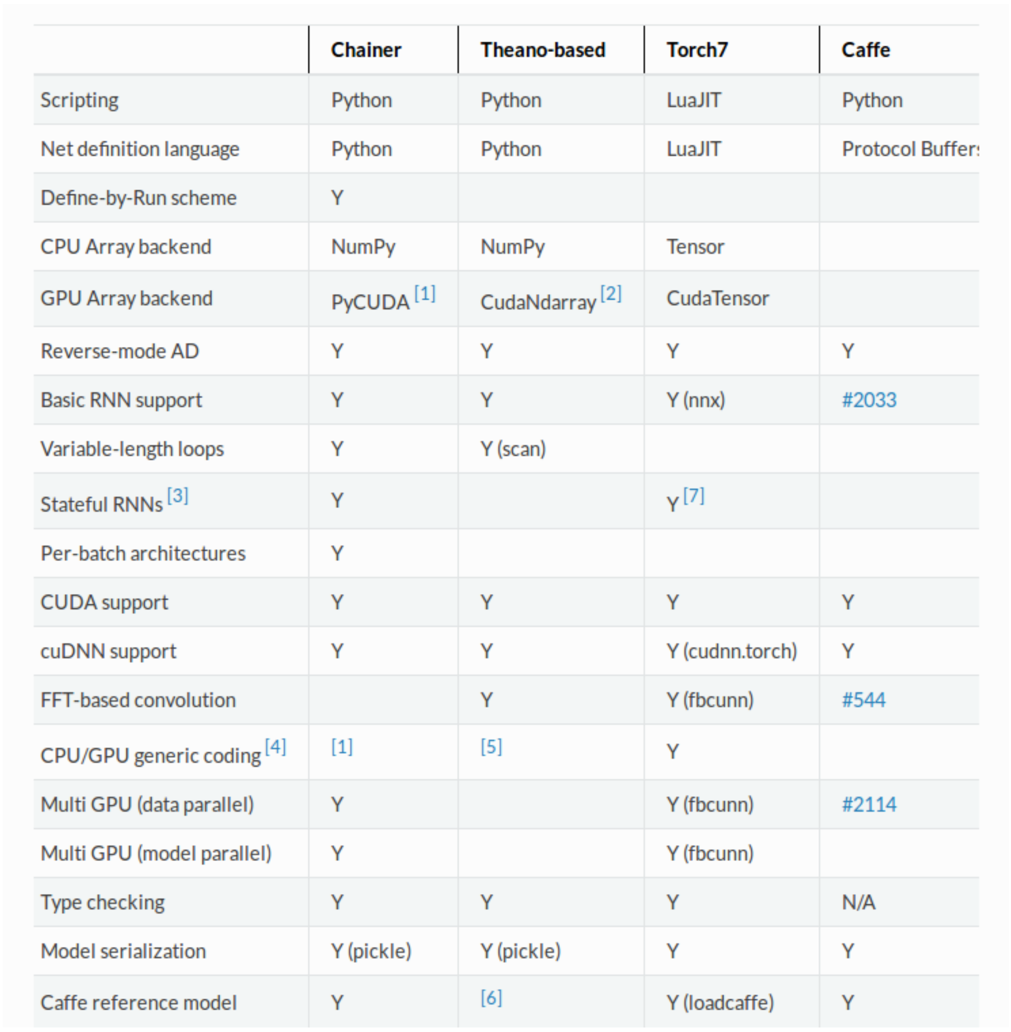


Figure 3.5: Back Propagation Through Time (*BPTT*)

3.6 Deep Learning frameworks

The computational requirements of Deep Learning methods makes a careful optimization of all components and the use of GPUs extremely important. It is therefore important, although not necessary, to rely on specialist libraries that offer low level (usually *C*) implementations of core routines, and simplify the management of *GPUs*. In recent years there have been a few libraries which have been proposed with this aim. The three most commonly employed are *Theano*, *Caffe* and *Torch* (mainly developed by the University of Montreal, the University of Berkeley and by Facebook, respectively). An additional framework that have been recently introduced is *Chainer*. The first release was in June 2015, at the beginning of this project. Compared to the previous, this last framework lacks of support and of an active community, since it has just been introduced, however has several features that make it very interesting.



	Chainer	Theano-based	Torch7	Caffe
Scripting	Python	Python	LuaJIT	Python
Net definition language	Python	Python	LuaJIT	Protocol Buffer
Define-by-Run scheme	Y			
CPU Array backend	NumPy	NumPy	Tensor	
GPU Array backend	PyCUDA [1]	CudaNdarray [2]	CudaTensor	
Reverse-mode AD	Y	Y	Y	Y
Basic RNN support	Y	Y	Y (nnx)	#2033
Variable-length loops	Y	Y (scan)		
Stateful RNNs [3]	Y		Y [7]	
Per-batch architectures	Y			
CUDA support	Y	Y	Y	Y
cuDNN support	Y	Y	Y (cudnn.torch)	Y
FFT-based convolution		Y	Y (fbcunn)	#544
CPU/GPU generic coding [4]	[1]	[5]	Y	
Multi GPU (data parallel)	Y		Y (fbcunn)	#2114
Multi GPU (model parallel)	Y		Y (fbcunn)	
Type checking	Y	Y	Y	N/A
Model serialization	Y (pickle)	Y (pickle)	Y	Y
Caffe reference model	Y	[6]	Y (loadcaffe)	Y

Figure 3.6: Comparison of alternative DL frameworks (from Chainer's documentation)

Table X lists the some relevant features of the different deep learning frameworks. For this project we chose to use the newly introduced *Chainer* framework. From a more high level perspective, the main characteristic of the different models that led us to choose Chainer among the available framework are described in the following:

- The greater advantage of *Theano* is that it is not only a scientific computing library and a deep learning framework, but offers also a run time compiler, that compiles python code to C and ensures excellent performances in terms of speed. The main disadvantages are that is quite low level, has a not very intuitive programming model, and its automatic differentiation system has some limitations: Loops, for example, are dealt with quite poorly, requiring the use of a special function (*scan*), which limits the type of loops that can be used.

- *Torch*, thanks to the choice of the *lua* programming language, has both a simple high level programming model and a very simple programming interfaces for coding parts of your model in C. It doesn't, however, compile the entire model to C as *Theano.* With respect to Theano, it slightly more high level, and offers more pre-built models and algorithms, especially for optimization.

- *Caffe* has even higher level abstractions than *Torch.* Most of the core code is written in highly optimized *C++* code and allows significant saving in terms of memory. It is quite efficient also in terms of speed, but is not so easy to modify the lower level structures.

- Concerning *Chainer*, the most important feature offered by this framework is the choice of a *define by run* scheme. It allows to specify the network by simply writing the python code that computes the forward pass, and has effective way of building from such code a computational graph to manage backward passes and other issues. This makes coding complex model much simpler, as you only need to focus on the logic of the model, and not on how to map such logic onto special programming models (such as *Theano*'s).

All models have pros and cons; however, beside the differences, the common feature is that all frameworks offer a simple way of accessing the *GPU* and using it to speed up computation. Although Theano (especially), and also Torch*,* can offer greater speed when run on the cpu, on the *GPU* the difference is not so crucial as training becomes incredibly efficient in all cases. For this reason, given that we were interested in exploring *GPU* computing anyway, the simpler programming model of Chainer was the crucial reason that led us to choose such technology. Although there are wrappers also around Theano (e.g. Keras) that simplify the programming model of Theano, these are usually limiting if you need to modify the abstractions they are based on. Chainer, instead, offers both the benefits of a truly python based programming model and significant flexibility.

Chapter 4

Experimental Results

In this chapter we present the evaluation of various extractive summarization systems. All are built according a supervised framework. However, the various systems differ in the computed sentence features, the type of regression used to model relevance, the word and sentence representations used to measure redundancy, and how extracted sentences are ordered. Both standard and novel ideas are evaluated.

4.1 Outline and Methodology

In the following sections we present our experiments in automatic summarization. First, we focus on relevance based summarization analyzing what features can help us generating good summaries and how such features can be combined into a relevance score using regression. Second we focus on redundancy analyzing how we can further improve the output of our system by explicitly penalizing redundant information. Finally, we describe some heuristics concerning the ordering of the output of the system. We present the results achieved using algorithms and ideas described in Chapter 2, together with some novel proposals. All steps are carefully evaluated both using the rouge software and by manual inspecting the results of our system. The rouge software was especially relevant in the first part of the project when different ideas could lead to significant improvements in the score. However, not all improvements could be captured effectively by such automatic evaluation: the perplexity feature based on language modeling was, for example, introduced to improve the performance of our system on raw real world data (although it only marginally improved the rouge score - computed on the much cleaner conference data). Also redundancy was a significant issue from a human reader perspective, but not so much for the automatic scoring. A combination of both manual and automatic evaluation was therefore crucial all along the project. In the following section we will clarify the nature and the reason of all our proposals, and the outcome of evaluation.

4.2 Conference Data and Evaluation

In order to evaluate the different features and approaches to summarization we used the data from the *Document Understanding Conferences* (*DUC*). The *DUC* conference ran from *2000* to *2007* and aimed at providing large collections of documents with corresponding human reference summaries to be used for evaluation of summarization systems. In particular we used the set of collections that were provided in *2005* and *2006*. Each year provided *50* collections of documents, with each collection being composed of *20* to *40* documents, related to a given topic (different for each collection), and associated with a *query*. The conference data includes human reference summaries (up to *9* per collection), and also the summaries generated by various systems proposed by researchers all over the world (more than *30* system participated each year). Such data is used in multiple ways for training our system. First of all, according to the methods presented in Chapter 2, each sentence in the collections is scored according to how much it covers the information content of the human reference summaries. This provides us with a data-set that can be used to train regression models capable of scoring the relevance of novel sentences given the computed features. Second the parameters that regulate the trade off between relevance and redundancy in the extraction of a summary are tuned on a set of *validation* collections. Third, the final systems are used to extract summaries of a set of test collections, and the generated summaries are compared to the reference ones using the Rouge software: this provides us a reliable evaluation of the performance of the system on novel collections. It also allows to compare our systems to the efforts of other researchers around the world. For evaluation, we have used a *Java* wrapper around *Rouge*, implementing the metrics described in Chapter *2*, as this significantly simplifies its use. Among the available conferences, we have selected the years *2005* and *2006* because they are among the biggest corpora available, and the former provides more classical *ideal* conference collections, while the latter attempts to insert more real world documents, with meta-data and other non relevant content often appearing in the text. The documents are mainly news articles and their sources are among the most important newspapers world wide (such as *Financial Times* and *New York Times*). The nature of the documents is compatible with training a system suitable for integration in Signal's platform, as the kind of source and the variety of domains are similar. The main difference between the training corpus and the actual data in the real world is that, in average, training data is cleaner because it has gone through more preprocessing (therefore a manual inspection of the results of applying the proposed techniques to Signal's data was always worthy). In Tables *4.1* and *4.2* we provide sample topics of the collections used for training and for evaluation, to show the variety of domains, significantly different in the collections on which the system has been tested with respect to those that we used for training.

Table 4.1: sample of the topics of training collections

Year	**Topic**	**#documents**
2005	American Tobacco Companies Overseas	25
2005	Boundary disputes involving oil	30
2005	New Successful Applications of Robot Technologies	27
2005	The Breakup of Czechoslovakia	28
2005	Journalist risks	25
2005	Electric Automobile Development	28
2005	Foreign minorities in Germany	31
2006	Recent developments and theories regarding Autism	25
2006	Solar energy around the world	25
2006	Development of Magnetic Levitation Rail Systems	25
2006	Drug Research and Development Activities	25
2006	Post Cold War Arms reduction in Europe	25
2006	World Court	25
2006	Hugo Chavez	25

Table 4.2: sample of the topics of test collections

Year	**Topic**	**#documents**
2005	Law enforcement with dogs	43
2005	Threat to wildlife by Poachers	33
2005	Health-related problems caused by working with computers	39
2005	Argentine British relations post Falkland War	28
2005	Education Reform	30
2005	Amazon Rain-forest Problems	25
2005	Prosecution for Tax Evasion	50
2006	Federal budget surplus	25
2006	Steroid use among female athletes	25
2006	Anti-smoking laws	25
2006	ADD/ADHD diagnosis and treatment	25
2006	Impacts of global climate change	25
2006	Native American Reservation System - Pros and Cons	25
2006	Civil unrest in China	25

4.3 First Architecture

The first architecture we will consider focuses on relevance only. Redundancy is not yet considered at this stage. The proposed system is thus sub-optimal. The main aim of the experiments with such architecture is to provide a first simple environment to test several features and algorithms (either found in literature or proposed in the context of this project). Although the system does not consider redundancy at all, thanks to some novel ideas presented in the following sections, still achieves competitive performance with many systems presented at the *DUC* conferences. An improved system including also redundancy is instead presented in Section 4.6.

