UNIVERSITÀ
DEGLI STUDI
FIRENZE

PhD Program in Smart Computing
Dipartimento di Ingegneria dell'Informazione (DINFO)

Language Models for Text Understanding and Generation

Andrea Zugarini

Dissertation presented in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Smart Computing

PhD Program in Smart Computing
University of Florence, University of Pisa, University of Siena

Language Models for Text Understanding and Generation

Andrea Zugarini

Advisor:

Prof. Marco Maggini

Head of the PhD Program:

Prof. Paolo Frasconi

Evaluation Committee:
Prof. Vincent Guigue, *UPMC - LIP6*
Prof. Massimo Piccardi, *University of Technology Sydney* (*UTS*)

XXXIII ciclo — December 2021

Acknowledgments

First of all, I would like to thank my advisor Marco Maggini for guiding and supporting me during this three years of Ph.D. Besides my supervisor, I would like to thank the rest of my Supervisory Committee: Marco Lippi and Marco Turchi, whose supervision during the entire program gave me precious insights and suggestions from different perspectives. I also would like to thank the Evaluation Committee, Vincent Guigue and Massimo Piccardi, for reviewing this thesis. A special mention goes to Stefano Melacci whose guidance has been of invaluable help to me since before I was a Ph.D student.

My sincere thanks go to Kata Naszadi, that mentored and supervised me during the internship period at Amazon, and to Manuel Giollo, my manager, that despite the distance (due to covid) made me feel closer.

I would like to thank all the administrative stuff of both the Florence and Siena universities. In particular, my thank goes to Simona Altamura, always kind and efficient. Without her, I would probably be still stuck in bureaucracy. My sincere thanks go to QuestIT s.r.l. (Siena) for funding the grant supporting my studies.

A special thank goes to all the members of SAILab. My experience in SAILab started since before it was called that way (Dario knows it well!). I have seen the lab growing and I met a lot of great friends and colleagues that made this journey amazing. I will always remember with affection all the time we spent together, dinners, soccer games, wine tastings and all the rest. It was really a pleasure. In particular, thanks to Michelangelo for his amazing cooking skills, his dinners were the best. I thank Vincenzo for not giving to me "il posto del morto" (the death's spot) and for his profound appreciation of automatically generated poetry. Thanks to Lisa, a certainty in the lab, she had always a "good word" for anybody. I thank Gabriele, proactive member of the lab, for his insightful questions, despite he stole my desk as soon as he got the chance! Thanks to Enrico, great guy I had the pleasure to work with during this last year. A special thank goes to Alberto, one of the first guys I met in Siena, a friend, and also a great beekeeper! I thank Francesco for the many beautiful movies he made me watch, as well as the many less beautiful ones, and for the endless discussions about them. I thank Giuseppe, code mate by choice, roommate by accident (literally), for the many scientific conversations had while eating half kilogram hamburgers. Thanks to Dario, he was always there in the funniest moments I can think of. Never do an hackathon without him, it's bad luck. I thank Matteo, desk mate, mayor behind the scenes, we went through the Ph.D together (summer schools included), but most of all, for being a fantastic friend!

It was a wonderful period of my life, but none of this would have been possible without the support of my family. Thanks to my parents for believing in me, to my little brother Francesco for reminding me the importance of playing sometimes, and

above all, thanks to Martina, my blonder half, for always being by my side.

Abstract

The ability to understand and generate language is one of the most fascinating and peculiar aspects of humankind. We can discuss with other individuals about facts, events, stories or the most abstract aspects of our existences, only because of the power and the expressiveness of language. Natural Language Processing (NLP) studies the intriguing properties of languages, the rules, their evolution, its connections with semantics, knowledge and generation, and tries to harness its features into automatic processes. Language is built upon a collection of symbols, and meaning comes from their composition. Such symbolic nature limited for many years the development of Machine Learning solutions for NLP. Many models relied on rule-based methods, and Machine Learning models where based on handcrafted task-specific features. In the last decade there have been incredible advances in the study of language, thanks to the combination of Deep Learning models with Transfer Learning techniques. Deep models can learn from huge amounts of data, and they are proven to be particularly effective in learning feature representations automatically from high-dimensional input spaces. Transfer Learning techniques aim at reducing the need for data by reusing representations learned from related tasks. In the scope of NLP, it is possible thanks to Language Modeling. Language Modeling related tasks are essential in the construction of general purpose representations from unlabelled large textual corpora, allowing the shift from symbolic to sub-symbolic representations of language.

In this thesis, motivated by the need of moving steps toward unified NLP agents capable of understanding and generating language in a human-like fashion, we face different NLP challenges, proposing solutions based on Language-Modeling. In summary, we develop a character-aware neural language model to learn general purpose word and context representations, and use the encoder to face several language understanding problems, included an agent for the extraction of entities and relations from an online stream of text. We then focus on Language Generation, addressing two different problems: Paraphrasing and Poem Generation, where in one the generation is tied with information in input, whereas in the other the production of text requires creativity. In addition, we also present how language models can offer aid in the analysis of language varieties. We propose a new perplexity-based indicator to measure distances between different diachronic or dialectical languages.

Contents

Chapter 1

Introduction

1.1 Motivation

Language is the main form of human communication that allows the exchange of information between individuals. It is known that many living species have some sort of verbal communication. Arguably, this strong communication abilities have been paramount for the development of our intelligence and dramatically increased the social skills of human beings. We distinguish from other species for the unique capability of creating (and believing in) fictive stories, which is considered to be a fundamental milestone of the Cognitive Revolution of humankind (Harari (2014)).

To us, understanding, interpretation, elaboration and generation of language seem natural. We acquire such capacities since the moment we are born, and gradually develop them during all our life. There are two key aspects of natural language: understanding and generation. Understanding is the ability to comprehend the meaning of text, i.e. the information received by another individual. Generation is the capability to convey information by producing your own message. In humans those properties are interleaved. This complex phenomenon has been studied for decades. Many have wondered and tried to wire human language into automatic programs. Natural Language Processing (NLP) is a discipline in between computer science and linguistics, aimed at studying and analyzing language by means of automatic processes. To tackle the many aspects of language, researchers have break down its complexity, dividing the problem into many single simpler problems. Some applications require both understanding and generation, as for example, in conversational agents.

The symbolic nature of language has been an obstacle to the development of Machine Learning techniques, that are essentially sub-symbolic, for many years. Most of the solutions indeed were based on rules conceived for each specific task at hand. Machine Learning or Statistical techniques were used only where enough supervised data was available and they required the design of handcrafted features

constituting the input of such models.

In the last decade there have been astonishing advances in Machine Learning. Arguably, most of the success has been achieved thanks to the rise of Deep Learning techniques. Computer Vision, Speech Recognition and Natural Language Processing too, have all seen major improvements and nowadays neural architectures dominate the state-of-the-art in these fields. Deep models can learn from huge amounts of data, and they are proven to be particularly effective in learning feature representations automatically from high-dimensional input spaces. However, these models have a lot of parameters (in the order of millions or even billions), so they are extremely data-hungry, which means that large amounts of annotated examples are necessary in order to learn effectively. Labelling examples is in general expensive and sometimes non-trivial, since it requires human intervention, limiting the power of these models in many scenarios.

Deep Learning has benefit a lot from Transfer Learning. Transfer Learning is based on the idea of reusing acquired knowledge to solve new problems. In its most general form, we refer to Transfer Learning anytime what has been learned in one setting is exploited to improve generalization in another setting. In the classical Machine Learning setup, each task is attached independently. However, there may be many problems that are related to each other. This means that at least part of the representations (features) that a model has to develop for one task are important for another task as well. Hence, if we treat related problems independently we have to re-discover the same features each time, from less data, which seems rather inefficient. Making a parallel to us humans, we do not start from scratch anytime we learn a new skill. The knowledge we acquired from previous problems is already there, so it would be a tremendous waste of energy to "re-train" ourselves without exploiting what we already know. Our strong generalization and adaptation abilities are the essence of our intelligence. For example, learning to ride a motorbike is easier for people who know how to ride bicycles. Learning to speak a new language is simpler when you already speak a similar one. In general, the more related the tasks, the easier is to learn. In the context of Natural Language Processing, problems like Sentiment Analysis or Topic Classification, both require the development of some sort of language understanding features. Differently, in Transfer Learning the acquired knowledge is shared/reused to address related problems. Sharing feature representations across multiple tasks reduces the data burden, since part of the effort was already done to acquire the previous knowledge. The simplest way to reuse previous knowledge from related task(s) is to pre-train a model on this task, then adapt its weights to solve the new task. This technique is particularly powerful when the previous task has a lot of data and it is generic enough to allow the development of general purpose representations. We highlight the differences between standard Machine Learning and Transfer Learning in Figure 1.1.

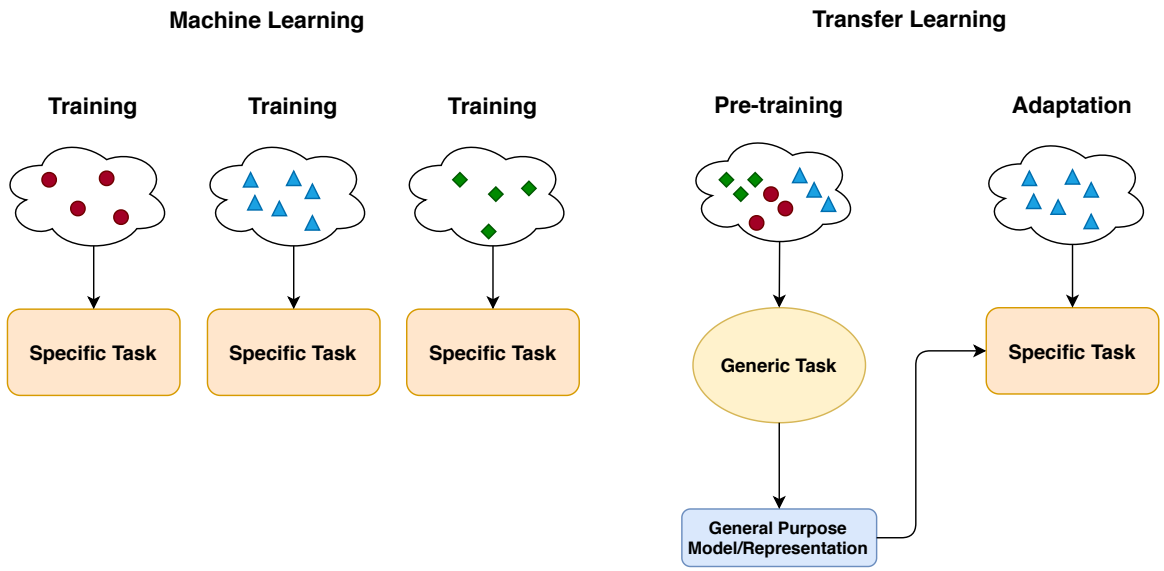

Figure 1.1: Illustration of classical Machine Learning vs Transfer Learning. In Machine Learning each task is treated separately, whereas in transfer learning all the data available is exploited to learn shared input representations, so that task-specific models only have to adapt their parameters from an already informative representation, instead of learning everything from scratch.

Natural Language Processing has seen dramatic improvements, that no long time ago, many would have thought to be impossible to achieve. The game changer in Natural Language Processing was the introduction of methods capable of learning robust, general purpose, representations of text from large self-supervised textual corpora. The ability of transferring such representations to any task brought tremendous progresses in both Language Understanding and Generation. All these techniques are based on the principle that the meaning of a word is correlated with the context in which it appears. This principle is the key of learning tasks that do not require annotated data, and they are all intimately related with the problem of Language Modeling.

Focus of this thesis. Motivated by the need of moving steps toward unified NLP agents capable of understanding and generating language in a human-like fashion, in this thesis, we investigate specific instances of NLP problems ranging from understanding to generation, proposing Language-Modeling-based solutions. We benefit from Language Modeling related tasks to train a character-aware neural model obtaining general purpose representations of words and contexts from large unsupervised corpora. These embeddings are exploited to address two well known Language Understanding problems: Chunking and Word Sense Disambiguation. On top of the same character-aware encoder, we build a novel agent for extract-

ing information from text streams. The system is furnished with memory components (initially empty) that are dynamically updated while reading one sentence at a time. The model learns online from sparse supervisions and self-supervised mechanisms. We then studied two language generation problems in low-resource conditions, Poem Generation and Paraphrasing, proposing different approaches to train those language models and generate text from them. In addition, we show how language models can be help in the analysis of language varieties, were we introduced a new perplexity-based indicator to measure distances between different diachronic languages.

1.2 Contributions

The major contributions of the thesis are summarized as follows:

- Proposal of a character-based contextual encoder to learn representations for words and contexts. The model consists in a hierarchy of two distinct Bidirectional Long Short Term Memory (Bi–LSTMs) networks (Schuster and Paliwal (1997)), to encode words as sequences of characters and word–level contextual representations, respectively. The unsupervised learning approach yields general purpose embeddings with features that turn out to be important for different NLP problems. *Based on Marra et al. (2018).*

- Proposal of an online-learning agent for extracting entities and relations that populates a not-given-in-advance Knowledge Base. In particular, (1) we introduce a new scheme to learn latent representations of entities and relations directly from data, either autonomously (self-learning) or using limited supervisions. Character-based encoders are exploited to handle small morphological variations (plurals, suffixes, ...) and typos, synonymy, semantic similarity. (2) We present a new problem setting where the system is evaluated while reading a stream of sentences organized into short stories, requiring online learning capabilities. (3) We showcase the validity of our approach both in an existing dataset made of Wikipedia articles and a new dataset that we introduce and make publicly available. *Based on Maggini et al. (2019).*

- Introduction of a syllable-based neural language model to generate poems with the style of a given author. To cope with the lack of data, we design a multi-stage transfer learning approach that exploits non-poetic works of the same author, and also other publicly available huge corpora of modern language data to learn robust representations of the target language. Furthermore, we devise an automatic selection mechanism, to rank generated poems according to measures that are designed around the poet style. *Based on Zugarini et al. (2019).*

- Proposal of a method for building a dataset of aligned sentences that can be used to train sequence-to-sequence models for paraphrasing. In particular, we apply it to the case of the Italian language, generating the dataset by crawling pairs of contents from Italian newspapers and blogs, and exploiting them to train a neural paraphrasing sequence-to-sequence architecture based on Pointer Networks (See et al. (2017)). *Based on Globo et al. (2019).*

- Presentation of *Vulgaris*, a project that studied a text corpus consisting of vulgar Italian language literary resources. We first provide an in-depth analysis through a corpus-driven study in dialectology and diachronic varieties. Then we consider perplexity-based distances to analyse the historical evolution process of the Italian varieties, introducing the use of neural language models for the estimation of perplexity and a novel indicator, that we named Perplexity-based Language Ratio (PLR). *Based on Zugarini et al. (2020)*

1.3 Structure of the Thesis

The rest of this thesis is organized as follows:

- Chapter 2 introduces all the fundamental concepts and terminologies that will be necessary in the rest of the dissertation. Particular attention is given to Language Modeling, Recurrent Neural Networks, that will be widely used in the following chapters, Input Representations and Natural Language Generation.

- Chapter 3 discusses the character-based model to learn contextual representations of words by exploiting an unsupervised learning approach. After collocating the work in the literature, a description of the architecture and the learning algorithm is given. An experimental analysis to evaluate the effectiveness of the learned representations in Chunking and Word Sense Disambiguation shows that the approach is competitive with other related methods. Finally, we provide a final discussion of the obtained results.

- Chapter 4 presents a model for extracting entities and relations from text streams by updating online a simple knowledge base. We outline the related works, the problem setting, the proposed agent architecture and its online learning dynamics. Then, we discuss the results on both novel synthetic data and benchmarks available in the literature.

- Chapter 5 introduces two Natural Language Generation models on different domains: Poem Generation and Paraphrasing. The models learn through data, and are trained as language models. In both problems we operate in a low-resource environment. In Poem generation the lack of resources is inherently

dependent on the kind of task, that involves ancient manuscripts that are rare. So we introduce a syllable-based language model trained in a multi-stage transfer learning fashion to acquire knowledge from large corpora of modern languages. Concerning paraphrasing, the lack of resources occurs especially in non-English languages, and can be addressed by retrieving paraphrase pairs automatically. Hence, we propose an algorithm to create a corpus with aligned pairs of sentences. Then, we train a Pointer Generator model on such data to prove the effectiveness of the retrieved pairs and, henceforth, the validity of our algorithm.

- Chapter 6 describes *Vulgaris*, a project to study the evolution of Italian Language Varieties in the middle age. We provide a statistical analysis of the data and show how language models can be exploited to provide insights about the evolution of diachronic varieties.

- Chapter 7 draws the final remarks of the work presented in this thesis. Furthermore, we illustrate possible future directions to extend and unify the presented approaches.

Chapter 2

Background

2.1 Language Modeling

In this Section, we define what is Language Modeling, how it is evaluated and traditional techniques to estimate the models. For the sake of simplicity, in the following discussion we describe language modeling by considering text as a sequence of words, however the narration is general and holds for any kind of token (e.g. characters, syllables etc...).

2.1.1 Definition

Language modeling is the problem of estimating the probability distribution of text. Given a sequence of n words, the goal of a language model is to determine the probability of the joint event $(w_1, w_2, \ldots, w_n)$, i.e.:

$$p(w_1, w_2, \ldots, w_n), \tag{2.1}$$

where, each w_i is a random variable. If we apply the the chain rule, Equation 2.1 is equivalently rewritten as a factorization of conditional probabilities:

$$p(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} p(w_i | w_{i-1}, \ldots, w_1), \tag{2.2}$$

where $p(w_i|w_{i-1}, \ldots, w_1)$ is the probability of word w_i to appear after all the previous words $w_{i-1}, \ldots, w_1$, that are also referred as (left) context. Since Equation 2.2 is more tractable, usually statistical models aim at estimating the conditional probabilities $p(w_i|w_{i-1}, \ldots, w_1)$ or approximations of them.

2.1.2 Evaluation

Language Models are usually evaluated in terms of perplexity. In information theory, perplexity is directly related to entropy, being defined as the exponential (base

two) of the entropy of a given distribution for a certain random variable. Therefore, perplexity is a measure of uncertainty, where lower values indicate more certainty, vice versa, when the entropy increases (and consequently perplexity grows too), the model decisions are more chaotic.

To evaluate a language model, perplexity is estimated at word level. The distribution learnt by the model is compared with a reference test sample. Let be $\boldsymbol{w}$ a sequence of words $\boldsymbol{w} := (w_1, \ldots, w_n)$, e.g. the words in a sentence or an entire corpus, and $p(w_i|w_{i-1}, \ldots, w_1)$ the learnt distribution, the perplexity pp of p in $\boldsymbol{w}$ is defined as:

$$pp(p, \boldsymbol{w}) := 2^{\frac{1}{n}\sum_{i=1}^{n} \log(p(w_i|w_{i-1},\ldots,w_1))}. \tag{2.3}$$

2.1.3 N-grams

The estimation of all the conditional probabilities in Equation 2.2 can be learnt from the observations available in a given, arbitrarily large corpus. Indeed, the probability of token w_i given the context $w_{i-1}, \ldots, w_1$ can be obtained by counting the number of times w_i appears after such context, normalized by the number of times the context $w_{i-1}, \ldots, w_1$ appears in total:

$$p(w_i|w_{i-1}, \ldots, w_1) = \frac{\#(w_1, \ldots, w_{i-1}, w_i)}{\sum_{w_j \in V} \#(w_1, \ldots, w_{i-1}, w_j)}, \tag{2.4}$$

where $\#(\cdot)$ is the counting function of an input observation. Unfortunately, storing and using all the possible sub-sequences in language is infeasible in terms of memory and computation as soon as the context length starts growing. N–gram models overcome this problem by approximating Equation 2.2 with a conditional probability on a truncated, fixed-size context of $N-1$ tokens. Hence, Equation 2.2 becomes:

$$p(w_1, w_2, \ldots, w_n) \simeq \prod_{i=1}^{n} p(w_i|w_{i-1}, \ldots, w_{i-N+1}). \tag{2.5}$$

The bigger N is, the more the approximation is reliable. In the simplest case, when $N = 1$, the probability estimation is not conditioned at all, and Equation 2.1 becomes a product of prior probabilities, which is equivalent to assume that each word occurrence is an independent event. Intuitively, a longer context window N requires the need for more observations. The problem is that the size grows exponentially with respect to N. If V is the token set referred to as vocabulary, containing all the words appearing in the corpus, the number of possible sequences of length N is equal to $|V|^N$. If we also notice that usually vocabularies have cardinality in the order of 10^5, it is immediately clear that statistics will quickly fail.

2.1.4 Neural Language Models

To overcome the generalization issues of N-grams, feed-forward neural networks can be adapted to learn the conditional probabilities in Equation 2.5:

$$p(w_1, w_2, \ldots, w_n) \simeq \prod_{i=1}^{n} p_\theta(w_i | w_{i-1}, \ldots, w_{i-N+1}). \tag{2.6}$$

where p_θ is the distribution estimated by a neural networks parameterized by its weights θ. The model is a multi-layer perceptron (MLP) taking as input the last $N-1$ words and optimized to maximize the probability of the next word in the whole sequence $(w_1, \ldots, w_n)$:

$$\max_\theta \prod_{i=1}^{n} p_\theta(w_i | w_{i-1}, \ldots, w_{i-N+1}). \tag{2.7}$$

Words can be represented in multiple ways, as we will describe shortly. For now we consider them as one-hot vectors.

Neural Networks introduce several advantages w.r.t. to N–gram models. They can generalize to unseen sequences and handle missing or unknown data. Furthermore, the model size is independent from the corpus size. The first Neural Language Model was proposed in Bengio et al. (2003). Despite their good properties, these models still share with N–grams the same limitation underneath the assumption formulated in Equation 2.5: the requirement of fixed size inputs, thus they cannot estimate directly Equation 2.2.

2.1.5 Recurrent Neural Language Models

Before discussing about Recurrent Neural Language Models, let us provide a brief introduction to Recurrent Neural Networks.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are one of the most successful Machine Learning models in NLP. They are a natural extension of multi-layer perceptrons (MLPs) designed to deal with sequences as inputs. An MLP learns a non-linear function $f : \mathcal{X} \subset \mathbb{R}^d \rightarrow \mathcal{Y}$ mapping fixed length multi-dimensional inputs into the output space. MLPs are memory-less, i.e. when processing an input, the output does not depend on the inputs seen in the past. However, there are lots of problems where the decision is inherently conditioned on past information. Time series, dynamical systems, language are all use cases where data is inherently sequential. Instead of mapping fixed length inputs, RNNs process sequences of variable length. These models naturally embed the temporal dynamics by storing past information into a state h_t.

Depending on the task, RNNs can either be trained for sequence classification or for mapping the input sequence to an output sequence (I/O transduction). Let us consider the latter case. Given an input sequence $\boldsymbol{x} := (x_1, \ldots, x_n)$ and a target sequence $\boldsymbol{y} := (y_1, \ldots, y_n)$, an RNN is a non-linear transformation of $\boldsymbol{x}$ into $\boldsymbol{y}$. In its simplest form, an RNN is composed by two functions, f and g. At each time step t, we have:

$$h_t = f(x_t, h_{t-1}) \tag{2.8}$$

$$y_t = g(h_t). \tag{2.9}$$

The temporal dynamics takes place in the function f, sometimes called as memory cell, where the state h_t is updated combining state information computed at the previous time step (h_{t-1}) and the current input x_t. All the relevant knowledge about the past is summarized into the state h_t thanks to the recursion of Equation 2.8. The function g is instead just a non-linear projection, mapping the hidden state to the output space. In the simplest case f is implemented by a single layer:

$$h_t = f(x_t, h_{t-1}) = \sigma(W \cdot x_t + U \cdot h_{t-1}),$$

with W and U being two weight matrices. There exists many other variants of RNNs, but we detail only Long-Short Term Memory Networks, that will be extensively used in the rest of this thesis.

Long-Short Term Memory Networks. Long short–term memory (LSTM) networks were introduced in Hochreiter and Schmidhuber (1997) and are nowadays the most popular architecture of recurrent neural network. The popularity of LSTM is mainly due to its ability of better capturing long–term dependencies (Bengio et al. (1994)), i.e. information coming from old past elements of a sequence. In problems like language modeling, sequences can be very long, and also the information present at the beginning of the sequence may be important to determine the next token. The structure of the function f in LSTMs is composed of four non-linear layers. Some of these layers act as gates, that ideally should regulate the flow of information in such a way that only relevant knowledge passes through. Gating is implemented combining squashing functions and point-wise matrix operations. The state h_t of an LSTM at time t is the result of the following equations:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{2.10}$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{2.11}$$

$$\hat{c}_t = \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right) \tag{2.12}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t \tag{2.13}$$

$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right) \tag{2.14}$$

$$h_t = o_t \odot \tanh(c_t) \tag{2.15}$$

where $\odot$ denotes point-wise multiplication, $[\cdot, \cdot]$ the vector concatenation and σ,tanh are respectively the sigmoid and hyperbolic tangent functions. Instead of having a single internal hidden state that stores past information, LSTMs have two states: h_{t-1} and c_{t-1} that is called *cell state*. At each time step t, c_t combines past and current inputs after being filtered by two gates: *forget* (see Equation 2.10) and *input* gates (see Equation 2.11), respectively. The last gate, in Equation 2.15, is used to compute the hidden state h_t taking into account c_t. We can summarize all the operations to compute the hidden state h_t as follows:

$$h_t = \texttt{LSTM}(x_t, h_{t-1}), \tag{2.16}$$

and use it in the rest of the dissertation.

Language Modeling with RNNs

Language Modeling can be seen as an I/O transduction problem. Recurrent Neural Networks naturally avoid the N–gram assumption, allowing the direct modelling of Equation 2.2. Text is a sequence of tokens (words) and, at each time step t, the goal of the model is to predict a word given the current one and the previous history encoded in h_{t-1}, which is equivalent to learning the probability distribution $p_\theta(w_i|w_{i-1}, \ldots, w_1)$.Thus, the model is optimized to maximize:

$$\prod_{i=1}^{n} p_\theta(w_i|w_{i-1}, \ldots, w_1),$$

that is the actual joint probability of language $p(w_1, \ldots, w_n)$. An example of recurrent language model training is shown in Figure 2.1.

2.2 Language Representations

Differently from other kinds of information, such as speech signals or images, language is purely symbolic, whereas Machine Learning models, neural networks included, are designed for dealing with sub-symbolic inputs. Roughly speaking, regardless of the NLP problem that we want to tackle, it is necessary to transform text into numbers. In this Section we discuss how to represent language in order to feed it to Machine Learning algorithms.

2.2.1 Tokenization

Tokenization is the process of converting a string into a sequence of tokens. This is the first aspect to address when processing text. Probably, the most natural way to look at language, at least for us humans, is to consider it as a sequence of words.

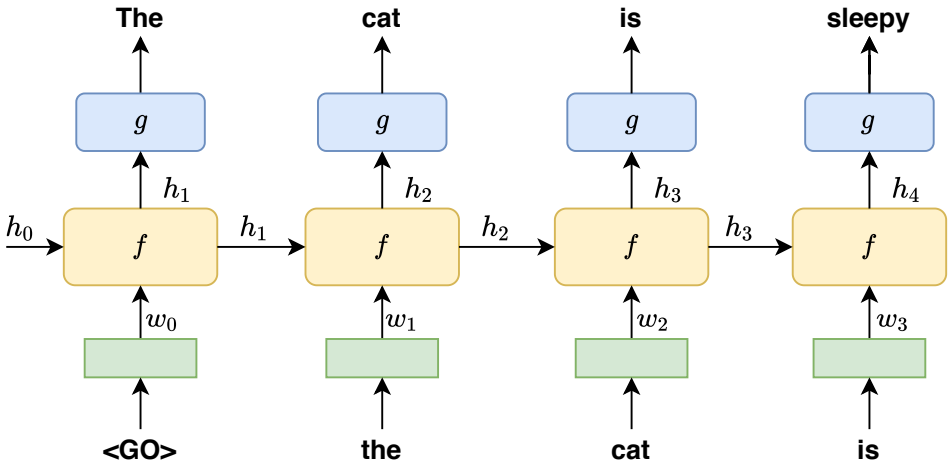

Figure 2.1: Example of a Recurrent Neural Network trained for language modeling. At each time step, the RNN predicts the next word given the current one and the past information carried out by h_{t-1}.

Indeed, words are the tokens that actually convey the semantics in the text. Their ordering and composition are what eventually determine the meaning of a textual span. By large, word-based tokenization has been dominating the scene. However, text can also be interpreted as a sequence of characters or syllables or any kind of sub-word tokenization. Recently, byte pair encoding, WordPiece and SentencePiece methods (Sennrich et al. (2015); Wu et al. (2016); Kudo and Richardson (2018)) have gained a lot of popularity.

The choice of tokenization influences all the next steps of an NLP pipeline. In some scenarios one tokenizer may be better suited than the other, depending on language characteristics, the task at hand, etc. Separating a string into words will create a large set of possible tokens, typically in the order of tens of thousands that can rise up to few millions. Moreover, morphological information is lost, albeit there are tasks or applications in which morphology plays a crucial role.

Character-level tokenization preserves without loss of information the content of a string and will produce much fewer possible elements, at the cost of making the tokenized sequences longer and breaking the word-level semantics into multiple tokens, which may be harder to be captured by a learning model.

Syllables, Byte Pair Encoding, WordPiece, SentencePiece tokenizers are somewhat a trade-off between character-level and word-level splits. Syllables are not a popular choice, because in general they require hyphenation tools, that are language dependent and can be hard to design. However, in problems like poetry or some languages, it can be the natural option. Byte Pair Encoding, WordPiece and SentencePiece are similar algorithms that, given a desired vocabulary size, split a string into a set of tokens according to their frequencies. Resulting symbols will be

a combination of entire words, single characters and word portions. We illustrate in Figure 2.2 an example of different tokenization strategies.

`The cat sleeps`
Words: `[The, cat, sleeps]`
Characters: `[T, h, e, <s>, c, a, t, <s>, s, l, e, e, p, s, <s>]`
WordPiece: `[The, cat, sl, ##e, ##eps]`

Figure 2.2: Example of different kinds of tokenization of the same sentence. In WordPiece, the symbol ## indicates that the token should be attached with the previous one.

2.2.2 One-hot Encoding

The problem of representing a number of discrete symbols goes beyond NLP. It is common to have scenarios in ML where some features are inherently symbolic, or categorical. Gender, blood type, citizenship, the color of a flower are all examples of possible features that can assume only a discrete number of values.

Let ϕ be a categorical feature and V the set of its possible values. We define an assignment function $H : V \to \mathbb{R}^d$ to map each symbol into a real vector. Discrete symbols must be uniquely assigned to a certain point in $\mathbb{R}^d$, otherwise two different elements would end up in the same representation, making them indistinguishable for the learning model. Clearly, it is possible to map any discrete set of values in $\mathbb{R}^d$ for any $d \geq 1$. However, in spaces with $d \leq |V|$, it is impossible to guarantee that all the elements are equally distant to each other. This may bias the model and introduce spurious, unintended similarities between uncorrelated values. Thus, without any other assumption, the best way is to assign symbols to one-hot vectors, such that all the nominal values are orthogonal. As obvious consequence, the size of feature representation d grows linearly w.r.t. the cardinality of V.

For example, let us create a one-hot encoding of the blood type feature. The set of possible values is $V := \{A, B, AB, O\}$, $d = |V| = 4$ and so a one-hot encoding of the values is:

$$\begin{aligned} H(A) &= [1,0,0,0] \\ H(B) &= [0,1,0,0] \\ H(AB) &= [0,0,1,0] \\ H(O) &= [0,0,0,1] \end{aligned}$$

We can see H as a hash function indexing the rows of a matrix E of size $R^{|V|\times|V|}$. In language the set of possible values V it is called vocabulary or dictionary and it

corresponds to the list of all the possible tokens. If we consider as tokens the words, the cardinality of V can be very large, in the order of tens of thousands up to few millions, depending on the application. This make one-hot encoding not scalable for complex NLP problems.

