UNIVERSITÀ DEGLI STUDI DI PISA

DIPARTIMENTO DI INFORMATICA
DOTTORATO DI RICERCA IN INFORMATICA
SETTORE SCIENTIFICO DISCIPLINARE: INF/01

PH.D. THESIS

Syntax-based machine translation using dependency grammars and discriminative machine learning

Antonio Valerio Miceli Barone

SUPERVISOR

Giuseppe Attardi

REFEREE

Miles Osborne

REFEREE

Christof Monz

Abstract

Machine translation underwent huge improvements since the groundbreaking introduction of statistical methods in the early 2000s, going from very domain-specific systems that still performed relatively poorly despite the painstakingly crafting of thousands of ad-hoc rules, to general-purpose systems automatically trained on large collections of bilingual texts which manage to deliver understandable translations that convey the general meaning of the original input.

These approaches however still perform quite below the level of human translators, typically failing to convey detailed meaning and register, and producing translations that, while readable, are often ungrammatical and unidiomatic.

This quality gap, which is considerably large compared to most other natural language processing tasks, has been the focus of the research in recent years, with the development of increasingly sophisticated models that attempt to exploit the syntactical structure of human languages, leveraging the technology of statistical parsers, as well as advanced machine learning methods such as *marging-based structured prediction* algorithms and *neural networks*.

The translation software itself became more complex in order to accommodate for the sophistication of these advanced models: the main translation engine (the *decoder*) is now often combined with a pre-processor which reorders the words of the source sentences to a target language word order, or with a post-processor that ranks and selects a translation according according to fine model from a list of candidate translations generated by a coarse model.

In this thesis we investigate the statistical machine translation problem from various angles, focusing on translation from non-analytic languages whose syntax is best described by fluid *non-projective dependency grammars* rather than the relatively strict *phrase-structure grammars* or *projective-dependency grammars* which are most commonly used in the literature.

We propose a framework for modeling word reordering phenomena between language pairs as transitions on non-projective source dependency parse graphs. We quantitatively characterize reordering phenomena for the German-to-English language pair as captured by this framework, specifically investigating the incidence and effects of the non-projectivity of source syntax and the non-locality of word movement w.r.t. the graph structure. We evaluated several variants of hand-coded pre-ordering rules in order to assess the impact of these phenomena on translation quality.

We propose a class of dependency-based source pre-ordering approaches that reorder sentences based on a flexible models trained by SVMs and and several recurrent neural network architectures.

We also propose a class of translation reranking models, both syntax-free and source dependency-based, which make use of a type of neural networks known as *graph echo state networks* which is highly flexible and requires extremely little training resources, overcoming one of the main limitations of neural network models for natural language processing tasks.

Acknowledgments

First I would like to thank my PhD supervisor, Professor Giuseppe Attardi for his expert scientific advice and personal guidance in the course of this project.

I thank the people of Human Language Technologies group at the Computer Science Department of University of Pisa for providing the infrastructure and state-of-the-art tools which have been indispensable to my work and for their insightful technical advice and practical help in running the experiments. Specifically, I would like to thank Professor Maria Simi, Dr. Stefano Dei Rossi, Dr. Zauhrul Islam, Dr. Atanas Chanev and Dr. Daniele Sartiano, and also visiting Professor Niladri Chatterjie for introducing me to the world of statistical machine translation.

I also thank Dr. Claudio Gallicchio of the Machine Learning group for his technical advice on the Graph Echo State Network model.
I also thank the coordinator of the Computer Science PhD program, Professor Pierpaolo Degano for his personal guidance and support.

A personal thank you to all my family and friends who supported me during the years I've spent working on this project.

Contents

Chapter 1

Introduction

1.1 Motivation

Automatic translation between human languages was one of the first applications of digital computers that researchers sought to realize since their introduction in the 1950s.

Early approaches, grounded in the *computationalist* philosophy of mind which viewed human cognition and linguistic ability essentially as a process involving discrete individually meaningful symbols manipulated according to well-defined rules, attempted to translate sentences by *parsing* them according to some formal *phrase-structure grammar* (Chomsky, 1956, 1957) (usually a *context-free grammar* (CFG)), hand-coded by human linguists and then transform these sentences into their target-language equivalents using dictionaries and syntactic transfer rules, also hand-coded.