In figure 4.1 we present the logical flow for this first system. First, the *DUC* conference data-sets, in *XML* format, are parsed and preprocessed. Then the data is divided in two sets containing the training and test collections, respectively (this is done according to a classic *80/20* split). Both the training and the test set include a set of collections, and, for each collection, the overall topic, the specific *query,* and the set of reference summaries written by human evaluators. All documents are segmented into sentences using nltk's *punkt* sentence tokenizer, which uses a unsupervised model trained on a large English corpus to model abbreviations, together with *start* and *end* words for sentences. Subsequently, for each sentence a set of features is computed, some based on the content of the sentence alone, some relying also on the overall topic of the collection and the specific *query* or *keywords* which we are interested in. Each training sentence is also assigned a label computed comparing the content of the sentence to the human reference summaries for the collection. The ground truth generation process is a crucial point for the overall success, and has the main limitation of providing a form of distant supervision (the actual relevance of each sentence is unknown). In all systems labels are generated according to the method presented in Chapter 2. Test sentences remain instead unlabeled.

At this stage, all the previous processing has provided us with all we need to learn a scoring function using standard regression techniques: i.e. a set of *(feature vector, relevance score)* pairs. Once a model for relevance is learned (see Section 4.4 for details on the various regression models we have evaluated), we can use it to score the sentences of the test collections (which have gone through the same processing, except that the relevance score is unknown). In this first architecture, this concludes the summarization process, as the summary is generated by simply extracting the top sentences from the ranked list, until the fixed bound in word or sentences is reached. Evaluation of different features and regression algorithms can then be performed by comparing the output of summarization with the reference summaries for the test collections (using the *Rouge* software, whose metrics have been presented and discussed in Chapter 2 in the *evaluation* section).

.

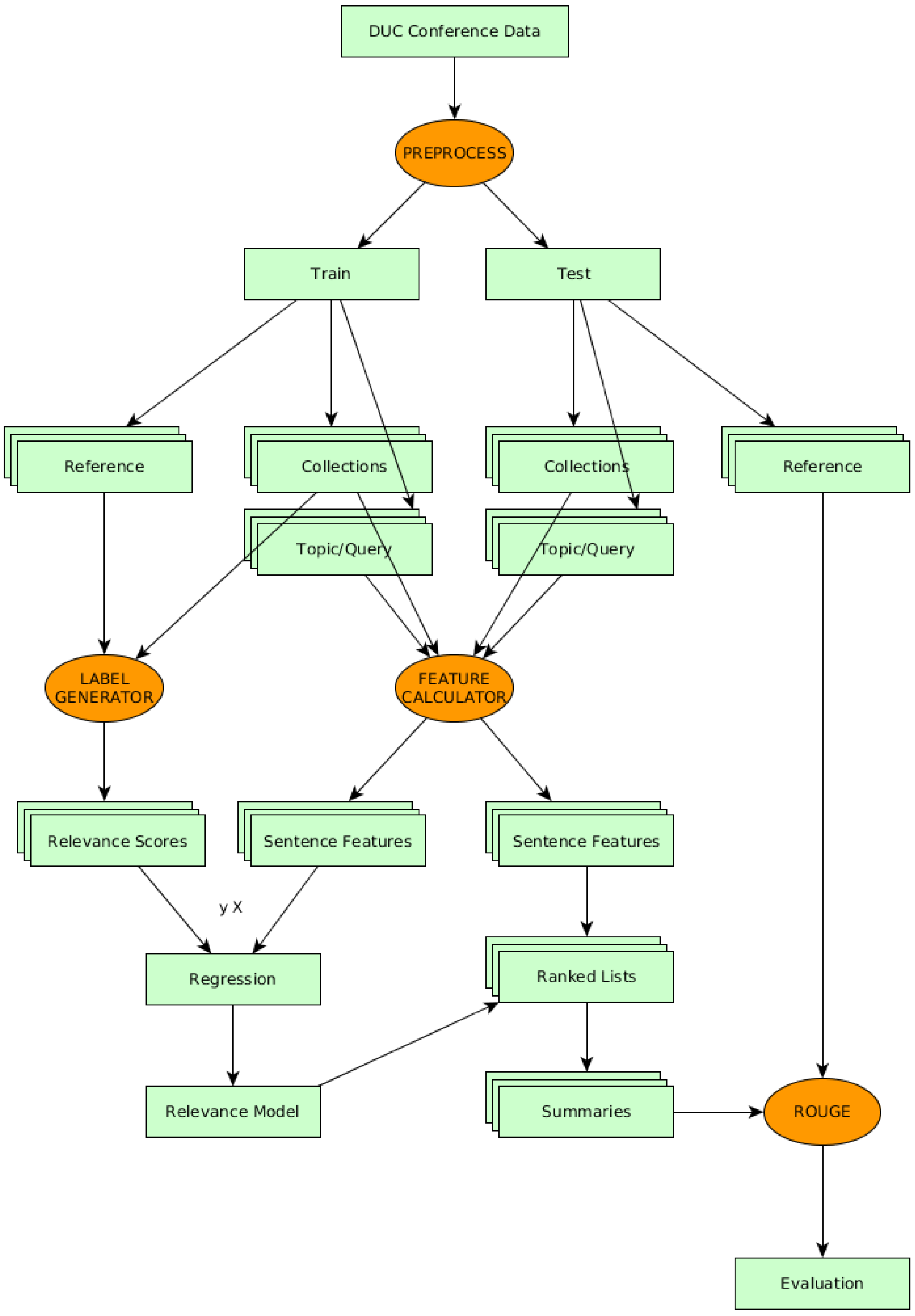


Figure 4.1: Relevance based summarization system's architecture

4.4 Non-linear Regression for Relevance Assessment

Given the pairs of sentence features and labeled score, a function that maps features to relevance can be approximated using any regression algorithm. However, the choice of the algorithm might have significant influence on the quality of the result. For summarization, we need a model capable of dealing with significant noise (we have only a form of distant labeling). It is not instead required to be able to deal with high dimensionality of data. In order to create a simple environment to evaluate if and how non linear regression algorithms can improve the performance of a summarization system, wrt linear models used in literature, we first implemented a few common features, and used the classic approach to ground truth generation presented by Lin et al. (see Section 2.5). The features employed in this experiments are: *P*, *LEN*, *inFirst5*, *Q*SS, *TSS* (see Section 2.5). More complex ways of labeling the training data, and additional novel features will be introduced in the next sections. The regression algorithms we evaluated are the following:

Decision trees **[DT]:** a decision tree for regression [13]is a tree structure predicting continuous values using one of multiple local regression models, whose selection is conditioned on the values assumed by a subset of the features. The partition is built to maximize homogeneity of the training set's splits (i.e. to minimize the standard deviation of each group). A suitable partition can be derived by recursively computing the standard deviation reduction of splitting the groups on every individual variable. Once a suitable partition is built, and the sequential assignment criteria represented as a tree structure, the label of a data point belonging to a specific partition of the feature space can be predicted in two ways: the simplest approach is to predict the average of the training points in the partition; alternatively, the value can be predicted using a local linear model (fit to the subset of the training points that belong to such partition). The hypothesis space is a set of piece-wise constant functions in the former case, or a set of piece-wise linear functions in the latter.

Random forests **[RF]:** a random forest [12] is a model that aims at correcting for the propensity of decision trees to overfit to the data, by training multiple decision trees (using each time only a random subset of the training set and random subspace selection [36]) and predicting the average of the individual prediction. This is an example of *ensembling*, a powerful technique in supervised learning, and more precisely of *bagging.* Its impact on the accuracy of the model can be understood in terms of the bias-variance decomposition of the generalization error. Louppe [48] proposed in *2014* a particularly interesting decomposition for randomized ensembles, such as Random Forests: given a novel data point $\boldsymbol{x}$ and an ensemble of M randomized models $\phi_{\mathcal{L},\theta_m}$, the expected error is expressed by the following equation:

$$E_{\mathcal{L}}[Err(\phi_{\mathcal{L},\theta_1,\ldots,\theta_m}(\mathbf{x}))] = noise(\mathbf{x}) + bias^2(\mathbf{x}) + var(\mathbf{x}) \tag{4.1}$$

$$noise(\mathbf{x}) = Err(\phi_{Bayes}(\mathbf{x})) \tag{4.2}$$

$$bias^2(\mathbf{x}) = (\phi_{Bayes}(\mathbf{x}) - E_{\mathcal{L},\theta}[\phi_{\mathcal{L},\theta}(\mathbf{x})])^2 \tag{4.3}$$

$$var(\mathbf{x}) = \rho(\mathbf{x})\,\sigma^2_{\mathcal{L},\theta}(\mathbf{x}) + \frac{1-\rho(\mathbf{x})}{M}\sigma^2_{\mathcal{L},\theta}(\mathbf{x}) \tag{4.4}$$

$$\rho(\mathbf{x}) = \frac{Var_{\mathcal{L}}[E_{\theta|\mathcal{L}}[\phi_{\mathcal{L},\theta}(\mathbf{x})]]}{Var_{\mathcal{L}}[E_{\theta|\mathcal{L}}[\phi_{\mathcal{L},\theta}(\mathbf{x})]] + E_{\mathcal{L}}[Var_{\theta|\mathcal{L}}[\phi_{\mathcal{L},\theta}(\mathbf{x})]]} \tag{4.5}$$

Where ϕ_{Bayes} is the optimal (*Bayes*) regression model, and the term $\rho(\mathbf{x})$ is the ratio between the variance due to the learning set and the total variance, accounting also for the effects of random perturbations. The decomposition shows that the bias of a random forest is equal to the one of a single *randomized* tree (and thus increases with randomization). Instead, variance is reduced by averaging and, for $M \to \infty$, tends to 0 as randomization gets stronger. Overall, the process can thus reduce the expected generalization error, as long as a suitable trade-off is found. Indeed, with suitable tuning, random forests are known to perform well on a wide range of tasks, and are among the best models also in our experiments on automatic summarization.

***Gradient boosting* [GB]:** Gradient boosting [29] is an alternative ensemble for regression, based, however, on very different principles with respect to random forests. The advantage of this approach is that can perform very good even if the base models are very *weak* learners. The main disadvantage is that it might be more prone to overfitting than bagging. The algorithm works by seeking a function expressed as a weighted sum of a set of base learners:

$$\hat{f} = \sum_{i=1}^{M} \gamma\, h_i(x) + c \tag{4.6}$$

Such function is constructed iteratively, fitting at each step some model to the *pseudo residuals* of the training samples (given the current function approximator $\hat{f}_m$), and computing γ by solving a one dimensional optimization problem.

$$\hat{f}_m(x) = \hat{f}_{m-1}(x) + \gamma_m h_m(x) \tag{4.7}$$

$$\gamma_m = \underset{\gamma}{argmin} \sum_{i=1}^{N} L(y_i,\, \hat{f}_{m-1}(x_i) + \gamma h_m(x_i)) \tag{4.8}$$

The impact of this ensembling procedure, in terms of the bias variance-decomposition, is a reduction of the bias of the model. Various base models can be used in this framework, in our experiments we use again decision trees as we did for random forests.