2.2.3 Word Embeddings

One-hot encoding is not well suited when features have a high number of discrete symbols. They produce one-hot sparse representation of symbols in very large spaces making the job of learning models difficult. So far, we have implicitly assumed that the input representation is determined a priori and kept fixed. Alternatively, one could randomly assign dense representations to represent each symbol, using relatively small vectors such that $d << |V|$ and consider them as learnable parameters of the model. This kind of representations in language goes under the name of embeddings. When text is segmented into words, we talk about Word Embeddings (WE). Word embeddings immediately reduce the size of matrix E from $R^{|V|\times|V|}$ to $R^{|V|\times d}$. Typical dimensions for d are between 50 up to 1000, usually around 300. In any case, d is significantly smaller than $|V|$, making the memory occupation cost for E growing as $O(|V| \cdot d)$ instead as $O(|V|^2)$. More importantly, a distributed learnable representation of words tremendously eases the learning process, because it allows to adjust the representations of the symbols (words) such that related elements are closer in the representation space than unrelated symbols, improving the generalization capabilities of the model.

The idea of continuous word vectors was originally introduced long time ago in Hinton et al. (1986), and re-proposed for the Neural Language Model in Bengio et al. (2003) that we discussed in Section 2.1.4. However, the extensive use of WEs became popular after the seminal works of Mikolov et al. (2010, 2013). They showed that WEs can grasp both syntactic and semantic regularities of text, and that words are disposed in the embedding space such that some algebraic properties between them emerge, as in the well known example of *king*, *queen* and *man*,*woman*:

$$we('king') - we('queen') \simeq we('man') - we('woman').$$

The main obstacle to the spread of distributed representations of words before Mikolov et al. (2010, 2013) was the fact that learning the matrix E of $|V| \times d$ parameters requires large amounts of data and computational resources. Annotated corpora available were generally too small, and training a neural model this way was still not affordable at that time. In Mikolov et al. (2010, 2013), they obtained word embeddings with simple shallow algorithms on unsupervised, arbitrarily large, textual corpora. Once trained, these representations are easily transferable and exploitable to train models in any NLP task. Literally dozens of methods designed to create WEs

have followed since then. Here we describe only CBOW and Skip-gram (Mikolov et al. (2013)), that rely on a basic principle that is shared by most of these methods, included the one proposed in this thesis in Chapter 3.

CBOW and Skip-Gram

CBOW stands for Continuous Bag of Words. The idea is inspired by a famous quote of Ruper Firth:

" You shall know a word by the company it keeps"

In other words, the word's meaning is given by the words in the context where it usually appears in. From such principle they defined an optimization problem where the goal is to estimate the probability of a word given the context in which it appears. So, given a sequence of words $\boldsymbol{w} := (w_1, \ldots, w_n)$, the goal is to estimate $p_E(w_i|w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_n)$, where p is a model parametrized by the embedding matrix E only. In practice, only a fixed length context window of size k before and after the target word w_i is considered, hence conditioning p_E on

$$w_{i-k}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+k}.$$

The task can be formulated as a standard classification problem by setting the context $w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_n$ as input and w_i as target.

The model is very simple, each word w_j in the context is associated to its embedding e_j corresponding to a row of the embedding matrix E (randomly initialized). This association can be seen as a hashing function, or as a linear layer (without bias) that projects the one-hot $|V|$ sized representation of a word into a d-dimensional space. The embeddings of the words in the context are summed together:

$$e_{c_i} = \sum_{j=i-k, i \neq j}^{i+k} \boldsymbol{e}_j, \tag{2.17}$$

then the resulting vector e_{c_i} is projected into the target space with a linear transformation, and normalized as a probability with the `softmax` function:

$$p(w_i|w_{i-k}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+k}) = \texttt{softmax}(e_{c_i} \cdot E') \tag{2.18}$$

where $'$ indicates the transpose operator.

Skip-gram is the dual formulation of CBOW. Instead of predicting a word given its context, the words in the context are predicted given the central word. A sketch of both models is presented in Figure 2.3. Both models are optimized with a log-linear classifier, usually the cross-entropy. The shallowness of these approaches makes them particularly efficient, allowing the processing of large textual corpora.

Connections with Language Modeling. Estimating the probability of a word given its context has several analogies with Language Modeling. In Section 2.1 we have defined Language Modeling as the problem of estimating the joint probability of a sequence of words, and shown in Equation 2.2 that it can be decomposed as product of conditional probabilities $p(w_i|w_1, \ldots, w_{i-1})$. Intuitively, instead of estimating the probability of a word w_i given the previous words, i.e. only the words in its left context, CBOW takes into account the entire surrounding context, before and after w_i. The conditional probability optimized in CBOW can be derived in the same manner from Equation 2.1, as follows:

$$p(w_1, \ldots, w_n) = p(w_i|c_i) \cdot p(c_i), \tag{2.19}$$

where $c_i := w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_n$. However, the problem defined in CBOW is not equivalent to Language Modeling, because the second term $p(c_i)$ in Equation 2.19 cannot be recursively expanded as it was done in Equation 2.2, so $p_E(w_i|c_i)$ is not enough to estimate Equation 2.1. Moreover, the approximation of the context c_i on a fixed-length window interval is analogous to the approximation of Equation 2.5 done in N-gram models, where in this case, the truncation occurs on both the left and right sides of a word.

Limitations

Despite their benefits, word embeddings have two major limitations. Being word-based representations, WEs lack of morphological knowledge about text, that, as we already detailed in Subsection 2.2.1, can be crucial in specific use-cases, and more importantly overcomes the problem of unknown tokens.

The second limitation comes from multi-sense words and the intrinsic ambiguity of language. The actual meaning of a word highly depends on the context in which it is placed, and its sense may change dramatically from one sentence to another. Assigning a unique embedding to words, without contextualizing it in the current context, inevitably limits the power of such representation. In the next two subsections, we will discuss how to address these problems, posing the basis for our proposed approach presented in Chapter 3.

2.2.4 Sub-word Encoding

There are multiple ways to create representations that are aware of sub-word information. We can either inject sub-word knowledge to create word embeddings or we can directly work with sub-word tokenization. In the former case, sub-word information enriches the representation of a word, whereas in the latter, CBOW-like approaches or Language Modeling are applicable in the same way, being the only difference the fact that tokens are not words (e.g. characters, syllables, etc).

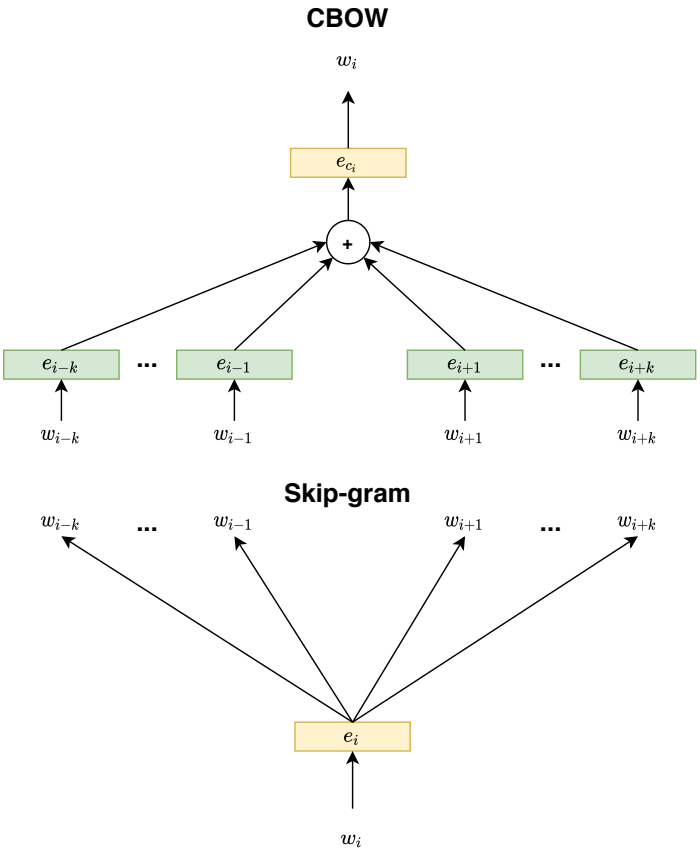

Figure 2.3: Sketch of CBOW and Skip-gram Algorithms.

2.2.5 Contextual Representations

A contextual representation of a token is an embedding that depends on the context where the token appears. In other words, the representation of a token in a sequence is obtained by taking into account the entire sequence (e.g. sentence or paragraph). Clearly, a model providing contextual representations of tokens is more expressive than word embeddings. The outcome of CBOW-like approaches is an embedding matrix E, reusable and transferable in other tasks. To produce contextual representations, we need a model capable of processing sequences adjusting the representations accordingly.

Bidirectional Recurrent Neural Networks are a class of models particularly effective to compute contextual representations of tokens. Let us describe them.

Bidirectional Recurrent Neural Networks. From a technical point of view, Bidirectional Recurrent Neural Networks (Bi-RNNs), proposed in Schuster and Paliwal (1997), are just a combination of two disjoint RNNs (presented in Subsection 2.1.5), processing the input sequence in opposite directions. At each time step, the states

of this two RNNs are concatenated. Given an input sequence $\boldsymbol{x} = (x_1, \ldots, x_n)$:

$$h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}] = [\overrightarrow{r}(x_1, \ldots, x_t), \overleftarrow{r}(x_t, \ldots, x_n)], \tag{2.20}$$

where $\overrightarrow{r}, \overleftarrow{r}$ are the cells of the two recurrent networks, respectively processing the input in forward and backward directions, and $[\cdot, \cdot]$ is the concatenation operator. In case of sequence classification, Bi-RNNs are usually exploited to compute a single hidden state, concatenating the final state of both the forward and backward networks:

$$h_f = [\overrightarrow{r}(x_1, \ldots, x_n), \overleftarrow{r}(x_1, \ldots, x_n)]. \tag{2.21}$$

When the cells $\overrightarrow{r}, \overleftarrow{r}$ are LSTMs, we obtain a Bi-LSTM, that is one of the most popular bidirectional RNN architectures. We will make use of Bi-LSTMs in the next Chapters, hence we rewrite equations 2.20, 2.21 for Bi-LSTMs:

$$h_t = [\overrightarrow{\texttt{LSTM}}(x_1, \ldots, x_t), \overleftarrow{\texttt{LSTM}}(x_t, \ldots, x_n)], \tag{2.22}$$

$$h_f = [\overrightarrow{\texttt{LSTM}}(x_1, \ldots, x_n), \overleftarrow{\texttt{LSTM}}(x_1, \ldots, x_n)]. \tag{2.23}$$

We concisely define the computations of Equation 2.22 as:

$$h_t := \texttt{Bi-LSTM}(x_i, h_{t-1}).$$

Despite their simplicity, bidirectional architectures have become quite popular because they alleviate the problem of learning long-term dependencies (Bengio et al. (1994)), since the distance between the current token and the past ones is reduced. At the same time, the hidden state h_t calculated at each step, naturally provides a contextual representation of the token x_t.

2.3 Language Generation

In its most general view, Natural Language Generation (NLG) is the problem of automatically generating text. The kinds of NLG problems can be various and heterogeneous. Text Summarization, Paraphrasing, Machine Translation, Poem Generation, Text Continuation and even Question Answering are all problems formulated as NLG tasks. Despite this variability, nowadays all NLG models can be reconducted to Language Models opportunely decoded at inference time. In this Section, we first describe the reasons of this connection with LMs, then we summarize the main decoding strategies that are chosen depending on the kind of task at hand.

2.3.1 Text Generation is Language Modeling

Natural Language Generation problems are all special instances of Language Modeling. To understand why, let us consider a sequence of tokens $(w_1, \ldots, w_{n+m})$ taken

from a text corpus in a target language. Since, most of NLG tasks are actually sequence-to-sequence problems, it is convenient to divide the tokens into two disjoint sequences $\boldsymbol{x}$ and $\boldsymbol{y}$, where $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_m)$. The former $(\boldsymbol{x})$ is the context provided to the text generator in advance, from which to base the new text to generate. More generally, $\boldsymbol{x}$ is a given source of information that conditions the generation of $\boldsymbol{y}$ ($\boldsymbol{x}$ could also be empty), while $\boldsymbol{y}$ is the sequence that is expected to be generated by the model. The goal of the LM is then to estimate the probability $p(\boldsymbol{y}|\boldsymbol{x})$, that is factorized as follows,

$$p(\boldsymbol{y}|\boldsymbol{x}) = \prod_{i=1}^{m} p(y_i|y_{<i}, \boldsymbol{x}), \tag{2.24}$$

being $y_{<i}$ a compact notation to indicate the words in the left context of y_i. Hence, the problem of estimating $\mathrm{p}(\boldsymbol{y}|\boldsymbol{x})$ reduces to the learning of the $p(y_i|y_{<i}, \boldsymbol{x})$ distribution. Standard state-of-the-art approaches estimate $p(y_i|y_{<i}, \boldsymbol{x})$ exploiting sequence-to-sequence models. Shortly, a sequence-to-sequence model is a neural network constituted by an encoder and a decoder. The encoder is responsible for creating a compact representation of $\boldsymbol{x}$, while the decoder yields a probability distribution over the tokens in V conditioned by the outcome of the encoder.

Let us show how most common language generation problems can be formulated as in Equation 2.24:

Machine Translation. The connection between Equation 2.24 and Machine Translation is straightforward. There are two sequences, an input text $\boldsymbol{x}$ from a given language and a target text $\boldsymbol{y}$ corresponding to the translation of $\boldsymbol{x}$ into another language. Clearly, the target translation $\boldsymbol{y}$ is highly conditioned by the source sequence $\boldsymbol{x}$.

Text Summarization. In Summarization the expected output sequence is a summary of a given input sequence. Therefore, also in this case $\boldsymbol{y}$ is highly correlated to $\boldsymbol{x}$.

Paraphrasing. An input text $\boldsymbol{x}$ is rephrased into $\boldsymbol{y}$, so that $\boldsymbol{y}$ conveys the same meaning while using different words.

Language Modeling. When the sequence $\boldsymbol{x}$ has size $n = 0$, we fall back to the traditional LM formulation as defined in Equation 2.2.

Text Continuation. The task is the generation of text $\boldsymbol{y}$ from an input text passage $\boldsymbol{x}$. The generation in this problem is inherently loosely correlated by the input, since there are many ways to continue an input text.

Poem Generation. As in Text Continuation, few verses may be provided as input $\boldsymbol{x}$, however, the previous verses are not enough to determine a unique continuation ($\boldsymbol{y}$) of the poem.

Open vs non-open ended text generation. The way $p(y_i|y_{<i}, \boldsymbol{x})$ will be related to the input sequence $\boldsymbol{x}$ depends on how strongly $\boldsymbol{x}$ is informative with respect to $\boldsymbol{y}$. We adopt the distinction in two categories of text generation problems that was provided in Holtzman et al. (2019). Tasks where the source sequence significantly biases the generation outcome are referred to as *non-open-ended* text generation, in contrast to *open-ended* text generation, where the source sequence loosely correlates with the output $\boldsymbol{y}$. Machine Translation, Text Summarization, Paraphrasing, Machine Comprehension are all instances of *non-open-ended* text generation, whereas Text Continuation and Poem Generation are examples of *open-ended* tasks. Of course, this categorization may be fuzzy, indeed there may be some particular problems or applications that are in between the two groups. In terms of optimization, all the models are learned based on the same objective, i.e. maximizing the likelihood of $p(\boldsymbol{y}|\boldsymbol{x})$. The distinction between open and non-open ended tasks is important to decide how to generate tokens at inference time.

2.3.2 Decoding Strategies

A trained Language Model estimates the probability $p(\boldsymbol{y}|\boldsymbol{x})$. At inference time, the model generates the sequence $\boldsymbol{y}$, one token at a time feeding itself with the previous output, i.e. by applying an autoregressive update scheme. There are different decoding strategies that are commonly used to determine the output sequence $\boldsymbol{y}$. Holtzman et al. (2019) have shown that the decoding strategy is essential for producing good quality results, and its choice intimately depends on the kind of text generation problem at hand. All the approaches can be divided into maximum-likelihood and sampling methods, that are better suited for *non-open-ended* and *open-ended* text generation, respectively.

Maximum-likelihood Decoding

Maximum-likelihood approaches aim to obtain the most probable output sequence, i.e.:

$$\boldsymbol{y} = (y_1, \ldots, y_m) = \arg\max_{\boldsymbol{y}} \prod_{i=1}^{m} p(y_i|y_{<i}, \boldsymbol{x}). \tag{2.25}$$

Unfortunately, finding the optimal $\boldsymbol{y}$ is intractable. The number of possible sequences grows exponentially w.r.t. the sequence length m. The computational cost for searching the optimal solution is $O(|V|^m)$, being $|V|$ the vocabulary size. Therefore, search methods that explore only a small subset of sequences have been devised. *Beam*

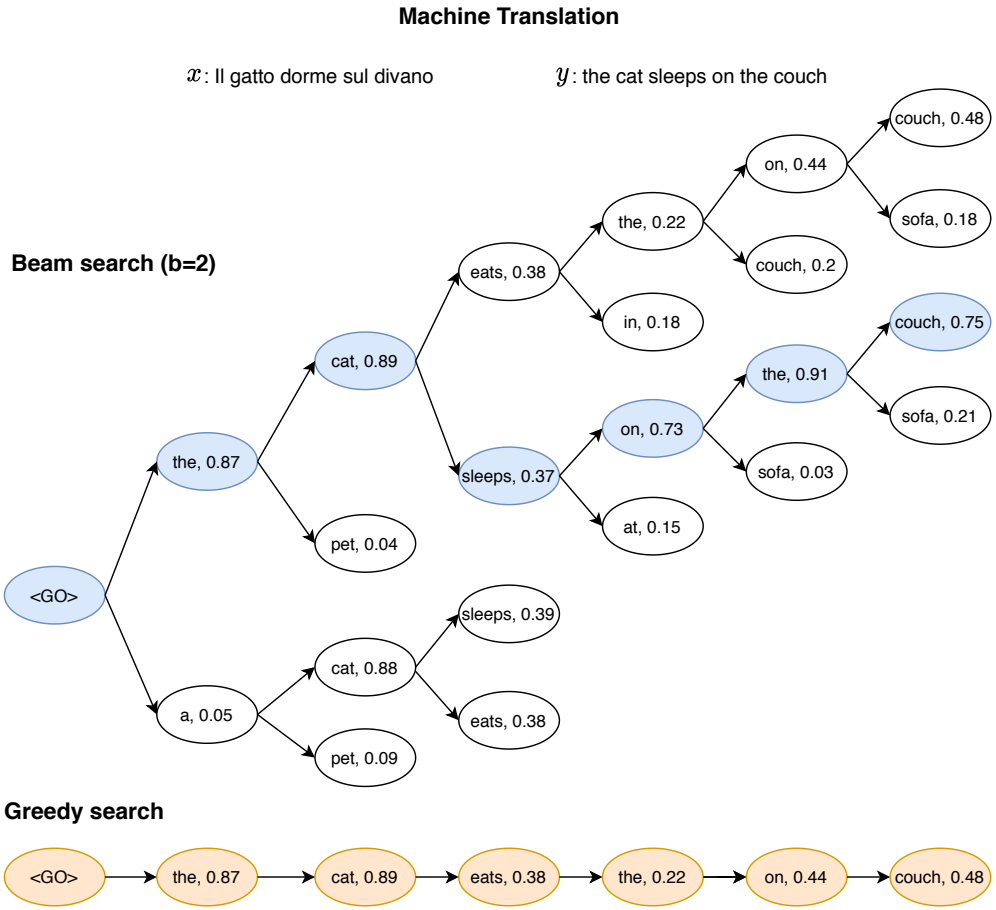

Figure 2.4: Illustration of *beam search* and *greedy search* decoding in a Machine Translation example. Both algorithms are heuristic, but beam search is exploring more solutions – in this case b is set to 2– generally leading to better results, as in this example, where beam search finds the correct sequence, while greedy search does not.

search and *greedy search* are the most popular ones. At each time step, beam search reduces the exploration to the b most promising sub-sequences. The computational cost drops to $O(m \cdot b^2)$, making the approach effective for a relatively small b. Greedy search adopts the same heuristics for the extreme case where only the best sub-sequence is expanded, which is equivalent to beam search with $b = 1$. Hence, Equation 2.25 becomes:

$$\boldsymbol{y} = \prod_{i=1}^{m} \arg\max_{y_i} p(y_i | y_{<i}, \boldsymbol{x}), \tag{2.26}$$

which has a linear cost w.r.t. the sequence length m. The two approaches are illustrated in Figure 2.4.

Despite their simplicity, those search algorithms are quite effective for decoding text because the distribution $p(y_i|y_{<i}, \boldsymbol{x})$, typically modelled by a neural model

with a softmax layer, has only few candidates that are worth to be explored, the rest is a long tail of tokens with very low probabilities. Maximum likelihood decoding, beam search in particular, generally performs well in *non-open-ended* text generation problems, such as Machine Translation, Text Summarization and Paraphrasing, where $\boldsymbol{x}$ highly conditions the probability distribution. On the contrary, in *open-ended* domains, likelihood maximization dramatically fails, leading to repetitive degenerate text (Holtzman et al. (2019)). More than a model flaw, it seems that the problem is related to the fact that humans naturally avoid to state the obvious (Grice (1975)), hence making highly probable sequences implausible, in stark contrast with maximum-likelihood approaches.

Sampling Methods

Surprisingly enough, when text generation is only marginally biased by the given input $\boldsymbol{x}$, randomization in the decoding is crucial for producing high-quality texts. Indeed, noise allows to pick less probable elements, emulating the human behaviour of not stating the obvious. Randomness in decoding also introduces variability to generation process. In fact, techniques based on probability maximization are deterministic, i.e. multiple generations with the same input will inevitably produce the same output.

Randomization is achieved by sampling the distribution. Intuitively, instead of picking the token y_i with highest probability, any token can be selected, depending on how probable it is. On average more likely elements are selected most of the times, whereas tokens from the tail have a very low chance to be chosen. At each time step i the next token is sampled from $p(y_i|y_{<i}, \boldsymbol{x})$:

$$y_i \sim p(y_i|y_{<i}, \boldsymbol{x}) = \frac{exp(h_i)}{\sum_{j=1}^{|V|} exp(h_j)}, \tag{2.27}$$

where h_i, h_j are the conditional hidden representations of the neural language model. This straightforward approach is known as *multinomial sampling*. It overcomes repetitive generations, but still leads to degenerate texts. The problem is that the vast majority of the tokens at time i is implausible and would make the text nonsensical, and albeit sampling one particular element in the tail of the distribution is unlikely to be selected, the probability mass of sampling a generic token from the tail is not neglectable, making sampling from the tail in a sequence of size m eventually extremely likely. To analyze this phenomenon let us define $\mathcal{T}$ as the tail set,. corresponding to the set of tokens with probability below the gold token. This set is typically large, and has a cardinality $|\mathcal{T}|$ comparable to the vocabulary size. At each time step, the probability of sampling from the tail is:

$$\epsilon_{i,\mathcal{T}} = p(y_i \in \mathcal{T}|y_{<i}, \boldsymbol{x}) = \epsilon_i \cdot |\mathcal{T}| \tag{2.28}$$

where ϵ_i is the average probability of the elements in the tail. Thus, the probability of not sampling the tail in the entire sequence is:

$$p(\boldsymbol{y} \notin \mathcal{T}|\boldsymbol{x}) = \prod_{i=1}^{m}(1 - \epsilon_{i,\mathcal{T}}), \tag{2.29}$$

that decreases exponentially w.r.t. the sequence length m.

One way to alleviate the problem is by sampling with temperature t (Goodfellow et al. (2016)). The softmax in Equation 2.27 is re-estimated introducing a scaling factor t as follows:

$$p(y_i|y_{<i}, \boldsymbol{x}) = \frac{exp(h_i/t)}{\sum_{j=1}^{|V|} exp(h_j/t)}. \tag{2.30}$$

The value of the temperature regulates the skeweness of the distribution. Typical values of t are in the interval $[0, 1)$ where the distribution becomes more skewed. In particular, when t approaches 0 it resembles greedy search decoding, whereas when t approaches to 1 it goes back towards multinomial sampling. Making the distribution more skewed reduces significantly the average value ϵ_i of the elements in the tail, and consequently $\epsilon_{i,\mathcal{T}}$, but it also reduces the generation diversity.

There exists several methods that overcome tail issues by truncating it. Essentially they differ in the truncation strategy. *Top-k sampling* (Fan et al. (2018)) truncates the distribution to the set of most probable k elements $V^{(k)}$. The number of tokens is fixed, determined a priory. In *Nucleus* (*top-p*) *sampling* (Holtzman et al. (2019)) instead, values are added in the non-tail set $V^{(p)}$ in decreasing order starting from the most probable one, until the probability mass does not outreach a certain p value threshold. In this case, the number of truncated tokens can vary. Regardless of the method, the final distribution is obtained by setting tail elements to 0 and normalizing the remaining values so that they sum up to 1:

$$p'(y_i|y_{<i}, \boldsymbol{x}) = \begin{cases} \frac{p(y_i|y_{<i}, \boldsymbol{x})}{\sum_{y_j \in V^{(*)}} p(y_j|y_{<j}, \boldsymbol{x})}, & \text{if } y_i \in V^{(*)} \\ 0, & \text{otherwise,} \end{cases} \tag{2.31}$$

where $V^{(*)}$ is either $V^{(k)}$ or $V^{(p)}$, for top-k and nucleus sampling, respectively.

Chapter 3

Character-aware Representations

Developing methods for learning powerful general purpose representations of language, words in particular, has been a breakthrough in Natural Language Processing. We have described word embeddings in Section 2.2, highlighting the impact and the diffusion of methods like CBOW and Skip-gram (Mikolov et al. (2013)), that allowed to learn efficiently word representations from large unlabeled datasets, solving a task related to language modeling. At the same time, we have also seen some limitations of word-based representations. Briefly, traditional word embeddings: (1) neglect morphological information coming from characters, struggling in scenarios with a large Vocabulary, especially in the case of morphologically rich languages in which the same lexical form can have many surface forms, or with many unseen or rare words (e.g. typos or Named Entities) ; (2) multi-sense words condense multiple meanings into the same lexical unit, regardless the actual context in which they appear in.

In this Chapter we address these limitations presenting a novel character-based neural model that effectively learns representations of words and contexts, in a completely unsupervised learning mechanism that follows the same principle of CBOW, discussed in Section 2.2. We first review the works in the literature, then we discuss in detail the architecture and the learning algorithm. Afterwards, we report the effectiveness and robustness of the learned representations in the experimental analysis carried out in two well known NLP problems: Chunking and Word Sense Disambiguation. In addition, we highlight the properties of the embeddings in a qualitative evaluation. Finally, we make a final discussion.

3.1 Related Works

There is a vast literature concerning language representation learning methodologies with different input representations.

Learning. The proposed model learns with an unsupervised mechanism that follows the scheme proposed in the CBOW algorithm (Mikolov et al. (2013)). Instead of using a fixed-length context window, we adopt recurrent neural networks to process sequences of variable size, following the approach proposed by Melamud et al. (2016). Instead of processing words directly, we consider words as sequences of characters and process them with bidirectional recurrent neural networks.

Character representations. There are many character-aware approaches. Some of them jointly learn word and character–based representations from supervised tasks. Word and character–based encodings are either concatenated or combined trough non linear functions. Miyamoto and Cho (2016) exploit a gate to decide how to interpolate the two representations. In (Santos and Zadrozny (2014); Santos and Guimaraes (2015)) instead, embeddings of words and characters are concatenated to address Part–Of–Speech (POS) Tagging and Named Entity Recognition (NER), respectively. We distinguish from them for the learning scheme, that is completely unsupervised, and for the fact that we construct word embeddings directly from characters without specific word-based parameters. Other works extract vectorial representations directly from character sequences, for Language Modeling (LM) or Character Language Modeling (CLM). These representations are usually obtained with Convolutional Neural Networks (CNNs) or Recurrent Networks. In particular, in Ling et al. (2015a) Bi-LSTMs (see Subsection 2.2.5) are exploited to learn task–dependent character level features for LM and POS tagging, whereas in (Jozefowicz et al. (2016)) various architectures, mostly CNN-based, are assessed in LM problems. In Hwang and Sung (2017), authors address CLM with a multi–layered Hierarchical RNN. All these approaches differ from ours for the learning method and for the architecture, that naturally develops character–aware representations of both contexts and words exploitable as task–independent features.

3.2 Model

The model is designed to build hierarchically representations of words and contexts from characters. The proposed model combines two Bi-LSTMs (see Equation 2.22) to process the input sequence. The architecture is illustrated in the example of Figure 3.1.

Tokenization. A sentence s is tokenized as a sequence of n words $s := (w_1, \ldots, w_n)$ and each word is further split into a sequence of characters, i.e. $w_i = (c_{i,1}, \ldots, c_{i,|w_i|})$, being $|w_i|$ the number of characters in w_i, or in other words, the length of word w_i. Despite the separation of words, in the end, s is a sequence of sequences, where the symbols within s are only characters. In such a way, we preserve the word-level

structure of language without the need for a word-based vocabulary V or the word embedding matrix E.

Encoding Characters. Each character c_{ij} is encoded as an index in a dictionary of C characters and it is mapped to a real vector $\hat{c}_{ij} \in \mathbb{R}^{d_c}$ as

$$\hat{c}_{ij} = W_c \cdot 1(c_{ij}), \tag{3.1}$$

where $W_c \in \mathbb{R}^{|C| \times d_c}$ is the character embeddings matrix, and $1(\cdot)$ is a function returning a one-hot representation of the character in input. Note that C is quite small, in the order of hundreds, and consequently W_c is small too, reducing of several orders of magnitudes the number of learnable weights of the model.

Encoding Words. The first Bi-LSTM, denoted as $\texttt{Bi-LSTM}_c$, encodes each word w_i, processing its characters $c_{i,*}$ in forward and backward directions. The final state of the forward and backward networks are combined as described in Subsection 2.2.5 to obtain the *word embedding*:

$$e_i = [\overrightarrow{\texttt{LSTM}_c}(\hat{c}_{i,1}, \ldots, \hat{c}_{i,|w_i|}), \overleftarrow{\texttt{LSTM}_c}(\hat{c}_{i,1}, \ldots, \hat{c}_{i,|w_i|})]. \tag{3.2}$$

Hence, the result of processing the character sequence s word-wise with the $\texttt{Bi-LSTM}_c$ is a sequence of word embeddings $e_s = (e_1, \ldots, e_n)$, that will be the input for the next Bi-LSTM layer.

Encoding Contexts. The second Bi-LSTM layer ($\texttt{Bi-LSTM}_e$) processes the sequence e_s. As we have already seen in Subsection 2.2.5, we can both produce the contextual representations of each word and the encoding of the entire sequence from the internal states of a Bi-LSTM. Moreover, we can also construct *context embeddings*, i.e. distributed representations of the surrounding context of each word, excluding the word itself. The context of a word e_i comprises the words that precede and follow e_i, i.e. $(e_1, \ldots, e_{i-1})$ and $(e_{i+1}, \ldots, e_n)$, respectively. Such encoding is obtained concatenating the RNN's states before processing the current word e_i:

$$\hat{e}_i = \texttt{MLP}([\overrightarrow{h}_{e_{i-1}}, \overleftarrow{h}_{e_{i+1}}]) = \texttt{MLP}([\overrightarrow{\texttt{LSTM}_e}(e_1 \ldots, e_{i-1}), \overleftarrow{\texttt{LSTM}_e}(e_{i+1} \ldots e_n)]), \tag{3.3}$$

where $\texttt{MLP}$ is a Multi-Layer Perceptron introduced in the architecture to project the representation into a lower-dimensional space. Inspired by (Mikolov et al. (2013)) and (Melamud et al. (2016)), we will use context embeddings to learn to predict a word given its context.