These early *syntax-based* approaches found some success in restricted application domains, but were largely unable to provide general-purpose machine translation systems capable of handling natural languages in a broad usage range, a goal which remained essentially unfulfilled until the introduction of *statistical* methods in the 1990s.

The *statistical machine translation* paradigm took at first a diametrically opposed approach at the translation problem: sentences are viewed as sequences of words, whose correlations are modeled by simple stochastic

models such as *Markov chains*, ignoring any information about their overall syntactic structure.
Translation models are learned from large corpora of bilingual text using statistical methods, estimating correlations between sentences in the source language and their reference translations in the target language, either at the level of words (*word-based models* (Brown et al., 1990, 1993)) or short segments of contiguous words (*phrase-based models* (Och et al., 1999; Och and Ney, 2002, 2004; Marcu and Wong, 2002; Koehn et al., 2003)).
Phrase-based translation yields performances good enough to be used in practical applications, but still performs substantially worse than human translators.

In recent years, there has been a tendency to reintroduce syntax in the framework of statistical machine translation: rather than using hand-coded strict grammars and transfer rules, these approaches estimate stochastic syntactical models from the training corpora, which enables them to take into account complex long-distance correlations that can't be represented by standard phrase-based models.

These approaches typically use a probabilistic version of context free grammars (PCFG) or related formalisms (Yamada and Knight, 2001; Chiang, 2005, 2007), or alternatively *dependency grammars*, which while based on a linguistic theory different than phrase-structure grammar (Tesnière, 1959; Mel'čuk, 1988), are often constrained to be *projective*, introducing an isomorphism with PCFGs.
These formalisms can be well suited to represent the syntax of *analytic languages* with a largely fixed word order such as English, Chinese or Japanese, while non-analytic languages with a relatively free word order such as German, Italian, Czech or Bulgarian might better represented by non-projective dependency grammars (Bosco and Lombardo, 2004; Chanev, 2005).
Moreover, the type of correlations that current syntax-based machine translation systems can take into account are typically "tree-local", meaning that while they may involve words and clauses that are far apart in the sentence, their distance in the syntax parse graph must be short. Indeed, most system

only consider parent-child and sibling-sibling relations. This limitation might fail to capture linguistic phenomena that may be relevant for translation quality.

Non-projective dependency grammars and "non-tree-local" syntactic models have not been significantly applied to statistical machine translation so far, which, we believe, leaves open promising research opportunities that we sought to undertake in this thesis.

Statistical machine translation can be seen as a *structured prediction* problem in the framework of *machine learning*.
Early approaches primarily used *generative* machine learning techniques, which estimate explicit, marginalizable representations of joint probability distributions. *Discriminative* machine learning techniques, more accurate but less flexible since they only estimate limited representations of conditional probability distributions, were initially used only for tuning a small set of parameters (Och, 2003).

More recently discriminative learning techniques have become more prominent, usually in the form of *linear models* with a large number of sophisticated input features (Och et al., 2004; Chiang et al., 2009; Hasler et al., 2011).
There has also been interest in non-linear models such as *neural networks* (Bengio et al., 2006; Auli et al., 2013; liu et al., 2013), but their applications have been limited by difficulties involved with training and decoding.
We believe that discriminative machine learning models, in particular neural networks and non-linear SVMs, can be beneficial to machine translation due to their ability to overcome the *linear separability* representation issue of linear models, thus we set forth to devise ways to exploit these techniques.

1.2 Object and Contributions of the Thesis

The object of this thesis is to develop techniques for statistical machine translation that exploit syntactic information modeled by non-projective dependency grammars, in order to assess the effectiveness of this framework, especially for translation from non-analytic languages.

We also aim to investigate the applicability of machine learning techniques which have not been commonly used in the field.

Modern statistical machine translation system have become quite complex in order to decompose the hard training and decoding problems into subproblems which can be more effectively addressed separately by specific algorithms.
The main translation engine (the *decoder*) can be coupled with preprocessing systems, usually performing *word reordering*, or with postprocessing systems performing *reranking* among a set of candidate translations.
We aim to investigate how to apply our dependency-based techniques on different steps of this translation pipeline, assessing where they yield the most promising results.