***Support vector regression* [SVR]:** Kernel support vector regression [18] allows to learn a non linear functions by mapping the training data to a transformed feature space where a linear model is fit. The support vector regression model is closely related to the standard SVM model for classification, so we will here present only a few relevant passages of the derivation. The start point for this is the choice of the loss function: we choose the ϵ-insensitive absolute error as loss function.

$$L(y,\, f(x)) = max(|y - f(x)| - \epsilon,\, 0) \tag{4.9}$$

Neglecting, for the moment, the non linear feature mapping of the training point, we can express the problem of finding the linear function minimizing the previous:

$$min\ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^{\ast}) \tag{4.10}$$

$$subject\,to:\ \left\{ \begin{array}{c} y_i - \langle w,\, x_i\rangle - b \leq \epsilon - \xi_i\ and\ \xi_i \geq 0 \\ \langle w,\, x_i\rangle + b - y_i \leq \epsilon + \xi_i^{\ast}\ and\ \xi_i^{\ast} \geq 0 \end{array} \right\} \tag{4.11}$$

A dual problem can be derived constructing the Lagrangian function, with multipliers α_i, $\alpha_i^{\ast}$, and looking for the saddle point setting partial derivatives to zero:

$$max\ -\frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i - \alpha_i^{\ast})(\alpha_j - \alpha_j^{\ast})\,\langle x_i, x_j\rangle - \sum_{i=1}^{N}[\epsilon(\alpha_i + \alpha_i^{\ast}) + y_i(\alpha_i - \alpha_i^{\ast})] \tag{4.12}$$

$$subject\,to:\ \sum_{i=1}^{N}(\alpha_i - \alpha_i^{\ast}) = 0\ and\ \alpha_i - \alpha_i^{\ast} \in [0,\, C] \tag{4.13}$$

Finally, the a non linear feature map, allowing for non linear regression in the original space, can be introduced by simply substituting $\langle x_i, x_j\rangle$ with the inner product in feature space, computing it with a suitable kernel function [65] for efficiency.

Experimental Results: The aggregated performances (over a 20% test split of the 2005 DUC conference data) using non linear score functions are shown in Figure 4.2. The results prove that non linear regression can indeed help. Figure 4.2 includes also the performance of the very simple standard baseline *Lead* as reference. In Figure 4.3 we show the detail of the performance of the various systems on sample test collections: we can easily see how the linear regression based system score systematically lower than the random forest based system (which provides the optimal results). The other regression algorithms have intermediate results with respect to these. Note that the uplift provided by non linear score functions can be greater it appears here, if more features and thus more complex interactions are introduced.

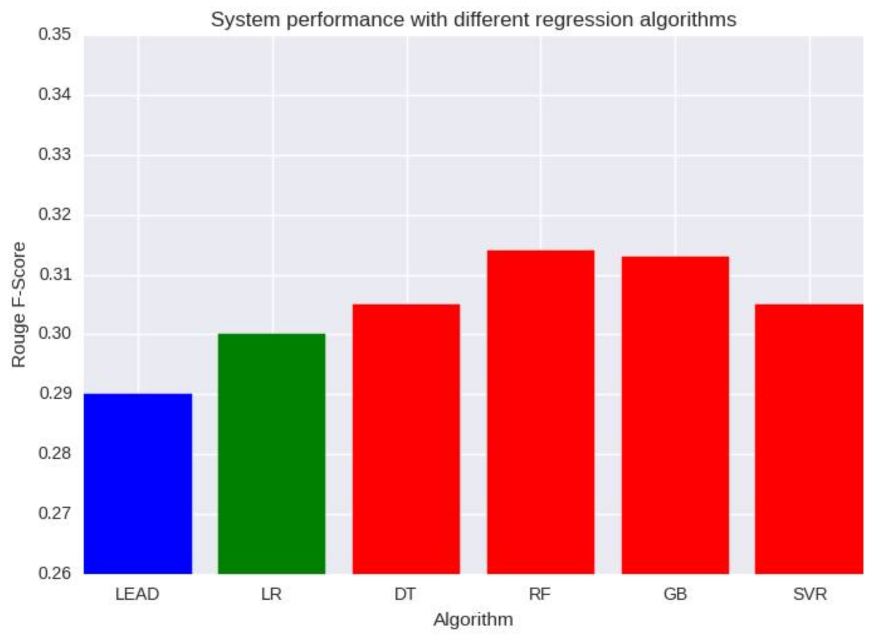


Figure 4.2: Comparison of summarization system's performance, using various regression algorithms, evaluation on a *20%* test split of the *2005* DUC data-set

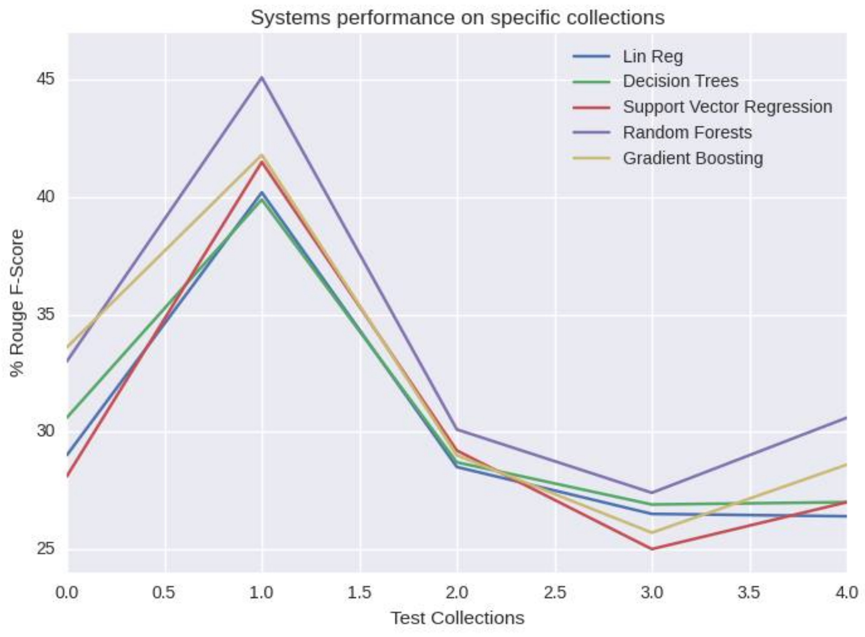


Figure 4.3: Comparison of the supervised summarization system's performance, using various regression algorithms, detail of the accuracy on the first 5 test collections

4.5 Additional Features

In order to improve the performance of our system we introduced some additional features. As we anticipated in Section 4.1, some features aimed at improving the raw rouge score, some were introduced to fix specific issues observed by manual inspection. In the following two sections we present the features and the reasons for their introduction: starting with some features already found in literature, and then analyzing more in depth the *perplexity* feature, computed using a language model (this feature was not previously used, to the best of our knowledge).

4.5.1 Additional Features

1. *Headlines*: First, we introduced the cosine similarity between the n-gram representation of a sentence and the n-gram representation of the headline of the specific document it belongs to. The idea is to try to enforce global (collection level) relevance of the extracted summary by allowing to take into account the local (per document) relevance of the sentences. This lead to summaries covering better the different subtopics of the collection, rather than focusing only on the global query and topic description for the entire collection.

2. *Alternative Vector Space Models*: the cosine similarity between a sentence and the query and between a sentence and the headline of the document it belongs to are, up to now, computed through a bag of words representation. It is known, however, that in many NLP tasks, using *TF-IDF* weighting to encode the vector representation can lead to better performances. In this case, we did not observe such improvement. The rouge score is approximately the same, and manual inspection of results revealed a worsening of the results: an unexpected result as stopping and stemming (already included in results of Section 4.4) had lead to improvements; thus, a systematic way of measuring the discriminative power of a word such as the tf-idf weighting was expected to help. We eventually reverted to the n-gram representation.

3. *POS tags*: we also evaluated the use of features counting the number of occurrences of the various POS tags in a sentence. To do so we used nltk's POS tagger to count the number of *NOUNS*, *VERBS*, *ADJECTIVES* and *ADVERBS*. Of these four features, the two that lead to improve the results were the counts of noun and verbs, and were therefore included. Especially the count of verbs had the positive result of eliminating some pathological sentences such as financial listings, which in some collections contained relevant keywords of the query. The observation that a simple syntactic feature could improve the output under this perspective, lead to the next feature: language modeling, which provides a more complete solution of the issue.

4.5.2 Perplexity Feature and Language Modeling

During our experiments with our automatic summarization system we soon realized that good performance on conference data not always ensured good performance on Signal's data. Real world data is indeed, in general, less polished than the conference data, especially with respect to the *2005* edition. This created serious problems to our system, with meaningless sentences corresponding to tables, caption, or financial data listings finding their way in the final summary. In order to address this issue we decided to use language models as additional feature. A language model is a probability distribution over words and n-grams, modeling their frequency of occurrence in a given corpus or domain. More specifically, we used language models to implement an additional feature with respect to the classical features presented in Chapter *2*: the *perplexity* of the sentence. In formal terms, given a language model specifying a distribution $p(x)$ over words (or n-grams) the *perplexity* of a sequence of words (or ngrams) x_1, ..., x_n is defined by the following expression:

$$c^{-\frac{1}{n}\sum_{i=1}^{n} log_c p(x)} \tag{4.14}$$

where c is a specified constant (usually *2*). In more intuitive terms, perplexity measures the likelihood of a sentence being generated given the language model: if the language model accurately describes the language of a given domain, low values of this metric will correctly identify well formed plausible sentence, while high values of perplexity will correspond to ill-formed sentences which are not to be introduced in the summary. Given a language model, it is easy and fast to compute this potentially very relevant feature, however, training a good language model can be less trivial. There are indeed several issues which must be dealt with, such as how to deal with previously unseen words or n-grams, and many completely different solutions. In this project we have evaluated the benefits to summarization of two alternative approaches to language modeling: a classical n-gram language model with *Kneser-Ney* smoothing, and a neural language model based on a recurrent network with *long short term memory* units.

Statistical Language Models The main issue with statistical language modeling us how to smooth the distribution in order to assign non zero probability also to unseen words and n-grams. Without this smoothing process all sentence where such words or n-grams occur will be assigned infinite perplexity. This is especially problematic if the distribution is over n-grams because the ratio of seen over unseen n-grams decreases with the order n. Various ways of doing such smoothing have been proposed in literature. One of the best performing smoothing methods is *Kneser-Ney*: this is an absolute discounting model that uses lower order models to estimate the probability of unseen n-grams, applying suitable discounting to other probabilities to keep the distribution normalized, as expressed by the following:

$$p_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{max(c(w_{i-n+1}^{i}, w_i) - \delta, 0)}{\sum_{w_i} c(w_{i-n+1}^{i})} + \frac{\delta * c(w_{i-n+1}^{i-1}) * p_{KN}(w_i|w_{i-n+2}^{i-1})}{\sum_{w_i} c(w_{i-n+1}^{i})} \tag{4.15}$$

In order to train on large corpora, which is essential for optimal results, I used an open source *C++* implementation (called *KenLM)* of such n-gram language model. This implementation of the *Kneser-Ney* statistical language model uses multithreading and allowed fast training. I used a n-gram order of 3.

Neural Language Models The fundamental problem with statistical language modeling is the curse of dimensionality: to describe a probability distribution over many discrete variables we need to specify an exponential number of parameters, making learning very difficult. Neural language models can provide better performance because they simultaneously constrain the hypothesis space and make more efficient use of the training data. Concerning the size of hypothesis space, the issue is bypassed by mapping words to vectors in a continuous space where smoothness can be assumed. Concerning the use of training data, mapping semantically similar words to metrically close vectors allows each training sentence to inform about an exponential number of semantically related sentences [77]. Neural language models may be built with convolutional or recurrent networks, however, best results have been achieved using RNNs. Therefore, we have used a recurrent network to train our language model on a *GPU* using *Back Propagation Through T*ime. More details concerning the architecture are given in Section 4.7.2.2, where this model is presented in details and used to generate sentence embeddings for redundancy assessment.

4.5.3 Experimental Results

In Figure 4.4 and 4.5 we show the improvement in the performance of a summarization system using non linear regression to learn a relevance score function, when the previous features are included, evaluating the result on a 20% split of the 2005 and 2006 DUC conferences, respectively. The performance of the system is indeed improved by new features, except for the TFIDF feature, which has little and inconsistent impact. The perplexity feature is instead very important, even more than it appears from the raw score. However, the choice between the neural or the standard n-gram language model has also no influence and is therefore not reported. The reason is likely that the fine grained differences in the perplexity feature are unlikely to count, with only particularly high perplexity mattering (as they signal ill-formed sentences). However, using the RNN language model has the advantage that it may also be used for sentence embedding (as shown in Section 6 and 7).