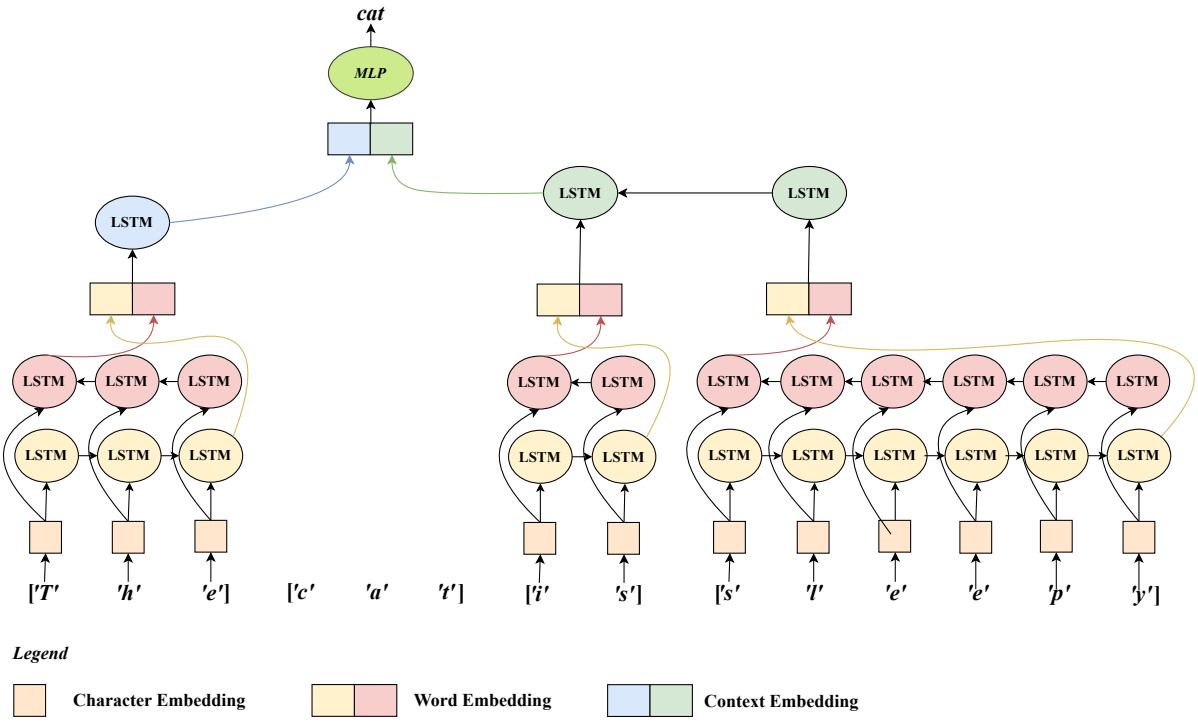

Figure 3.1: Example: the sentence "*The cat is sleepy*" is fed to our model, with target word *cat*. The sequence of character embeddings (orange squares on the bottom) are processed by the word–level Bi-LSTM yielding the word embeddings (yellow–red squares in the middle). The context-level Bi-LSTM processes the word embeddings in the left and right contexts of *cat*, to compute a representation of the whole context (blue–green squares on the top). Such representation is used to predict the target word *cat*, after having projected it by means of an MLP.

3.3 Learning Representations

In this Section we describe the learning mechanisms that allow word and context representations of our character-aware model to emerge. First we describe the algorithm and its optimization, then we provide details about the actual training.

The algorithm. The learning mechanism is inspired by CBOW (see Section 2.2) and *context2vec* (Melamud et al. (2016)). We follow the unsupervised schema that aims at predicting each word given its surrounding context on an arbitrarily large textual corpus (Equation 2.19). Our model encodes the context representations of word w_i into a vector $\hat{e}_i$ as defined in Equation 3.3. Such embedding is projected into the space of words through a linear projection:

$$\hat{y}_i = W_w \cdot \hat{e}_i \tag{3.4}$$

where $W_w \in \mathbb{R}^{|V| \times d}$ and d the size of embedding $\hat{e}_i$. We optimize the model by choosing the Noise Contrastive Estimation (NCE) as loss function (Gutmann and Hyvärinen (2010)), instead of softmax activation in combination with cross-entropy. NCE allows to spare computations by approximating the softmax function, that has a linear cost w.r.t. $|V|$, on a random smaller subset of sampled words.

Clearly, a word dictionary is required during learning, but this does not limit our model, since this decoding block is no longer used after training. We also tried to get rid of words at training time as well, using the context e_i as input to a character-based decoder, i.e. using as target the sequence of characters in a word. However word-level predictions turned out to give the best results. As in (Melamud et al. (2016)), the recurrent networks allow us to process contexts of any length, but we reset the RNN states at the beginning of each sentence.

Training details. The model is implemented in Tensorflow[1]. The model is trained on a 2 billion words dataset, the ukWaC corpus[2], constructed from the Web crawling the *.uk* domain only. Concerning the architecture, where it is possible we take inspiration from the *context2vec* architecture. We set the sizes of characters, word and context embeddings to 50, 1,000, 600, respectively. The hidden layer in the `MLP` of Equation 3.3 has 1,200 units with ReLU activation functions. The structure of the decoder of Equation 3.4 used in training is the same as that in (Melacci and Gori (2012)). The overall encoding architecture has around 7 million parameters, that is about 16 times smaller than the *context2vec* model, due to the use of characters to encode words instead of a word embedding matrix.

3.4 Experiments

Once word and context representations are learned with the proposed model, we can detach the encoders and evaluate the performances of such embeddings as regular features for any task-specific classifier, as shown in Figure 3.2. We consider word embeddings for Chunking and context representations for Word Sense Disambiguation (WSD). Additionally, we also evaluate the robustness of such embeddings w.r.t. character-level noise in the case of WSD. Finally, we present some qualitative examples to illustrate the properties of the obtained word and context representations. In the following, we discuss the experiments one by one.

[1]Source code of model and experiments is available at `https://github.com/GiuseppeMarra/char-word-embeddings`.

[2]`http://wacky.sslmit.unibo.it/doku.php?id=corpora`

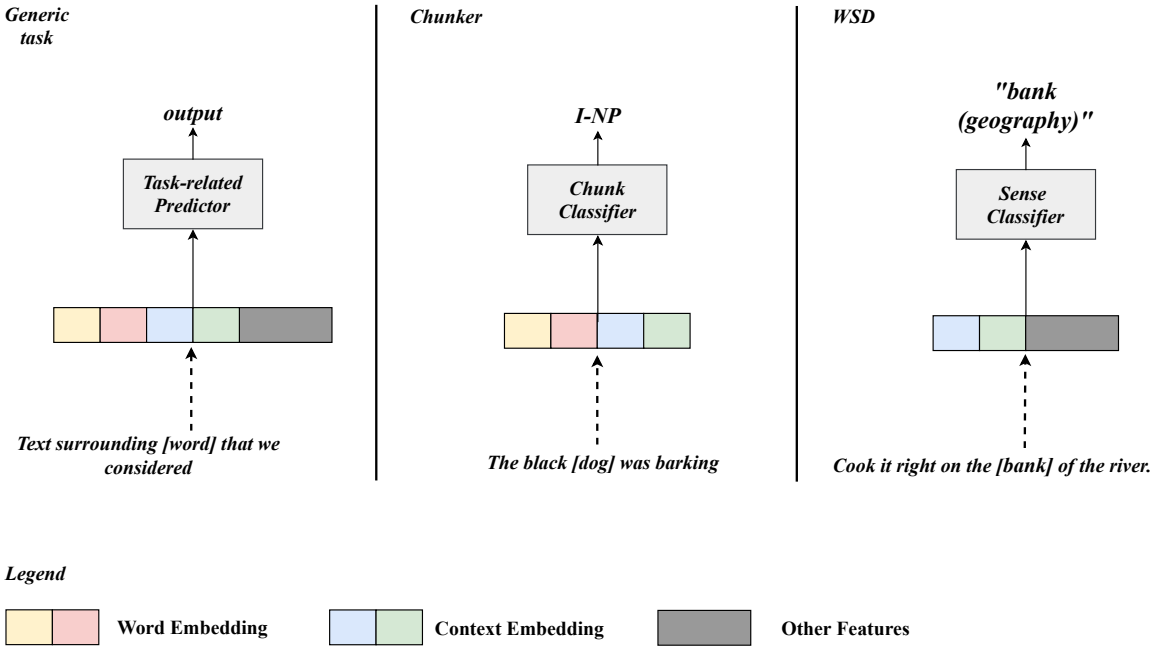

Figure 3.2: Examples of word and context embeddings usage. From left to right, a generic task, Chunking and Word Sense Disambiguation.

3.4.1 Chunking

Chunking is a classical problem in Natural Language Processing. The goal of Chunking is to identify phrases from text. Each text segment is expected to be tagged with a label defining its syntactic role, e.g. noun phrase (NP), verbal phrase (VP), etc. A word is uniquely associated to a unique tag. A Chunking classifier is a model that automatically assigns a word to its chunk. An example is illustrated in Figure 3.2, where *dog* is marked with the label *I-NP* (Inside-chunk Noun Phrase).

Dataset. We consider CoNLL 2000, a standard benchmark in Chunking. The dataset is composed of a training set and a test set. The training set has 211,727 tokens, whereas the test set contains 47,377 tokens, belonging to one of the 23 chunk tags, i.e. the targets.

Classifier. As classifier we exploit a Bi-LSTM. The network is fed at each time step with a 600 dimensional vector, the result of the projection of the concatenation of word and context representations. We optimize the network with Adam with default hyper-parameters and a weight decay of 10^{-3}.

Results. We compare the same classifier with different input features. In particular we train the model with pre-trained Word Embeddings only (**Our WE**), only

with pre-trained Context Embeddings (**Our CE**) and both of them (**Our WE + CE**). Furthermore, we also take into account the case of training WE and CE features from scratch (**WE + CE from scratch**), using the same architecture, but without pre-training it. We report F1 scores in Table 3.1. The best result is obtained when using both embeddings (**Our WE + CE**) as features, which indicates that both of them are needed to achieve better performances, as expected. Experiments clearly outline the importance of knowledge transfer achieved by pre-training the representations, that allows us to reach an F1 score of 93.30. This value is comparable with results reported in literature, such as Collobert et al. (2011) (94.32) and Huang et al. (2015) (94.46), that, it is important remark, in our case was achieved without making use neither of hand-crafted features nor of any kind of post-processing to adjust incoherent predictions. In case we add POS tagging features to our classifier, performances reach an F1 score of 93.94, the same value of the state-of-the-art architecture (Huang et al. (2015)), but without exploiting Conditional Random Fields on the output sequence of tags. Furthermore, we remind that our model, being based on characters has significantly less parameters w.r.t. the competitors. Hence, we can conclude that our proposed architecture and learning framework is beneficial and brings a positive transfer of information.

Table 3.1: Results (F1 score) on Chunking, using different input features. Best result is achieved when using both word and context pre-trained representations.

Input Features	F1 %
Our WE	89.68
Our CE	89.59
Our WE + CE	**93.30**
WE + CE from scratch	89.83

3.4.2 Word Sense Disambiguation

WSD is the problem of determining the actual meaning, or *sense*, of a word in the context where it appears in. Words can have multiples senses, this varies from word to word. The goal of a WSD is to assign the proper sense of a word among its possible senses.

Datasets. Word Sense Disambiguation is traditionally benchmarked on the framework proposed by Raganato et al. (2017), that gathers multiple datasets (Senseval*, SemEval*, and a merged collection - ALL).

Classifier. Our model follows the commonly used IMS approach proposed in (Zhong and Ng (2010)), that is based on an SVM classifier on top of conventional WSD fea-

tures. We extend it by introducing the context embeddings as input features (using word embeddings of the current word is pointless in this task).

Results. We refer to our model as **IMS + Our CE**, and compare it against the original **IMS** and other instances with different augmented context embeddings (**IMS + word2vec**, **IMS + context2vec**). Results are reported in Tables 3.2 and 3.3. Our embeddings outperform both **IMS** with only conventional features and IMS + word2vec, i.e. *word2vec* embeddings, opportunely averaged (Iacobacci et al. (2016)). Moreover, it is competitive against *context2vec* representations, despite, as already mentioned, our model has less learnable parameters, in particular 16 times less weights than *context2vec*. It is also worth to point out that, to the best of our knowledge, the use of *context2vec* features as input of the IMS is a novel attempt in the literature.

Table 3.2: Word Sense Disambiguation in the benchmarks collected in (Raganato et al. (2017)). SE2 and SE3 stand for SensEval2 and SensEval3. The best results (F1 %) are obtained by the *contex2Vec* model that however has 16 TIMES MORE PARAMETERS THAN THE PROPOSED MODEL and no capability to deal with OOV tokens.

Model	SE2	SE3	SemEval2007	SemEval2013	SemEval2015	ALL
IMS	70.2	68.8	62.2	65.3	69.3	68.1
IMS+word2vec	72.2	69.9	62.9	66.2	71.9	69.6
IMS+context2vec	**73.8**	**71.9**	**63.3**	**68.1**	**72.7**	**71.1**
IMS+Our CE	72.8	70.5	62.0	66.2	71.9	69.9

Table 3.3: Overall results (F1 %) grouped by Part of Speech (ALL benchmark Raganato et al. (2017)).

Model	Noun	Adjective	Verb	Adverb
IMS	70.0	75.2	56.0	83.2
IMS+word2vec	71.8	76.1	57.4	83.5
IMS+context2vec	**73.1**	**77.0**	**60.5**	83.5
IMS+Our CE	71.3	76.6	58.1	**83.8**

3.4.3 Robustness to Typos

There are many real world scenarios in NLP where the actual data is noisy. Misspelled words are hardly included in word dictionaries, therefore word-based model treat them as unknown elements, resulting in a loss of information. Our model being based on characters may be more robust to this kind of problems. To prove it, we devise a task to measure the robustness to random perturbations of characters.

We consider WSD (ALL benchmark) and compare the context embeddings of our model against *context2vec*, introducing random noise to the words in the sentence context. Conventional WSD features are completely removed in both cases, so we only use context-level representations. We show how F1 decreases with the noise probability growth in Figure 3.3. Clearly, both the models suffer for perturbations, however our character-aware features yield a slower loss of performances, eventually outperforming *context2vec* for a high degree of perturbations.

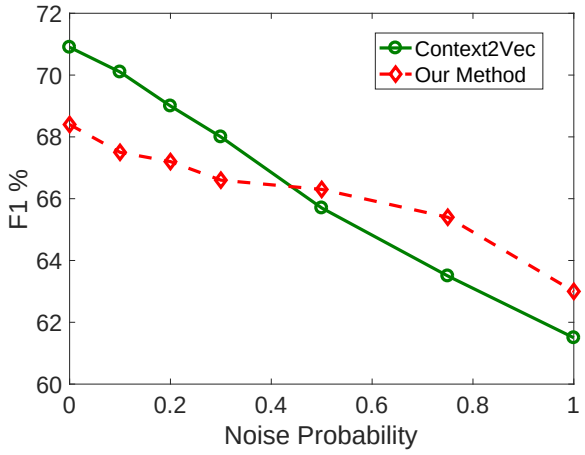

Figure 3.3: Analysis of robustness to typos in a WSD task (ALL benchmark Raganato et al. (2017)). "Noise probability" in the x-axis corresponds to the probability of having a typo in a word in the input context.

3.4.4 Qualitative Analysis

One of the most intriguing properties of embeddings is their capability to capture semantic and syntactic similarities into the topology of the embedding space. We outline such properties for our word and context embeddings by means of examples. In both cases we compute the distance between representations to retrieve k-nearest elements. We chose cosine similarity, that is considered a better measure than euclidean distance in high-dimensional spaces. Given some target words, we report their 5 nearest neighbours in Table 3.4. From the examples it emerges the character-based nature of the model, that captures morphological similarities. However, the model encodes also semantic information, as proven by some of the examples.

For the evaluation of context embeddings, we proceeded analogously. We took eight sentences related to two distinct topics (4 sentences each): pizza and state capitals. For each sentence we consider the context embeddings of words *capital* or *pizza*. Then, a random sentence is chosen as query, and the remaining sentences are sorted in increasing order w.r.t. the distance between the query context embedding and their vectors. The experiment is presented in Table 3.5. The query sentence

is about *pizza*, and we can see clearly that all the sentences related to the topic are closer to the query than sentences about *capitals*.

Table 3.4: Top-5 closest words for a given target word. Target words are in the headline of the table.

turkish	*sometimes*	*usually*	*happiness*
danish	somehow	normally	weirdness
welsh	altogether	basically	fairness
french	perhaps	barely	deformity
kurdish	nonetheless	typically	ripeness
swedish	heretofore	formerly	smoothness

Table 3.5: Some contexts sorted by descending cosine similarity with respect to the query context *"I like eating* [ ] *with cheese and ham"* of (unused) target word *pizza*.

	Query: *I like eating* [ ] *with cheese and ham.*	pizza
Contexts sorted by descending cosine similarity	Do you like to eat [ ] with cheese and salami ?	pizza
	Did you eat [ ] at lunch ?	pizza
	What is the best [ ] i can eat here ?	pizza
	Paris is the [ ] and most populous city in France	capital
	London is the [ ] and most populous city of England	capital
	Rome is the [ ] of Italian Republic .	capital
	Washington , D.C. , ... , is the [ ] of the United States .	capital

3.5 Discussion

We have presented a character-aware neural model that develops task-independent representations of words and contexts by learning from an unsupervised language modeling-related task. The embeddings that emerged from the training on a 2 billion word dataset, were exploited as input features for two popular NLP problems, Chunking and Word Sense Disambiguation, leading to competitive results against state-of-the-art embeddings of models with significantly more parameters. Moreover, the character-level information makes the representations more robust to noise introduced by misspelling, which may be significant in real world scenarios, such as conversational agents.

Chapter 4

Information Extraction in Text Streams

Most of nowadays machine-learning-based approaches to language-related problems are defined in the classical setting where offline predictors learn from data to tackle a given task. Little has been done when considering the setting in which we process a continuous stream of text and build systems that learn and respond in an online manner. However, actual real-world applications, such as conversational systems (Yu et al. (2016)), information extractors from streams of news (Del Corso et al. (2005)), or social network data (Ritter et al. (2012)), and those systems that require interactions with the environment (Christakopoulou et al. (2016)), all belong to the latter setting.

In this Chapter we face these challenges to learn an information extractor from online streams. In particular, we develop an agent living in an online environment that learns to discover and disambiguate entity/relation instances in a text stream, exploiting a small number of sparse supervisions and learning in an online fashion.

The Chapter is organized as follows. Related work is reviewed in Section 4.1. Section 4.2 formally describes the problem setting. The architecture of the proposed system is presented in Section 4.3, while the online learning dynamics is summarized in Section 4.4. We discuss the experiments in Section 4.5, then in Section 4.6 we make some final remarks.

4.1 Related Works

Mining over *text streams* has been studied in a number of works (Banerjee and Basu (2007); Krawczyk et al. (2017); Aggarwal and Zhai (2012)), with several purposes, that, however, are different from what we consider in this Chapter. Our approach to the learning problem is based on simple sentences that have the same structure of the ones used in many tasks of the *bAbI project* by Facebook (Sukhbaatar et al. (2015); Kumar et al. (2016)). However, none of such tasks is conceived for online learning or for entity/relation extraction and disambiguation. Interesting ideas on entity-

oriented sub-symbolic memory components have been recently proposed by Henaff et al. (2017) and Ji et al. (2017), and extended to the case of relations by Bansal et al. (2017): their formulation is developed to comply with the aforementioned bAbI tasks. The idea of considering small text passages could resemble the task of *Machine Comprehension*, where, however, such passages are read with the purpose of answering a question (Richardson et al. (2013); Rajpurkar et al. (2016); Kobayashi et al. (2016)).

Our approach disambiguates mentions using their contexts, so it shares several aspects with *Word Sense Disambiguation* (WSD) and *Entity Linking* (EL), that, differently from our case, assume to work with a given ontology and are not meant to learn online. In WSD (Zhong and Ng (2010); Raganato et al. (2017)), the set of target words is known and the senses of each word are taken from a given dictionary. EL (Shen et al. (2015)) is similar to WSD (Moro et al. (2014); Moro and Navigli (2015)), but it is about linking "potentially partial" entity mentions to a target KB, that has an encyclopedic nature (Hachey et al. (2013); Moro and Navigli (2015)). The EL problem is presented in several variants and focusing on different types of data (Ling et al. (2015b); Guo et al. (2013); Pan et al. (2015); Sil and Florian (2016); Pappu et al. (2017); Lin et al. (2017)), and it has been the subject of task-oriented evaluation procedures and benchmarks (Ma et al. (2017); Van Erp et al. (2016)). A few EL systems work in an unsupervised way (Han and Sun (2011); Pan et al. (2015)), but the KB is still given. *Named Entity Recognition* (NER) focuses on discovering mentions to entities, and it is also a basic module of several EL systems (Luo et al. (2015)). However NER usually deals with proper nouns, as frequently EL does (Ling et al. (2015b)), while here we also consider common nouns. Moreover, NER systems output the entity type (person, location, etc.) without producing any instance-level information (Lample et al. (2016); Chiu and Nichols (2016)). *Relation Extraction* (RE) has been recently approached with end-to-end and advanced embedding-based models (Miwa and Bansal (2016); Obamuyide and Vlachos (2017)). The entities involved in the target relation are usually known, and a pre-defined ontology is given (distant supervisions are also used, as in (Mintz et al. (2009))). There are a number of discussions to better state the RE problem and build accurate gold labels (Martin et al. (2016)).

Our work is also inspired by *Semantic Parsing* (SP) (Zettlemoyer and Collins (2012); Berant et al. (2013); Yih et al. (2014); Berant and Liang (2014)), that has been recently approached with largely supervised networks (Herzig and Berant (2017)). The idea of mapping text into a logical form that represents its meaning in a symbolic way is a direction to which we plan to extend our approach, once we introduce entity/relation types (see Chapter 7). Differently from common SP approaches, we are giving emphasis to the process of learning latent sub-symbolic representations of the KB instances.

Finally, learning the KB component is the subject of those tasks of automatic *KB Construction* (Niu et al. (2012)) and *KB Population* (Dredze et al. (2010); Ji and Grishman (2011)), that, differently from our case, either make some application-specific assumptions to implement the KB, or exploit a given ontology schema, also combining unsupervised and supervised learning with ensembles and stacking techniques (Rajani and Mooney (2016)).

4.2 Problem Setting

Text Stream. We consider a continuous stream of text. At each time step t it produces a sentence s_t. Groups of contiguous sentences are organized into small *stories* about a (not-known-in-advance) set of actors/objects, so that the narration is discontinuous whenever a new story begins. An example of a stream with two stories is illustrated in Figure 4.1.

Agent's goal. The agent reads one sentence at a time. Its goal is to extract salient text fragments and link them to its own KB. The Knowledge Base, initially empty, is constantly updated by the system itself (Figure 4.1).

Knowledge Base structure. We think of the KB as a set of *instances*, and the considered text fragments of s_t are *mentions* to them. Some mentions are about *entities*, others are about *relations*. For each instance, the KB stores (possibly) multiple mentions that are commonly used to refer to the instance itself, as for entities 1 and 2 in Figure 4.1, respectively mentioned as {*Clyde Radcliffe*, *He*, *Clyde*} and {*the office*, *the new office*} . On the other way around, the same mention can be shared among multiple instances. Looking again at the example in Figure 4.1, *Clyde* is a textual form that refers to different entities (entities 1 and 6), on different portions of the stream. KB instances also include information about the *contexts* in which instances have been mentioned in the stream so far, where the notion of *context* compactly indicates the whole sentence in which the instance was mentioned. Here and throughout the Chapter, we simplify the descriptions by frequently using the generic term *instance* to refer to both the cases of relations and entities, without making a precise distinction (if not needed).

Learning setup. We consider the case in which text fragments of s_t are matched with the mentions in the KB to detect compatible instances. Then, instances are disambiguated by observing the current context (that is the portion of s_t around the mention), and exploiting the knowledge about the story to which s_t belongs (up to the current sentence). At the beginning of each story, a small number of sentences are supervised with the identity of the mentioned entities and relations.

The system is expected to follow the narration by disambiguating mentions, learning from such sparse supervisions (and quickly reacting to them), and discovering new instances. The natural ambiguity of language, the discontinuous narration, and the dynamic nature of the KB, require the system to develop advanced disambiguation procedures to interpret s_t.

Evaluation. It is worth noticing that, in the proposed setting, the system decisions on a given sentence are evaluated immediately when processing the following sentences, regardless of whether the sentence was supervised or not. This schema gives rise to a novel online dynamics of the learning process, which is profoundly different from most of the existing approaches, being it more challenging and realistic than common batch-mode offline approaches (consider, for example, the case in which it is framed within conversational applications). We encourage the development of new methods, models and datasets, for this extremely relevant, but largely unexplored setting.

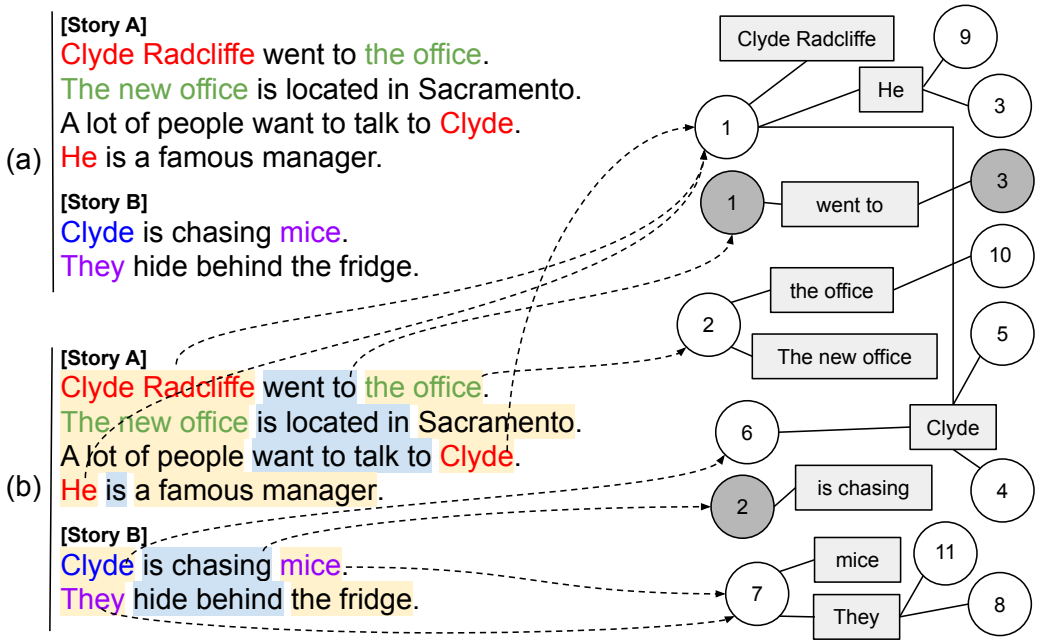

Figure 4.1: Left: a text stream composed of two stories. Right: Example of a KB. (a) Input: sentences from the stream. (b) System output: mentions to entities and relations are detected (pale yellow and pale blue background, respectively), and linked (dashed lines) to KB instances (circles) - only some links are shown, for clarity. Empty circles are *entity* instances, while filled-grey circles are *relation* instances (circles are intended to also include context-related information that characterize the instances). Boxes indicate known mentions, and they are connected with the compatible instances. *We printed with the same color those mentions that should be linked to the same instance.*

4.3 Model

At each time step, the system processes an input sentence s. From now on, we drop the time index, for simplicity. To reach its goal, the agent has to first detect those segments z (i.e. one or more adjacent tokens) that are expected to be mentions of KB instances. Then for each one, the system has to find possible compatible instances of the KB and finally disambiguate among them, linking the segment to the most likely instance. The whole processing involves a pipeline of four neural modules, namely Mention Detector, Encoder, Candidate Generator, Disambiguator. We sketch them in Figure 4.2. The Mention Detector segments s by identifying mentions to entities and relations. For example, the sentence *Parry is chasing a mouse* is segmented into two mentions to entities (i.e "Parry" and "a mouse") and a mention to a relation (i.e "is chasing"). The Encoder module is responsible for embedding the mention z and its context into a dense representation. The combination of such embeddings, the surface form of a mention (plain text), the temporal coherence in the current story are mixed together by the Candidate Generator to obtain a probability distribution over the KB instances. Finally, a Disambiguator takes the final decision of which is the most likely KB instance to link a segment z with, using the aforementioned probability distribution and the representation of the mention context.

Subsections 4.3.1, 4.3.2, 4.3.3, 4.3.4 describe each computational block in details.

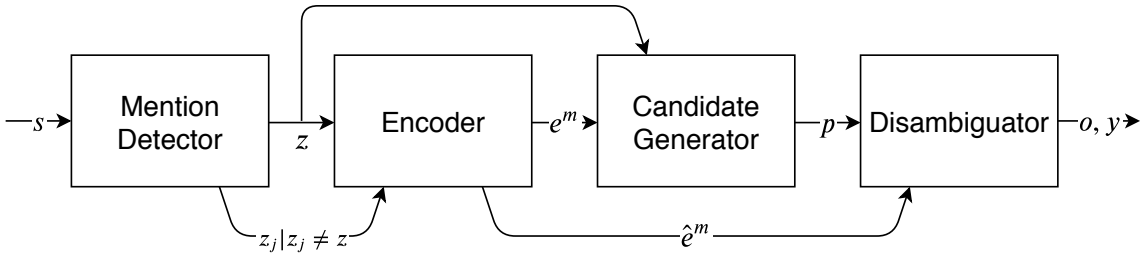

Figure 4.2: Computational flow of the model for mention-instance linking. The system can be seen as a sequential composition of sub-systems that process the input sentence s and finally output the identifiers of the instances that are linked to the mentions detected in s (z is a generic mention detected in s).

4.3.1 Mention Detection

The goal of the Mention Detector (first block of the pipeline in Figure 4.2) is to segment each sentence into non-overlapping text fragments, which are mentions to yet-unknown entity or relation instances.

Motivated by the need of developing models that are robust to morphological changes and that do not depend on a pre-defined vocabulary of words (as needed by interactive/conversational applications), we process the input data using the character-level encoding described in Chapter 3 (Marra et al. (2018)).

Instead of encoding the context of a word as in Equation 3.3, that are representations not including the word itself, we retrieve a *contextual representation* $\tilde{\boldsymbol{e}}_i^w$ of the word itself:

$$\tilde{\boldsymbol{e}}_i^w = \texttt{Bi-LSTM}_e(w_i, h_{i-1}) \,, \tag{4.1}$$

An MLP classifier processes (one-by-one) the contextualized embeddings $\tilde{\boldsymbol{e}}_i^w$ of the words in the input sentence. The predictions are the output of the module. The MLP is trained using supervised learning, with a tagging scheme similar to the approach proposed in (Lample et al. (2016)). In particular, each word in the sequence is tagged, among a total of 6 classes: depending whether a word belongs to an *entity* or a *relation* mention and its position within the mention segment (*begin, inside, end*). As a result, each mention z is composed by the sequence of words where the first word is tagged with the *begin* tag, the last word with the *end* tag, and the other words are predicted as *inner* (tags must be all of the same type, either entity or relation).

As remarked in Section 4.2, we exploit sparse supervisions, so that training the MLP-tagger (and, in turn, the character-aware network) might be difficult. However, we follow the intuition that syntax has a crucial role in text segmentation: noun phrases are mentions to entities, while fragments that start with a verb and end with a preposition (if any) are mentions to relations (Fader et al. (2011)). Hence, we can use these rules to automatically generate artificial supervisions on large collections of text to pre-train the Mention Detector.

4.3.2 Mention and Context Encoding

The Encoder module (second block of Figure 4.2) is responsible for encoding the mentions detected and their contexts into vectorial representations. Once again, we rely on the character-aware model presented in Chapter 3. From the output of the previous computational block, the Encoder receives the input sentence segmented into a sequence of mentions of one or more words each. In details, at this stage, s is a sequence of m_s mentions $z_i, i = 1, \ldots, m_s$. Two different vectorial representations are computed for each mention.

Encoding Mentions. First, the *mention embedding* $\boldsymbol{e}_i^m$ of z_i is obtained as

$$\boldsymbol{e}_i^m = \texttt{Bi-LSTM}_c(\hat{\boldsymbol{c}}_{i,1} \ldots \hat{\boldsymbol{c}}_{i,|z_i|}) \,, \tag{4.2}$$

being $\hat{\boldsymbol{c}}_{i,1} \ldots \hat{\boldsymbol{c}}_{i,|z_i|}$ the sequence of characters of the mention.

Encoding Mention Contexts. Second, the *context embedding* $\hat{\boldsymbol{e}}_i^m$ is computed from the other mentions z_j, $j \neq i$ in s, as

$$\hat{\boldsymbol{e}}_i^m = [\overrightarrow{\texttt{LSTM}_e}(\boldsymbol{e}_1^m \ldots, \boldsymbol{e}_{i-1}^m), \overleftarrow{\texttt{LSTM}_e}(\boldsymbol{e}_n^m \ldots \boldsymbol{e}_{i+1}^m)]. \tag{4.3}$$

Notice that, differently from Equation 4.1, here we are encoding *only* the context around the considered mention, excluding the mention itself. Equation 4.3 is analogous to Equation 3.3, except for its input, that is made of a sequence of mention embeddings instead of word vectors.

Once the Mention Detector has been pre-trained accordingly to what suggested in Section 4.3.1, the Encoder module can be pre-trained as well without any human intervention, by processing large collections of text and learning to decode each mention.