1.2.1 Research questions

The main research questions that thesis investigates are:

1. The effects of non-projectivity and tree non-locality on reordering phenomena between language pair. Specifically we investigate the German-to-English pair, which is both well studied and still considered significantly hard. We intend to determine whether non-projectivity and tree non-locality are linguistic phenomena relevant to machine translation.

2. The viability of non-projective and non-tree-local syntax-based pre-reordering systems automatically trained from source-side parsed and word-aligned parallel corpora. We intend to determine whether a system trained by machine learning can successfully discover how to exploit these type of syntactic relations that are not usually considered by typical syntax-based approaches.

3. The viability of several translation reranking approaches based on the topology of the source-side dependency graph.

1.2.2 Main contributions

Our main contributions address the research questions as follows:

1. We investigated the incidence of *non-projectivity* and *tree non-locality*, how they correlate with each other and with reordering accuracy and translation accuracy of pre-reordered phrase-based systems, both under ideal "oracle" permutations derived from word alignments and under hand-coded pre-reordering rules. We found that both non-projectivity and non-tree-local reordering are common linguistic phenomena in the German-to-English language pair and that they are relevant to reordering and translation quality. Specifically, they correlate with the pseudo-upper bounds on improvements that can be obtained using heuristic "oracle" permutations [1]. When reordering with dependency-based hand-coded rules, non-projectivity and tree non-locality also predict improvement, although the measured effect of non-projectivity is weaker.

2. We investigated several novel syntax-based pre-reordering models based on *SVMs* and *recurrent neural networks* and compared them with a standard phrase-based baseline system and hand-coded rules and correlated the reordering and translation accuracy with non-projectivity and tree non-locality. Specifically, we designed a system based on *transitions on the source dependency graph* guided by SVM classifiers. We evaluated it on the German-to-English and the (easier) Italian-to-English language pairs but we couldn't obtain an improvement over the phrase-based baseline. We attributed this issue to an apparent bias of the model towards an in-order depth-first traversal of the graph. We developed a class of *RNN* models that overcame this difficulty by allowing greater freedom of movement on the graph at the cost of greater computational complexity ($O(L^2)$ or $O(L^3)$ instead of approximately $O(L)$). We obtain a significant improvement of translation quality w.r.t. the baseline, with performance comparable to the best hand-coded

[1] Al-Onaizan and Papineni heuristic (Al-Onaizan and Papineni, 2006).

rules. We found a positive correlation both between non-projectivity and translation accuracy and tree non-locality and translation accuracy.

3. We introduce a novel class of *reranking* approaches based on graph echo state neural networks (*GraphESN*), both syntax-free and syntax-based, and we compare these models to each other and to the baseline. GraphESNs are a flexible and efficient type of neural networks for prediction tasks in domains where inputs are complex data structures represented as arbitrary graphs. Their main point of attraction is that they are highly non-linear models but they are trained as linear models, owing to their *reservoir computing* paradigm. These models require minimal training resources and feature engineering, hence in addition to being interesting on their own, they can be used to investigate a lower bound to the performance gains that can be obtained by reranking with dependency topology information. We also consider a linear reranking model based on engineered dependency topology features. This system examines the output of the upstream phrase-based decoder in order to analyze, for each candidate translation of a source sentence, how the phrase segmentation cuts its dependency parse tree and how these phrases are reordered in the translation with respect to the source parse. We evaluated these models on the Italian-to-English language pair and found that these approaches significantly improve translation quality over the phrase-based baseline.

1.3 Map and origin of the chapters

The Thesis is structured as follows:

- Chapter 2 describes the background theory and techniques relevant to this work. Specifically, section 2.2 introduces the general concepts of *natural language processing*, sec. 2.3 describes constituency and dependency *parsing*, sec. 2.1 provides a perspective on machine learning techniques used in this work, sec. 2.4 introduces *language modeling*

techniques and finally sec. 2.5 describes *statistical machine translation* approaches.

- Chapter 3 describes our *characterization of German-to-English reordering as transitions on a dependency tree*.

- Chapter 4 describes our *discriminative non-tree-local syntax-based sentence pre-reordering approach*. This approach was inspired by an earlier work presented by a at the SSST-5 "Fifth Workshop on Syntax, Semantics and Structure in Statistical Translation" associated with the ACL HLT 2011 conference. (Attardi et al., 2011). Part of this work (the SVM model) was presented at the ACL 2013 WMT "Eight workshop on Statistical Machine Translation" (Miceli Barone and Attardi, 2013). Other parts of this work (the recurrent neural network models) will be presented at the ACL 2015 main conference and the NAACL HLT 2015 SSST-9 "Ninth Workshop on Syntax, Semantics and Structure in Statistical Translation".