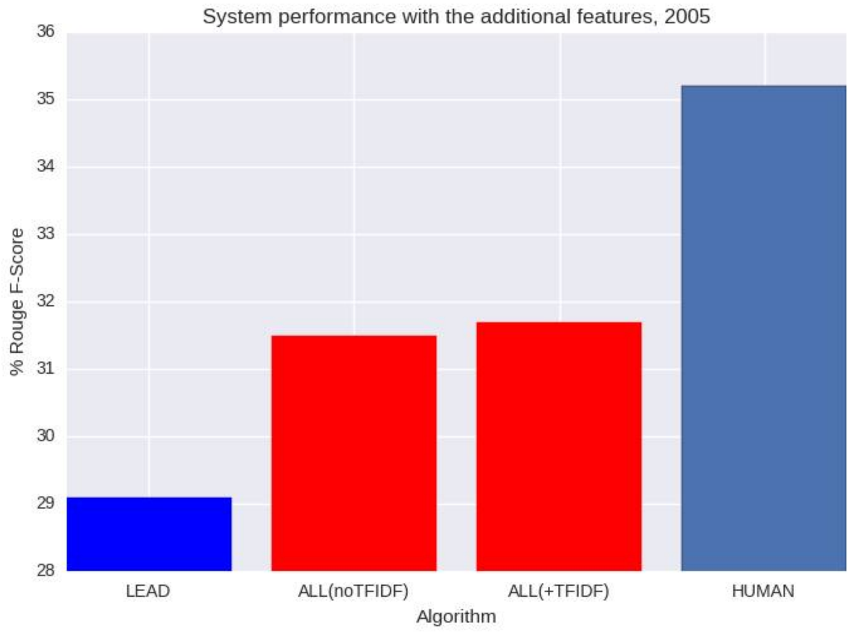


Figure 4.4: Summarization system's performance, using the additional features, evaluation on a *20%* test split of the *2005* DUC data-set

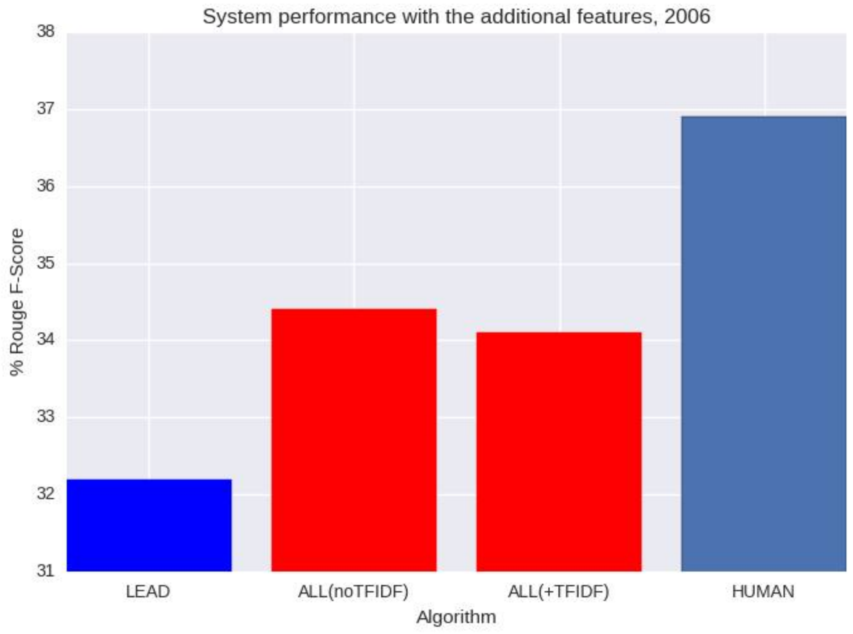


Figure 4.5: Summarization system's performance, using the additional features, evaluation on a *20%* test split of the *2006* DUC data-set

4.6 Analysis of Results and Improved Architecture

The experiments and the ideas developed in the previous sections do allow to tackle effectively the first issue within automatic summarization: i.e. the assessment of the relevance of a sentence with respect to a given query or set of keywords. The rouge scores of the systems built using the previously described techniques are indeed competitive with many of the systems proposed in the *DUC* conferences. Depending on the time and computational resources a company wants to invest, the previous system might be a suitable choice. It will be the first one to be integrated within the Signal's platform. However, there are still some shortcomings in a automatic summarization system which relies only on relevance in order to extract a summary of a multidocument collection. As we anticipated in Chapter 2, the main issue is that there is no control over the introduction of redundant content in the summary. The problem is particularly serious if the different documents of the collection are strongly related, in this case there might be multiple very similar sentence that receive very similar scores and might consume a relevant part of the available word budget. Lets consider, for example, the output of summarizing a collection of documents concerning drug traffic. The following sentences are both extracted by the system:

1. *Cuba's interior minister, the Cabinet officer in charge of domestic law enforcement, was fired Thursday as that nation's drug purge continued, but the crackdown has failed to touch other leaders who U.S. officials say are involved*

2. *Cuba's interior minister, Gen. Jose Abrantes, has been fired for "great deficiency" in this Communist nation's first drug trafficking scandal, the official newspaper Granma said today*

Although the information content of the extracted sentences can be indeed considered relevant for the purposes of our system, there is some clear redundancy, wasting a significant amount of words: any human summarizer would most certainly have selected only one of the two sentences. In order to achieve optimal results in the summarization tasks we need to modify the architecture defined in Section 4.3 to deal with the redundancy issue. The novel pipeline introduces an additional component for extracting the summary, as shown in Figure 4.6. Such component takes as input the ranked list of sentences, according to relevance, and implements the *maximum marginal relevance algorithm* described in Chapter 2. Given the computational requirements of Signal, we have implemented the greedy version of such algorithm, as the *ILP* and *Dynamic Programming* algorithm are not fast enough. This means recursively selecting the sentence that maximizes an objective function that linearly combines the absolute relevance of a sentence with the redundancy between such sentence and the ones extracted at the previous iterations. The trade-off between relevance and redundancy is determined by an additional hyperparameter λ.

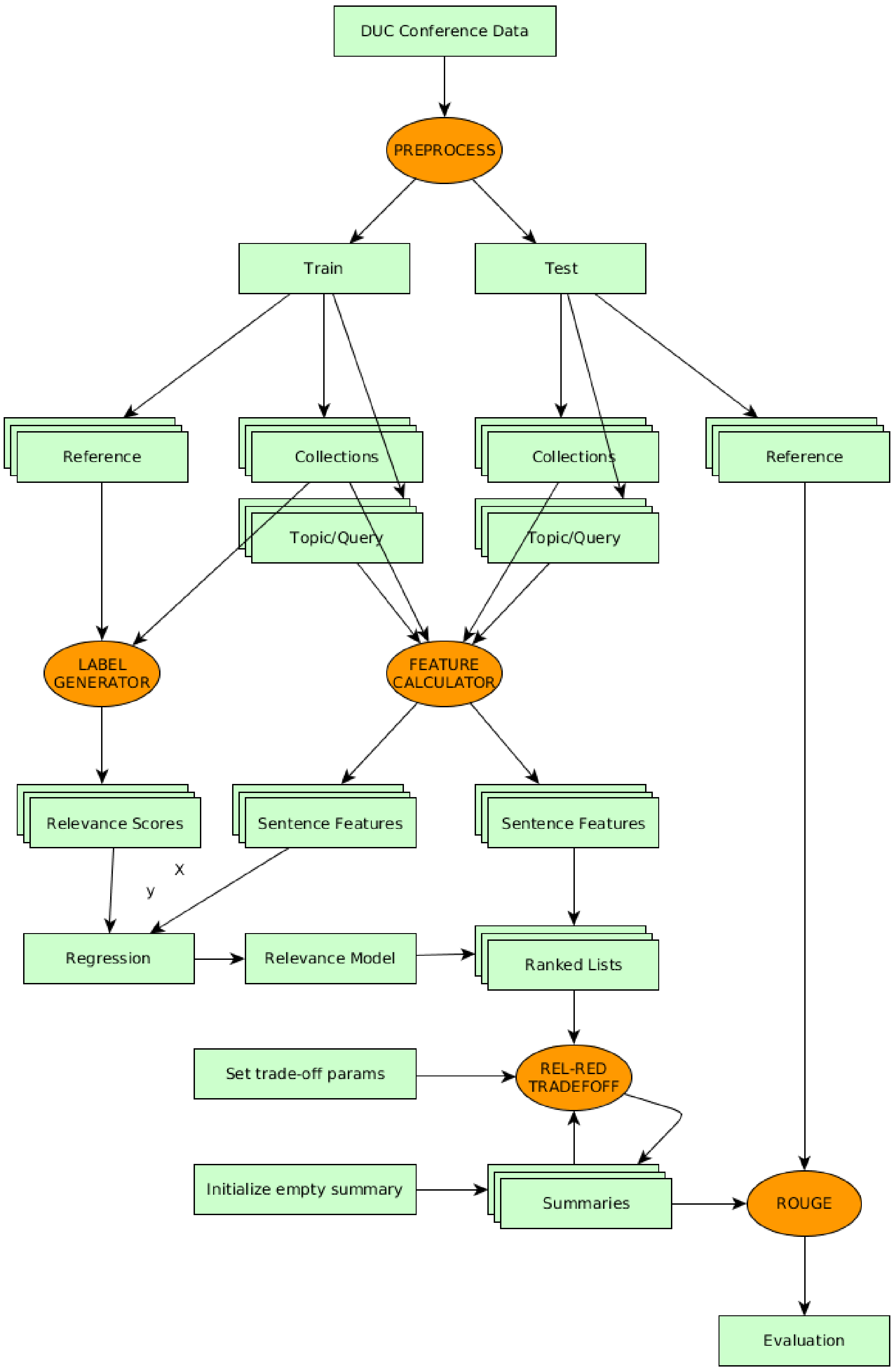


Figure 4.6: Full summarization system's architecture

4.7 Redundancy Assessment

The crucial part of the maximum marginal relevance algorithm (implemented by this improved system) is the redundancy measure assessing how much the information content of two sentence is overlapping. In this section we present three approaches to building such redundancy measure: one is a classic, purely syntactic, n-gram based approach, while the other two are more sophisticated; this second class of metrics relies on neural sentence embeddings (derived using deep convolutional and recurrent networks, respectively) to encode in vectors the actual meaning of the sentence.

4.7.1 Classic n-gram based Redundancy Metrics

The redundancy metric based on a bag of word representation we have implemented is very simple and has been widely employed in literature (see Chapter 2). Such metric measures redundancy through the cosine similarity between the bag of words representations of the two sentences (computed after applying stemming and removing stop words - to avoid non discriminative words dominating the result).

$$Red_2(s_1,\, s_2) = cos(s_1, s_2) = \frac{\langle onehot(s_1),\, onehot(s_2)\rangle}{\|s_1\|\,\|s_2\|} \tag{4.16}$$

The score is based on a uni-gram model. Note that in information retrieval, where redundancy is computed at a document level, it is usually preferred the cosine similarity between higher order n-gram representations of documents; however, when comparing two sentences such representations are too sparse as most pair of sentence may not have even a single n-gram in common if n is set equal to 2 or 3. For this reason, for sentence level redundancy, uni-grams are preferable. The results achieved using this metric within the maximum marginal relevance framework are presented and compared to those achieved using other redundancy metrics (after reviewing the other metrics based on neural models).

4.7.2 Sentence Modeling for Redundancy Assessment

The fundamental limitation with the previous metric is the fact that it is based on a bag of word representation. This implies, as it has been widely discussed in the chapter devoted to deep learning for natural language processing, that it cannot manage effectively synonyms or paraphrases. In order to overcome such limitation we have adopted a different approach to sentence representation, using the internal representation of a deep neural network to encode the meaning of the sentence, as described in Chapter 3. The cosine similarity between such encodings can then provide us the more robust redundancy measure we are looking for.