4.3.3 Candidate Generation

Given an input mention z from the current sentence and its embedding $\hat{\boldsymbol{e}}^m$, the Candidate Generator (third block of Figure 4.2) implements four *memory components* that are used to generate a list of candidate KB instances compatible with z, and, afterwards, to store the information on the disambiguated instance. Initially, all the components are empty and they are progressively filled while learning. Before providing further details on the candidate generation process, we describe the four memory components, as shown in Figure 4.3.

Memory Components

Mentions Set. The memory component $\mathcal{H}$ is an ordered set that collects all the mentions that were processed up to the current sentence; this allows a fast lookup of previously predicted instances for specific mentions. For example, if the mention *John Doe* was previously assigned to instance k, when processing the same mention again the system could easily hypothesize that it still belongs to instance k. In other words, $\mathcal{H}$ is a dynamic vocabulary storing mentions encountered by the system up to that moment.

Mention Embeddings. The matrix E stores (row-wise) the embeddings of the mentions in $\mathcal{H}$, computed with the encoder of Section 4.3.2, Equation 4.2. Thanks to a similarity measure in the embedding space, this component allows the system to associate KB instances to never-seen-before mentions which are small variations of previously seen ones, or that refer to semantically similar elements. For example, given the never-seen-before mention *John D.*, the system could easily predict it still

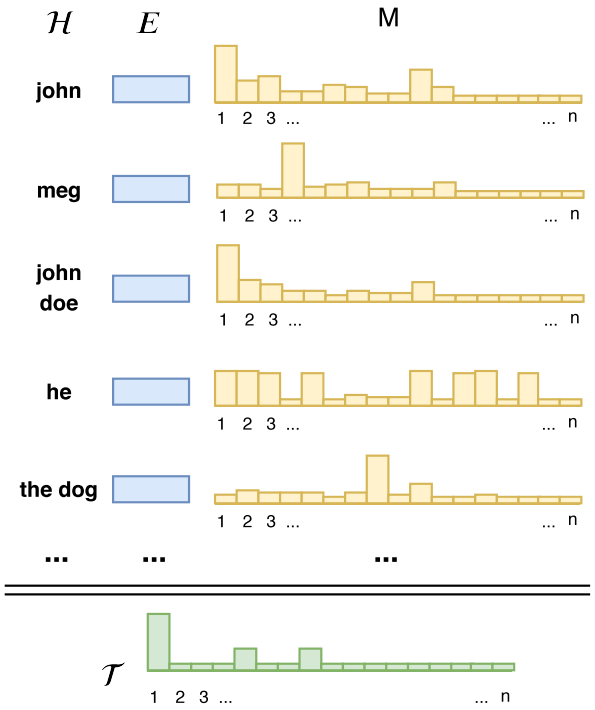

Figure 4.3: A graphical representation of the four memory components $\mathcal{H}$, E, M, and $\mathcal{T}$. $\mathcal{H}$ collects the (lowercase) mentions that are stored in the KB. For each of them, the embedding vector (blue rectangle) is stored in a row of the matrix E. The affinity scores of the considered mention, with respect to each of the n instances in the KB, are stored in a row of the matrix M. Finally, $\mathcal{T}$ contains the KB instances that have been recently linked by the system (here represented as a histogram of the number of times each KB instance was recently linked).

belongs to instance k, since its char-level embedding is close to the one of *John Doe*, even though the exact lookup in $\mathcal{H}$ failed.

History Buffer. The set $\mathcal{T}$ keeps track of the last disambiguated instances (with repetitions). This memory is implemented as a buffer. Despite being naive, it allows the system to handle co-references. As we will see shortly, the system can learn that some specific inputs (e.g. pronouns, category identifiers, etc.) are often assigned to recently mentioned instances, making valuable temporal hypotheses when it has to disambiguate such inputs. The buffer resets at the end of each story.

Mention to Instance Map. The matrix M stores (row-wise) the instance-activation scores of each mention in $\mathcal{H}$. Each row is associated to a mention in $\mathcal{H}$, each column corresponds to a KB instance. The row of M associated to a certain mention $u \in \mathcal{H}$

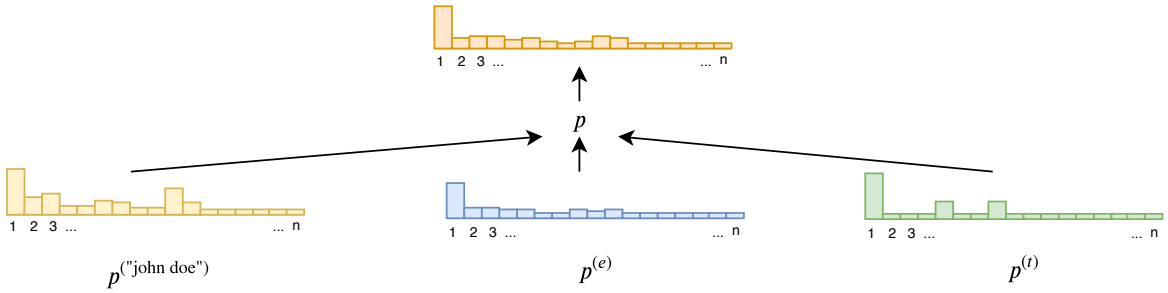

Figure 4.4: A visual representation of the combination of the three hypotheses. $\mathbf{p}^{(z=\text{“john doe”})}$ suggests that the current mention (i.e. “john doe”) has been often linked to entity instance 1. $\mathbf{p}^{(e)}$ indicates that the embedding of the current mention is very similar to the embeddings of mentions usually linked to entity 1. Finally, $\mathbf{p}^{(t)}$ indicates that entity 1 has been recently mentioned several times.

models how strongly each KB instance is associated to the mention u. The matrix M is learned while reading the text stream, as we will describe in Section 4.4.

Hypotheses Formulation

In the following, we will show how the CANDIDATE GENERATOR exploits these memory components to generate candidates of KB instances, given an input mention. For the purpose of this description, we suppose that the current memory consists of n instances and m mentions. For each mention, the *candidate generation* routine first outputs three distinct hypotheses, that are represented by three vectors $\boldsymbol{p}^{(z)}$, $\boldsymbol{p}^{(e)}$, $\boldsymbol{p}^{(t)}$, of n scores each. Each score is in $[0, 1]$, and it represents how strongly an instance is a candidate for being linked to the mention z. For example, $p_i^{(z)}$ (the i-th component of $\boldsymbol{p}^{(z)}$) models the probability of the instance i to be a link candidate accordingly to the first hypothesis. Then, the three hypotheses are combined together into a final vector $\boldsymbol{p}$. A visual representation of this combination is shown in Figure 4.4.

String-match hypothesis. The first vector, $\boldsymbol{p}^{(z)}$, named STRING-MATCH HYPOTHESIS, contains the activation scores of the n instances given the string z. It models the idea that the *surface form* of z is a strong indicator for spotting candidate links. Intuitively, the degree of ambiguity of a mention is generally limited to few possible instances (e.g. there are only few individuals that are referred with the noun *Clyde*). Formally,

$$\boldsymbol{p}^{(z)} = \sigma\left(M_{h(z,\mathcal{H})}\right) , \tag{4.4}$$

where the function $h(z, \mathcal{H})$ returns the index of z in $\mathcal{H}$, $\sigma(\cdot)$ is a sigmoidal function that operates element-wise on its input (yielding output values in $(0, 1)$) and the subscript after the matrix M indicates the matrix row.

Embedding-match hypothesis. The second hypothesis, $\boldsymbol{p}^{(e)}$, named EMBEDDING-MATCH HYPOTHESIS, collects the activations of the instances given the embedding $\boldsymbol{e}^m$ of z. The rationale behind it is that if $\boldsymbol{e}^m$ is similar to an embedding of a known mention (in the sense of the cosine similarity $\cos(\cdot,\cdot)$), then it is likely to activate the same instances. Due to the way our embeddings are computed (Section 4.3.2), we expect that two embeddings are "close" if they have similar roles in the processed sentences (semantic-similarity, synonymy), and similar morphological properties (due to the character-level input). Formally:

$$\boldsymbol{p}^{(e)} = \left(\left[\frac{\cos(\boldsymbol{e}^m, E_i) + 1}{\sum_{j=1}^{m} \cos(\boldsymbol{e}^m, E_j) + m} \right]_{i=1}^{m} \right)' \cdot \sigma(M) \tag{4.5}$$

where the notation $[v_i]_{i=1}^m$ indicates the (column) vector $[v_1, \ldots, v_m]$. Notice that $\sigma(M)$ is a matrix of the same size of M, and $\boldsymbol{p}^{(e)}$ involves a vector-times-matrix operation that basically computes a weighted sum of the rows of M accordingly to the similarity between $\boldsymbol{e}^m$ and the stored embeddings[1]. We can consider the embedding-match hypothesis $\boldsymbol{p}^{(e)}$ as a smooth version of string-match $\boldsymbol{p}^{(z)}$.

Temporal hypothesis. The third hypothesis, $\boldsymbol{p}^{(t)}$, named TEMPORAL HYPOTHESIS, implements the idea that recently disambiguated instances are good candidates for coreference resolution (temporal locality). In other words, if a story is talking about a given entity, it is likely that the narration will make references to it using some new surface forms. For example, given an entity labeled *Donald Trump*, there could be ambiguous mentions like *Donald*, *Mr. Trump* (other people called this way), or *the president* (that cannot be captured by the other two hypotheses). In these cases, the temporal locality has a crucial role, that is even more evident when using pronouns, that are shared by several instances. Formally, we have

$$\boldsymbol{p}^{(t)} = \frac{[u(i, \mathcal{T})]_{i=1}^{n}}{\max [u(j, \mathcal{T})]_{j=1}^{n}}, \tag{4.6}$$

where $u(j, \mathcal{T})$ returns the number of occurrences of instance j in $\mathcal{T}$.

Candidate generator. The CANDIDATE GENERATOR merges the three hypotheses and combines them into a single one FINAL HYPOTHESIS $\boldsymbol{p}$ defined in Equation 4.7. Instead of a simple average, we combine them such that we give more priority to $\boldsymbol{p}^{(z)}$ than to $\boldsymbol{p}^{(e)}$ when the former is strongly activated (since $\boldsymbol{p}^{(z)}$ is about "exact" matches in terms of surface forms). Moreover, The importance of the temporal component $\boldsymbol{p}^{(t)}$ in the hypothesis formulation depends on the current mention z. For example,

[1] In our implementation, we kept only the top-k cosine similarities ($k = 5$), forcing the other ones to -1.

if z is a pronoun, the system must trust $\boldsymbol{p}^{(t)}$ more than the others, while, in case of some unambiguous mentions, it must consider less $\boldsymbol{p}^{(t)}$. We let the system learn the importance of $\boldsymbol{p}^{(t)}$ depending on the mention embedding $\boldsymbol{e}^m$. Formally, the system computes $\gamma = q(\boldsymbol{e}^m) \in [0, 1]$, where $q(\cdot)$ is a learnable function (whose form will be defined shortly), and the vector of merged hypotheses is

$$\boldsymbol{p} = (1 - \gamma) \cdot \left(\boldsymbol{p}^{(z)} + \left(1 - \boldsymbol{p}^{(z)} \right) \boldsymbol{p}^{(e)} \right) + \gamma \cdot \boldsymbol{p}^{(t)} \, . \tag{4.7}$$

The vector $\boldsymbol{p}$ contains a set of scores that model how strongly each KB instance is related to the current input mention. We kept the model as simple as possible by considering only the three hypotheses that were necessary to cover all the ambiguities present in the task at hand. However any number of hypotheses can be attached to the candidate generation module, making our model adaptable to different NLP problems.

4.3.4 Disambiguation

While the candidate generation routine only focuses on the mention z, the Disambiguator (Figure 4.2, fourth block) is responsible for determining what are the most likely KB instances given the context of z. The representation of the context $\hat{\boldsymbol{e}}^m$ is computed by the Encoder (Section 4.3.2) by Equation 4.3. The Disambiguator is based on the functions $d_i(\hat{\boldsymbol{e}}^m)$, $i = 1, \ldots, n$, also referred to as *disambiguation units*. In details, each d_i is associated to a KB instance, and it is learned while reading the text stream in a supervised or unsupervised way, as we will describe in Section 4.4. In particular, in the considered problem, we do not have the use of any discriminative information: when we receive the supervision that the input mention is about a certain KB instance (or when the model decides this in an unsupervised manner), we cannot infer that the context of the considered mention is not compatible with any other KB instance. As a matter of fact, the same context may be shared by several instances, so each $d_i(\cdot)$ must have the capability of learning from positive examples only. For this reason we implement d_i with a locally supported similarity measure,

$$d_i(\hat{\boldsymbol{e}}^m) = \frac{1}{2} + \frac{1}{2} \max_{j=1,\ldots,\kappa} \cos(\hat{\boldsymbol{e}}^m, \boldsymbol{w}_{ij}) \in [0, 1] \, , \tag{4.8}$$

that models the distribution of the contexts for instance i by means of κ centroids $\{\boldsymbol{w}_{ij}\}$. As we will describe Section 4.4, these centroids are developed in an online manner, and, in the unsupervised case, we end up in an instance of online spherical K-Means. Even the previously mentioned function $q(\cdot)$ needs to be locally supported (for the same reasons), so we implemented it following Equation 4.8 as well.

We combine the activation of the candidates (i.e., the vectors $\boldsymbol{p}$), and the disambiguation-unit outputs, $\boldsymbol{d} = [d_i(\hat{\boldsymbol{e}})]_{i=1}^n$, to get the output $\boldsymbol{o}$ of the system,

$$\boldsymbol{o} = \delta(\boldsymbol{p} > \tau_r) \cdot (\eta \cdot \boldsymbol{p} + (1 - \eta) \cdot \boldsymbol{d}) \, , \tag{4.9}$$

where δ returns a binary vector with 1's in those positions for which the (element-wise-evaluated) condition in bracket is true. The scalar $\tau_r > 0$ is a *reject* threshold, and $\eta \in [0, 1]$ is a tunable parameter that controls the role of the hypothesis $\boldsymbol{p}$ in the decision process[2]. The reject threshold allows us to avoid computing the $d_i(\cdot)$ associated to very-low-probability candidate instances.

Formally, the mention z is linked to the most-likely instance y as

$$\boldsymbol{o} = \phi(z, s), \quad y = \arg\max(\boldsymbol{o}) \,, \tag{4.10}$$

where ϕ is the function that computes the affinity scores of z with respect to all the KB instances, given the context of the current sentence s. Here we replace the second argument of ϕ by the sequence of mentions detected in s, excluding z itself, i.e., $\phi(z, \{z_j : z_j \in s, z_j \neq z\})$.

4.4 Online Learning Dynamics

The agent learns while reading the data from the text stream in an online fashion. Before going into further details, we recall that the learnable parameters of the proposed model are the matrix M in the memory component, the vectors $\boldsymbol{w}_{ij}$ of the disambiguation units and of the temporal relevance function q (i.e., the parameters of the Candidate Generator and of the Disambiguator), and the weights of the LSTMs in the Mention Detector and in the Encoder. The system starts with an empty KB (so M is not allocated yet), and with randomly initialized model parameters. As already outlined in Subsections 4.3.1 and 4.3.2, we can pre-train those modules that constitute the preliminary stages in the system pipeline of Figure 4.2, i.e., the Mention Detector first, and then the Encoder, in both cases, without human annotations. In other words, the system will start reading data from the text stream and it will progressively acquire the skill of detecting mentions to entities and relations, and the skill of encoding such mentions and their context. When the agent has robust enough detector and encoders, the system can start to develop also the KB-based disambiguator, and to eventually refine and improve the pre-trained modules.

While processing the text stream, accordingly to Equation 4.10, each detected mention z is associated to a disambiguated KB instance y. Before starting the disambiguation, the system verifies if z is already in $\mathcal{H}$. If it is not the case, then z is included in $\mathcal{H}$, its embedding $\boldsymbol{e}^m$ is appended to E, and a new row is added to M (with values such that $\sigma(M_{h(z,\mathcal{H})}) \approx 0$). The learning stage consists in an online process to optimize the model parameters accordingly to either self-learning or a supervision about the target instance $\overline{y}$. A sketch of the whole learning stage is shown in Algorithm 1.

[2]For simplicity, we set $\eta = 0.5$.

```
foreach mention in the current sentence do
    if supervision not provided then
        if recognized some instances then
            reinforce the associated disambiguation units
        end
        else if uncertain disambiguation then
            no actions
        end
        else if no recognized instances then
            create new instance
            reinforce the new disambiguation unit until > τ_a
        end
    end
    else
        if already known supervision then
            reinforce the associated disambiguation unit until > τ_a
            penalize the other disambiguation units until < τ_a
        end
        else
            disambiguate instance y
            if y was associated to another supervision then
                create new instance
                reinforce the new disambiguation unit until > τ_a
            end
            else
                associate supervision to the y-th disambiguation unit
                reinforce the associated disambiguation unit until > τ_a
                penalize the other disambiguation units until < τ_a
            end
        end
    end
end
```

Algorithm 1: Learning Dynamics

Self-Learning. When no supervision is provided, the learning dynamics changes in function on the confidence that the system yields in formulating hypotheses ($\boldsymbol{p}$) and in disambiguating the mention z ($\boldsymbol{o}$). We distinguish among three cases:

i. $\max \boldsymbol{o} \geq \tau_a$: RECOGNIZED SOME INSTANCES

ii. $\max \boldsymbol{p} > \tau_r \wedge \max \boldsymbol{o} < \tau_a$: UNCERTAINTY

iii. $\max \boldsymbol{p} \leq \tau_r$: UNKNOWN INSTANCE ,

where τ_r is the aforementioned *reject* threshold, and $\tau_a \in (\tau_r, 1)$ is an *accept* threshold.[3]

Case *i*. The response of the system has been rather strong in indicating at least one instance, therefore the prediction of the model is considered reliable enough, such

[3]We set $\tau_r = 0.1$ and $\tau_a = 0.9$.

that the decision must be reinforced for all those disambiguation units that generated outputs in $\boldsymbol{o}$ above the *accept* threshold τ_a. This is done in a self-learning fashion (Nigam and Ghani (2000)), by means of a single online gradient-based update, with the aim of minimizing the quadratic loss that measures the distance between the selected disambiguation unit outputs (indexed by j) and 1, that is $\sum_j (d_j - 1)^2$ (i.e., we reinforce the selected outputs).

Case *ii*. The system activates some candidates but it is uncertain in the disambiguation, so no further actions are taken.

Case *iii*. This case is triggered when $\boldsymbol{p}$ is composed of only low-confidence candidate activations. This situation happens when the candidate generation module does not find a known instance that is compatible with the current mention, that is likely to indicate the occurrence of a new entity/relation. Therefore, the system creates a new instance in the KB and reinforces its disambiguation unit until its response is above τ_a, to develop the new instance model (i.e., multiple gradient-based updates).

Supervision. When a supervision $\overline{y} \in \overline{\mathcal{Y}}$ is provided for the mention z, we want the system to immediately learn from it. The system keeps track of the mapping between the set of user-provided supervisions $\overline{\mathcal{Y}}$ and the set of instances in the memory components (the user is not aware of how the system handles instances). When $\overline{y}$ is a known supervision, the index of the corresponding instance is found, and the output of the disambiguation unit associated to this instance is reinforced until it is greater than the accept threshold τ_a. We also push the output of the other disambiguation units towards 0, by minimizing the quadratic loss, $\sum_j (d_j)^2$. This implements a penalization process, that involves multiple gradient-based updates until all the involved outputs fall below τ_a[4]. When $\overline{y}$ is a never-seen-before supervision for the system, then it is associated with the disambiguated instance y. On the other hand, if y was previously associated to another supervision symbol in $\overline{\mathcal{Y}}$ different from $\overline{y}$, then we have a collision in the mapping and we solve it by creating a new instance and by associating it to $\overline{y}$. Then, we follow the same steps as in case *iii* above[5].

[4]Notice that, whenever the system needs to reinforce the j-th output o_j, and it also holds that $p_j \leq \tau_r$, then, due to $\delta(\cdot)$ in Equation (4.9), we get no gradient, so we first increase p_j until it is above τ_r.

[5]Supervisions may not only be related to the instance-label of the detected mentions, but they can also be associated with the detection of the mention. For example, the user could label a mention that does not correspond with the detected ones. This supervision signal is propagated to the mention detector, that can be refined and improved. In turn, the mention and context encoders could be refined as well. The investigation of these refinement procedures goes beyond the scope of this proposal.

4.5 Experiments

4.5.1 Datasets

The experimental campaign is carried out on two different datasets: **Simple Story Dataset** and **WikiFacts**. In the following, we describe both in detail.

Simple Story Dataset

We introduce a new synthetic dataset[6], specifically designed to emulate the problem setting described in Section 4.2, because any of the existing benchmarks, albeit somewhat related, do not really fit the defined setup. The dataset is organized in stories and presented to the system as a stream of sentences, similarly to what shown in Figure 4.1. A story is a list of not-repeated facts (mostly) about a certain entity, that we call "main entity". Entities and relations are mentioned multiple times, in different stories and with different surface forms, including synonyms, nicknames, co-references, etc. Overall, we collected 564 different stories resulting in a stream of $10,000$ sentences. In the dataset there are 130 entities and 27 relations, belonging to a pre-defined ontology. The aforementioned ontology, illustrated in Figure 4.5 has 21 and 28 entity and relation types that allows us to build the facts that constitute the stories. Each fact is a triple of mentions, two entities connected by a relation. We automatically generate sentences from facts. Entity and relation instances are substituted with their textual forms, that are occasionally perturbed with different kind of noise, such as character-level perturbations to emulate typos. We also introduce noise in the stories, injecting some facts not related to the main entity, so that the narration slightly departs from the main topic.

We end up having a dataset consisting of a word dictionary of $2,176$ elements, $1,526$ and 288 distinct mentions to entities and relations, respectively, including different variations (due to typos, determiners, etc.) from an original set of 354 base entity mentions and of 176 base relation mentions. The stories are also rich of co-references ($7,975$), and there are $6,830$ mention occurrences that are ambiguous, i.e. that refer to multiple entity/relation instances.

WikiFacts

The WikiFacts dataset was originally proposed in (Ahn et al. (2016)) for knowledge-aware language modeling. The dataset is a collection of Wikipedia summaries, weakly aligned with triples from Freebase. Each summary contains a textual description of a certain entity. Anytime such entity is mentioned in the text, the mention is annotated with its Freebase identifier that can be exploited to get facts involving such entity.

[6]The dataset is publicly available at `https://sailab.diism.unisi.it/stream-of-stories/`.

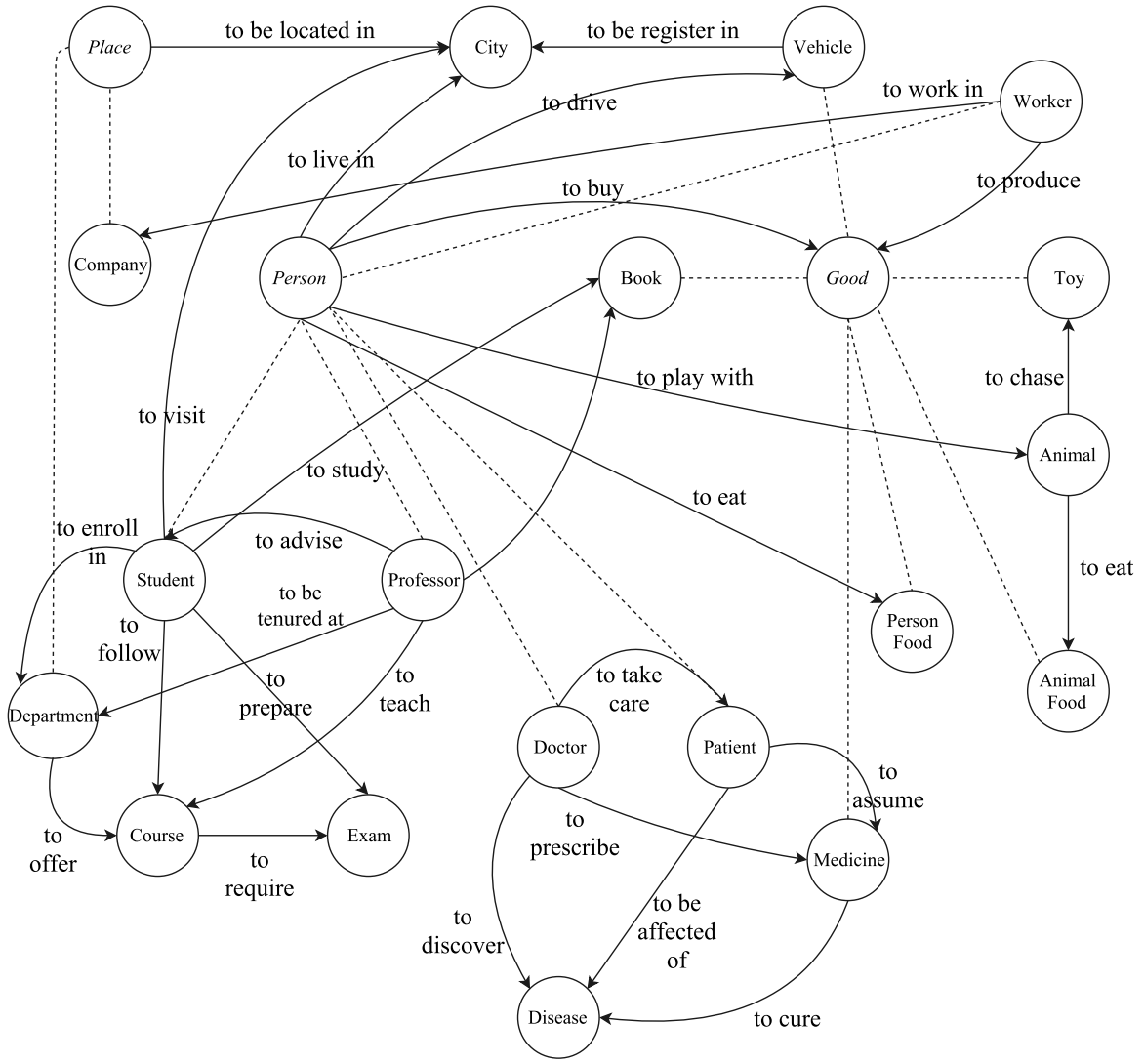

Figure 4.5: Ontology Graph of entity (nodes) and relation (edges) types in the Simple Story Dataset. Dashed lines between node pairs indicate a hierarchy between the two types.

We adapt WikiFacts to the problem setting of Section 4.2. The dataset is composed of 10,000 pages that include about 560,000 entities. Each summary is a story. We consider only a portion of WikiFacts, precisely 1,112 wiki pages in order to have about 10,000 facts. We kept only pages with at least three sentences. A sentence must include two or more entities and we distantly supervise the text within two consecutive entities as relation, in the case we can identify a relation involving those two entities in Freebase. Because this alignment between relations and textual spans is manual and not entirely reliable, we only evaluate performances on entities in this dataset. Moreover, despite being made of real world data, entities are rarely repeated across different stories, making the data significantly less ambiguous. In-

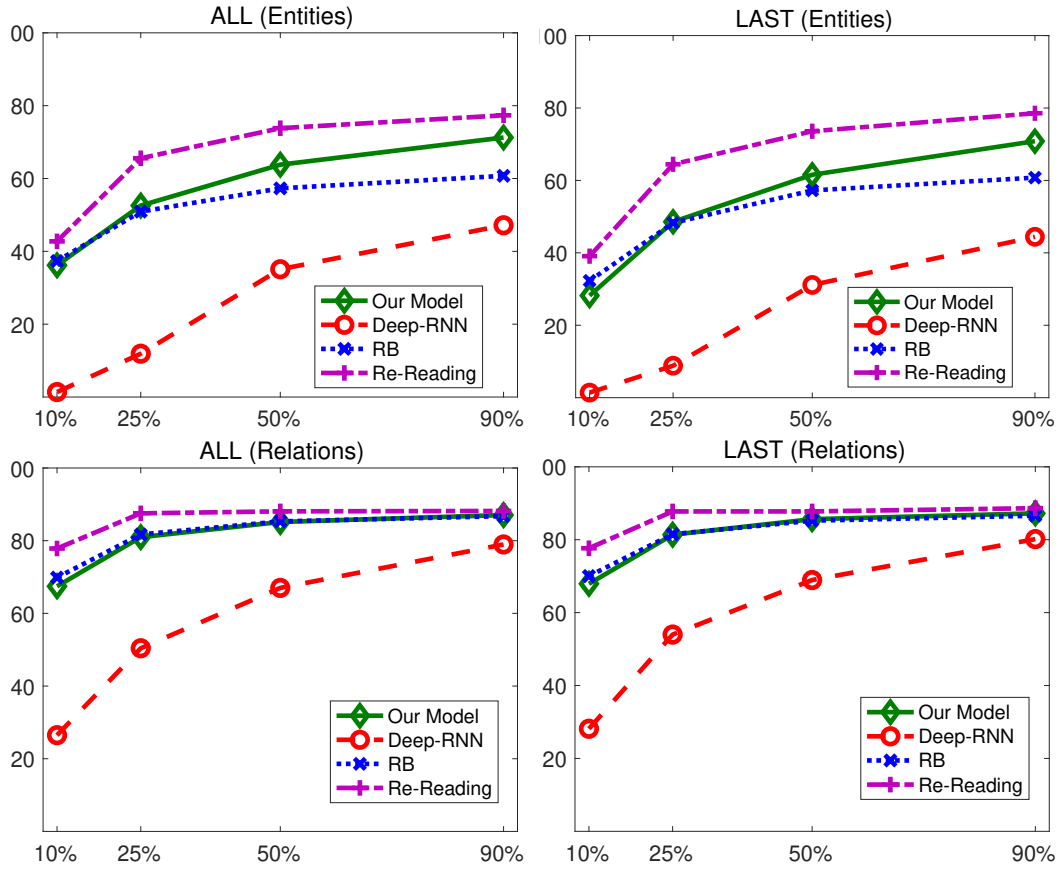

Figure 4.6: Accuracy (%) for different amounts of supervision in the Simple Story Dataset, in the case of entities (two topmost graphs), and relations (in the bottom). We include two competitors (*Deep-RNN*, *RB*), our system, and our system reading the stream again (*Re-Reading*).

Table 4.1: Accuracy (%) for different amounts of supervision in the Wikifacts Dataset.

	Model	10%	25%	50%	90%
All	RB	16.84	40.44	48.28	49.55
	Deep-RNN	0.6	3.01	12.34	21.78
	Our Model	39.25	**54.57**	**69.64**	**75.45**
	Re-Reading	**44.75**	**54.66**	66.88	70.55
Last	RB	17.28	40.87	48.04	49.37
	Deep-RNN	0.6	3.25	12.11	21.37
	Our Model	37.44	52.93	**67.45**	**75.37**
	Re-Reading	**43.41**	**53.38**	65.13	70.39

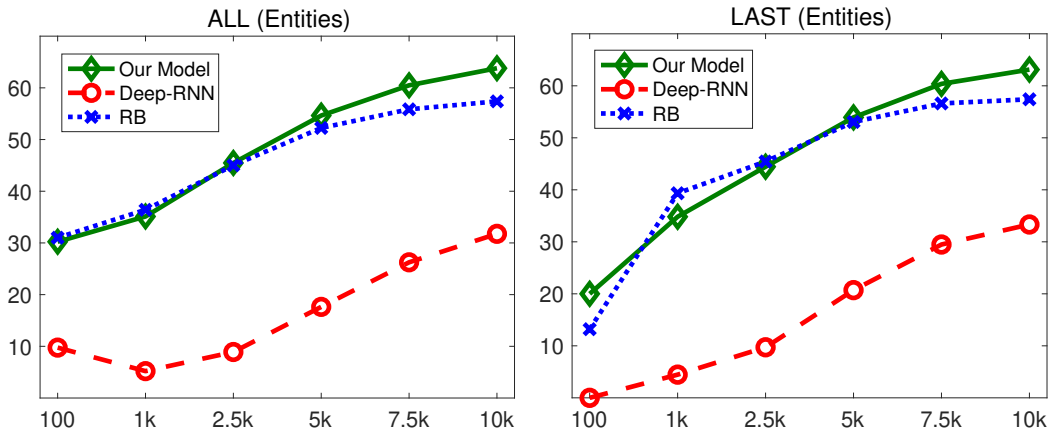

Figure 4.7: Accuracy (%) on entities at different time instants in the Simple Story Dataset (with 50% of supervised sentences).

deed, in this dataset there are 4,431 entities in 10,000 facts, about 34 times more instances than in the simple story dataset.