- Chapter 5 introduces our *reranking models*. Specifically, section 5.1 describes reranking with *graph echo state networks* and sec. 5.2 reraking using *source phrase dependency features*. The latter work will be presented at the NAACL HLT 2015 SSST-9 "Ninth Workshop on Syntax, Semantics and Structure in Statistical Translation".

Chapter 2

Background

In this chapter we describe the existing background theory and methods this work was built upon.

We will first describe some machine learning primitives that are relevant to this work, then we will the general concepts of the framework of *natural language processing*, focusing in particular on syntactic representations and language models. Then we will describe *statistical machine translation* techniques.

2.1 Machine learning

Machine learning is the set of techniques used to make computer system learn how to perform tasks from the automated analysis of data.

Considered a branch of both *Inferential statistics* and *Artificial intelligence*, machine learning is centered on identifying *representations* that can be used to describe and compute useful predictions from suitably large classes of models, and training procedures to efficiently compute these representations from data. The models learned from data must both represent the training data accurately and they must *generalize* well on unseen data, as long as it is sufficiently homogeneous to the training data.

A full taxonomy of machine learning tasks and techniques is beyond the scope of this document, therefore, in the following sections we will only describe those which are relevant to this work.

2.1.1 Discriminative models

A typical way of framing machine learning problems is to identify a set of random variables: the inputs (independent variables) X and outputs (dependent variables) Y. We assume that there exists an underlying probability distribution over these variables that is unknown to us, but that we can observe a set of observations (examples) independently sampled from it.

Given a set of m examples, known as the *training set* T, we want to estimate the properties of the underlying conditional probability distribution $P(Y|X)$. Specifically we want to obtain a representation that allows us to compute the probability of an output value $y \in Y$ given an arbitrary input value $x \in X$, or to predict the most likely output given an arbitrary input: $\overset{*}{y} \equiv \text{argmax}_{y \in Y} P(Y = y|X = x)$ [1].
This approach is known as *supervised learning*, and the class of model it produces are known as *discriminative models*.

We can distinguish these learning tasks according to the type of the outputs. The typical cases include:

- *Regression*, if the outputs are continuous (scalar or vectorial) values.
- *Classification*, if the outputs are discrete labels in a small set.
- *Structured prediction*, if the outputs are discrete data structures in a large [2] set, usually dependent on the input: $y \in GEN(x) \subset Y$.

We will focus on classification and structured prediction, as they are more relevant to statistical natural language processing. In fact, broadly speaking, most NLP tasks can be viewed as structured prediction problems, where the input x is typically a sentence of text or speech and the output y is a complex property or transformation of this sentence, such as parse tree or a translation in another language.

[1] As a slight abuse of notation, we are going to use capital letters to denote both random variables and their domain

[2] By "large" we mean that it will be typically too big to allow efficient exhaustive enumeration.

2.1.2 Generalized linear models

Linear binary classification

Binary classification using a *linear model* is one of the simplest problem that can be considered in machine learning, which is nevertheless significant since many other problems can be reduced to it.

We assume that there are only two possible output labels ("true" and "false", or "positive" and "negative") and that the inputs are real-valued vectors of fixed dimension: $X \equiv \mathcal{R}^n$.
Given an input x, the model computes its prediction according to thresholded linear decision function:

$$h(x, \theta) \equiv \sum_{i=1}^{n} \theta_i x_i \geq -\theta_0 \tag{2.1}$$

where $x_1, \ldots, x_n$ are the n components of the input vectors and $\theta_0, \ldots, \theta_n$ are the $n+1$ adjustable real-valued parameters that characterize the model.
Or, in a simplified notation:

$$h(x, \theta) \equiv \theta^T \cdot x \geq 0 \tag{2.2}$$

where we assume that x is a vector in $\mathcal{R}^{n+1}$, with $x_0 = 1$ (the "bias" feature), and that the dot operator denotes the inner product between vectors.