4.7.2.1 Convolutional Neural Networks

This work presented in this specific subsection (4.7.2.1) has been done in collaboration with Miljan Martic, another MSc student working on his final project in collaboration with Signal. We will here present the *CNN* architecture we used and an algorithm for hyper-parameter search. The results achieved using the corresponding redundancy metric for summarization are reviewed at the end of Section 7.

Architecture The architecture we chose, in order to use convolutional neural networks for sentence modeling, is inspired by Kalchbrenner's recent paper on the topic [44]: we encode each sentence as a fixed size matrix of 48 words, padded with zeros in case of shorter sentences, then we feed such matrix to *2* convolutional layers with uni-dimensional filters and 2 max pooling layers, followed by a dropout, a folding layer (average pooling layer) and finally a fully connected layer that attempts to predict the sentiment of the sentence. With respect to Kalchbrenner's architecture, showed in Chapter 3, we introduced the following differences: First, we implemented the model using the novel framework Chainer, instead of using Matlab. Second, Kalchbrenner uses a very limited vocabulary of less than 20.000 words and trains the word and sentence vectors simultaneously with the rest of the architecture by using the classic Movie Review data-set for sentiment mining (augmented using the parsed sub-sentences labeled by Socher). This approach however, is debatable. If the sentence embeddings are to be used for a general domain, such very limited vocabulary, together with the limited domain, allows for significant overfitting. Given that we want to learn compositional rules but we want a wide vocabulary and as much generalization as possible, we used *word2vec* pretrained vectors, and maintained the word vectors fixed to reduce overfitting the domain. The word2vec vectors we used are 300 dimensional, compared to the 48 dimensional vectors used by Kalchbrenner, and were trained on a 100 billion words Google News corpus: the exposure to wider domain helps to provide better generalization. The price payed is a slightly reduced precision in predicting the sentiment, but it's something we are less interested in, as we are actually focused on sentence modeling. Finally, we used *relu* as activation function instead of *tanh*, we did not implement the dynamic k-max pooling feature used by Kalchbrenner in the max-pooling layer, and we trained the model with the *Adam* optimizer [45] (provided by Chainer) instead of *SGD*.

Hyper-parameter Search The choice of a deep model, combined with high dimensional word embeddings made the training of the model non trivial and time consuming. As a consequence, it was complex to pursue an effective hyperparameter search (which is usually very important to train high quality deep models). In order to optimize the performance on the sentiment mining task, and, consequently, the quality of the sentence embeddings, we developed a novel hyperparameter search

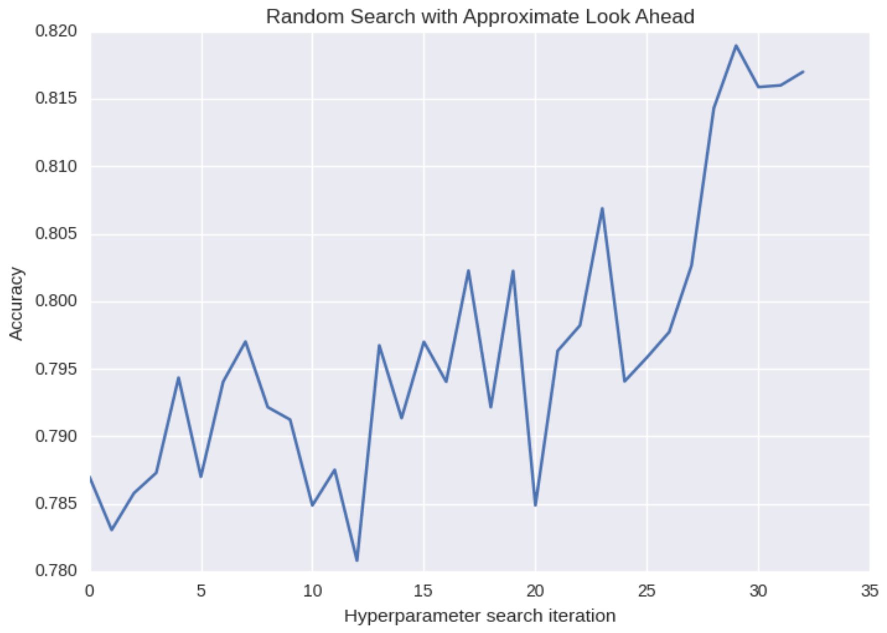


Figure 4.7: Hyperparameter Search

Algorithm 4.1 Random Search with Approximate Look Ahead

```
1: $iterations \leftarrow$ N;$width \leftarrow$ M;
2: $i \leftarrow$ 0;
3: $history \leftarrow$ initialHist()
4: $model \leftarrow$ trainModel(history)
5: $best \leftarrow$ initialGuess()
6: $bestAcc \leftarrow$ evaluate(best)
7: repeat
8:     $curr \leftarrow$ None;$currEst \leftarrow$ 0;
9:     $j \leftarrow$ 0;
10:    repeat
11:        $cand \leftarrow$ generateCandidate())
12:        $lookAhead \leftarrow model$.predict(curr)
13:        if $lookAhead \geq currEst$ then
14:            $currEst \leftarrow lookAhead$
15:            $curr \leftarrow cand$
16:        end if
17:    until $j \leq$ width
18:    $currTrue \leftarrow$ evaluate(curr)
19:    $history$.append((curr,currTrue))
20:    $model \leftarrow$ trainModel(history)
21: until $i \leq$ iterations                    ▷ or other termination criteria
```

method. Standard approaches to hyperparameter search usually rely on *grid search* or on *random search*. Due to the exponential number of hyper-parameters combinations, random search is usually the best choice, as shown by *Bergstra et al.* [9]. However, within randomized forms of hyper-parameter search, very little work has been done in using regression to model the loss function and use such model to guide exploration. Together with Miljan Martic we have thus proposed a form of hyper-parameter search which we called *Random Search with Approximate Look-Ahead*, using random forests to model the loss function while we explore the set of hyperparameter configurations and guide us in such exploration. We have successfully used such algorithm in order to optimize the following architectural hyper-parameters of the network, after specifying for each a range of values we are interested in exploring.

1. The number of *feature maps* at the first and second convolutional layer - in the (10,20) and (20,50) range respectively.

2. The *width* and *height* of the convolution's filter at the first and second layer - in the (1,5) and (4,10) range respectively.

The hyperparameter search is a form of stochastic hill climbing that, however, does not proceed blindly, but builds and uses a model of the loss function, to keep track of the past and try to infer the future. In order to do so, first, we initialize a default hyperparameter configuration according to some initial manual experiments (seeds); then we initialize a random forest regressor, using the seeds and the corresponding training error that could be achieved with each hyper-parameter configuration training for a small number of epochs k; we also assign to each hyperparameter a variance σ proportional to the range of feasible values for that hyperparameter. Then, search is performed according to the following iterative procedure. Each iteration is composed of two steps: first, the algorithm samples n candidate hyperparameter configurations from a Gaussian (with null co-variance and variance σ) centered in the best configuration found up to the current iteration; candidates are rounded to the nearest integer in case of integer hyper-parameters; subsequently, the algorithm uses the regression model to pre-evaluate the n candidate configurations (predicting what accuracy can be achieved by training the neural network with the given set of hyper-parameters - this is the *approximate look ahead*); then, the procedure runs k training epochs with the best set of hyper-parameters among the candidates (best according to our model of the loss function); after each iteration, the model of the loss function, as function of the hyper-parameters, is updated using the seed evaluations and the <*hyperparameters, accuracy*> pairs evaluated up to (and including) the current iteration . Listing 4.1 details the sudo code of the algorithm, while Figure 4.7 shows the improvement in accuracy running the hyperparameter search on the binary class sentiment mining task on the movie review data-set.

4.7.2.2 RNN for Sentence Modeling

Convolutional models can deal very naturally with word order and multiple dimensions of semantics, and, as most feed-forward neural networks, can be trained quite efficiently. Their main limitation is that cannot deal as naturally with variable length inputs as recurrent and recursive models do. This lead us to conduct some experiments with recurrent neural networks to analyze if this could give us better results. Sentence embeddings based on vanilla recurrent neural network are also sensitive to word order as convolutional ones, however they are usually strongly biased towards the last inputs which have been read [53]. This is excellent for pure language modeling tasks, can lead to sub-optimal sentence embeddings that are largely independent from large portions of the sentence. As more extensively argued in Chapter 3, we therefore turned our attention to LSTM recurrent networks. These long-short-term-memory networks can indeed solve the issue of preserving information for a greater number of steps, and, therefore, are likely to provide better representations of the overall meaning of sentences. In our experiments with LSTM recurrent networks for sentence embedding, we used an architecture inspired by the one recently presented by Zaremba et al. [78] for language modeling. For this Section we also relied on some of the tutorial code of the Chainer framework on recurrent networks.

Network Architecture and Regularization The innovation presented in [78] is how to correctly apply *regularization* to LSTM networks. The main tool for neural net regularization is drop-out (as reviewed in Chapter 3), however its use with recurrent networks has been considered problematic for a long time. For example, Bayer et al. [5], suggested that the vanilla application of dropout might hurt training of RNNs because noise is amplified by recursion. Zaremba et al. showed, however, that this does not hold if dropout is applied only to a subset of the connections of the network. More specifically, they suggest that the correct way of applying regularization is to apply dropout only on the *non-recurrent connections*. The reason behind this is that with such approach dropout does force the feed-forward connections to be more robust to noise, but it avoids corrupting the information stored by the stateful components of the network (see Figure 4.8). The architecture we used to generate the sentence embeddings implements this idea as shown in Figure 4.9. The input fed to the network are words, to which are associated word vectors with 250 dimensions. Then we have two structurally identical blocks connected in sequence (note that although structurally the same, their parameters are not tied). Each of these blocks combines an input, received through a dropout, with the values of a hidden layer, and through a LSTM layer updates both the LSTM memory and the hidden layer, which is returned as output. The second block is then topped with a fully connected layer returning a distribution over the vocabulary, specifying how likely it is for each word to occur next in the current sentence.

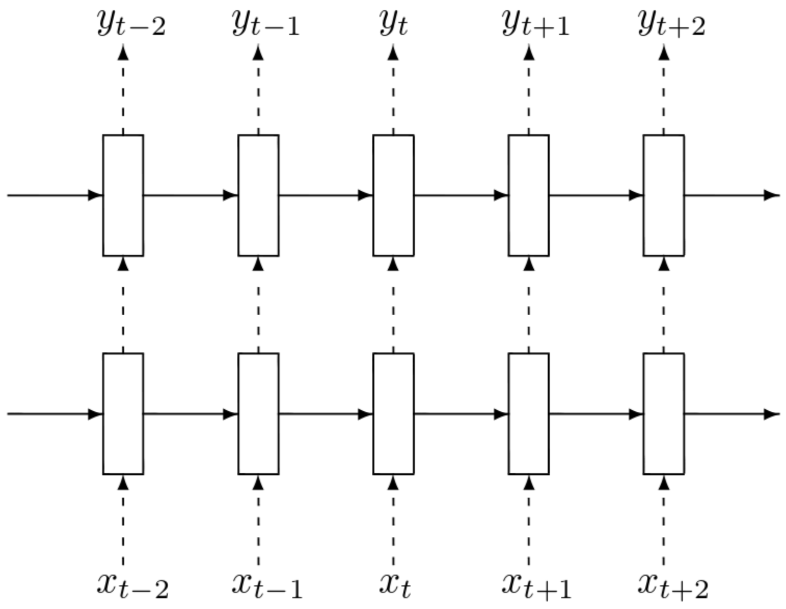


Figure 4.8: Dashed lines identify the connections dropout is applied to, solid lines represent the connections through which information can flow uncorrupted [78]