4.5.2 Learning Settings

Each story is split into two parts, separating supervised from unsupervised sentences. The supervised portion covers the first sentences of the story. Depending on the experimental setting, the percentage of supervised sentences varies; in particular, we considered four setups: 10%, 25%, 50% and 90% of the sentences in the story. The system reads the data, receives supervisions, and makes predictions on the unsupervised sentences, accordingly to the stream order.

Evaluation. The accuracy of a prediction is measured at the moment the prediction is made. We use cluster purity measure, as in (Manning et al. (2008)), to map unsupervised outputs to the ground truths, where, in case of conflicting assignments, we only keep the mappings that are determined using the largest statistics. The map used to convert predictions is computed with the statistics accumulated up to the time the prediction is made. We define two scores, namely LAST and ALL, used for the evaluation of the unsupervised sentences. ALL computes the average accuracy for all the unsupervised sentences of a story, thus its value depends on the portion of supervised sentences. LAST instead measures the average accuracy on a single sentence, which allows us to compare performances across different percentages of supervised sentences. Both are averaged over each story first, and, then, over the whole set of stories.

Model Details. The model parameters were validated on a small out-of-sample dataset. The investigation regarded the number k of centers of the locally supported functions (disambiguation units). The number of centers for the locally supported function has been chosen low ($k = 10$) for entities, since they appear in a limited number of contexts, whereas it has been chosen higher for relations ($k = 50$) since their contexts are much more variable. The memory size for entities and relations were set up respectively to $2,500$ and $1,500$, while we chose 10 as the maximum number of recent instances ($|\mathcal{T}|$) for both. The **mention detector** reads the input stream of text word by word, processing words as sequences of characters. The character embedding matrix has dimensions 100×50, where 100 is the vocabulary cardinality and 50 the character embedding size. Both bi-LSTMs have an hidden state of size $500(\times 2)$. The final output layer is composed of 6 sigmoidal units. Concerning the **mention and context encoder** the same parameters of the mention detector were chosen for the character embedding matrix. The size of mention contextual encodings has been set to 300 (2×150) to match the commonly used size of Word2Vec embeddings. The decoder, exploited in the pre-training phase, was composed of an LSTM having a hidden state of 300 neurons and a final softmax output layer over the character vocabulary (i.e. 100 output units).

Training details. Our model was bootstrapped accordingly to the scheme of Section 4.4. In particular, before streaming the stories, we generated a stream of text composed of simple language sentences, taken from Simple English Wikipedia and the Children's Book Test (CBT) data[7], getting overall a total of 1.5 million sentences. We automatically generated supervisions for the mention detector, accordingly to the procedure described in Section 4.3.1, and we processed the stream. We randomly injected typos with probability $5 \cdot 10^{-3}$ to any input character in order to make the model be more robust to this kind of perturbations. Moreover, we truncated words with more than 15 characters and we also limited the maximum length of a sentence to 20 words. During pre-training we considered a batch size of 256 sequences. Afterwards, we stopped updating the mention detector and we streamed the same data again to allow the system to develop the mention and context encoders (Section 4.3.2). In this case we set the maximum length of an input segment and the maximum sequence length to 50 and 10, respectively. Finally, we stopped updating the encoder and we started to stream the stories.

4.5.3 Competitors

We compared our model with two competitors and we also provide a variant of the model.

[7] `https://simple.wikipedia.org`, `https://research.fb.com/downloads/babi/`

Deep-RNN. *Deep-RNN*, is a deep neural architecture, where bottom layers are analogous to the ENCODER module of the agent, thus they have the same character-aware model described in Chapter 3, already bootstrapped as for our model. In order to classify mentions, the *Deep-RNN* architecture uses an MLP (1 hidden layer with 600 units and tanh activation; softmax activation in the output layer) on top of the concatenated representation $[\boldsymbol{e}^m, \boldsymbol{e}^{\hat{m}}]$ (Equation 4.2, Equation 4.3). This network is aware in advance of the number of instances present in the datasets, that is the size of the output layer of the MLP, hence it does not incur in errors related to the self-discovering of new instances. We followed a classic online supervised learning scheme, where a single gradient-based update is performed for each processed sentence, otherwise we experimented that the network simply overfits the last supervisions and forgets about the others. In few words, *Deep-RNN* is a simplified version of our system that is adapted for online-learning in a standard Machine-Learning setup.

Rule-based model. We also devised a very informed *rule-based model*, *RB*, that buffers statistics on the supervisions received up to time t. Given an input mention, *RB* predicts the most-common supervision received when encountering it. When never-supervised-before mentions are encountered, *RB* predicts the most-frequent supervision of the story, that is likely to be the main entity of the story itself. This information is very precious (our model has no access to it), since several co-references refer to the main entity. We tested a large number of different rule-based classifiers, and *RB* was the one obtaining the best results. These heuristics are very effective since they are shaped upon prior knowledge about the dataset construction, which provides strong biases.

Re-Reading. We also report a variant of our model referred as "*Re-Reading*". We let the system read the whole stream another time, without providing supervisions again, that is where the self-learning skills are most emphasized.

4.5.4 Results

We summarize the results on the dataset described in Subsection 4.5.1 and compare the system with the methods discussed in Subsection 4.5.3. Figure 4.6 reports the accuracy of discovering and disambiguating mentions to entity and relation instances, in the Simple Story Dataset, while Table 4.1 shows the accuracy in the case of the Wikifacts dataset (entities).

Our system outperforms the competitors in all our experiments. Differently from *Deep-RNN*, our model exploits its capability of building local models of the instances, while *Deep-RNN* is not able to capture this locality, either not-learning

enough or overfitting supervisions. Moreover, *Deep-RNN* has difficulties in storing information on the whole story, as shown in the LAST case. When very few supervisions are provided, in the case of entities, *RB* shows an accuracy that is similar to our system on the Simple Story Dataset. Differently, the proposed model significantly outperforms RB on WikiFacts, showing interesting generalization capabilities on real world data. In the case of relations, *RB* is not different from our system, mostly due to the fact that relations have poor ambiguity in the Simple Story Dataset, also confirmed by the improved results of the *Deep-RNN*. The re-reading variant of our model shows better classification capabilities in most of the cases, comparing favourably with all the other approaches. This property is even more evident in the few-supervision settings. These results indicate that, despite the dynamic nature of the online problem we face, the proposed model has strong memorization skills, and the capability of improving its confidence by self-learning.

Model Evolution. In the Simple Story Dataset, we also investigate the evolution of the model, evaluating its performances at different time instants, using 50% of the supervisions (entities). Basically, we paused the system at a certain point of the stream, measured the accuracy, and activated the system again. This process is repeated until the end of the stream is reached. Results are reported in Figure 4.7. Our model reaches better results than *RB* after having read roughly the $20 - 25\%$ of the input stream. We analyzed this result, and it turned out that this is mostly due to the fact that our system takes some time to learn how to handle the temporal information ($\boldsymbol{p}^{(t)}$) needed to resolve co-references. As a matter of fact, co-reference resolution clearly depends on the number of supervisions, as reported in Table 4.2.

Table 4.2: Accuracy (%) in the case of pronouns.

Supervision	10%	25%	50%	90%
ALL	12.74	36.04	43.61	53.55
LAST	8.57	42.85	44.76	54.28

The role of γ. We also analyzed the values of γ, that weighs the importance of the temporal locality in the disambiguation process. In the case of pronouns the average value of γ is 0.33, while in the case of the other mentions (that might include some other co-references different from pronouns) is 0.21, thus showing that the system learns to give more importance to the temporal component when dealing with pronouns than other mentions. This is confirmed by the low instance-activation scores in the case of mentions that are pronouns (the average of max $\boldsymbol{p}^{(z)}$ is 0.005, remarking their inherent ambiguity), with respect to the ones of the other mentions (average of max $\boldsymbol{p}^{(z)}$ is 0.5).

Table 4.3: Accuracy (%) of our Full Model, of a system based on Word-Level Encoding (WL Enc.), and of a system not-provided with information on the recently disambiguated instances (No Recent).

	Entities		**Relations**	
	ALL	LAST	ALL	LAST
WL Enc.	47.39	46.18	84.33	85.10
No Recent	55.56	54.95	–	–
Full Model	**63.79**	**63.12**	**85.41**	**85.81**

4.5.5 Ablation Study

We analyse two variants of the model in order to emphasize the role of some components, and report the results in Table 4.3 for the Simple Story Dataset.

Character-level encoding. To understand the benefits brought by the character-level encoding for mentions, we compare it with respect to classical word-level embedding approaches. To this end, we built a vocabulary of words, including all the correct-spelled words of our dataset (and an out-of-vocabulary token), and we limited the character-based encoder to such words, thus simulating a word-level encoder. We call this variant as Word-Level Encoding (WL Enc.). Our model is able to encode in a meaningful way those words that contain typos, and to exploit them in context encoding, while the word-level encoder faces several out-of-vocabulary words, that also create ambiguity while comparing contexts. Once more, the importance of sub-word level information embedded thanks to our character-aware model (Chapter 3) is proven to be impactful in language understanding problems.

Impact of the temporal hypothesis. Another aspect that we investigate is the contribute of the temporal hypothesis $\boldsymbol{p}^{(t)}$ when disambiguating entities. Hence, we consider a version of our model that considers only the hypotheses $\boldsymbol{p}^{(z)}$ and $\boldsymbol{p}^{(e)}$. Please note that this variant is equivalent to setting γ always equal to zero in Equation 4.7. We name this version as No Recent. Table 4.3 shows that the temporal locality has an important role, and disabling it degrades the performances. This is not only due to its positive effects in co-reference resolution, but also when disambiguating mentions to the main entity of the story. Finally, thanks to $\boldsymbol{p}^{(t)}$, the system learns to develop the tendency of associating new mentions to already existing instances, instead of creating new ones, that is an inherent feature of each story (in the worst case it creates only 183 entity instances).

4.5.6 Dealing with Long Text Streams

The proposed model explicitly memorizes the information about entities and relations of the current and past stories into distinct (i.e., independent) memory locations (Section 4.3.3). This strongly alleviates the catastrophic forgetting problem (McCloskey and Cohen (1989)), that is usually due to memory interference among multiple entities or relations (e.g., due to weight sharing). The experimental analysis of Section 4.5.4, where the same stream is read a second time ("Re-Reading"), suggests that the proposed system does not incur into serious catastrophic forgetting issues, since its recognition accuracy improves once reading again the text stream, without additional supervisions (Figure 4.6). Moreover, even if the encoding module of Section 4.3.2 is based on RNNs, we are not exposed to the problem of learning long-term dependencies with such RNNs (Bengio et al. (1994)). As a matter of fact, the RNNs are only responsible of encoding each mention and its context within a single sentence, whose usual length is efficiently manageable with LSTMs without any long-term dependency problems.

We performed a further experimental analysis aimed at evaluating the memorization skills of the system when dealing with long text streams. In particular, we provided the system an incremental number of stories, and we periodically evaluated its capability of correctly recognizing the entities of the first story of the stream. As long as the system reads more data, this procedure provides a clear indication of the persistence of the system memory. However, the system is still exposed to ambiguity issues, since multiple instances might share the same mention. For this reason, we considered different experimental settings with different levels of ambiguity.

In detail, we considered four scenarios in the Simple Story Dataset, referred to as A, B, C, D, respectively. In each of them, we generated 10 different long streams ($\approx 2,600$ sentences in each stream), randomly shuffling the involved stories. For each stream, we selected the first story as *target* story. The model reads the stream and learns in an online manner, receiving supervisions about the mentioned entities, and constantly adapting its internal parameters as in the experiments of Section 4.5.4. After having processed t stories, we temporarily freeze the whole system, and we process the target story again, measuring the prediction accuracy on the entities of this story. By increasing the number t of intermediate stories, we evaluate how the systems keeps memory of the target story as long as the stream grows.

The four settings differ in how the stream has been constructed, and they introduce an increasing level of ambiguity. In the first setting (A), intermediate stories in the stream neither contain entities nor mentions belonging to the target story. Reading the stream does not introduce any ambiguity in the contents of the target story. This allows us to evaluate whether the system is forgetting information as long as time passes, completely discarding ambiguity issues. In setting B we include pronouns in the intermediate stories. Mentions related to pronouns are obviously very

Table 4.4: Average recognition accuracy (on 10 different streams) of the entities of target stories, with an increasing number t of intermediate stories for settings A, B, C, D (Acc_A, Acc_B, Acc_C, Acc_D), and average number of entities per mention (degree of ambiguity Amb_A, Amb_B, Amb_C, Amb_D).

	$t = 0$	$t = 10$	$t = 50$	$t = 100$	$t = 150$
Amb_A	1.03	1.03	1.03	1.03	1.03
Amb_B	1.05	1.42	2.14	2.42	2.65
Amb_C	1.04	1.50	2.49	3.28	3.64
Amb_D	1.03	1.83	3.48	4.54	5.06
Acc_A	98.41	98.41	98.41	98.41	98.41
Acc_B	99.62	96.88	92.03	94.03	94.03
Acc_C	98.89	85.05	72.25	70.30	71.85
Acc_D	97.98	85.83	71.32	65.93	63.25

ambiguous, hence the model will have to rely on the temporal hypothesis (Section 4.3.3) to be able to disambiguate them. The intermediate stories of setting C can contain mentions that are also used in the target story, and these mentions may or may not refer to the same entities of the target story. In this case, the system must not only remember what was read in the target story, but it must also gain stronger skills in disambiguating entities. Finally, in the last scenario (D), we ensure that the intermediate stories never refer to the entities of the target story. However, we still allow the stream to use mentions that are shared with the ones that are used in the target story, simulating a strong data drift.

We explicitly computed the degree of ambiguity of each stream for all the settings (referred to as Amb_A, Amb_B, Amb_C, Amb_D), that is the average number of entities associated to each mention of the target story up to step t. Table 4.4 reports both the degrees of ambiguity and the recognition accuracy of the entities of the target story (averaged over the 10 generated streams). We report the results after $t = 0$, 10, 50, 100, 150 intermediate stories (0, 180, 880, 1700, 2600 intermediate sentences, respectively).

All the four settings show an accuracy close to 100% at $t = 0$, as expected, where the few errors are mostly due to pronouns, since the system has not developed strong co-reference skills, as expected. For $t > 0$, results in setting A confirm that the model does not forget past information when processing text streams that do not interfere with it. Setting B shows an interesting behaviour, in which accuracy is reduced as long as t grows up to 50, and then it increases again. This indicates that while at the beginning the system misclassifies pronouns in the target story, it is able to improve pronoun resolution while reading several stories, thus adjusting the temporal hypothesis of Equation 4.6. In both settings C and D, as expected, results

indicate that performances degrade as the number of intermediate stories increases, due to the increasing number of ambiguous mentions. The model clearly performs better in setting *C* than in setting *D*, remarking the positive effects of refining the disambiguation units while reading information about the entities of the target story in multiple occasions. Finally, it is worth noticing that the degradation of the performances is directly related to the degree of ambiguity in the streams. This is a clear hint that long streams increase the difficulty of the task not due to their length but to their intrinsic ambiguity.

4.6 Discussion

We presented an end-to-end model to process text streams, where mentions to entities and relations are disambiguated and eventually added to an internal KB, that, differently from many existing works, is not-given-in-advance. Our model is capable of performing one-shot learning, self-learning, and it learns to resolve co-references. It has shown strong disambiguation and discovery skills when tested on a stream of sentences organized into small stories (we also created a new dataset that we publicly made available for further studies), even when a few, sparse supervisions are provided. We also showed how it can improve its skills by continuously reading text.

The proposed model has the potentiality of being enhanced by introducing a more structured and advanced knowledge components (types, facts, abstract notions for higher level inference - such as logic formulas), providing a powerful bridge between sub-symbolic and symbolic approaches. A knowledge-grounded model maybe also important to make language generation more controllable. We will discuss these aspects in details in Section 7.2.

Chapter 5

Natural Language Generation

Poem Generation and Paraphrasing are two very different Natural Language Generation problems. The former aims at the free generation of verses that follow precise rhyming schemes and meter rules, while conveying emotions and/or narrating events. Instead, paraphrasing is the task of rewriting a text passage without altering its meaning. Although we have shown in Section 2.3 that NLG problems can all be framed as particular instances of Language Models, the nature of these two tasks is unalike. Referring again to Section 2.3, we can categorize Poem Generation into the class of *open-ended* problems, whereas Paraphrasing falls in the *non-open-ended* domain. However, in this Chapter we address an issue that is common for both problems: the lack of resources. In Poem generation data is inherently poor, since it involves ancient manuscripts that are rare, and not enough to train a robust language model. Regarding paraphrasing, the lack of aligned text pairs of paraphrases occurs especially in non-English languages, such as Italian.

This Chapter presents two distinct solutions to cope with low-resources. After reviewing the literature in Section 5.1, we introduce a novel syllable-based language model trained in a multi-stage transfer learning fashion to acquire knowledge from large corpora of modern languages (Section 5.2). In Section 5.3, we propose an algorithm to create a corpus for paraphrasing, that we exploit to create a dataset of aligned text pairs for Italian. Then we make use of such data to train a Pointer Generator model to generate paraphrases.

5.1 Related Works

Natural Language Generation (NLG) has seen a steady increase of attention in the last few years inside the NLP field. We focus in this Section only on the works most related to the two problems tackled in the Chapter, i.e. Poem Generation and Paraphrasing. Due to their very different nature, we discuss separately their related works.

Poem Generation

The scientific literature includes several works on machines that are either programmed to generate poems or that approach the problem of poem generation using machine learning algorithms. Early methods (Colton et al. (2012)) rely on rule-based solutions, while more recent approaches focus on learnable language models. As a matter of fact, nowadays, most of NLG approaches are based on recurrent nets (Wen et al. (2015); Chopra et al. (2016); Hasan et al. (2016)).

Word-based language models usually require large vocabularies to store all the (most frequent) words in huge textual corpora, and, of course, they cannot generalize to never-seen-before words. Some other approaches tried to overcome this issue, exploiting sub-word information. A character-level solution was proposed in (Hwang and Sung (2017)), while other authors (Miyamoto and Cho (2016)) combine word embeddings with character-level representations. It has been shown (Marra et al. (2018); Akbik et al. (2018)) that character-based models can be adapted to produce powerful word and even context representations, that capture both morphology and semantics. Sub-word information is very important in poetry, since it represents a crucial element to capture the "form" of a poem.

The first approach that proposes a deep learning-based solution to poem generation is described in (Zhang and Lapata (2014)), where the authors combined convolutional and recurrent networks to generate Chinese quatrains. Then, a number of approaches focussing on Chinese poetry were proposed. In particular, a sequence-to-sequence model with attention mechanisms was proposed in (Yi et al. (2017)) and (Wang et al. (2016)). In (Yu et al. (2017)) the authors extend Generative Adversarial Networks (GANs) (Goodfellow et al. (2014)) to the generation of sequences of symbols, exploiting Reinforcement Learning (RL). They consider the GAN discriminator to be the reward signal of a RL-based generator, and, among a variety of tasks, Chinese quatrains generation is also addressed. Another RL-based approach is proposed in (Yi et al. (2018)), where two networks learn simultaneously from each other with a mutual RL scheme, to improve the quality of the generated poems.

In the context of English poem generation, transducers were exploited to generate poetic text (Hopkins and Kiela (2017)). Meter and rhyme are learned from characters by cascading a module that focusses on the content and a weighted state transducer that explicitly models the form of the generation. Differently, the more recent Deep-speare (Lau et al. (2018)) combines three neural modules, sharing the same character-based representation, to generate English quatrains. These models consist in a word-level language model fed with both word and character representations, a network that learns the meter, and another net that identifies rhyming pairs. At the end, generations are selected after a post-processing step that picks the best quatrains combining the output of the three modules. We notice that the

authors of (Lau et al. (2018)) exploited a collection of poems from several authors in order to train the model.

Paraphrasing

Paraphrasing is the task of re-writing an input text using other words, without altering the meaning of the original content. Conversational systems can exploit automatic paraphrasing to make the conversation more natural, e.g., talking about a certain topic using different paraphrases in different time instants. Recently, the task of automatically generating paraphrases has been approached in the context of Natural Language Generation (NLG). In particular, the task of rewriting text has been classified into three different categories (Madnani and Dorr (2010)): lexical paraphrasing, that consists in replacing words with other words with same meaning (Bolshakov and Gelbukh (2004)); phrasal paraphrasing, when the paraphrase is created acting on fragments with the same meaning (Ganitkevitch et al. (2013)); sentential paraphrasing when the paraphrase is generated at a sentence level, considering sentences with the same meaning (Barzilay and McKeown (2001); Barzilay and Lee (2003); Dolan et al. (2004)). Here we follow the idea that Paraphrasing can be approached as a sequence-to-sequence problem é (Sutskever et al. (2014)), and we formulate it as discussed in Section 2.3. We make use of Deep Neural Networks as for Machine Translation problems (Kalchbrenner and Blunsom (2013)). In particular, we exploit Pointer networks (Gu et al. (2016); Vaswani et al. (2017); See et al. (2017)) widely successful in Machine Translation and Text Summarization.

5.2 Neural Poetry

In this Section we propose a model for Poem Generation, an instance of NLG that is particularly fascinating for its unique features. The precise structure of poems, rhymes, meters, convey an aesthetic and rhythmic sound. Each poet has their own style, expressing emotions in their own peculiar way. Such constraints on language make automatic poem generation more challenging. Furthermore, the resources available are much poorer than in other NLG problems, and the lack of examples becomes a serious issue when considering ancient poets.

We address those aspects by proposing a novel syllable-based language model to allow strong transfer learning from modern general purpose texts that is trained in a multi-stage fashion on different data sources. Then, we introduce a poem selection mechanism that is based on a scoring function shaped on the poem and author characteristics. In the following we describe the language model, the multi-stage transfer learning approach, the generation procedure and finally we report the ex-

periments, carried out on the Italian poet Dante Alighieri, widely known for his Divine Comedy (Alighieri et al. (1998)).

5.2.1 A Syllable-based Model

The model, sketched in Figure 5.1, is composed by two blocks: a hyphenation module and a syllable-based language model. The first block is a tokenizer that splits text into syllables, while the second module is the actual learnable model that captures the statistics from the language. In the following, we refer to the entire model as SY-LM.

Hyphenation Module. The hyphenation module is responsible for the tokenization of text sequences into syllables. To do so, the input is first split into words, then each word is divided into syllables. Syllabification is a language-dependent process. This module is a not learnable tokenizer. We implemented the common hyphenation rules for Italian, that, apart from rare exceptions, correctly separate words into syllables. Of course, the same procedure can be designed for any other language. We removed punctuation and enriched the sequence of syllables with special tokens to consider spaces separating words (<sep>), begin (<go>) and end of tercet (<eot>) to delimit the generation, and an end of verse symbol (<eow>), since each input sequence is a tercet composed of three verses.

Language Model. The sequence of syllables $\boldsymbol{y} = (y_1, \cdots, y_T)$ produced by the hyphenation module is consumed by a neural language model. Following what discussed in Section 2.1, we exploit a recurrent neural network to learn the conditional distribution of Equation 2.2. In particular we selected an LSTM network. Syllables and special tokens are encoded with dense embeddings, named as "syllable embeddings", stored row-wise in the embedding matrix $W_{s_y} \in \mathbb{R}^{|V_{s_y}| \times d}$, where $|V_{s_y}|$ is the size of the vocabulary. At time t, the token y_t is mapped into its syllable embedding $\boldsymbol{e}_t$. In detail, we have:

$$\boldsymbol{e}_t = W_{s_y} \cdot \mathbf{1}(y_t) \,,$$

where $\mathbf{1}(\cdot)$ is a function returning the 1-hot column vector that has 1 in the position associated to the vocabulary index of its argument. The syllable representations are learnt during the task. It is important to notice that V_{s_y}, i.e. the set of all syllables (and a few special tokens), has a cardinality significantly smaller than in word-based dictionaries. Therefore, the embedding matrix W_{s_y} has a much less parameters, making SY-LM lighter and easier to train. In addition to that, syllables practically solve the problem of unknown words too. At each time step t, the internal recurrent state $\boldsymbol{h}_t$ is updated as follows:

$$\boldsymbol{h}_t = \texttt{LSTM}(\boldsymbol{e}_t, \boldsymbol{h}_{t-1}) \,. \tag{5.1}$$

A non-linear layer (weights W, bias b, activation σ indicating the hyperbolic tangent) projects back $\boldsymbol{h}_t$ to a d-sized vector $\mathbf{z}_t$, and a dense layer followed by the softmax activation function computes the probability distribution $\hat{\boldsymbol{y}}_t$,

$$\mathbf{z}_t = \sigma(W\boldsymbol{h}_t + b) \tag{5.2}$$

$$\boldsymbol{o}_t = W'_{s_y}\mathbf{z}_t \tag{5.3}$$

$$\hat{\boldsymbol{y}}_t = \texttt{softmax}(\boldsymbol{o}_t)\ . \tag{5.4}$$

It is worth noticing that the dense layer of Equation 5.3 shares its parameters with the syllable embedding matrix W_{s_y} (being $'$ the transpose operator), further reducing the number of parameters of the model. The model architecture is illustrated in Figure 5.1 after the hyphenation module.

Sy-LM is optimized to maximize the likelihood of the sequence of syllables in the training poem, i.e. the Divine Comedy, given the language model that estimates the conditional probability $p(y_t|y_1,\ldots,y_{t-1})$. This is implemented by minimizing the cross-entropy between each prediction $\hat{\boldsymbol{y}}_t$ and the ground truth from the Divine Comedy, thus pushing toward 1 the entry of $\hat{\boldsymbol{y}}_t$ associated to the t-th syllable of the current tercet in the Divine Comedy.

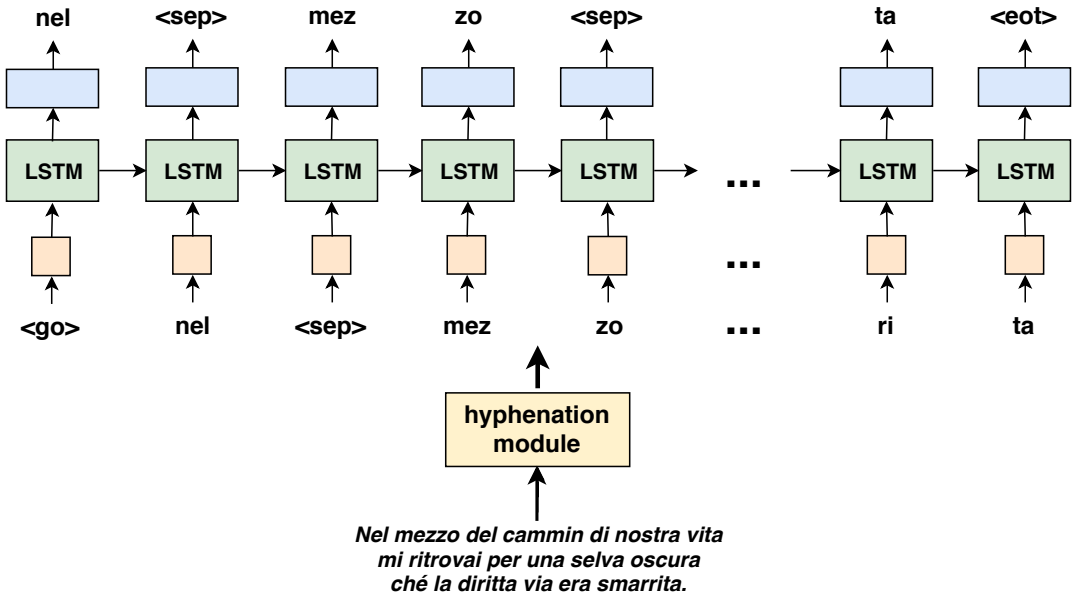

Figure 5.1: Sketch of sy-LM. Text is first tokenized into sequences of syllables by the hyphenation module and enriched by some special tokens: word-separator <sep>, begin-of-tercet <go>, end-of-verse <eov>, end-of-tercet <eot>. Orange squares are syllable embeddings, green rectangles represent the `LSTM` cell unfolded through time, blue blocks depict the network of equations (5.2-5.4). As in all language models, the target is the next syllable (see Equation 2.2).

5.2.2 Multi-Stage Transfer Learning

Neural Language Models require large amounts of data to generalize. This is generally not an issue, since LMs do not necessitate annotated data and the web provides plenty of data. In the case of Poetry however, resources are generally scarce, especially when focusing on a single author from an ancient language variety, as in the case of Dante Alighieri. In particular, the Divine Comedy contains about $4,000$ tercets ($\sim 157k$ syllables) only, several order of magnitude less than traditional corpora. Just to make a comparison, the ukWaC corpus[1] exploited to train the char-aware model in Chapter 3 contains 2 billion words, at least forty thousand times bigger. Even considering all the author's manuscripts, including prose and poems, even written with a different style, the total amount of data is barely more than doubled. To alleviate the lack of available resources we propose a multi-stage transfer learning procedure. The goal is to progressively grasp knowledge, from generic syntactical and grammatical information about the language itself, up to the author's style. Knowledge transfer starts by exploiting a large generic corpus written in the same language. A second training step exploits other manuscripts by the author to adapt the network toward the author style, and finally, the model is fine-tuned on the desired content only. The multi-stage steps are shown in Figure 5.2.

The role of syllables. Unfortunately, there is not a large enough corpus in Dante Alighieri's language. He lived in the middle ages and wrote the Divine Comedy in Tuscan/Florentine Vulgar dialect of that time. He gave a strong contribute in the development of the Italian Language that is currently spoken. Thus, we consider for the first step a large modern Italian dataset. Clearly the language has evolved since then, and the modern Italian is different from original Vulgar. New words and forms have replaced some obsolete expressions. This changes would cause little or no transfer when using a classical word-based language model, but with our SY-LM, that is based on syllables, much more information is preserved and can be transferred.

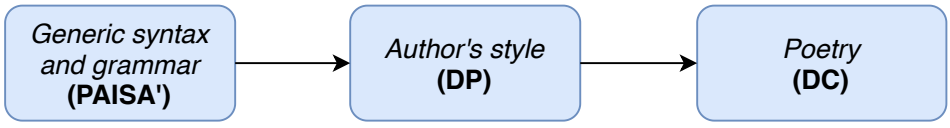

Figure 5.2: Multi-stage Transfer Learning. The language model is progressively adapted toward task-specific data. First generic grammatical properties are acquired from large general purpose data (**PAISA'**). Then, the model is refined on author's content to learn its style and peculiarities (**DP**). Finally, the model is fine-tuned on the author's poetic material only (**DC**).

[1] `http://wacky.sslmit.unibo.it/doku.php?id=corpora`

5.2.3 Generation Procedure

We formulate Poem Generation as in Equation 2.24 without any given conditional input $\boldsymbol{x}$. As previously discussed in Section 2.3, the problem clearly belongs to the *open-ended* class of NLG tasks, hence decoding requires sampling strategies. Moreover, after decoding we introduce a novel author-based scoring system to select only a portion of the generated tercets. Lets discuss first the decoding and then the selection mechanism.

Decoding. To generate new tercets, at inference time we adopt a Monte Carlo sampling strategy as done in (Yu et al. (2017)). Generation starts feeding SY-LM with the <go> input symbol and a zero vector as initial hidden state $\boldsymbol{h}_0$. The predicted token feeds autoregressively the network at the next time step. Sampling terminates either when the end-of-tercet symbol (<eot>) is generated or, when the number of syllables reaches a fixed maximum limit that was set to 75. The randomness in multinomial sampling guarantees different generated sequences sampled from the learnt distribution.

Tercets selection. We generate a batch of M tercets. A scoring function $R(\boldsymbol{y}) \in \mathbb{R}$ assigns a value to each tercet $\boldsymbol{y}$. The ones with highest score are selected, the rest is discarded. In our experiments, we generated $2,000$ tercets and selected the best 20. The scoring function $R(\boldsymbol{y})$ is actually an average of $s = 4$ different scores that are based on known properties of the Divine Comedy and its author, in terms of form and language:

$$R(\boldsymbol{y}) = \frac{1}{s} \sum_{j=1}^{s} R_j(\boldsymbol{y}). \tag{5.5}$$

We describe each one in detail.