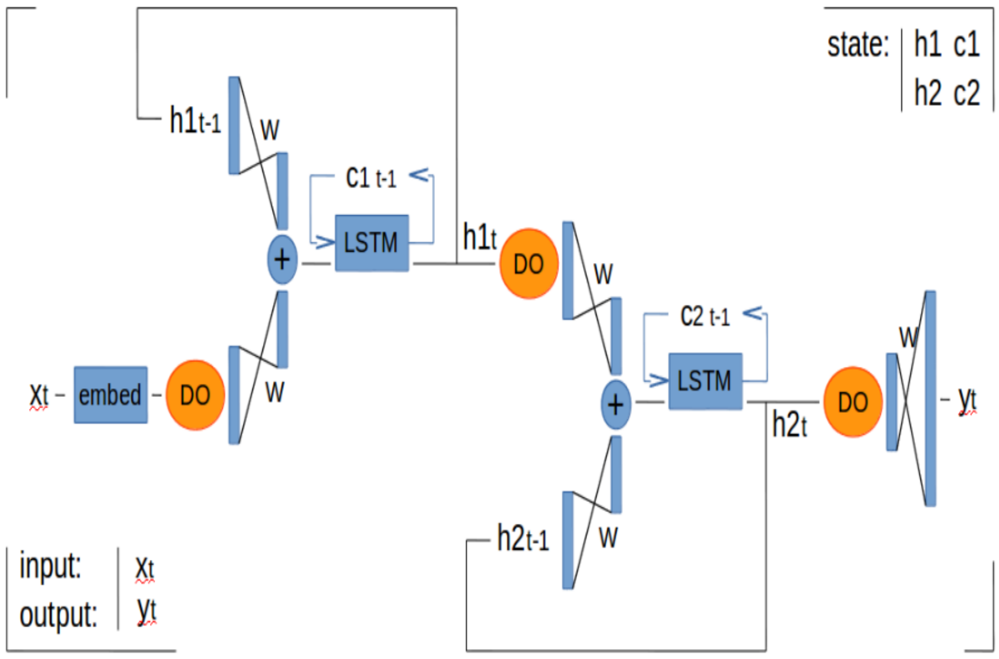


Figure 4.9: Architecture of the recurrent neural network with LSTM units for language modeling, which we employed to generate sentence embeddings

4.7.2.3 Redundancy Assessment:

Once a trained model for predicting the sentiment of a sentence is available, sentence embeddings can be generated at run time by encoding sentences as a matrix using the word2vec model and feeding them to the network. For CNNs the computation up to the folding layer is then executed, and the input of the final fully connected layer is output as representation of the sentence. For RNNs the network is fed the words of the sentence sequentially and the state of the network after reading the entire sentence is output as embedding. In both cases pairwise redundancy of two sentences can then be assessed using cosine similarity between their embeddings:

$$Red(s_1,\, s_2) = cosine(emb(s_1), emb(s_2)) \tag{4.17}$$

The same idea for redundancy assessment was also used with the RNN based sentence embeddings, although the generation of the embedding itself was be obviously different. We could not instead replicate our approach to hyperparameter search, as training RNNs, even using graphic processing units, was considerably slower; this limited significantly the possibility of exploring appropriately the possible configurations, and we were forced to limit ourselves to some manual tuning.

4.7.2.4 Experimental Results

The experimental results show that n-gram redundancy metric, at least if applied naively, does not improve significantly the performance of the summarization system. The rouge score is indeed approximately the same, even slightly lower in one of the conference data-sets. It does at least succeed in avoiding trivial repetitions such as the one exemplified in Section 4.6, when most of the sentence is overlapping word by word. However, it does not achieve any significant result in broadening the topic coverage of the extracted sentences. Concerning this, using neural sentence embeddings to measure redundancy leads to much better results, and this translates also in a significant improvement of the rouge score in both conference test sets. More specifically, embeddings based on convolutional neural network seem to perform especially well, providing the best system among those we have implemented. This might be due to the fact that convolutional embeddings have a more global view of the sentence, while recurrent nets tend to focus on the last words. Convolutional neural networks are also more efficient at training and prediction time. The performance of our system using such convolutional embeddings ranks 3rd and 10th in the two conference respectively. The rank could be further increased if we were to tune the system on the data of each conference separately, however we preferred to focus on generalization across different corpora, given that we aim at developing a system that can be used on novel real world collections within Signal's platform.

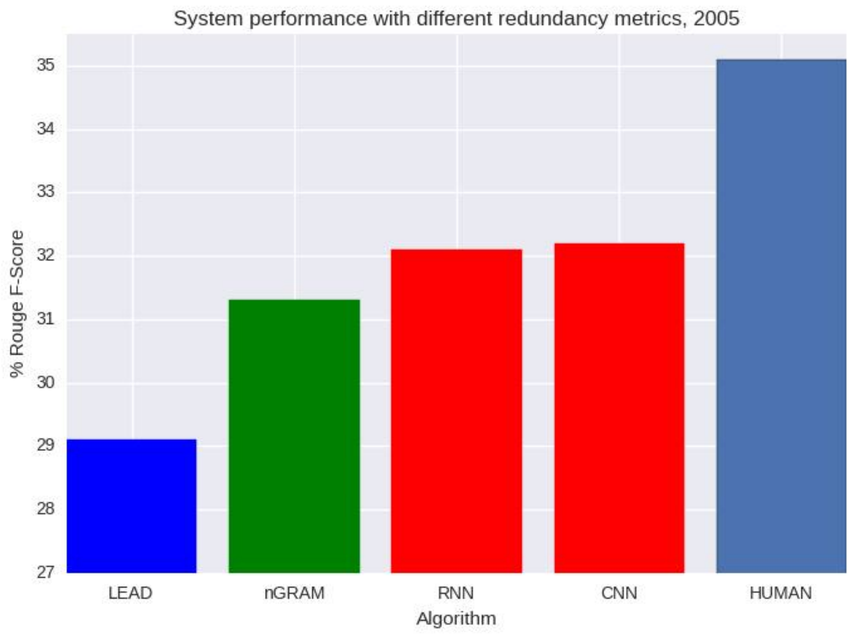


Figure 4.10: Summarization system's performance, using different redundancy assessment metrics, evaluation on a *20%* test split of the *2005* DUC data-set

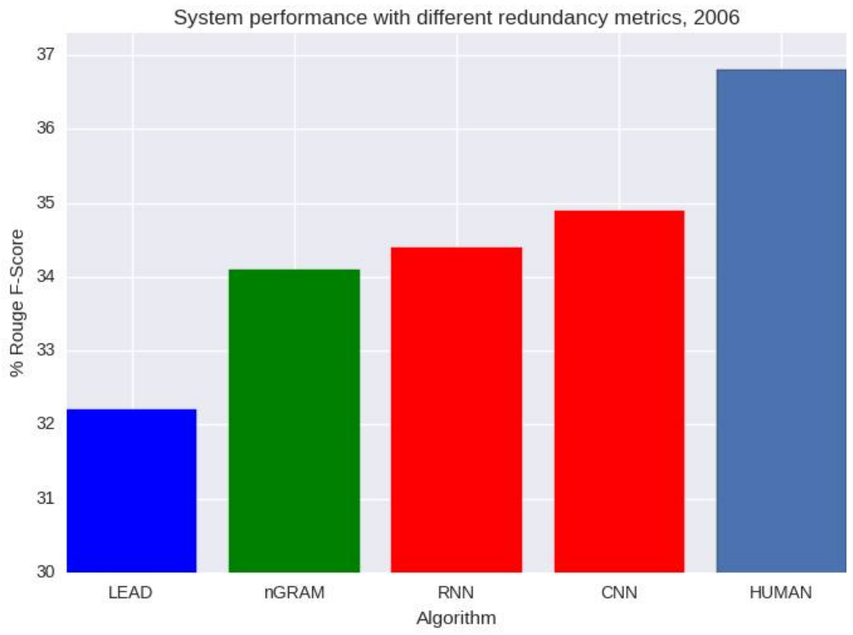


Figure 4.11: Summarization system's performance, using different redundancy assessment metrics, evaluation on a *20%* test split of the *2006* DUC data-set

4.8 Ordering

As anticipated in Chapter *2*, in order to maximize readability and understandability of a summary, the order in which the sentences are presented is potentially relevant. Particularly in an argumentative text the order could be really crucial. In factual summaries, as those we are mostly interested in, the specific order is not as important to understand the content, but a suitable ordering can still improve readability. Barzilay et al. [59] have shown for example in their experiments, that the average score assigned by a set of human judges to a summary may vary significantly (even though the information content is the same). In Chapter 2 we have presented a few possible approaches that are inspired in various ways by this idea, which have been proposed in literature. The common idea, underlying the various methods for arranging sentences in a summary, is that sentences more strictly related and concerning the same sub-topic should be placed nearby, as abrupt changes of topics are not perceived positively by human readers. None of the methods found in literature, however, was fully satisfying, for the reasons explained in Chapter 2. In the context of this project we have therefore implemented two alternative approaches. The main issue in creating algorithms for ordering sentences is that there is no systematic way of evaluating the output of the algorithm. The results presented in this Section are therefore inherently more qualitative, with respect to the rest of the thesis (where quantitative evaluation metrics where available). Careful *observation* of the results allowed us however to identify some recurrent patterns, that highlight the shortcoming and advantages of the two alternative approaches to sentence ordering.

4.8.1 Clustering Based Ordering

The first algorithm we implemented is a simple baseline, that can be thought as a simplified version of the *majority* and the *chronological ordering* algorithms (proposed by Barzilay et al. [57–59] and described in Chapter 2). The algorithm clusters the sentences using the standard k-means algorithm, and then arranges the various clusters using a very rough measure of importance: the size of the cluster itself. The distance among data-points (essential to run k-means) may be computed using any of the redundancy metrics of Section 7. The algorithm has the advantage of being very simple and efficient. With respect to the majority and chronological ordering algorithms, it relies on a sound and well tested approach to clustering, and solves the subsequent issue of ordering the clusters in a more straightforward way. We had considered using chronological ordering instead (following Barzilay et al. [59]) but it was not suitable for the target application domain. Signal aims at generating summaries of news feeds created in real time and whose document publication timestamps may differ of a few minutes only and carry little to none information. Results are commented at the end of Section 8 together with the alternative algorithm.

4.8.2 Spring model

Besides this first baseline, we have also developed a more novel proposal, based on *spring models*. Spring models have been used extensively in a wide range of problems. For example, they have been used for *data visualization*, especially for deciding the layout in 2 or 3 dimensions of graph structures. The most famous algorithm based on spring models for arranging a graph in a plane was proposed by Eades [23] and is outlined in Listing 4.2. Such algorithm computes a combination of repulsive and attractive forces, acting on each of the nodes of the graph, and lets the system naturally evolve through time towards an equilibrium configuration. The attractive forces among the nodes are modeled using variants of Hook's law for springs. The basic form of such law states that the forces exerted on two nodes connected by a spring are proportional to the distance between them and directed towards each other. Such basic approach is however usually modified making the functional dependency upon the distance scale logarithmically:

$$|F_{a \to b}| = k * log(\frac{dist(a,b)}{c}) \tag{4.18}$$

In such formulation the two constants c and k specify the length at rest of the spring and its strength, respectively. It is intuitively clear that, in addition to these forces, also repulsive forces are required, in order for the system not to collapse in a degenerate uni-dimensional point. Such repulsive forces are usually made proportional to inverse of the square root of the distance. Our problem of ordering sentences is inherently one dimensional, but the same equations can be applied also in this setting. Forcing a uni-dimensional arrangement just simplifies the computation of forces. Furthermore, we can use any of the redundancy metrics described in the previous Sections to determine how strongly each pair of sentences is connected, and then we can let the system arrange into a configuration where each sentence is strategically positioned near the sentences which are more strongly related, as intuitively depicted in Figure 4.12, where equal strength springs are assumed.

Algorithm 4.2 Eades's Algorithm for Graph Visualization

```
1: $pos \leftarrow$ AssignAtRandom()
2: $forces \leftarrow zeros(N)$
3: for $iteration in 1,$ M do
4:     for $sent in 1,$ N do
5:         $forces[sent] \leftarrow Elastic(sent) + Repulsive(sent)$
6:     end for
7:     $pos \leftarrow pos + c * forces$
8: end for
```

.