Tercets' structure. The first score $R_1(\boldsymbol{y})$ penalizes generations that are not tercets, i.e. that are not made of three lines. We define the function as:

$$R_1(\boldsymbol{y}) := -\texttt{abs}(|\boldsymbol{y}| - 3) + 1\,, \tag{5.6}$$

where $|\boldsymbol{y}|$ indicates the number of verses in the tercets and `abs` is the absolute value function.

Hendecasyllabic verses. Differently, $R_2(\boldsymbol{y})$ promotes sequences with verses in $\boldsymbol{y}$ that follow a hendecasyllabic meter. Since our model is based on syllables, it is easy to count the number of syllables in a generated verse v, and we define R_2 as follows,

$$R_2(x) = -\sum_{v \in \boldsymbol{y}} (\texttt{abs}(|v| - 11)) + 1\,. \tag{5.7}$$

where v indicates a verse and $|v|$ is its number of syllables.

Rhyme scheme. We remind that tercets in the Divine Comedy follow an ABA rhyming scheme. Thus we measure $R_3(\boldsymbol{y})$,

$$R_3(\boldsymbol{y}) = \begin{cases} 1, & \text{if } (v_1, v_3),\, v_1, v_3 \in \boldsymbol{y} \text{ are in rhyme} \\ -1, & \text{otherwise} \end{cases}, \tag{5.8}$$

assigning a positive score to tercets where the first and third lines are in rhyme. To establish the rhyme we have to check the last two words of each line. Since the model generates a sequence of syllables, first we reconstruct words, merging syllables between word-separators (<sep> token).

Word vocabulary coherence. The last scoring function $R_4(\boldsymbol{y})$ also operates at word level. It assigns a small positive value a to words in $\boldsymbol{y}$ that appear in the Divine Comedy and strongly discourage not valid words. Although we experienced to be pretty unlikely, with this score we avoid the generation of words that are far from the poet's style. Formally,

$$R_4(\boldsymbol{y}) = \sum_{w \in \boldsymbol{y}} f_w(\boldsymbol{y}), \qquad f_w(\boldsymbol{y}) = \begin{cases} a, & \text{if } w \in V \\ -b, & \text{otherwise} \end{cases} \tag{5.9}$$

where w indicates a word in tercet $\boldsymbol{y}$. In the experiments, a was set to 0.05 and b to 1.

5.2.4 Experiments

We report two kinds of experiments to evaluate the quality of the proposed poem generator: a quantitative analysis, to assess the impact of the multi-stage transfer learning approach and a human evaluation by expert and non-expert individuals. Before illustrating the results, we describe the corpora that are used to train the model and some details about the implementation of the sy-LM architecture.

Datasets

Modern Italian Dataset (PAISA'). PAISA',[1] is a large corpus of Italian web texts. To train sy-LM, we sampled a sub-portion of 200k documents, consisting of about 836k sentences ($\sim$ 67M syllables).

[1] http://www.corpusitaliano.it/en/contents/paisa.html

Dante's Production (DP). We gathered most of Dante's non-latin prose and poetry production. In particular, we collected the entire *Convivio*, *Le rime* and *La vita nuova*, obtaining overall 1,752 sentences ($\sim$ 157k syllables) from prose and 2,727 verses ($\sim$ 48k syllables) from poems.

The Divine Comedy (DC). Divine Comedy is the most important Dante Alighieri's manuscript. This poem is a collection of 100 "cantos" belonging to three *cantiche*. Each canto is a poem with a variable number of tercets also known as "Dante's tercet", made of hendecasyllabic verses that follow a chained rhyming scheme (ABA BCB ... VZV Z). Sy-LM was trained on 3,768 tercets and evaluated on a test set of 472. We also kept a validation set of 471 tercets to set the network hyper-parameters. Overall, there are about 180k syllables in the Divine Comedy.

Implementation Details

The neural model and the experiments were implemented in Tensorflow[1]. We selected the hyper-parameters of the neural language model by picking the configuration yielding the best perplexity (see Section 2.1.2) on the validation set held-out from the **DC** corpus. The best setup was obtained with the syllable embeddings size d set to 300, the state of the `LSTM` cell with size 1,024 and the state neurons were dropped out with probability 0.3 (Srivastava et al. (2014)). The size of the vocabulary V_{s_y} was set to 1,884, including all the syllables in the Divine Comedy and the special tokens. When pre-training on **PAISA'** and then refining on **DP** (Section 5.2.2), we kept a small validation set for early stopping, and multiple batch sizes and learning rates have been validated. In the end, we found that batch size 32 and learning rate of 0.001 were the best optimization parameters.

As already mentioned in Section 5.2.3, we adopted Monte Carlo sampling in the reported experiments. Later on, top-k and nucleus sampling strategies have been released in our implementation too.

Transfer Learning Evaluation

We analyze the impact of the multi-stage learning procedure described in Section 5.2.2. The multi-stage transfer learning approach is progressively trained on three datasets: **PAISA'**$\rightarrow$ **DP** $\rightarrow$ **DC** (arrows specify the training order). To show that the approach is beneficial, we evaluate the perplexity measured on the validation and test sets held out from the Divine Comedy (**DC**), using the model trained with different dataset combinations. Results are reported in Table 5.1. As expected, pre-training on additional data always improves the LM quality. Pre-training on **PAISA'** is what gives the most substantial improvements, proving there is a significant knowledge

[1]The code is publicly available at `https://gitlab.com/zugo91/nlgpoetry/`

Table 5.1: Evaluation of the model trained using multiple datasets in a multi-stage transfer learning manner. $A \rightarrow B$ means that we train on data A first, and then we train on data B. Perplexities reported are on the validation and test sets from the Divine Comedy.

Datasets	**Validation** pp	**Test** pp
DC	12.45	12.39
PAISA′ $\rightarrow$ **DC**	10.83	10.82
DP $\rightarrow$ **DC**	11.95	11.74
PAISA′$\rightarrow$ **DP** $\rightarrow$ **DC**	**10.63**	**10.55**

transfer from modern Italian to Vulgar, i.e. its past diachronic variety. Even pre-training the model on **DP**, that is a rather small corpus, brings a gain in perplexity. In any case, the best performances are achieved by multi-stage transfer learning (**PAISA′**$\rightarrow$ **DP** $\rightarrow$ **DC**).

Human Evaluation

Evaluating the quality of generated text is challenging. Perplexity gives insights about the quality of a language model, but we already showed in Section 2.3 how this is not enough to guarantee good quality texts. In some NLG problems, the ones belonging to the *non-open-ended* domains, it is possible to compare the generated output with a reference text, and there are several metrics (BLEU - (Papineni et al. (2002)), ROUGE - (Lin (2004)), METEOR - (Banerjee and Lavie (2005))) that correlate with human judgments. In *open-ended* scenarios, as Poem Generation, there are no reference targets, therefore, human evaluation is paramount. We employed human judges for two different evaluations of the generated tercets.

In the first assessment, we involved 13 graduate and not graduated students, mostly from humanistic degrees. We refer to them as "non-expert" judges, since they were not specialized in Dante's production, but very well aware of the author and his works. Ten tercets were submitted to each judge, one at a time. They were asked to identify which tercets were real and which ones were generated by SY-LM. They always received five tercets from Dante and five generated by our model, but they were not aware of such distribution. Results of such evaluation are presented in Table 5.2. We report the number of times (in percentages) that tercets from a certain population were judged to be authored by Dante. It is clear that, given the humanistic background of the evaluators, judgements are rather thoughtful, however our generated tercets are able to fool the judges almost half of the times of the real ones from Dante, with a relative difference of 56.25%. We deep-dive by distinguishing the judges in two groups: the ones that struggled in the identification of real

Table 5.2: The number of times (percentages) that tercets from either SY-LM or Dante Alighieri (POET) are considered to be authored by Dante (i.e., they were marked as "real"). The model fooled the humans almost half of the times of real Dante's production.

Generator	Real-Mark
sy-LM	28%
Poet	64%

Dante's tercets as real[2], and the remaining ones. In Figure 5.3 we can observe that the "less-capable judges" were even more attracted by SY-LM than by real Dante's tercets. We believe that these judges may better represent the average population of users, hence this result suggests that SY-LM is very positively perceived. On the other hand, more capable evaluators are less frequently fooled by SY-LM with about 67% relative difference from Dante's poems.

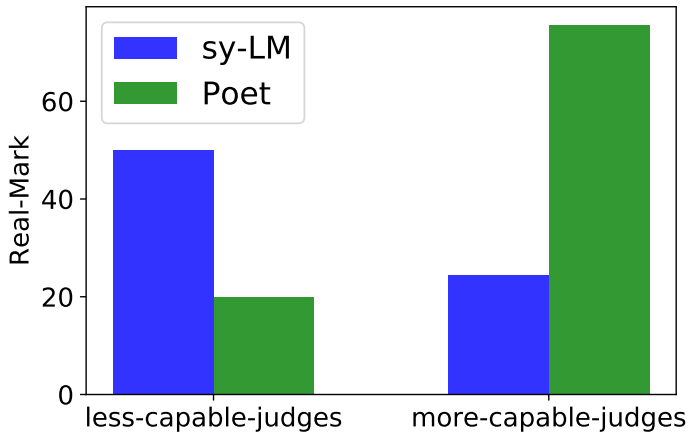

Figure 5.3: Results of Table 5.2 when further dividing the population into two groups of individuals: capable vs less capable judges.

In the second human evaluation experiment, we involved 4 expert judges with academic experiences on Dante Alighieri's production. Trying to fool them is clearly pointless, since they are well aware of the Divine Comedy and its contents. Hence we ask to each expert to evaluate twenty tercets, scoring each poem (from 0 to 5) on the following properties: *emotion*, *meter*, *rhyme*, *readability* and adherence to the *author's style*. Also in this case, half of the assigned tercets was real, half was generated by SY-LM, without informing them about such distribution. The results of the test are summarized in Table 5.3. Real tercets are better scored, as expected, however, we

[2]We decided to assign to this group the evaluators that correctly marked real tercets as real less than 50% of the times.

observe a good evaluation of the quality of the rhymes produced by SY-LM. Considering that judges know deeply Dante Alighieri, it is interesting to see that they are experiencing some of the his style in the generated tercets. Evaluators emphasized how the semantics behind the generated verses are sometimes hard to appreciate since they do not convey enough emotion, that is the motivation behind the lower scores on the first two columns of Table 5.3. The meter was poorly evaluated by the judges, because they applied very strict criteria in evaluating the meter, giving low scores whenever a small incoherence with Dante's meter was apparently detected, even if they reported that it was not far from the ideal case.

Table 5.3: Evaluation of tercets generated by SY-LM by each expert and the average rates. Five semantical aspects have been considered: *readability, emotion, meter, rhyme and style*. Votes range from 0 to 5. For comparisons, in the last line we also report the average rates in the case of actual Dante's verses (POET).

	Readability	**Emotion**	**Meter**	**Rhyme**	**Style**
Judge 1	1.57	1.21	1.57	3.36	2.29
Judge 2	1.64	1.45	1.73	3.00	2.27
Judge 3	2.83	2.33	2.00	4.17	2.92
Judge 4	2.17	2.00	2.33	2.92	2.50
Average	2.04	1.73	1.90	3.37	2.49
Poet (Average)	4.34	3.87	4.45	4.50	4.34

Finally, we present few examples of generated tercets in Table 5.4 just to give some insights of the kind of the tercets that were generated. Three of them were rated positively by expert and non-expert judges, the last one instead (bottom right), was badly scored.

Table 5.4: Four examples of tercets generated by SY-LM. The last one (bottom right) never fooled the judges, whereas the first three tercets were marked as real Dante's tercets by 88.00%, 55.56% and 45.45% of the evaluators, respectively.

e tenendo con li occhi e nel mondo *che sotto regal facevan mi novo* *che 'l s'apparve un dell'altro fondo*	*per lo mondo che se ben mi trovi* *con mia vista con acute parole* *e s'altri dicer fori come novi*
in questo imaginar lo 'ntelletto *vive sotto 'l mondo che sia fatto moto* *e per accorger palude è dritto stretto*	*non pur rimosso pome dal sospetto* *che 'l litigamento mia come si lece* *che per ammirazion di dio subietto*

5.3 Neural Paraphrasing

Paraphrasing is the problem of rephrasing a text passage without affecting its meaning. We have already seen in Section 2.3 that paraphrasing can be formulated as an NLG task, and in particular it belongs to the *non-open-ended* domains. In order to properly training neural language models for the task of paraphrasing, large amounts of aligned pairs of paraphrases are required.

In this Section we propose a novel method for the automatic construction of large corpora of aligned pairs of paraphrases. The proposed method poses its basis on the assumption that news and blogs websites talk about the same facts, using often different narrative styles. Based on a similarity search procedure with linguistic constraints, our algorithm retrieves and aligns paraphrase candidates and then selects only the most promising ones. We apply our method to the case of Italian language, then we use such data to train a neural model to generate paraphrases and report the results.

5.3.1 Automatic Dataset Construction

We design a method for building a dataset of sentence pairs. The resulting dataset is a collection of pairs of textual sentences, each of them composed of an *input* sentence and a *target* paraphrase. The method is based on the idea that websites normally report the same news using different *idiolects*. An *idiolect* is defined as the individual distinctive and unique use of language, concerning the morpho-syntactic and stylistic features. This assumption allows us to formulate the problem of paraphrase discovery as a Highly Constrained Sentence Similarity Search (HCSSS), where the results of a search in a document collection must satisfy hard linguistic constraints (morphological, syntactic and semantic constraints) in addition to word co-occurrences. The automatic dataset construction consists of a pipeline with different stages. We discuss each step of the pipeline in the following.

Crawling

In the first stage of the proposed approach a large number of articles are retrieved from the web using a focused crawler. In particular, we exploit a list of newspaper and blog websites, and limit the search to contents belonging to a set of domains (or topics): *news*, *culture*, *economics*, *nature*, *politics*, *society*, *sports* and *technology*. The topic oriented crawling can be driven by groups of keywords for each domain, so that only articles containing these keywords are retrieved. All the listed websites can be crawled daily and their contents are organized by category (topic) and by date in a such way to ease the subsequent alignment phase. At this step, the consid-

ered content consists exclusively of raw text and it is saved for the following content analysis phase.

Pre-processing

The crawled documents are pre-processed through a pipeline of NLP techniques before being indexed by the search engine. The pipeline aims at enriching the representation of text with the identified collocations and named entities, where a collocation is an aggregation of tokens having a specific meaning if co-occurring together, while a named entity represents a specific entity of the real world, belonging to a set of predefined types (people, organization, location, etc.). For each collocation we identify lemmas and Part-Of-Speech (POS) information. We also keep track of common nouns and proper nouns that are found in the sentence, that will play a central role in the steps described in the following subsections.

After the NLP analysis, the text is segmented into sentences which are subsequently stored using a multi-field search engine. For each sentence, the following linguistic features are indexed as search engine fields: *Raw text* (*tokens*), *Collocations, Lemmas* (*of collocations*), *POS* (*of collocations*), *Named entity type*. Storing and indexing the linguistic features allows us to use linguistic constrained search queries, that are crucial to improve the precision of the search results, as we will describe in the next subsection.

Sentence Alignment

In order to build a dataset composed of (*sentence, paraphrase*) pairs, we define a group of reference *sentences*, and, for each of them, we perform search engine queries to identify a set of candidate *paraphrases*. Queries are based on the aforementioned linguistic features, which allow us to constrain the resulting candidate paraphrases to share a large amount of information with the reference sentence used as search seed. Moreover, the system can use information on the crawled article, to which the reference sentence belongs, to improve the precision of the search, such as the publication date and the article topic (paraphrases are expected to be found on articles with the same/similar publication dates). As a result, we get a search system with very high precision and low recall, and the details of the whole sentence selection and alignment are reported in the following.

Selection of the reference set. Selecting the sentences that belong to the reference set is a procedure that is applied to all the crawled sentences. The selection depends on the number of detected proper nouns and common nouns, since we are interested in picking sentences that are not too short and that involve the description of real-world facts. In particular, given a sentence s_i, let us indicate with $CN(s_i)$ the set of

common nouns in s_i and with $PN(s_i)$ the set of proper nouns in s_i. The reference set RS is defined as

$$RS = \{rs_i : |CN(rs_i)| \geq min_{CN} \text{ AND } |PN(rs_i)| \geq min_{PN}\}\,, \tag{5.10}$$

where rs_i is a sentence of the set, min_{CN} is a system parameter indicating the minimum number of common nouns required in each sentence and min_{PN} is a system parameter indicating the minimum number of proper nouns required in each sentence.

Constrained search (HCSSS). The sentences in the reference set are used as seeds in the search of candidate paraphrases. Given a reference sentence rs_i, a search query is executed in order to retrieve sentences s_j that satisfy a number of constraints in terms of the common and proper nouns that they share with rs_i. In detail, we search for sentences s_j such that:

- The proper nouns of rs_i are included in s_j,

$$PN(rs_i) \subseteq PN(s_j). \tag{5.11}$$

- There is a strong intersection between the common nouns in rs_i and the ones in s_j,

$$\text{If } |CN(rs_i)| > min_{CN} \text{ then } \frac{|CN(rs_i) \cap CN(s_j)|}{|CN(rs_i)|} \geq \alpha$$

$$\text{If } |CN(rs_i)| = min_{CN} \text{ then } CN(rs_i) \subseteq CN(s_j),$$

where $\alpha \in [0, 1]$ is a coverage parameter which filters out results having a small percentage of shared common nouns. We compactly report the function that checks for potential paraphrase pairs in Algorithm 2.

Filtering the candidate paraphrases. The search procedure returns a large number of candidate paraphrases $cs_i := \{para_1(rs_i)), \ldots, para_{n_i}(rs_i)\}$ for each reference sentence rs_i. Candidates in cs_i are ranked according to an internal confidence score computed by the search engine itself. The score is a value in $[0, +\infty)$ and measures the degree of similarity between each result and the query (enriched with linguistic features). From the n_i candidates, a variable subset of $\tilde{n}_i \leq n_i$ sentences is selected by choosing all the candidates with the score (opportunely normalized in the range $[0, 1]$) above a threshold $\beta \in [0, 1]$. From this subset of good paraphrases of the reference sentence rs_i we build the list of aligned pairs:

$$ps_i := \{(rs_i, para_1(rs_i)), \ldots, (rs_i, para_{\tilde{n}_i}(rs_i))\}.$$

Function `isParaphrase`(rs_i, s_j)**:**
  Data: reference sentence rs_i, candidate paraphrase s_j
  Result: *true* if s_j is a paraphrase of rs_i, $false$ otherwise
  if $PN(rs_i) \subseteq PN(s_j)$ **then**
    if $|CN(rs_i)| = min_{CN}$ **then**
      return $\left[CN(rs_i) \subseteq CN(s_j)\right]$
    end
    else if $|CN(rs_i)| > min_{CN}$ **then**
      return $\left[\frac{|CN(rs_i) \cap CN(s_j)|}{|CN(rs_i)|} \geq \alpha\right]$
    end
  end
  else
    return $false$
  end

Algorithm 2: The algorithm checks if s_j is compatible with a paraphrase of the reference sentence rs_i ($[\cdot]$ is an operator returning the Boolean value of the expression inside).

Building the final dataset pairs. Each of the elements in ps_i is made of a sentence and a candidate paraphrase $(rs_i, para_j(rs_i))$. We have seen that the candidate sentence is constrained to include all the proper nouns of the reference sentence (see Equation 5.11), but in some cases the candidate paraphrase may contain additional proper nouns, which likely corresponds to a more detailed description of the fact/event. With the aim of ordering each pair with the same criterion, we ensure that the most informative sentence is in the first position of the pair. In our context, the informativeness of a sentence is measured by counting the number of proper nouns that it contains. Applying the procedure that is formalized in Algorithm 3, we get the final aligned dataset composed of what we generically refer to as (*input*, *target*).

5.3.2 Case Study

The proposed method was applied to the case of Italian Language, where there are few available resources to train models for paraphrasing. The generated dataset will be used in the experiments of Section 5.3.4. Crawling was performed by a proprietary web monitoring system[3]. The pre-processing was realized with a proprietary NLP platform[4] developed by QuestIT[5], that consists of a pipeline-based processing system with more than 20 layers of linguistic analysis, including a sliding-window

[3]http://www.mysnooper.net/
[4]https://www.quest-it.com/natural-language-processing/
[5]https://www.quest-it.com

Function `buildFinalPair`$((rs_i, para_j(rs_i)))$**:**
  Data: aligned pair $(rs_i, para_j(rs_i))$
  Result: final pair $(input, target)$
  if $|PN(para_j(rs_i))| > |PN(rs_i)|$ **then**
    return $(para_j(rs_i), rs_i)$
  end
  else
    return $(rs_i, para_j(rs_i))$
  end

Algorithm 3: Build final dataset pairs. In case the candidate paraphrase is more informative than the reference sentence (i.e. it contains more proper nouns), reference and paraphrase are swapped.

SVM approach for POS Tagging, Lemmatization and Named Entity Recognition, Search Tree combined with gazetteers to identify collocations. Overall, 86,000 articles were downloaded, spanning over a time interval of about 2 months, retrieved from Italian news and blog sites. From these documents, we extracted more than 1 million sentences. Sentences were indexed using Elastic Search[6]. We selected $\alpha = 0.70$ and $\beta = 0.70$, and the proposed method ($min_{CN} = min_{PN} = 3$) identified a reference set consisting of about 430,000 sentences, that yielded a final paraphrasing dataset of about 85,000 aligned pairs in Italian language. Such final number of pairs is due to the fact that no paraphrases were identified for most of the reference sentences, since we imposed a high confidence score in the HCSSS procedure. In Figure 5.4 we report few examples of the aligned pairs $(input, target)$ generated by our method.

5.3.3 Model

We have seen in Section 2.3 how paraphrasing can be framed as a language modeling problem, where the generation of the paraphrase $\boldsymbol{y}$ is highly conditioned to a given input sequence $\boldsymbol{x}$ (the reference sentence). The problem reduces to the estimation of the conditional probability of Equation 2.24, that is typically estimated with sequence-to-sequence neural models. In particular, we exploit for this task a sequence-to-sequence model with attention (Bahdanau et al. (2014)) and a pointer generator. Pointer networks (Vinyals et al. (2015)) are a special class of the so-called sequence-to-sequence models (i.e., models that process an input sequence and output another sequence (Sutskever et al. (2014))), where the token to predict at each time step is obtained from a combination of two probability distributions. When

[6]https://www.elastic.co/products/elasticsearch

Input: Brexit, no al secondo referendum: Parlamento boccia l'emendamento Brexit, May non si arrende: "Accordo entro 29 marzo alla nostra portata" Brexit, Parlamento affonda anche "No Deal": rinvio piu' vicino del divorzio Brexit, i Labour contro un secondo referendum.

Target: Brexit, il Parlamento boccia l'accordo Brexit, il Parlamento vota no al ''no deal'' Brexit, no al secondo referendum: Parlamento boccia l'emendamento Brexit, i Labour contro un secondo referendum.

Input: Philippe Barbarin, 68 anni, cardinale e arcivescovo di Lione e uno dei maggiori prelati della Chiesa cattolica francese, e' stato condannato a 6 mesi di carcere con la condizionale: la sentenza e' stata pronunciata dal tribunale di Lione.

Target: Il cardinale Philippe Barbarin, arcivescovo di Lione, condannato per aver coperto abusi sui minori, offre le sue dimissioni.

Input: La valutazione dell'agenzia di intelligence, nella quale i dirigenti affermano di avere un alto grado di fiducia, e' la piu' definitiva per ora tra quelle che legano il principe Bin Salman al delitto e complica gli sforzi dell'amministrazione Trump di salvare le relazioni con il suo stretto alleato in Medio Oriente.

Target: La valutazione dell'agenzia di intelligence Usa e' la piu' autorevole per ora tra quelle che legano il principe Bin Salman al delitto e complica gli sforzi dell'amministrazione Trump di salvare le relazioni con il suo stretto alleato in Medio Oriente.

Input: Dorothy con le magiche scarpette rosse e tutti i suoi strampalati amici protagonisti dello spettacolo ''Il mago di Oz'', il musical per bambini e famiglie in programma domenica 24 febbraio alle ore 16 al Teatro condominio di Gallarate.

Target: Gallarate - Dorothy balla con le scarpette rosse nel musical ''Il mago di Oz'' - Bambini - Varese News.

Figure 5.4: Examples of aligned pairs automatically generated with the proposed method (Italian).

generating the output sequence, the model either generates a word from the predefined vocabulary or it copies a word from the input sequence. This mechanism is helpful to catch and handle Out-Of-Vocabulary (OOV) words, such as named entities, allowing the network to improve its performances in several NLP tasks (Machine Translation, Text Summarization, and others). Our model is similar to the one of (See et al. (2017)), but without any coverage mechanism, where both the input sentence and target paraphrases are tokenized as a sequence of words. An illustration of the overall architecture is given in Figure 5.5. In the following, we summarize the encoder and decoder separately.

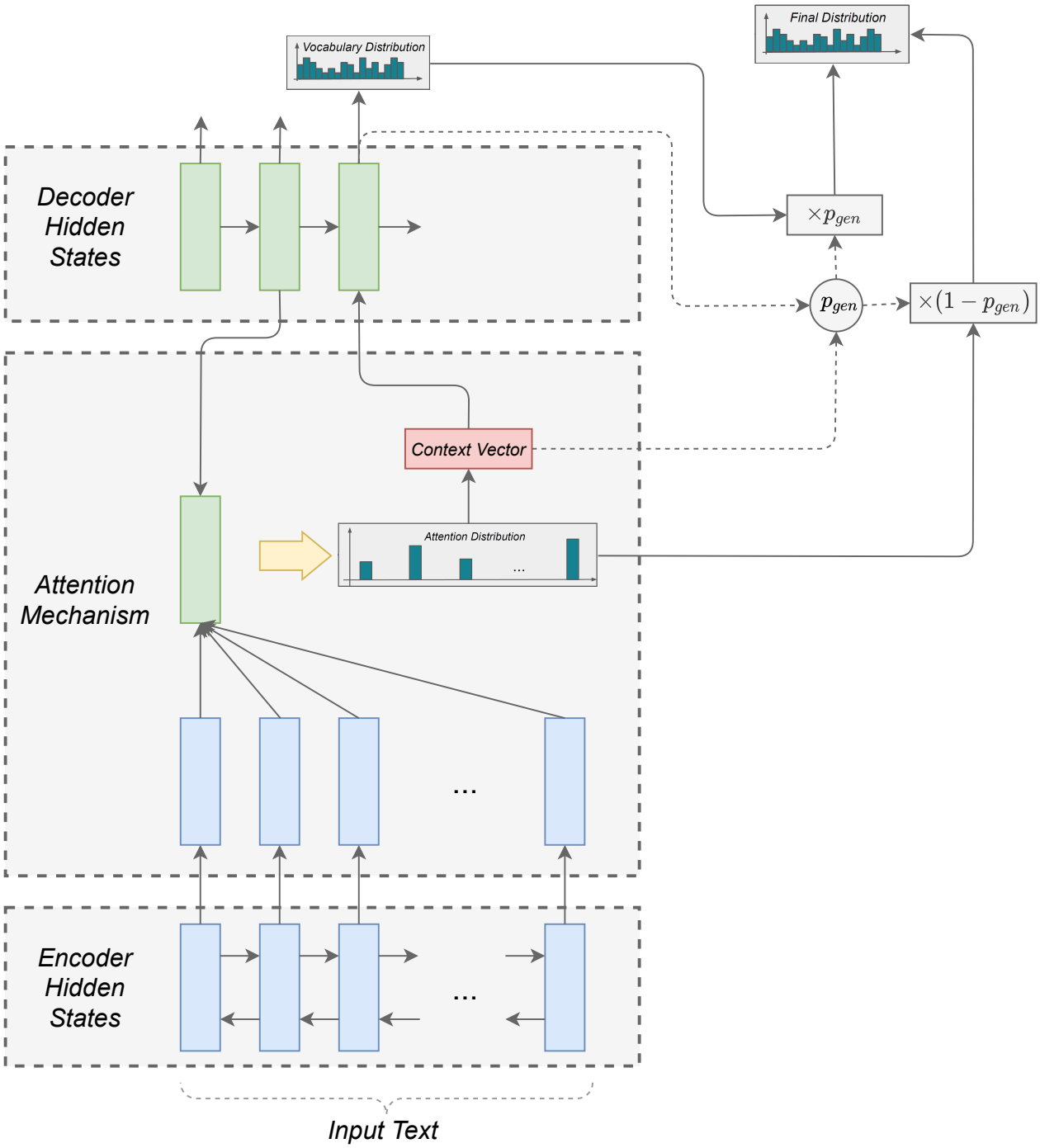

Figure 5.5: Sketch of the neural pointer generator.

Encoder

The input sequence $\boldsymbol{x}$, corresponding to the reference sentence, is mapped into a fixed-length internal representation using a Recurrent Neural Network (RNN). In particular we encode $\boldsymbol{x}$ with a Bi-LSTM. The encoder processes the whole sequence $\boldsymbol{x}$ of length n, and its internal state at the end of the sequence, referred to as h_n, is used to initialize the state of the decoder. The encoder also returns all the intermediate states generated while processing x and computed with the classic recurrent scheme

$$h_i = \texttt{Bi-LSTM}(x_i, h_{i-1})$$

as already defined in Section 2.2. So, the output of the encoder that is passed to the decoder is a sequence of states $\mathcal{H} := \{h_1, \ldots, h_n\}$.

Decoder

The decoder estimates the probability distribution of a word given the previous words and the input text. The decoder is an LSTM, conditioned by the output of the encoder $\mathcal{H}$ processed through an attention layer (Bahdanau et al. (2014)). Formally, the attention layer outputs a vector $\hat{h}_t$, computed as follows:

$$\hat{h}_t = \sum_{i=1}^{n} a_{t,i} \cdot h_i, \tag{5.12}$$

where $a_{t,i}$ is a scaling factor of h_i decided by the attention. Overall $\hat{h}_t$ is the result of a weighted average, where the weights $a_{t,i}$ are computed:

$$e_{t,i} = v'_a \cdot \texttt{tanh}(W_a \cdot [h_i, s_{t-1}] + b_a), \tag{5.13}$$

$$a_t = \texttt{softmax}(e_t). \tag{5.14}$$

The function `tanh` is the hyperbolic tangent activation, and the vectors v_a, b_a and the matrix W_a are parameters associated to the attention layer, learned while training the network. The decoder state is then updated with

$$s_t = \texttt{LSTM}([y_{t-1}, \hat{h}_t], s_{t-1})\ . \tag{5.15}$$

Finally, a projection layer transforms the state s_t, concatenated with $\hat{h}_t$, into a probability distribution on the words in the vocabulary

$$P^t_{\text{vocab}} = \texttt{softmax}(W_V \cdot [s_t, \hat{h}_t] + b), \tag{5.16}$$

where the matrix W_V and vector b are learnable parameters. We will use the notation $P^t_{\text{vocab}}(\texttt{y})$ to indicate the probability of generating word `y` at time t.

What we described so far is a plain decoder with attention that selects the next word to produce at each time instant, i.e., the one that is most likely accordingly to

Equation 5.16. Pointer networks are furnished of a decoder that combines the classical distribution over all the words in the vocabulary, with a copy mechanism that yields the probability of emitting (copying) a word taken from the input sequence $\boldsymbol{x}$. The latter probability, indicated with P^t_{copy}, is computed by exploiting the attention scores a_t of Equation 5.14. In detail,

$$P^t_{\text{copy}}(\mathtt{y}) = \sum_{i:\mathtt{y}_i==\mathtt{y}} a_{t,i} \ , \tag{5.17}$$

where the summation is only needed in the case in which a word is repeated multiple times in the input sentence, and all the probabilities $a_{t,i}$ associated to the instances of such word must be accumulated. In order to combine $P^t_{\text{vocab}}(\mathtt{y})$ and $P^t_{\text{copy}}(\mathtt{y})$ into the final generation probability $P^t(\mathtt{y})$, a weighting score $p^t_{\text{gen}} \in [0,1]$ is introduced, that depends on the decoder state s_t, the context vector $\hat{h}_t$ and the embedding of the previously generated word y_{t-1},

$$p^t_{\text{gen}} = \sigma(W_{\text{gen}} \cdot [y_{t-1}, s_t, \hat{h}_t] + b_{\text{gen}}). \tag{5.18}$$

The parameters W_{gen} and b_{gen} are learned jointly with the whole networks, and σ is the sigmoid activation function. Qualitatively, the model can learn when it should copy from the source input and when it should not. The final probability of generating a word $\mathtt{y}$ at time t is a linear combination of $P^t_{\text{vocab}}(\mathtt{y})$ and $P^t_{\text{copy}}(\mathtt{y})$:

$$P^t(\mathtt{y}) = p^t_{\text{gen}} \cdot P^t_{\text{vocab}}(\mathtt{y}) + (1 - p^t_{\text{gen}}) \cdot P^t_{\text{copy}}(\mathtt{y}) \ . \tag{5.19}$$

It is worth noticing that $\mathtt{y}$ may be an OOV word. In such a case $P^t_{\text{vocab}}(\mathtt{y})$ would be zero. Analogously, if $\mathtt{y}$ does not belong to the input sentence, $P^t_{\text{copy}}(\mathtt{y}) = 0$.