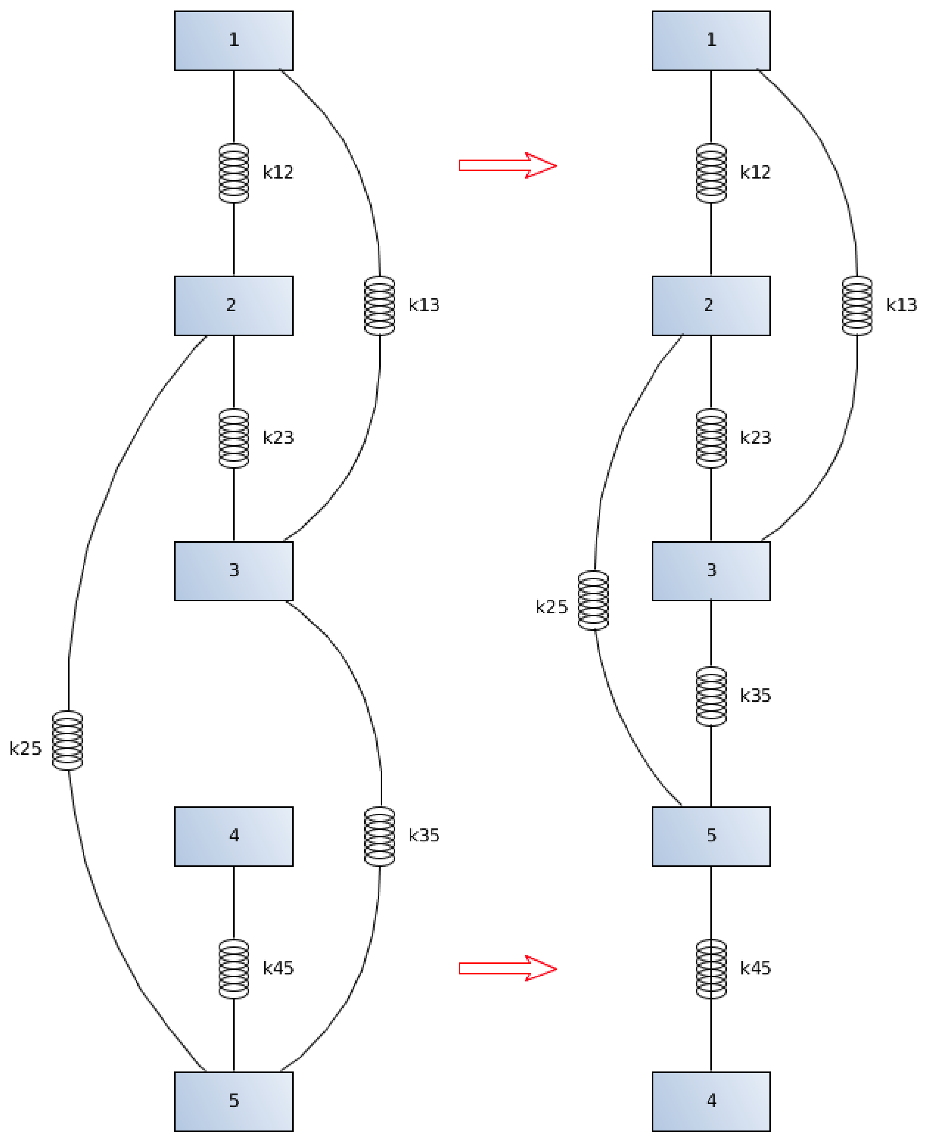


Figure 4.12: Spring model for reordering sentences

4.8.3 Qualitative Evaluation of Ordering Algorithms

Both algorithms have pros and cons. The main common lesson that we have learned by experimenting with both is that it is difficult to evaluate a bad summary, where there is not a clear piece of information to be delivered. All systems for summarization, independently from the underlying principles, will usually be more or less effective depending on the specific collection, and as a consequence the resulting summary may therefore be more or less easy to summarize.

Clustering Besides these general considerations, analyzing the results of the *clustering* based ordering algorithm showed that the typical summary for which the algorithm performs presents clearly defined and cohesive subtopics. If n-gram based redundancy metrics are used to compute distances, stopping and stemming is essential, good performance will also be usually associated to distinctive keywords. For example a summary that was ordered very effectively is the one associated to the collection of documents concerning the ADD/ADHD syndrome. In such summary, there are at least two clearly defined subtopics: diagnosis and treatment. Furthermore, for optimal results, the value k selected when running k-means should be equal to the number of subtopics. In the case of the ADD/ADHD collection choosing more than two clusters will indeed brake partially the clustering, with sentences belonging to either of these subtopics being assigned to a third cluster without any clear criteria. Only under these quite restrictive conditions, the results are reasonably good. Especially the problem of selecting the value k is the biggest limitation.

Spring Model Analysis of the results of the spring model algorithm lead to different considerations. The first observation is that trivial application of the previously defined algorithm, without modifications, leads to even worse results than the previous algorithm. In order to achieve good results it is better to make the spring graph sparse, by eliminating connections between pairs of weakly related sentences. This can be done by setting a threshold below which the elastic constant is set to zero or by retaining the N stronger connections, I chose the former approach. The value of the threshold is a parameter that can only be set by manual tuning (given the lack of a quantitative evaluation metric), however in our experiments was simpler to set than the parameter k of the alternative algorithm, as it is not so dependent on the specific summary to be reordered. A threshold sufficiently high to remove at least half of the connections is most summaries was used in our best experiments with the spring model based ordering algorithm. Overall, the algorithm seems to perform reasonably well on wider range of situations, with respect to the previous one, however when the optimal conditions for the clustering algorithm to work are met the first algorithm can give really impressive results.

Chapter 5

Conclusions

In this final chapter, we summarize what we learned experimenting with automatic summarization. In the following we shortly recap both the results concerning the sentence features used to assess relevance and those concerning sentence representations for redundancy. In the second section of this chapter we also describe the planned follow ups to the project and possible extensions to our research.

5.1 Summary

Overall, from a practical perspective, the results of this project have been very satisfying, as a state of the art system for multidocument summarization system was built. The main lessons that have been learned in the process are the following:

A supervised approach is the best choice when complex trade-offs are to be found among conflicting constraints, as it happens in industry applications. Many complex models and ideas can find a place in this framework. In particular relevance based systems and maximum marginal relevance methods have the flexibility required.

Available training data now allows to combine this approach with sophisticate regression algorithms. Automatic evaluation is a powerful tool for fast prototyping, but it's still essential not to forget manual inspection of results. Testing on real world data is also important (to not overfit to particular features of the conference data).

In order to use deep models, extensive hyperparameter searches proved themselves crucial. This is true both at the structural level (number of layers, feature maps, etc...) and at the training level (learning rate, epochs, etc...). In order to make this computationally feasible, model based approaches can be very useful.

The most important result is that deep learning's abstract representations of high level concepts can indeed benefit also summarization, as it has benefit other fields of natural language processing, but the availability of GPUs for training is essential. Both recurrent networks and convolutional networks can be successfully applied to generate such representations, but convolutional network seem to have an edge.

5.2 Future Work

The first follow up of the project will be rewriting part of the system using the *Clojure* programming language to simplify integration with *Signal's* media monitoring platform. I will personally be involved in the first stages of this, as support to the team. It would also be interesting to apply sentence compression and information fusion techniques to improve the quality of the resulting extractive summaries.

As extension of the present work, it would be interesting to apply the techniques used for multidocument summarization of large collections of documents also to a related task: the *update* challenge. This task requires, while extracting the summary, to avoid introducing redundant information with respect to a second group of documents. We believe that sentence embeddings could play an important role also for this problem. This would interesting also for Signal as they can track the documents read by the user on that platform and use this knowledge to generate summaries presenting exclusively the novel content which the user must be informed about.

Besides summarization the work done on deep learning and sentence embeddings could also find application in other parts of Signal platform. More specifically, it would be interesting to use these representations, at a document level, to unify the encoding used by different predictive models within the platform. Relying on a common representation could make the pipeline faster and models more effective.

Even though both convolutional and recurrent network have been proved capable of generating meaningful embeddings, convolutional networks have provided better results. It would be interesting to analyze if this is due to fundamental reason (convolutional network provide a global view of the sentence while RNNs focus on the latest inputs) or, if better architectures or suitable forms of hyperparameter search could improve also the quality of RNN based sentence representations.

Besides summarization, we believe that the work done on model based hyperparameter search has potential for further development and for application beyond this topic. It would be interesting to evaluate the random search with approximate look ahead algorithm on multiple deep architectures and multiple challenging tasks. Different models for regression could also be evaluated within this context, to improve the effectiveness of the algorithm in guiding the hyperparameter exploration.

Finally, concerning ordering of sentences, introductory experiments have shown that RNNs with LSTM units might be suitable to model long term cross-sentences dependencies. In particular using a simple RNN, and given a human generated summary and a random permutation of it, I was able to predict which was the correct order with an accuracy way above that achievable by chance alone. This suggest that an approach similar to the one based on HMMs and reviewed in Chapter 2 may be developed also using recurrent networks. Time constraints did not allow me to investigate further in this direction, but it may be a fruitful line of research.

Bibliography

[1] Aker A., Cohn T., and Gaizauskas R. Multi-document summarization using a* search and discriminative training. *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 482–491, 2010.

[2] Nenkova A. and Passonneau R. Evaluating content selection in summariation: The pyramid method. *Proceedings of HLT/NAACL 2004*, 2004.

[3] Marco Baroni and Georgiana Dinu. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. *Proceedings of Association for Computational Linguistics (ACL)*, 2014.

[4] Marco Baroni and Roberto Zamparelli. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 2010.

[5] Justin Bayer, Christian Osendorfer, Nutan Chen, Sebastian Urban, and Patrick Van der Smagt. On fast dropout and its applicability to recurrent networks. *ArXiv preprint, Arxiv: 1311.0701*, 2013.

[6] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5:157–166, 1994.

[7] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2009.

[8] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Departement d'informatique et recherche operationnelle - Universite de Montreal*, 1282, 2006.

[9] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[10] M. Bianchini, M. Gori, and M. Maggini. On the problem of local minima in recurrent neural networks. *Neural Networks, IEEE Transactions*, 5:167–177, 1994.

[11] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *The Journal of Machine Learning research*, 3:993–1022, 2003.

[12] Leo Breiman. Random forests: From theory to practice. *Machine Learning*, 45.

[13] Leo Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. Classification and regression trees. *Wadsworth and Brooks-Cole Advanced Books and Software*, 1984.

[14] Hori C., Hori T., and Furui S. Evaluation methods for automatic speech summarization. *Proceedings of Eurospeech 2003*, 2003.

[15] Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 335–336, New York, NY, USA, 1998. ACM.

[16] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. *Proceedings of the 25th International Conference on Machine Learning*, 2008.

[17] C. E. Leiserson Cormen, T. R. and R. L. Rivest. Introduction to algorithms. *The MIT Press*, 1989.

[18] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20, 1995.

[19] Lin C.Y. and Hovy E. Manual and automatic evaluation of summaries. *Proceedings of the Workshop on Automatic Summarization post conference workshop of ACL-02*, 2002.

[20] Lin C.Y. and Hovy E. Automatic evaluation of summaries using n-gram co-occurrence statistics. *Proceedings of HLT-NAACL*, pages 71–78, 2003.

[21] G. Cybenko. Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, 2.

[22] S. Deerwester, S.T. Dumais, G. W. Furnas, T.K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41.

[23] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[24] H. P. Edmundson. New methods in automatic extracting. *J. ACM*, 16(2):264–285, April 1969.

[25] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, and Pascal Vincent. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, pages 625–660, 2010.

[26] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *Journal of Machine Learning Research*, 2009.

[27] Katrin Erk and Sebastian Pado. A structured vector space model for word meaning in context. *Proceedings of EMNLP.*

[28] Günes Erkan and Dragomir R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *J. Artif. Int. Res.*, 22(1):457–479, December 2004.

[29] J.H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

[30] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J Schmidhuber. A novel connectionist system for improved unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Learning*, 31, 2009.

[31] A. Graves, A.R. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. *Acoustics, Speech and Signal Processing (ICASSP), IEEE International Conference*, pages 6645–6649, 2013.

[32] Saggion H., Radev D., Teufel S., and Lam W. Meta-evaluation of summaries in a cross-lingual environment using content-based metrics. *Proceedings of COLING-2002*, 2002.

[33] Harris. Distributional structure. *Word 10*, pages 146–162, 1954.

[34] G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation 18*, 2006.

[35] G.E. Hinton, J.L. McClelland, and D.E. Rumelhart. Distributed representations. *Parallel distributed processing: Explorations in the microstructure of cognition.*, 1, 1986.

[36] Tim Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Learning*, 1998.

[37] S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen (the fundamental deep learning problem). *Diploma thesis, TU Munich*, 1991.

[38] S. Hochreiter and J.. Schmidhuber. Long short-term memory. *Neural Computation*, 8:1735–1780, 2006.

[39] Sutskever I. and Q. Le O., Vinyals. Sequence to sequence learning with neural networks. *NIPS*, 2014.

[40] Pennington J., Socher R., and Manning C.D. Glove: Global vectors for word representation. *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

[41] Miller J.E. and Mccoy K.F. Experimental design to improve topic analysis based summarization. *International Language Generation Conference*, 2014.

[42] Hongyan Jing. Sentence reduction for automatic text summarization. In *Proceedings of the sixth conference on Applied natural language processing conference*, pages 310–315, 2000.

[43] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. *Association for Computational Linguistics*, October 2013.

[44] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, June 2014.

[45] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ArXiv*, 2014.

[46] Chin-Yew. Lin. Rouge: A package for automatic evaluation of summaries. In *Proceedings of the Workshop on Automatic Summarization*, Philadelphia, USA, 2002.

[47] Chin-Yew. Lin. Looking for a few good metrics: Automatic summarization evaluation - how many samples are enough? In *Proceedings of the NTCIR Workshop 4*, Tokyo, Japan, 2004.

[48] Gilles Louppe. Understanding random forests: From theory to practice. *ArXiv*, 2014.

[49] H. P. Luhn. The automatic creation of literature abstracts. *IBM J. Res. Dev.*, 2(2):159–165, 1958.

[50] Andre F. T. Martins and Noah A. Smith. Summarization with a joint model for sentence extraction and compression, 2009.

[51] Andre F. T. Martins and Noah A. Smith. Fast and robust compressive summarization with dual decomposition and multi-task learning, 2013.

[52] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into texts. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 404–411, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[53] Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048, 2010.

[54] Jeff Mitchell and Mirella Lapata. Vector-based models of semantic composition. *Proceedings of ACL*.

[55] Abdel-rahman Mohamed, G.E. Dahl, and G.E. Hinton. Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions*, 2012.

[56] Srivastava Nitish, G.E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, pages 1929–1958, 2014.

[57] Barzilay R. and Lee L. Catching teh drift: Probabilistic content models with application to generation and summarization. *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, 2004.

[58] Barzilay R. and Lapata M. Modeling local coherence: An entity based approach. *Computational Linguistics*, 34, 2008.

[59] Barzilay R., Elhadad N., and McKeown K. Inferring strategies for sentence ordering in multidocument news summarization. *Journal of Artificial Intelligence Research*, 17, 2002.

[60] McDonald R. A study of global inference algorithms in multi-document summarization. *ECIR'07 Proceedings of the 29th European conference on IR research*, 2007.

[61] Li S., Ouyang Y., Wang W., and Sun B. Multi-document summarization using support vector regression. *Proceedings of DUC 2007*, 2007.

[62] G. Salton, A. Wong, and C.S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18:613–620, 1975.

[63] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. *INTERSPEECH*, 2011.

[64] Yohei Seki. Sentence extraction by tf/idf and position weighting from newspaper articles, 2002.

[65] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

[66] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ILSVR 2014 - arXiv:1409.1556*, 2014.

[67] Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. Parsing with compositional vector grammars. In *In Proceedings of the ACL conference*, 2013.

[68] Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. *Advances in Neural Information Processing Systems 24*, 2011.

[69] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

[70] Ilya Sutskever, Oriol Vinyals, and V. Le Quoc. Sequence to sequence learning with neural networks. *ArXiv*, 2014.

[71] K. Chen G. Corrado T., Mikolov and Dean J. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013.

[72] Mikolov T., Sutskever I., Chen K., Corrado G., and Dean J. Distributed representations of words and phrases and their compositionality. *Proceedings of NIPS, 2013*, 2013.

[73] Hatzivassiloglou V., Klavans J.L., Holcombe M.L., Barzilay R., Kan M.Y., and McKeown K. Simfinder: A flexible clustering tool for summarization. *Proceedings of the NAACL Workshop on Automatic Summarization*, pages 41–49, 2001.

[74] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural network using dropconnect. *ICML 2013*, 2013.

[75] Paul Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1:339–356, 1988.

[76] Cohen W.W., Schapire R.E., and Singer Y. Learning to order things. *Journal of Artificial Intelligence Research*, 1998.

[77] P. Vincent Y. Bengio, R. Ducharme. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[78] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *Under review for ICLR 2015*, 2015.

Appendix A

Configuring GPU Instances

In the following appendix, we attach a slightly edited version of a tutorial document we wrote for internal use at Signal on the configuration of a GPU instance on Amazon Web Services and the setup of the Chainer framework for deep learning (together with all the required dependencies). All the required software and tools that are necessary are either open-source or made available to download for free.

A.1 Graphical Processing Units (*GPUs*)

Graphical Processing Units (*GPUs*) are a kind of specialized hardware initially developed in order to focus on graphical processing. This is a intrinsically data-parallel but computationally intensive task, that could easily consume all the available computational power of a processor and slow down significantly any machine. *Data parallelism* is a form of parallelism that emerges when the same set of instructions must be applied to multiple elements (this is often the case in graphics).

The exceptional performance that *GPUs* achieved, driven by the massive investments of the *gaming industry*, made widely available at reasonable prices a form of specialized hardware capable of easily managing the huge computational effort required by modern graphics, but in principle suitable for other applications. It was indeed natural to explore how this computational power could be exploited to solve other computationally intensive and data parallel problems (such as those in the field of *Machine Learning*). This lead to the development of *General Purpose GPUs* (*GPGPU*), which are designed to offer developers a simple interface and a reasonably flexible programming paradigm for developing data parallel applications.

The first proprietary framework for *GPGPU* computing was developed by *NVIDIA* and is known as *CUDA* (*Compute Unified Device Architecture*). In recent years other *GPU* vendors have developed an alternative called *OpenCL* (*Open Computing Language*). The two frameworks are similar, however, most scientific computing libraries still offer support for *CUDA* only, and we therefore chose this model to experiment.

A.2 Amazon Web Services (*AWS*)

Amazon Web Services (*AWS*) is a set of cloud computing services, including virtual machines rent, remote storage, and high performance computing clusters and devices. In particular, Amazon allows to access *NVIDIA GPUs* with up to *1536* cores and *4 GB* of their own *RAM*. Unless you own yourself a modern *GPGPU* with support for *CUDA* (see https://developer.nvidia.com/cuda-gpus for a list of compatible devices), a low cost solution for experimenting with *GPU* computing is thus to use *Amazon Web Services* to access a remote *GPU*. An additional advantage of using a remote server's *GPU* is that the *AWS* environment (accessed by a remote terminal only) does not overload the GPU with useless computations required for rendering the graphical interface as it usually happens if you use your laptop's own graphic card. The price of renting computational resources on *AWS* is very low: for example, a *g2.2xlarge* instance, i.e. a virtual machine with *8* core *CPU*, 1 *NVIDIA GPU* (with *1536* cores), *15 GB RAM*, and *60 GB SSD* costs less than 1 *US*\$/h.

A.3 Create a *GPU* instance on *AWS*

In order to setup a *GPU* instance on *AWS* the first step is to sign up for an *AWS* account on http://aws.amazon.com/. Then we can select the *EC2* service (that allows to rent virtual machines), and create an instance of virtual machine connected to its own *GPU* in a few simple steps. We need however to be careful while instantiating the virtual machine, and select the following configurations:

1. *Choose Image*: When we creating an instance, the first relevant choice is what operating systems to load in the virtual machine. The best choice, if we aim at using *GPU* computing for *Machine Learning*, is to use *Ubuntu*. This, because, most deep learning frameworks and libraries that we can use to exploit *GPU* computing are much easier to install and configure on *Linux*. We therefore select from *Amazon's Community AMIs* the image of an *Ubuntu Server*, version *12.04*, with *HVM* virtualization (we chose *ubuntu/images/hvm/ubuntu-precise-12.04-amd64-server-20140127*). The *HVM* virtualization is necessary to access the *GPU* and the older version *12.04* instead of the more recent *14.04* version is strongly recommended (a kernel component of version *14.04* conflicts with the *CUDA* drivers, and, although it is doable, working around this issue can prove quite problematic).

2. *Choose instance*: to run the previously cited virtual machine with *8* core *CPU*, *1 NVIDIA GPU* (with *1536* cores), *15 GB* of *RAM*, and *60 GB* of *SSD* select the *g2.2xlarge* instance (*g2.8xlarge* if we want to access multiple GPUs simultaneously).

3. *Add Storage*: it is important to note that the available space on the 60 GB SSD is ephemeral. This means that everything saved in such space will be wiped every time the machine instance is stopped or terminated. In order to have a reasonable

amount of persistent storage we need to add here *30 GB* of *EBS* storage (which is independent from *EC2* instance life cycle). The ephemeral nature of the *SSD* disk storage can cause additional issues if you have chosen *Windows* operating systems as several of the available windows installers of the required dependencies install across all available disks, but this should not be an issue if the *Ubuntu* OS has been chosen.

4. *Launch:* when the review and launch button is clicked, you will be asked to generate a key for log in. Store the key locally in a **.pem* file. Then *Amazon* will launch the instance (might take several minutes) and assign a public *IP* to the machine. Once a suitable instance is up we need to install some additional software.

A.4 Setup CUDA

First you need to connect to the remote machine via *SSH*: if you have *Mac* or *Ubuntu* on your own laptop, this can be done easily from the terminal, with the following command (where *key.pem* is the path to the previously generated key for securely accessing the remote machine).

```
$ ssh -i key.pem ubuntu@serverIP
```

Through the remote terminal, some basic libraries for compiling and package management can then be downloaded and installed:

```
$ sudo apt-get update
$ sudo apt-get install python-pip
$ sudo apt-get install python-dev build-essential
$ sudo pip install --upgrade pip
$ sudo apt-get install gcc g++ make
$ sudo apt-get binutils linux-headers-`uname -r`
```

Then the CUDA library's installer itself is downloaded from the NVIDIA website, extracted and run

```
$ wget http://developer.download.nvidia.com/compute/
  cuda/6_5/rel/installers/cuda_6.5.14_linux_64.run
$ rs/cuda_6.5.14_linux_64.run
$ chmod +x cuda_6.5.14_linux_64.run
$ mkdir nvidia_installers
$ ./cuda_6.5.14_linux_64.run
  -extract=`pwd`/nvidia_installers
$ cd nvidia_installers
$ ./NVIDIA-Linux-x86_64-340.29.run
```

After accepting all the default configuration when prompted to choose, and after waiting several minutes for the installer to complete its execution, we can compile the CUDA samples downloaded together with the library:

```
$ cd NVIDIA_CUDA-6.0_Samples/1_Utilities/deviceQuery
$ make
```

Finally we can verify that the CUDA environment is properly setup by running any of the cuda samples automatically downloaded with the CUDA drivers, for example we can run the following command, from which we expect *Result = PASS*

```
$ ./deviceQuery
```

If we get the expected result we can now be confident that the CUDA environment is correctly setup on our AWS instance and we can install the other dependencies which are needed in order to build and train deep learning models using the framework Chainer.

A.5 Install Chainer Dependencies

The first requirements to use the Chainer deep learning framework are Numpy (essential) Scipy and ScikitLearn (useful to run several tutorial examples):

```
$ sudo apt-get install python-numpy python-scipy
```

Then, we need to set several Environmental Variables in order to allow PyCuda and scikit.cuda to find the relevant CUDA drivers

```
$ export CUDA_PATH=/usr/local/cuda/
$ export PATH=/usr/local/cuda/bin/:$PATH
$ export LD_LIBRARY_PATH=/usr/local/cuda/lib/
  :/usr/local/cuda/lib64:/usr/local/cuda/libnsight/
  :/usr/local/cuda/libnvvp/:$LD_LIBRARY_PATH
```

Finally, we can install Chainer itself and the remaining Chainer dependencies (needed to enable GPU computing): PyCuda and scikit.cuda:

```
$ sudo pip install chainer
$ sudo pip install chainer-cuda-deps
```