Generation. Given the *non-open-ended* nature of the NLG problem, at inference time we adopt a beam search as decoding strategy (see Section 2.3). In the experiments we set the beam size b to 10, that is a good trade-off between performances and computational efficiency.

5.3.4 Experiments

We evaluate the performances of different variants of the aforementioned sequence-to-sequence model in the task of paraphrasing. Results are compared using the ROUGE score (Lin (2004)), a popular metric in paraphrasing, with the aim of showing the quality of the created dataset. Furthermore, we also present some qualitative results to emphasize the limits of the compared models. In detail, we compared a plain sequence-to-sequence model with attention and the same model enriched with the pointer-based generator. We also evaluate the case in which we pre-train a part of the Pointer network in an autoencoding task, where the network is asked

to simply generate the input sentence. During such pre-training, both the attention and the pointer mechanisms were turned off, to avoid the network to simply learn the trivial solution of always copying the input text.

Datasets. We consider the dataset created for Italian language in Subsection 5.3.2 with the procedure described in Subsection 5.3.1. The corpus was divided into two disjoint subsets of about 72*k* and 12*k* pairs, used as training and validation sets, respectively. Furthermore, we left a test set of 1384 pairs (excluded from the training and validation sets). Since, we found that the networks were badly biased by the presence of multiple paraphrases of the same sentence, we decided to filter the data to ensure that for each input sentence there was only one target paraphrase, and we augmented the data including also flipped pairs. Afterwards, we ended up with a training, validation, test set of 35*k*, 10*k* and 1485 examples, respectively. For pre-training the model, we exploited the **PAISA'** dataset (Lyding et al. (2014)), a large corpus of Italian texts taken from the web already exploited in Section 5.2.

Training details. We used the same configuration of (See et al. (2017)), apart from the size of the vocabulary that we set to 30*k* words, and the maximum length of input and output sequences set to 70 tokens. Learning was stopped after 15 consecutive epochs without any improvements on the validation set. All models were optimized with Adam (Kingma and Ba (2014)), with learning rate 0.001 and clipping the gradient norm to 4.0. All the models are implemented in Tensorflow.

Table 5.5: ROUGE-1,2,*l* F_1 score on test data. Three variants of the model described in Subsection 5.3.3 have been trained. POINTER NETWORK is exactly as described in Subsection 5.3.3, S2S + ATTENTION is a simplified version without the pointer generator and PRETRAINED POINTER NETWORK is the full model (partially) pretrained in an autoencoding task.

	ROUGE		
	1	2	*l*
S2S + ATTENTION	44.39	31.71	41.36
POINTER NETWORK	**60.66**	**50.21**	**56.64**
PRETRAINED POINTER NETWORK	60.51	49.88	56.29

Results. The results are reported in Table 5.5. We evaluated the performances measuring different ROUGE-based metrics, that are the F1 scores in the case of ROUGE-1, ROUGE-2 and ROUGE-*l* (see Lin (2004)). These metrics quantify the word-overlap, bigram-overlap, and longest common sequence between the target text and the predicted paraphrase, respectively. Results are in-line with the ones commonly obtained in Text Simplification and Single-sentence Summarization (Zhang

et al. (2017)), which are tasks related to the one we consider. They suggest that the proposed dataset construction procedure is a feasible way to collect data to train neural models for paraphrasing. The copying mechanism of Pointer networks sig-

INPUT: - tutto il programma velate, varese - domenica 23 a velate un'ospite illustre: ada cattaneo vestirá i panni della regina delle nevi per raccontare le incantevoli tradizioni di questo periodo dell'anno.

TARGET: in arrivo la prossima domenica 23 a velate un'ospite illustre: ada cattaneo vestirá i panni della regina delle nevi per raccontare le incantevoli tradizioni di questo periodo dell'anno.

S2S + ATTENTION: tutto il programma 2019, varese - domenica 23 a novembre un'ospite illustre : gino cattaneo, i panni della regina delle nevi per raccontare le <UNK> tradizioni di questo periodo dell'anno.

POINTER NETWORK: varese, domenica 23 a velate un'ospite illustre: ada cattaneo vestirá i panni della regina delle nevi per raccontare le incantevoli tradizioni di questo periodo dell'anno.

INPUT: manovra economica sotto la lente il ministro all'economia, giovanni tria, in audizione alle commissioni bilancio di camera e senato, ha sottolineato che i recenti livelli di rendimento dei titoli di stato non riflettono i dati fondamentali del paese e ha inoltre commentato che lo spread scenderá.

TARGET: in audizione sul def davanti alle commissioni bilancio di camera e senato tria ha confermato che "lo scenario tendenziale (del def, ndr) <UNK> gli incrementi dell'iva e delle <UNK> dal 2020-2021".

S2S + ATTENTION: il ministro dell'economia, giovanni tria, in audizione alle commissioni bilancio di camera e senato, ha sottolineato che i recenti livelli di rendimento dei titoli di stato non riflettono i dati fondamentali del paese e ha inoltre commentato che lo spread bancarie.

POINTER NETWORK: manovra, sotto la lente il ministro all'economia, giovanni tria, in audizione alle commissioni bilancio di camera e senato, ha sottolineato che i recenti livelli di rendimento dei titoli di stato non riflettono i dati fondamentali del paese e ha inoltre commentato che lo spread scenderá lo spread scenderá in spread.

Figure 5.6: Examples of a common issue in the considered neural models. Generated sentences are sometimes too similar to the input sentence, in particular with the pointer generator.

nificantly improves the ROUGE scores with respect to plain sequence-to-sequence, while the pre-training stage does not introduce further improvements. While the former results were expected, the latter is surprising and might be caused by the pretraining stage applied on only part of the network, without any pointers and/or attention (for the reasons we described earlier), making it hard to transfer the pre-

trained model toward the final Pointer network. An aspect that we observed is that all the trained models are strongly biased toward the reproduction of the exact input sentence. As a matter of fact, the input sentence and the target paraphrase are sometimes very similar, due to the intrinsic property of the paraphrasing task. This behaviour is evident when looking at the generations reported in the example of Figure 5.6. We leave this point open for further investigation on improved neural models for paraphrasing.

5.4 Discussion

In this Chapter we have addressed two very different NLG problems: Poem Generation and Paraphrasing. In both cases the models learns through Language Modeling and in low-resource conditions. However, the different nature of the problems required different solutions.

Poetry. In the case of Poem Generation, we presented a syllable-based language model to generate tercets with the style of a given author. The approach is general and it has been studied in the context of Dante Alighieri, an Italian poet of the middle ages, well known for being Divine Comedy's author. The problem of low-resources is addressed exploiting syllables to allow transfer learning from large scale collections of modern Italian by means of a multi-stage transfer learning technique. Despite its simplicity and the lack of large-scale collections of data from the target author, our model produces tercets that are considered real by evaluators with humanistic background roughly half of the times of Dante's verses. This is due to a scored generation mechanism that helps to keep Divine Comedy's meter and rhyme, and also due to a multi-stage training procedure that improves the quality of the content, exploiting all the poet's production and text in modern Italian. However, the outcome of the evaluation from expert judges clearly showed that, while the rhyme and style are positively captured by the model, the generations are still weak on meter and on conveying enough emotion.

Paraphrasing. Concerning paraphrasing, the language model is conditioned to the reference sentence to generate the paraphrase. We proposed a novel methodology to automatically build large corpora of paraphrases, that can be exploited to train neural language models. The algorithm consists of a pipeline involving massive/focused crawling, extraction of linguistic features from documents and an alignment and selection procedure. We showed its effectiveness in the case of Italian language, obtaining overall a dataset with more than $85,000$ examples of paraphrases. The quality of the dataset has been evaluated by exploiting it to train a neural paraphrasing sequence-to-sequence model with pointer-based generator.

Chapter 6

Language Varieties

Understanding the evolution of a language is a challenging problem. What is its origin, how is it related to other languages, these are crucial questions that the study of language evolution attempts to answer. In this Chapter we study the evolution of Italian, a Romance language that has its roots in Vulgar Latin. Modern Italian was born in Tuscany around the 14th century, thanks to the works of Dante Alighieri, Francesco Petrarca and Giovanni Boccaccio, the most prominent authors of the middle age. Italy has been characterized by an high variety of dialects, which are often loosely related to each other, due to the past fragmentation of the territory. Italian has absorbed influences from many of these dialects, as also from other languages due to dominion of portions of the country by other nations, such as Spain and France. We study the evolution of Italian in its early stages, analyzing textual resources from authors of different regions, ranging in a time period between 1200 and 1600.

In summary, the contributions of this Chapter are: (1) a project, *Vulgaris*, that studied a text corpus consisting of vulgar Italian language literary resources, organized in such a way to ease language research, (2) the study of historical and geographical background, the statistical properties of the collected data and its composition and (3) a corpus-driven study in dialectology and diachronic varieties exploiting perplexity-based distances. In particular, we introduce Neural Language Models to estimate the perplexity and provide a new indicator, named as Perplexity-based Language Ratio (PLR), to analyse the historical evolution process of dialects and diachronic varieties.

The Chapter is organized as follows. Related works are summarized in Section 6.1. In Section 6.2, we describe in detail the characteristics of the data collected, including the historical background, its structure and several statistics. Then, in Section 6.3, we discuss about perplexity-based measures, that combined with neural language models (Section 6.4) are then used to carry out experiments on the diachronic varieties within *Vulgaris* in Section 6.1. Finally, we draw the conclusions

in Section 6.6.

6.1 Related Works

Natural Language Processing techniques are powerful tools that can support researchers in the analysis of dialects and diachronic language varieties (Zampieri and Nakov (2020); Ciobanu and Dinu (2020)). There exists several lines of research that approach the problem of defining distances between languages or varieties. Linguistic phylogenetics (Borin (2013)) aim at determining a rooted tree to describe the evolution of a group of languages or varieties. Trees are built based on the so called *lexicostatistics* technique, that takes into account words with common origin to determine a taxonomic organization of the languages. Language distance approaches instead rely on the distributional hypothesis of words and require cross-lingual corpora. Similarity is based on word co-occurrences (Asgari and Mofrad (2016); Liu and Cong (2013)), or using perplexity-based methods (Basile et al. (2016); Gamallo et al. (2017); Campos et al. (2018, 2020)). Perplexity (see Section 2.1) is estimated from Language Models, typically n-grams LMs of characters, trained on one corpus and evaluated on another variety.

Differently from previous work, we consider Neural Language Models (NLMs) (Bengio et al. (2003); Mikolov et al. (2010)), that as we have seen in Section 2.1 are more robust estimators, well known for their generalization capabilities and currently the state-of-the-art approaches in Language Modeling tasks. There is a vast literature on NLMs. Many works also address the problem of character language modeling (Jozefowicz et al. (2016); Hwang and Sung (2017)) or character-aware LMs (Marra et al. (2018); Kim et al. (2016)). We focus on Italian, a Romance language derived from Vulgar Latin. The uniquely fragmented political situation that occurred in Italy during the middle age makes Italian an extremely variegate and complex case of study, rich of dialects that are still spoken nowadays. We consider a corpus of medieval text collections, with the purpose of easing the research activity on diachronic varieties. Moreover, the dataset can be also a valid resource to study the problem of Text Generation in low-data and variegate styles conditions. Similar corpora have been already collected for other languages. Colonia (Zampieri and Becker (2013)), is a Portuguese diachronic dataset of about 5 milion tokens grouped by century in five sub-corpora. In (Campos et al. (2020)) the authors gathered three corpora for English, Spanish and Portuguese.

6.2 Data Collection

With the project *Vulgaris*[1] we aim at investigating the diachronic evolution and variance of the vulgar italian language. We collected a heterogeneous literary text corpus, comprehensive of poetry, prose, epistles and correspondence by the most important Italian authors ranging from the dawn of the vulgar language to the Reinassance Age. Henceforth, for compactness, we refer to such data as *Vulgaris*. The dataset represents a fundamental timeframe for the Italian language, including the first steps and diachronic evolutions departing from the Latin language. Moreover, through *Vulgaris* it is possible to gain evidence of the early language fragmentation deriving from the complex historical geo-political context of the Middle Age. We first summarize the hystorical background of Italy in Middle Age. Then we detail the structure of the corpus, and finally, guided by the results of the statistical analysis, we provide hints about how the vulgar spread.

6.2.1 Historical background

The earliest years of the 13th century were characterized by a novel and complex civilisation. The rise of medieval Communes, associations among citizens of towns belonging to the same social class, influenced the rise of a novel school of secular thought increasingly unhindered by the religious influences. For these reasons, along with the establishment of the first universities, beside latin literature the vulgar Italian language started to appear in various literary works. The heterogeneous political and geographical context led to a linguistic fragmentation, characterized by various contact points. The first literary evidence of vulgar poetic, which we denote as belonging to the **Archaic text** family, is a collection of verses still connected with religious and moral themes, written in regions of the central Italy, in particular Umbria and Tuscany. Amongst the main authors, we mention *Francesco d'Assisi*. Inspired by this works, in the middle of the century (about 1250) some vulgar authors (e.g. *Jacopone da Todi* et al.), in the same geographical zone, composed several **Laude**, enriching the religious and mystical poetry theme.

The **Northern Didactic poetry** family, flourished in the same years, was influenced by these religious and moral guidelines. We point out *Bonvesin da la Riva* from Milan and *Giacomino da Verona* among the representatives, having the goal to instruct readers about morality, philosophy and doctrine.

The prosperous and thriving Imperial court of Federico II fostered the birth of a **Sicilian School** (1230-1250), where the figure of an angelic woman and the stereotype of love play a central role. This group laid the foundations of modern poetry, introducing a specific metric and organization in *Stanzas*, creating Sonnets and uni-

[1]The project is available at `https://sailab.diism.unisi.it/vulgaris/`.

fying the language lexicon and structure. Among the authors we highlight *Giacomo da Lentini* and *Pier della Vigna*.

With the death of Federico II, the cultural axis moved to Tuscany, thanks to the proliferation of Communes. Differently from the Sicilian School we cannot talk of a unique literary school. Indeed, in several important cities, such as Pisa, Lucca, Arezzo, Siena, other than Florence, which became only afterwards the most important cultural center, emerged themes inspired by the Sicilian similar ones.

The **Northern/Tuscan Courtly poetry** arises from poets belonging to the Sicilian school who moved after the decadence of the Svevian Empire, influencing the themes and style of local authors (*Guittone D'Arezzo*, *Bonagiunta Orbicciani*, *Compiuta Donzella*). In the meanwhile, **Central Italy Didactic poetry** (*Brunetto Latini*) and **Realistic Tuscan poetry** (*Cecco Angiolieri, Folgòre da San Gimignano, Cenne de la Chitarra*) emerged, differentiated by the themes, goal of the poetry, and style. Departing from the literature inspired by court life, a more popular and playful genre, the **Folk and Giullaresca Poetry**, was mainly due to jesters such as *Ruggieri Apuliese*.

Finally, thanks to the influence of Sicilian School and Tuscan poetry, the **Stilnovisti** family (*Guido Guinizzelli*, *Dante Alighieri*, *Guido Cavalcanti*, *Lapo Gianni*, *Gianni Alfani*, *Dino Frescobaldi*, *Cino da Pistoia*) and some authors close to them (**Similar to Stilnovisti** - *Lippo Pasci de Bardi*) evolved and refined the poetry of their predecessors. Metaphors, a noble symbolism and introspection characterize this movement, which was born in Bologna and developed in Florence reaching its climax.

Boccaccio and **Petrarca** compose, together with Dante, the three *Crowns* of Italian literature. Their poetry and prose are inspired by *Dolce Stil Novo*, with an evolution toward a more wordily thematic, rather than spiritual. Their linguistic style is the offspring of an evolved society.

The works developed by these families highly influenced following authors. In particular, **Ariosto** and **Tasso**, at the beginning of the 16th century, were deeply inspired by Petrarca and the Stilnovisti, respectively. However, their different temporal context was reflected in their literary works.

Therefore, the vulgar Italian language, starting from the beginning of the 13th century, became more and more popular amongst various authors, evolving during the following years in several families, which we summarize in Figure 6.1. The highly fragmented geo-political context gave rise to different schools, groups, communities (depicted in Figure 6.1) and hence many language varieties, dialects, that even nowadays are noticeable.

Through the Vulgaris project, we aim to provide a rich resource to analyze the diachronic evolution of the early Italian language, in particular in poetry, prose and correspondence texts.

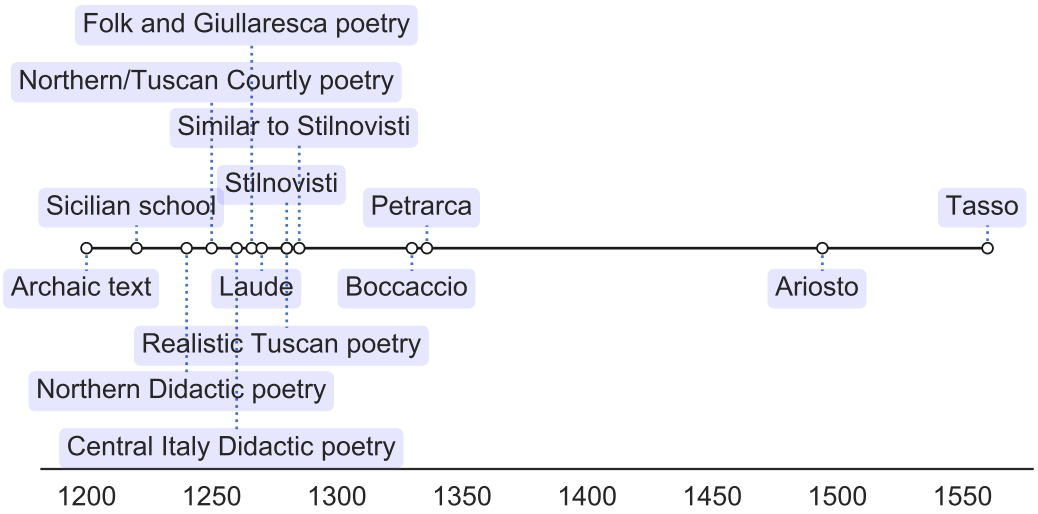

Figure 6.1: Timeline representing the temporal sequence of the different families we described in the main text.

6.2.2 Dataset structure

The examined corpus contains texts retrieved from Biblioteca Italiana[2], a digital library project collecting the most significant texts of the Italian literature, ranging from the Middle Age to the 20th century. The code to retrieve and analyse the data can be found in `https://github.com/sailab-code/vulgaris`. *Vulgaris* provides the following filtered type of information, extracted from the parsed data: **author**, **title**, **collection**, **family**, **type**, **text**. In details, the corpus is composed by **text** produced by 104 **authors** belonging to the 14 **families** described in Section 6.2.1. The corpus contains 177 *collections*, consisting of groups of poetry, single poems, personal epistles. Each item of the dataset is a single composition, for instance a poetry, a chapter or a letter. Moreover, we split the resources by the **style** attribute into *poetry* and *prose*, with the latter containing both the prose and the correspondence documents.

The structure of a poetic composition represents an important information in tasks such as Poem Generation (Lau et al. (2018); Zugarini et al. (2019); Zhang and Lapata (2014)). The verse organization of the poetry is encoded by tags denoting each line break `<EOL>` (end of a verse), as well as the end of each stanza `<EOS>`. In the case of prose, only the organization in paragraphs is represented by the tag `<EOS>`.

6.2.3 Statistical Analysis

The dataset is analyzed through simple statistics to unravel the heterogeneity of the collection. In Table 6.1 we report some statistics on the families (first column, or-

[2] `http://bibliotecaitaliana.it/`

Table 6.1: Analysis of the composition of the dataset. We report the families, their most representative authors, total number of provided texts and their distribution in poetry and prose.

Family	*Authors*	*#Texts*	*#Poetry*	*#Prose*
Archaic text	Francesco d'Assisi, Ritmo Laurenziano, Ritmo Cassinese	5	5	-
Sicilian School	Giacomo da Lentini, Guido delle Colonne, Pier della Vigna, Pronotaro da Messina	46	46	-
Northern Didactic poetry	Girardo Patecchio Da Cremona, Bonvesin Da La Riva, Giacomino Da Verona, Anonimo Genovese	29	29	-
Northern/Tuscan Courtly poetry	Guittone D'Arezzo, Bonagiunta Orbicciani, Chiaro Davanzati, Monte Andrea Da Firenze	101	101	-
Central Italy Didactic poetry	Brunetto Latini,Garzo, Detto Del Gatto Lupesco, Dal Bestiario Moralizzato Di Gubbio	8	8	-
Folk and Giullaresca poetry	Ruggieri Apugliese, Castra Fiorentino, Matazone Da Caligano, Rime Dei Memoriali Bolognesi	23	23	-
Laude	Jacopone Da Todi, Laude Cortonesi, Lauda Dei Servi Della Vergine	41	41	-
Stilnovisti	Guido Guinizzelli, Guido Cavalcanti, Cino da Pistoia, Dante Alighieri, Lapo Gianni	769	704	65
Realistic Tuscan poetry	Rustico Filippi, Cecco Angiolieri, Folgore da San Gimignano, Cenne de la Chitarra	69	69	-
Similar to Stilnovisti	Dante's Friend, Lippo Pasci de' Bardi	709	70	-
Boccaccio	-	1,058	296	762
Petrarca	-	872	747	125
Ariosto	-	363	144	219
Tasso	-	3,366	1,604	1,762
Total	-	6,820	3,887	2,933

dered by date), including their most representative authors (second column), the total amount of collected texts, divided into poetry and prose (third, fourth and fifth column, respectively). Whilst the older families are underrepresented, families belonging to a later period are mostly characterized by a larger amount of texts. This fact is a good indicator of the diffusion that the Italian language has undergone during this timeline.

The corpus investigated in *Vulgaris* is extremely heterogeneous and composed by 4 million word occurrences, whose texts have been written by authors from a wide range of geographical regions and time periods, as shown in Figure 6.1. In Table 6.2 we summarize some statistics on the total amount of word occurrences, the number of unique words and the average occurrences per word for each text **type**. The total number of words in poetry and prose is almost balanced, whereas their composition is remarkably different. Indeed, poetry has a richer lexicon than prose, containing almost twice unique words.

To depict the contribution of each family to the dataset, Figure 6.2 reports the total number of word occurrences for each family and the poetry/prose proportion. Once more, these statistics confirm the increasing spread of the Italian language. We can also notice how vulgar spread. Initially, it was mainly used in poetry and only later vulgar prosaic forms appeared. Only 5 out of 14 families contain prose, and, as we can see from the timeline in Figure 6.1, they correspond to the latest families.

Table 6.2: Statistics on words for each text category. The distribution of text is almost balanced between *Poetry* and *Prose* (# word occurrences). However, *Poetry* has a richer lexicon than prose, with almost two times unique words.

	Global	*Poetry*	*Prose*
# word occurrences	4,090,166	1,925,838	2,164,328
# unique words	180,450	136,195	69,135
Avg occurrences per word	22.67	14.14	31.31

Finally, in the top row of Figure 6.3, we report the average distribution of the text length, in both the styles (i.e, *poetry* on the left and *prose* on the right) among all the families. The bottom row of Figure 6.3 shows the average number of words contained in each collection, hence texts having similar characteristics or theme.

6.3 Perplexity-based Language Measures

We have already seen in Section 2.1 that perplexity is exploited to evaluate the quality of a language model. Perplexity $pp(p, \mathcal{D})$ is a function of the probability distribution p and the reference corpus $\mathcal{D}$. When the evaluation set $\mathcal{D}$ is fixed, we can compare different language models and select which one is best. Analogously, we

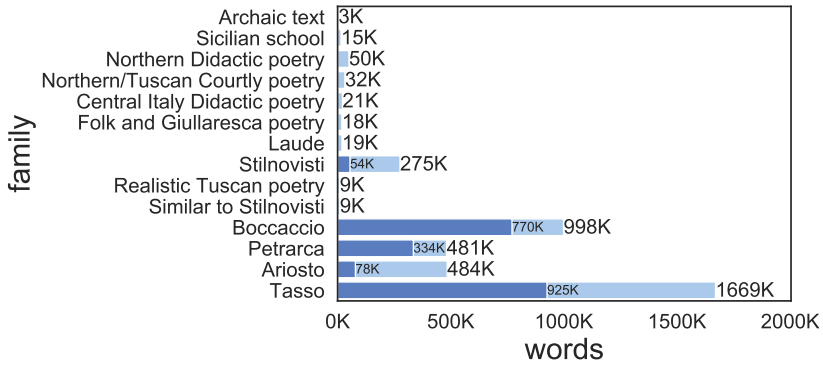

Figure 6.2: The figure reports the total amount of word occurrences for each family (both in poetry and prose), at the right of each family bin. The darker blue bar denotes the portion of word occurrences in prose texts only.

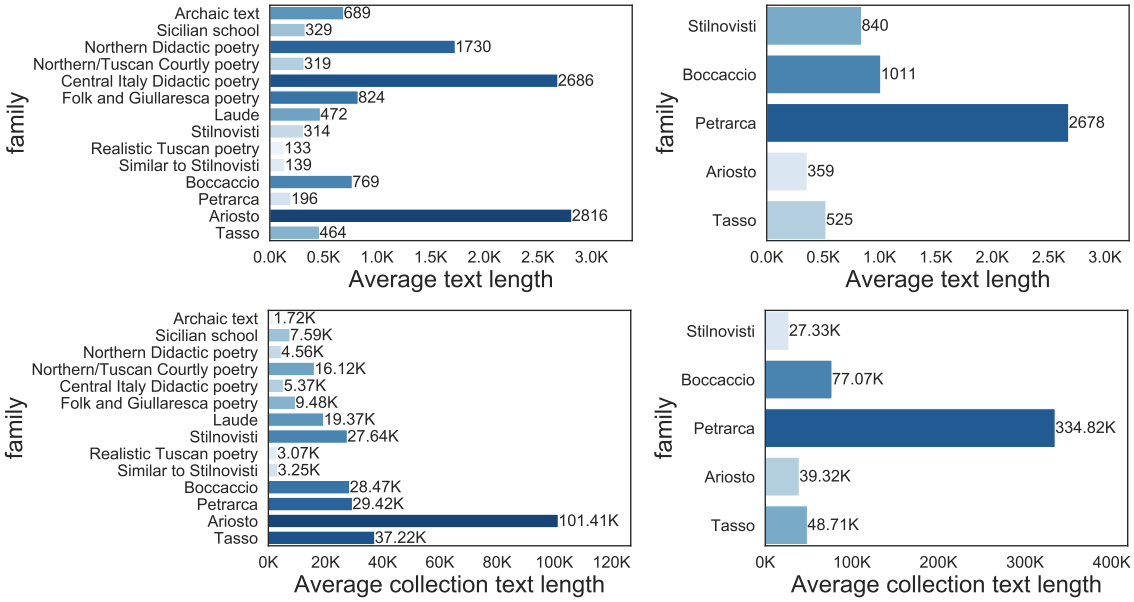

Figure 6.3: In the top row, the average text length of poetry (left) and prose (right) texts. On the bottom, average number of words contained in each collection in poetry (left) and prose (right).

can fix the language model and change the evaluation corpus. Comparing the perplexity on different datasets for the same LM gives to us indications of which is the corpus most similar to the distribution estimated by the model. Since the distribution of language models is a direct consequence of the data used to estimated it, we can argue that if a language model trained on $\mathcal{D}_1$ has a low perplexity when evaluated on $\mathcal{D}_2$, then $\mathcal{D}_1$ and $\mathcal{D}_2$ are related. Hence we can use perplexity and language models to provide a distance between language corpora. This idea has been already

exploited in (Gamallo et al. (2017)), and this approach has been effectively applied for language discrimination and the analysis of historical varieties (Campos et al. (2018, 2020)).

Perplexity-based Language Distance. Let us consider two language corpora, namely $\mathcal{L}_1$ and $\mathcal{L}_2$, and let $\texttt{LM}_{\mathcal{L}_1}, \texttt{LM}_{\mathcal{L}_2}$ be two language models trained on $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. We can argue that the more the corpora are related to each other, the more accurate is the estimate provided by the LM trained on one language when evaluated on the other. By denoting the two measures of perplexity as $pp_{\mathcal{L}_1 \to \mathcal{L}_2}(\mathcal{L}_2, \texttt{LM}_{\mathcal{L}_1})$ and of $pp_{\mathcal{L}_2 \to \mathcal{L}_1}(\mathcal{L}_1, \texttt{LM}_{\mathcal{L}_2})$, the Perplexity-based Language Distance (PLD) is defined in (Gamallo et al. (2017)) as the average of these two values:

$$PLD(\mathcal{L}_1, \mathcal{L}_2) := \frac{pp_{\mathcal{L}_1 \to \mathcal{L}_2}(\mathcal{L}_2, \texttt{LM}_{\mathcal{L}_1}) + pp_{\mathcal{L}_2 \to \mathcal{L}_1}(\mathcal{L}_1, \texttt{LM}_{\mathcal{L}_2})}{2}. \tag{6.1}$$

Perplexity-based Language Ratio. PLD copes with the fact that $pp_{\mathcal{L}_1 \to \mathcal{L}_2}(\mathcal{L}_2, \texttt{LM}_{\mathcal{L}_1})$ and $pp_{\mathcal{L}_2 \to \mathcal{L}_1}(\mathcal{L}_1, \texttt{LM}_{\mathcal{L}_2})$ are not symmetric, mostly because LMs are trained and tested on different data distributions. However, the asymmetry in these values can be a good indicator of the language evolution on diachronic/dialect varieties, since it can enlighten either a language compression/simplification or a language expansion over time. Indeed, in the process of language unification, words are reduced, and dialectal expressions are suppressed, thus reducing the overall richness of the language. Hence, we introduce a novel measure, namely, Perplexity-based Language Ratio (PLR):

$$PLR(\mathcal{L}_1, \mathcal{L}_2) := \frac{pp_{\mathcal{L}_1 \to \mathcal{L}_2}(\mathcal{L}_2, \texttt{LM}_{\mathcal{L}_1})}{pp_{\mathcal{L}_2 \to \mathcal{L}_1}(\mathcal{L}_1, \texttt{LM}_{\mathcal{L}_2})}. \tag{6.2}$$

PLR values greater than 1 indicate that $\mathcal{L}_1$ is likely to be a more various language than $\mathcal{L}_2$, whereas values less than 1 are likely to indicate $\mathcal{L}_2$ as the more complex language.

6.4 Conditional Language Modeling

In this section we briefly describe the structure of the language models that have been exploited in the experimental evaluation. We extend Equation 2.2 by adding other features of the text to condition the NLM. In particular, we leverage the external meta information about author a, family f and kind of composition k (prose or poetry) available in the dataset. Hence Equation 2.2 becomes:

$$p(\boldsymbol{x}) = \prod_{i=1}^{m} p(x_i | x_{i-1}, \ldots, x_1, a, f, k), \tag{6.3}$$

where $\boldsymbol{x}$ is a sequence of characters.

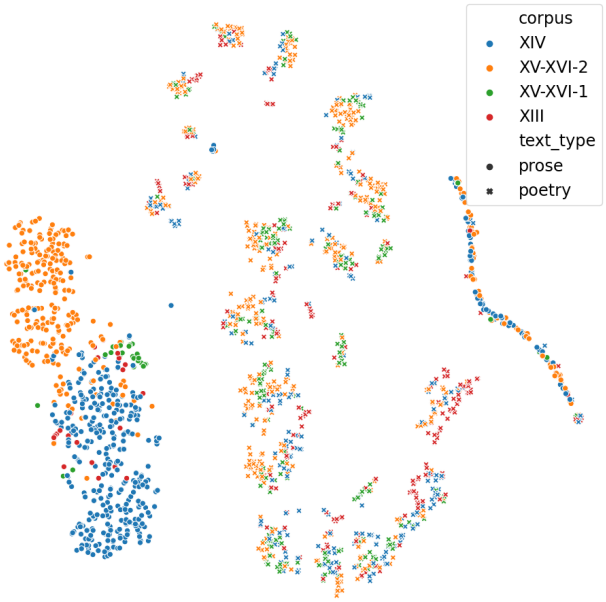

Figure 6.4: Two dimensional t-SNE representation of sentences' state h_T. Different colours indicate different groups, dots for poetry, crosses for prose.

We model the distribution in Equation 6.3 by means of a recurrent neural network. Each token from the vocabulary V of size $|V|$ is associated to a latent embedding $\boldsymbol{e}$ of dimension d. The set of the $|V|$ embeddings are collected in the $|V| \times d$ matrix $\mathbf{E}$. In particular, we consider an LSTM. At time t the state $\boldsymbol{h}_t$ is updated as follows,

$$\boldsymbol{h}_t = \texttt{LSTM}(\boldsymbol{e}_t, \boldsymbol{h}_{t-1}) \,, \tag{6.4}$$

The external features (a, f, k) are concatenated to $\boldsymbol{h}_t$ and then linearly projected into a d-dimensional vector $\boldsymbol{s}_t$:

$$\begin{aligned} \boldsymbol{c}_t &= [\boldsymbol{h}_t, \boldsymbol{a}, \boldsymbol{f}, \boldsymbol{k}], \\ \boldsymbol{s}_t &= W \cdot \boldsymbol{c}_t + b, \end{aligned}$$

where , is the concatenation operator, and $\boldsymbol{a}, \boldsymbol{f}, \boldsymbol{k}$ the embedding representations associated to author a, family f and kind k, respectively. The probability distribution $\hat{\boldsymbol{y}}_t$ is the output of a softmax layer sharing the weights of the input embeddings to apply a back-projection of the contextual state $\boldsymbol{s}_t$ into the vocabulary space:

$$\begin{aligned} \boldsymbol{o}_t &= \mathbf{E}^T \cdot \boldsymbol{s}_t, \\ \hat{\boldsymbol{y}}_t &= \texttt{softmax}(\boldsymbol{o}_t). \end{aligned}$$

The PLD and PLR are estimated exploiting this conditional neural language model where input tokens are characters. We chose NLMs over n-grams because of their

notorious generalization capabilities, as discussed in Section 2.1. A more robust estimation of LMs will improve the quality of the PLD and PLR measures. The same kind of architecture is used to build and visualize the sequence representations shown in Figure 6.4, learnt from a word-based language model on the entire *Vulgaris*.

6.5 Experiments

Text collected in *Vulgaris* spans over a time period of about four centuries. In the experiments we analyse the diachronic varieties within the dataset using perplexity-based distances, PLD and our novel PLR, subdividing it into groups of different centuries.

The 14 families of *Vulgaris* are arranged in four language corpora, based on their time periods, as shown in Figure 6.1. The first group, referred to as XIII, includes all the families belonging to the 13th century. In this century there are 10 out of 14 families of the dataset, making this language variety the most heterogeneous one, including many authors from different areas of the Italian territory. In the second one (XIV), we consider *Petrarca* and *Boccaccio* families/authors, whereas *Ariosto* and *Tasso* constitute the third and forth corpora, respectively, XV-XVI-1 and XV-XVI-2. Clearly the boundaries are not neat, since the activity of some authors may span across two centuries. From Table 6.3 we can see that the diachronic corpora are unbalanced. Despite the high number of families and authors, the XIII corpus is the less represented one, followed by XV-XVI-1 that is slightly larger. They both are small compared to XIV and XV-XVI-2. However, XIII is also the dataset with lowest average number of occurrences per word, indicating a high variance of the collection caused by the rich variety of styles and authors.

Table 6.3: Number of words and proportions between the four diachronic groups.

	XIII	XIV	XV-XVI-1	XV-XVI-2
# words	455,583	1,480,379	484,276	1,669,928
dataset proportion (%)	11.14	36.19	11.84	40.83
# unique words	57,343	73,530	42,594	72,369
Avg occurrences per word	7.94	20.13	11.37	23.08

Qualitative results. As a first qualitative analysis, we trained a word-based conditional NLM on the entire corpus, using a vocabulary of $50,000$ words. The final cell state h_T of a text sequence $x = (x_1, \cdots, x_T)$ is projected to a 2-dimensional representation using t-SNE (Maaten and Hinton (2008)). Figure 6.4 visualizes the

2-d representation of 2,000 examples, colored accordingly to the corpus they belong to, and styled differently in case of prose or poetry works. Prose and poetry are easily discriminated by the NLM. The corpus provenance is also captured by the NLM, although not completely, suggesting that the diachronic varieties share a similar structure.

Perplexity-based analysis. Then, both the PLD and the PLR described in Section 6.4 are computed for each pair of corpora. For the character LMs, we consider input character sequences with a maximum length of 50. The state h_t has size 256, with $(\boldsymbol{a}, \boldsymbol{f}, \boldsymbol{k})$ of size 16, 16 and 32, respectively. Special tokens delimiting end of sentence, end of verse and white space are included in the vocabulary of characters. For each $\mathcal{L}_i \rightarrow \mathcal{L}_j$, the network is trained on 90% of the $\mathcal{L}_i$ corpus, whereas the remaining 10% is used for early stopping, and it is finally evaluated on the whole $\mathcal{L}_j$. Results

Table 6.4: PLD among pairs of diachronic language varieties. Each element (i, j) in the table corresponds to $PLD(\mathcal{L}_i, \mathcal{L}_j)$.

	XIII	XIV	XV-XVI-1	XV-XVI-2
XIII	3.90	5.38	5.99	6.08
XIV	5.38	3.52	4.76	4.65
XV-XVI-1	5.99	4.76	3.30	4.47
XV-XVI-2	6.08	4.65	4.47	3.28

are shown in tables 6.4 and 6.5. PLD is lower in diachronic varieties closer in time, as expected. Interestingly enough, PLR highlights a strong asymmetric behaviour on perplexity pairs involving the set XIII. Indeed, while training a language model on a heterogeneous corpus, as it is XIII, makes the LM well performing when testing on simpler varieties, a language model trained on a poorer corpus underperforms when evaluating it on a richer corpus, as XIII.

Table 6.5: PLR among pairs of diachronic language varieties. Each element (i, j) in the table corresponds to $PLR(\mathcal{L}_i, \mathcal{L}_j)$.

	XIII	XIV	XV-XVI-1	XV-XVI-2
XIII	1.00	0.81	0.65	0.72
XIV	**1.23**	1.00	0.86	0.95
XV-XVI-1	**1.53**	1.16	1.00	1.14
XV-XVI-2	**1.39**	1.05	0.88	1.00

6.6 Discussion

In this chapter we analyzed a collection of literary texts covering the production of Italian authors mainly from the middle age. The dataset contains both poetry and prose, and each document is enriched by metadata that provide both information on the text characteristics and structure (the verse and stanza organization for poems and the paragraph splitting for prose). To measure the distance between different language varieties we exploit language models in conjunction of perplexity-based measures. To the best of our knowledge, we are the first to employ neural language models for the analysis of language varieties with perplexity-based metrics. We make use of Perplexity-based Language Distance and a novel asymmetric indicator that we defined as Perplexity-based Language Ratio (PLR). Our findings on the dataset analysis give some insights on the main features of the collection, that reflect the dialectical and diachronic properties of Italian language in its early stages.

Chapter 7

Conclusions

This Chapter summarizes the work presented in this thesis. We point out the main contributions and then, we discuss about some possible future research directions for extending and unifying our work.

7.1 Summary

In this Section, we summarize all the findings of this thesis. In the previous Chapters we have faced some crucial aspects of language understanding and generation.

Character-aware representations. We have presented a character-aware neural model that develops task-independent representations of words and contexts by learning from an unsupervised language modeling-related task. The character-level information makes the embeddings more robust to morphological variations and noise caused by misspelling, which may be significant in real world scenarios, such as conversational agents. The obtained representations were exploited as input features for language understanding tasks, achieving competitive results, despite the small number of learnable parameters.

Information Extraction in text streams. The character-level encoder also constitute the base of an end-to-end model that processes text streams to extract entities and relations. This agent operates in an online scenario. It detects mentions to entities and relations and then disambiguates them by aligning the instances to its (not-given-in-advance) internal KB. We have shown that our model is capable of learning with sparse supervisions and self-learning. It demostrated strong disambiguation and discovery skills when tested on a stream of sentences organized into small stories[1], even when a few, sparse supervisions are provided. We also showed how it can improve its skills by continuously reading text.

[1] We also created a new synthetic dataset that we publicly made available for further studies.

Neural poetry. We proposed a syllable-based neural language model to generate tercets with the style of a given author. This general approach has been studied in the context of Dante Alighieri, a medieval Italian poet, famous for being the Divine Comedy's author. Syllable tokenization, that is a natural choice for poetry, allows also transfer learning from large scale collections of modern Italian by means of a multi-stage transfer learning technique. In addition, we devise a scoring mechanism to select only the tercets that are more coherent with Divine Comedy's meter and rhyme and the author's style. The multi-stage training procedure and the scoring mechanism together improve the quality of the content, that, despite the poorness of author's data, let our model create original and realistic tercets.

Paraphrasing. Paraphrasing can be formulated as an NLG problem (Section 2.3), i.e. a language model conditioned by an input sequence, that is nicely modeled by sequence-to-sequence architectures. Unfortunately, unlike other sequence-to-sequence problems (e.g. Machine Translation, Text Summarization), there are few datasets with aligned pairs of paraphrases, especially for not-English languages. Hence, we proposed a novel algorithm to automatically build large corpora of paraphrases by crawling news from the Web. The method follows a pipeline involving the extraction of linguistic features from the crawled documents and an alignment and selection procedure. We showed its effectiveness in the case of Italian language, collecting overall more than $85,000$ examples of paraphrases. The quality of the dataset has been evaluated by exploiting it to train a neural paraphrasing sequence-to-sequence model. In particular, we implemented a pointer-based generator.

Language Varieties. So far, we have used textual corpora to train language models. Vice versa, language models are a valuable asset to analyze the corpora themselves. Our last contribution is the definition of a perplexity-based (asymmetric) indicator, namely Perplexity-based Language Ratio (PLR), that can be used to measure the distance between different language varieties, outlining the evolution of diachronic or dialectical varieties. Perplexity is computed from neural language models. We carry out the analysis of a collection of literary texts covering the production of Italian authors mainly from the middle age. Our findings on the dataset analysis give some insights on the main features of the collection, that reflect the dialectical and diachronic properties of Italian language in its early stages.

7.2 Future Works

Transformers. All the language models that we have used in this thesis for developing representations of text, extracting information, generating language, analyze language varieties, are implemented by means of Recurrent Neural Networks. How-

ever, recently Transformers (Vaswani et al. (2017)) have consistently outperformed RNNs in the vast majority of NLP problems, becoming the new state-of-the-art in NLP community. Transformers are attention-based models that allow a faster and more effective processing of data sequences w.r.t. Recurrent Neural Networks, overcoming the inherent sequentiality of RNNs' computations. Our findings remain intact regardless of the neural architecture underneath. Therefore, straightforward next steps will consist in replacing RNNs with transformer-based architectures. In particular, we can substitute language encoders (Chapter 3 and Chapter 4) with BERT-like models (Devlin et al. (2018); Lan et al. (2019)). In (Ma et al. (2020)), authors already proposed a character-aware pretrained language model that is similar to what we presented in Chapter 3, but exploiting a BERT architecture. Concerning language generation, language modeling can be estimated with models like GPT (Radford et al. (2018)), GPT-2 (Radford et al. (2019)), T5 (Raffel et al. (2019)).

Controlling text generation in open-ended domains. We have seen that open-ended generation problems loosely correlate with the source of information provided as input. As a consequence, models in open-ended domains generate text freely, without any (formal, semantic) restrictions. Models like GPT-2 (Radford et al. (2019)) have achieved impressive results in generation, producing sentences or even paragraphs almost indistinguishable from the ones written by humans. However, the lack of control harms the usability of these models in actual business use-cases. This issue is also clear in the context of Poem Generation, where text must respect predefined meter and rhyming rules. The scoring mechanism devised in the poem generator of Section 5.2 is a preliminary attempt toward the control of generated text. However, one could think of more sophisticated solutions. For instance, such constraints may be injected directly into the model, either during learning as reward functions of Reinforcement Learning (Sutton and Barto (2018)) algorithms and/or by enriching the LM input with information that biases significantly the generation. Alternatively, revising strategies could be devised in order to make a model compliant with some writing rules.

Toward a unified model for understanding and generation. The final future direction that we believe is worth to explore is the design of a unified model with strong understanding and generation capabilities. Research efforts, including the contributes of this thesis are all aimed at improving single aspects of NLP, instead of focusing on the entire picture. Some of the contributes of this thesis pose the basis for the development of agents capable of learning through reading, reason on top of the knowledge acquired, and provide coherent, accurate answers according to such knowledge and the agent's intent. Indeed, we plan to extend the system proposed in this Chapter 4 with a more structured and advanced knowledge component that

would allow to exploit symbolic reasoning. Furthermore, we expect to make use of such knowledge as a way to ground the language generation of the agent.

Appendix A

Publications

Journal papers

1. Marco Maggini, Giuseppe Marra, Stefano Melacci, **Andrea Zugarini**, "Learning in Text Streams: Discovery and Disambiguation of Entity and Relation Instances", *IEEE Transactions on Neural Networks and Learning Systems*, , 2019. **Candidate's contributions**: joint definition of the problem and method, design and implementation of the agent, joint design and implementation of the experiments.

2. Ottavia Spiga, Vittoria Cicaloni, Andrea Zatkova, Lia Millucci, Giulia Bernardini, Andrea Bernini, Barbara Marzocchi, Monica Bianchini, **Andrea Zugarini**, Alberto Rossi and others, "A new integrated and interactive tool applicable to inborn errors of metabolism: Application to alkaptonuria", *Computers in biology and medicine*, pages:1–7, 2018. **Candidate's contributions**: correlation Analysis, development of AKU database and web interface.

Peer reviewed conference papers

1. **Andrea Zugarini**, Stefano Melacci, Marco Maggini, "Neural Poetry: Learning to Generate Poems Using Syllables", *International Conference on Artificial Neural Networks*,pages:313–325,2019. **Candidate's contributions**: collection of the data, definition of method, design of the multi-stage pre-training steps, implementation of the neural model and experiments, joint design of experiments and evaluations.

2. Giuseppe Marra, **Andrea Zugarini**, Stefano Melacci, Marco Maggini, "An unsupervised character-aware neural approach to word and context representation learning", *International Conference on Artificial Neural Networks*, pages:126–136, 2018. **Candidate's contributions**: joint definition of the method, design

and implementation of the agent, joint design and implementation of the experiments.

3. Achille Globo, Antonio Trevisi, **Andrea Zugarini**, Leonardo Rigutini, Marco Maggini, Stefano Melacci, "Neural Paraphrasing by Automatically Crawled and Aligned Sentence Pairs", *2019 Sixth International Conference on Social Networks Analysis, Management and Security* (*SNAMS*), pages:429–434, 2019. **Candidate's contributions**: design and implementation of paraphrasing experiments.

Workshop papers

1. **Andrea Zugarini**, Matteo Tiezzi, Marco Maggini, "Vulgaris: Analysis of a Corpus for Middle-Age Varieties of Italian Language", *Seventh Workshop on NLP for Similar Languages, Varieties and Dialects*,pages:150–159, 2020. **Candidate's contributions**: joint statistical analysis of the corpus, joint design of the experiments, implementation of the experiments, design of perplexity-based indicator.

2. **Andrea Zugarini**, Jérémy Morvan, Stefano Melacci, Stefan Knerr, Marco Gori. "Combining deep learning and symbolic processing for extracting knowledge from raw text", *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pages:90–101, 2018. **Candidate's contributions**: joint design of model and algorithm, implementation and execution of the experiments.

Papers under review

1. **Andrea Zugarini**, Enrico Meloni, Alessandro Betti, Andrea Panizza, Marco Corneli, Marco Gori. "An Optimal Control Approach to Learning in SIDARTHE Epidemic model", submitted to *IEEE Transactions on Neural Networks and Learning Systems*. **Candidate's contributions**: joint definition of the learning task, joint design of model and algorithm, joint implementation and execution of the experiments.

Other

1. Francesco Giannini, Vincenzo Laveglia, Alessandro Rossi, Dario Zanca, **Andrea Zugarini**, "Neural networks for beginners. A fast implementation in matlab, torch, tensorflow", *arXiv preprint arXiv:1703.05298*, 2017. **Candidate's contributions**: designed and illustrated algorithms by examples in torch.

2. Vittoria Cicaloni, **Andrea Zugarini**, Alberto Rossi, Matteo Zazzeri, Annalisa Santucci, Andrea Bernini, Ottavia Spiga,"Towards an integrated interactive

database for the search of stratification biomarkers in Alkaptonuria", *PeerJ Preprints*, 2016. **Candidate's contributions**: correlation Analysis, development of AKU database and web interface.

Bibliography

Aggarwal, C. C. and Zhai, C. (2012). *Mining text data*. Springer Science & Business Media.

Ahn, S., Choi, H., Pärnamaa, T., and Bengio, Y. (2016). A neural knowledge language model. *arXiv preprint arXiv:1608.00318*.

Akbik, A., Blythe, D., and Vollgraf, R. (2018). Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649.

Alighieri, D., Sisson, C., Sisson, C., and Higgins, D. (1998). *The Divine Comedy*. Oxford University Press. Oxford University Press.

Asgari, E. and Mofrad, M. R. (2016). Comparing fifty natural languages and twelve genetic languages using word embedding language divergence (weld) as a quantitative measure of language distance. *arXiv preprint arXiv:1604.08561*.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Banerjee, A. and Basu, S. (2007). Topic models over text streams: A study of batch and online unsupervised learning. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 431–436. SIAM.

Banerjee, S. and Lavie, A. (2005). Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72.

Bansal, T., Neelakantan, A., and McCallum, A. (2017). Relnet: End-to-end modeling of entities & relations. *arXiv:1706.07179*.

Barzilay, R. and Lee, L. (2003). Learning to paraphrase: an unsupervised approach using multiple-sequence alignment. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human*

Language Technology-Volume 1, pages 16–23. Association for Computational Linguistics.

Barzilay, R. and McKeown, K. R. (2001). Extracting paraphrases from a parallel corpus. In *Proceedings of the 39th annual meeting of the Association for Computational Linguistics*.

Basile, P., Caputo, A., Luisi, R., and Semeraro, G. (2016). Diachronic analysis of the italian language exploiting google ngram. *CLiC it*, page 56.

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

Bengio, Y., Simard, P., Frasconi, P., et al. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Berant, J., Chou, A., Frostig, R., and Liang, P. (2013). Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

Berant, J. and Liang, P. (2014). Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1415–1425.

Bolshakov, I. A. and Gelbukh, A. (2004). Synonymous paraphrasing using wordnet and internet. In *Int. Conf. on Application of Natural Language to Information Systems*, pages 312–323. Springer.

Borin, L. (2013). The why and how of measuring linguistic differences. *Approaches to measuring linguistic differences, Berlin, Mouton de Gruyter*, pages 3–25.

Campos, J. R. P., Gamallo, P., and Alegria, I. (2018). Measuring language distance among historical varieties using perplexity. application to european portuguese. In *Proceedings of the Fifth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial 2018)*, pages 145–155.

Campos, J. R. P., Otero, P. G., and Loinaz, I. A. (2020). Measuring diachronic language distance using perplexity: Application to english, portuguese, and spanish. *Natural Language Engineering*, 26(4):433–454.

Chiu, J. P. and Nichols, E. (2016). Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370.

Chopra, S., Auli, M., and Rush, A. M. (2016). Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the NAACL: Human Language Technologies*, pages 93–98.

Christakopoulou, K., Radlinski, F., and Hofmann, K. (2016). Towards conversational recommender systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 815–824. ACM.

Ciobanu, A. M. and Dinu, L. P. (2020). Automatic identification and production of related words for historical linguistics. *Computational Linguistics*, 45(4):667–704.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Colton, S., Goodwin, J., and Veale, T. (2012). Full-face poetry generation. In *ICCC*, pages 95–102.

Del Corso, G. M., Gulli, A., and Romani, F. (2005). Ranking a stream of news. In *Proceedings of the 14th international conference on World Wide Web*, pages 97–106. ACM.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dolan, B., Quirk, C., and Brockett, C. (2004). Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proceedings of the 20th international conference on Computational Linguistics*, page 350. Association for Computational Linguistics.

Dredze, M., McNamee, P., Rao, D., Gerber, A., and Finin, T. (2010). Entity disambiguation for knowledge base population. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 277–285. Association for Computational Linguistics.

Fader, A., Soderland, S., and Etzioni, O. (2011). Identifying relations for open information extraction. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1535–1545. Association for Computational Linguistics.

Fan, A., Lewis, M., and Dauphin, Y. (2018). Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*.

Gamallo, P., Campos, J. R. P., and Alegria, I. (2017). A perplexity-based method for similar languages discrimination. In *Proceedings of the fourth workshop on NLP for similar languages, varieties and dialects (VarDial)*, pages 109–114.

Ganitkevitch, J., Van Durme, B., and Callison-Burch, C. (2013). Ppdb: The paraphrase database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 758–764.

Globo, A., Trevisi, A., Zugarini, A., Rigutini, L., Maggini, M., and Melacci, S. (2019). Neural paraphrasing by automatically crawled and aligned sentence pairs. In *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 429–434. IEEE.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Grice, H. P. (1975). Logic and conversation. In *Speech acts*, pages 41–58. Brill.

Gu, J., Lu, Z., Li, H., and Li, V. O. (2016). Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*.

Guo, S., Chang, M.-W., and Kiciman, E. (2013). To link or not to link? a study on end-to-end tweet entity linking. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1020–1030.

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS*, pages 297–304.

Hachey, B., Radford, W., Nothman, J., Honnibal, M., and Curran, J. R. (2013). Evaluating entity linking with wikipedia. *Artificial intelligence*, 194:130–150.

Han, X. and Sun, L. (2011). A generative entity-mention model for linking entities with knowledge base. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 945–954. Association for Computational Linguistics.

Harari, Y. N. (2014). *Sapiens: A brief history of humankind*. Random House.

Hasan, S. A., Lee, K., Datla, V., Qadir, A., Liu, J., Farri, O., et al. (2016). Neural paraphrase generation with stacked residual lstm networks. In *International Conference on Computational Linguistics: Technical Papers*, pages 2923–2934.

Henaff, M., Weston, J., Szlam, A., Bordes, A., and LeCun, Y. (2017). Tracking the world state with recurrent entity networks. *ICLR*, pages 1–14.

Herzig, J. and Berant, J. (2017). Neural semantic parsing over multiple knowledge-bases. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Short Papers)*, pages 623–628.

Hinton, G. E., Mcclelland, J. L., and Rumelhart, D. E. (1986). Distributed representations, parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations.

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

Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. (2019). The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*.

Hopkins, J. and Kiela, D. (2017). Automatically generating rhythmic verse with neural networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 168–178.

Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Hwang, K. and Sung, W. (2017). Character-level language modeling with hierarchical recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 5720–5724. IEEE.

Iacobacci, I., Pilehvar, M. T., and Navigli, R. (2016). Embeddings for word sense disambiguation: An evaluation study. In *ACL (Volume 1: Long Papers)*, pages 897–907.

Ji, H. and Grishman, R. (2011). Knowledge base population: Successful approaches and challenges. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1148–1158. Association for Computational Linguistics.

Ji, Y., Tan, C., Martschat, S., Choi, Y., and Smith, N. A. (2017). Dynamic entity representations in neural language models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1830–1839. Association for Computational Linguistics.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. (2016). Exploring the limits of language modeling. *arXiv:1602.02410*.

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709.

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2016). Character-aware neural language models. In *AAAI*, pages 2741–2749.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kobayashi, S., Tian, R., Okazaki, N., and Inui, K. (2016). Dynamic entity representation with max-pooling improves machine reading. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 850–855.

Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J., and Woźniak, M. (2017). Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132–156.

Kudo, T. and Richardson, J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulrajani, I., Zhong, V., Paulus, R., and Socher, R. (2016). Ask me anything: Dynamic memory networks for natural language processing. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1378–1387, New York, New York, USA. PMLR.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. In *Proceedings of NAACL-HLT*, pages 260–270.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Lau, J. H., Cohn, T., Baldwin, T., Brooke, J., and Hammond, A. (2018). Deep-speare: A joint neural model of poetic language, meter and rhyme. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics* (*Volume 1: Long Papers*), pages 1948–1958.

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

Lin, Y., Lin, C.-Y., and Ji, H. (2017). List-only entity linking. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics* (*Volume 2: Short Papers*), volume 2, pages 536–541.

Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fermandez, R., Amir, S., Marujo, L., and Luis, T. (2015a). Finding function in form: Compositional character models for open vocabulary word representation. In *EMNLP*, pages 1520–1530.

Ling, X., Singh, S., and Weld, D. S. (2015b). Design challenges for entity linking. *Transactions of the Association for Computational Linguistics*, 3:315–328.

Liu, H. and Cong, J. (2013). Language clustering with word co-occurrence networks based on parallel texts. *Chinese Science Bulletin*, 58(10):1139–1144.

Luo, G., Huang, X., Lin, C.-Y., and Nie, Z. (2015). Joint entity recognition and disambiguation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 879–888.

Lyding, V., Stemle, E., Borghetti, C., Brunello, M., Castagnoli, S., Dell'Orletta, F., Dittmann, H., Lenci, A., and Pirrelli, V. (2014). The paisa' corpus of italian web texts. In *9th Web as Corpus Workshop (WaC-9)@ EACL 2014*, pages 36–43. EACL.

Ma, W., Cui, Y., Si, C., Liu, T., Wang, S., and Hu, G. (2020). Charbert: Character-aware pre-trained language model. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 39–50.

Ma, X., Fauceglia, N., Lin, Y.-c., and Hovy, E. (2017). Cmu system for entity discovery and linking at tac-kbp 2017. *Proceedings of TAC2017*.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Madnani, N. and Dorr, B. J. (2010). Generating phrasal and sentential paraphrases: A survey of data-driven methods. *Computational Linguistics*, 36(3):341–387.

Maggini, M., Marra, G., Melacci, S., and Zugarini, A. (2019). Learning in text streams: Discovery and disambiguation of entity and relation instances. *IEEE Transactions on Neural Networks and Learning Systems*.

Manning, C. D., Raghavan, P., Schütze, H., et al. (2008). *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge.

Marra, G., Zugarini, A., Melacci, S., and Maggini, M. (2018). An unsupervised character-aware neural approach to word and context representation learning. In *International Conference on Artificial Neural Networks*, pages 126–136. Springer.

Martin, T., Botschen, F., Nagesh, A., and McCallum, A. (2016). Call for discussion: Building a new standard dataset for relation extraction tasks. In *Proceedings of the 5th Workshop on Automated Knowledge Base Construction*, pages 92–96.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.

Melacci, S. and Gori, M. (2012). Unsupervised learning by minimal entropy encoding. *IEEE transactions on neural networks and learning systems*, 23(12):1849–1861.

Melamud, O., Goldberger, J., and Dagan, I. (2016). context2vec: Learning generic context embedding with bidirectional lstm. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 51–61.

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

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

Mintz, M., Bills, S., Snow, R., and Jurafsky, D. (2009). Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics.

Miwa, M. and Bansal, M. (2016). End-to-end relation extraction using lstms on sequences and tree structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1105–1116.

Miyamoto, Y. and Cho, K. (2016). Gated word-character recurrent language model. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1992–1997.

Moro, A. and Navigli, R. (2015). Semeval-2015 task 13: Multilingual all-words sense disambiguation and entity linking. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 288–297, Denver, Colorado. Association for Computational Linguistics.

Moro, A., Raganato, A., and Navigli, R. (2014). Entity linking meets word sense disambiguation: a unified approach. *Transactions of the Association for Computational Linguistics*, 2:231–244.

Nigam, K. and Ghani, R. (2000). Analyzing the effectiveness and applicability of co-training. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 86–93. ACM.

Niu, F., Zhang, C., Ré, C., and Shavlik, J. W. (2012). Deepdive: Web-scale knowledge-base construction using statistical learning and inference. *VLDS*, 12:25–28.

Obamuyide, A. and Vlachos, A. (2017). Contextual pattern embeddings for one-shot relation extraction. In *6th Workshop on Automated Knowledge Base Construction (AKBC)*, pages 1–8.

Pan, X., Cassidy, T., Hermjakob, U., Ji, H., and Knight, K. (2015). Unsupervised entity linking with abstract meaning representation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1130–1139.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Pappu, A., Blanco, R., Mehdad, Y., Stent, A., and Thadani, K. (2017). Lightweight multilingual entity extraction and linking. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 365–374. ACM.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Raganato, A., Camacho-Collados, J., and Navigli, R. (2017). Word sense disambiguation: A unified evaluation framework and empirical comparison. In *Proc. of EACL*, pages 99–110.

Rajani, N. F. and Mooney, R. (2016). Combining supervised and unsupervised enembles for knowledge base population. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1943–1948.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, page pages 2383?2392.

Richardson, M., Burges, C. J., and Renshaw, E. (2013). Mctest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 193–203.

Ritter, A., Etzioni, O., Clark, S., et al. (2012). Open domain event extraction from twitter. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1104–1112. ACM.

Santos, C. D. and Zadrozny, B. (2014). Learning character-level representations for part-of-speech tagging. In *ICML*, pages 1818–1826.

Santos, C. N. d. and Guimaraes, V. (2015). Boosting named entity recognition with neural character embeddings. *arXiv preprint arXiv:1505.05008*.

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

See, A., Liu, P. J., and Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*.

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

Shen, W., Wang, J., and Han, J. (2015). Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460.

Sil, A. and Florian, R. (2016). One for all: Towards language independent named entity linking. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 2255–2264.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015). End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

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

Sutton, R. and Barto, A. (2018). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press.

Van Erp, M., Mendes, P. N., Paulheim, H., Ilievski, F., Plu, J., Rizzo, G., and Waitelonis, J. (2016). Evaluating entity linking: An analysis of current benchmark datasets and a roadmap for doing a better job. In *LREC*, volume 5, page 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in NIPS*, pages 5998–6008.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In *Advances in NIPS*, pages 2692–2700.

Wang, Q., Luo, T., Wang, D., and Xing, C. (2016). Chinese song iambics generation with neural attention-based model. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 2943–2949. AAAI Press.

Wen, T.-H., Gasic, M., Mrkšić, N., Su, P.-H., Vandyke, D., and Young, S. (2015). Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Yi, X., Li, R., and Sun, M. (2017). Generating chinese classical poems with rnn encoder-decoder. In *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data*, pages 211–223. Springer.

Yi, X., Sun, M., Li, R., and Li, W. (2018). Automatic poetry generation with mutual reinforcement learning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3143–3153.

Yih, W.-t., He, X., and Meek, C. (2014). Semantic parsing for single-relation question answering. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics* (*Volume 2: Short Papers*), volume 2, pages 643–648.

Yu, L., Zhang, W., Wang, J., and Yu, Y. (2017). Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Yu, Z., Xu, Z., Black, A. W., and Rudnicky, A. (2016). Strategy and policy learning for non-task-oriented conversational systems. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 404–412.

Zampieri, M. and Becker, M. (2013). Colonia: Corpus of historical portuguese. *ZSM Studien, Special Volume on Non-Standard Data Sources in Corpus-Based Research*, 5:69–76.

Zampieri, M. and Nakov, P. (2020). *Similar Languages, Varieties, and Dialects: A Computational Perspective*. Cambridge University Press.

Zettlemoyer, L. S. and Collins, M. (2012). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv:1207.1420*.

Zhang, C., Sah, S., Nguyen, T., Peri, D., Loui, A., Salvaggio, C., and Ptucha, R. (2017). Semantic sentence embeddings for paraphrasing and text summarization. In *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 705–709.

Zhang, X. and Lapata, M. (2014). Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 670–680.

Zhong, Z. and Ng, H. T. (2010). It makes sense: A wide-coverage word sense disambiguation system for free text. In *ACL*, pages 78–83.

Zugarini, A., Melacci, S., and Maggini, M. (2019). Neural poetry: Learning to generate poems using syllables. In *International Conference on Artificial Neural Networks*, pages 313–325. Springer.

Zugarini, A., Tiezzi, M., and Maggini, M. (2020). Vulgaris: Analysis of a corpus for middle-age varieties of italian language. In *Proceedings of the 7th Workshop on NLP for Similar Languages, Varieties and Dialects*, pages 150–159.