Probability estimates In this simple formulation, the classification model only computes the most likely output label given the input.
If we also want to compute an estimation of the probability of class membership we can do so by converting the linear score $\theta^T \cdot x$ to a number in the $[0, 1]$ range which can be interpreted as a probability. This is accomplished using a suitable non-linear *activation function*:

$$P(Y = y | X = x) = \pi\left(\theta^T \cdot x\right) \tag{2.3}$$

Where $\pi : \mathcal{R} \to [0, 1]$ is typically defined as the *logistic sigmoid* function:

$$\mathrm{lgc}(u) \equiv \frac{\exp(u)}{\exp(u) + 1} = \frac{1}{\exp(-u)} \tag{2.4}$$

Training as an optimization problem The problem of training a classification model from a set of labeled examples is usually formulated in terms of an optimization problem, where we seek to minimize the *cost* (or *error*) of the model with respect to the training set, which is a measure of how much the predictions made by the model differ from the training examples.

Let X be the set of input values, Θ be the set of values the model parameters can take. Let

$$T \equiv \left(\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right) \tag{2.5}$$

be a list of m training examples. Then the training problem is defined as computing

$$\overset{*}{\theta} \equiv \underset{\theta}{\operatorname{argmin}} \operatorname{cost}(T, \theta) \tag{2.6}$$

The cost function is often defined as the average (or sum) of per-example costs:

$$\operatorname{cost}(T, \theta) = \frac{1}{m} \sum_{i=1}^{m} L\left(h\left(x^{(i)}, \theta\right), y^{(i)}\right) \tag{2.7}$$

where $L : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is a *loss* function, which measures how far the predicted value is from the reference value for that particular example.
In this case, it is usually defined as the "0-1 Loss", which is equal to 0 if the labels are the same and it is equal to 1 if they are different, or, if the model provides probability estimates, as the empirical cross-entropy loss:

$$L_{CE}(z, y) \equiv -y \log(z) - (1 - y) \log(1 - z) \tag{2.8}$$

(Using a logistic activation function trained with the cross-entropy loss is called *logistic regression*, which, despite the name, is actually a classification approach)

An optional additive "*regularization*" term that only depends on the parameters θ and not on the training examples can be included in the cost function. This can be interpreted as a measure of the intrinsic "complexity" of the model. Intuitively, simpler models are expected to generalize better to inputs not seen during training.
Common choices for the regularization term that work well in practice are the L1-norm or the L2-norm of the parameter vectors, scaled by an hyperparameter λ.

Training algorithms A large number of training algorithms that solve the optimization problem defined above have been proposed in the literature. They differ in which loss function and regularization function (if any) they support, whether they require the full training set to be available at the beginning of the computation (batch algorithms) or they can accept training examples incrementally (online and mini-batch algorithms), whether they solve the optimization problem to a global optimum or to a local optimum, and so on.

A description of these algorithms is beyond the scope of this document. We will briefly mention the most common in use, which include the *averaged perceptron* (Freund and Schapire, 1999; Collins, 2002) (an implicitly regularized variant of the earlier *perceptron* algorithm), "large margin" algorithms such as *linear support vector machines* (SVM) (Vapnik, 1982; Cortes and Vapnik, 1995) and the *margin-infused relaxed algorithm* (MIRA) (Crammer and Singer, 2002).
Logistic regression models have a cost function that is differentiable w.r.t. the model parameters, and are therefore trained using first-order or second-order unconstrained continuous optimization algorithms, such as *stochastic gradient descent* (Ferguson, 1982), the *conjugate gradient method* (Press et al., 1988), or the *limited-memory BFGS method* (Liu and Nocedal, 1989).

Linear separability No matter which training algorithm is used, linear models can't learn to accurately represent arbitrary relations, not even arbitrary deterministic relations.

Mathematically, a linear binary classifier separates the original $\mathcal{R}^n$ input space in two halves divided by the (oriented) hyperplane $\sum_{i=1}^{n} \theta_i x_i \geq -\theta_0$ and classifying all examples on one side as positive and all examples on the other side as negative.

This implies that linear classifiers can't represent arbitrary input-output relations, since there exist relations that are not linearly separable.
Linear classifiers can still perform well if the underlying relation is approximately linearly separable, but fail in case of severe non-linearity. In these scenarios, non-linear machine learning techniques, such as *neural networks*

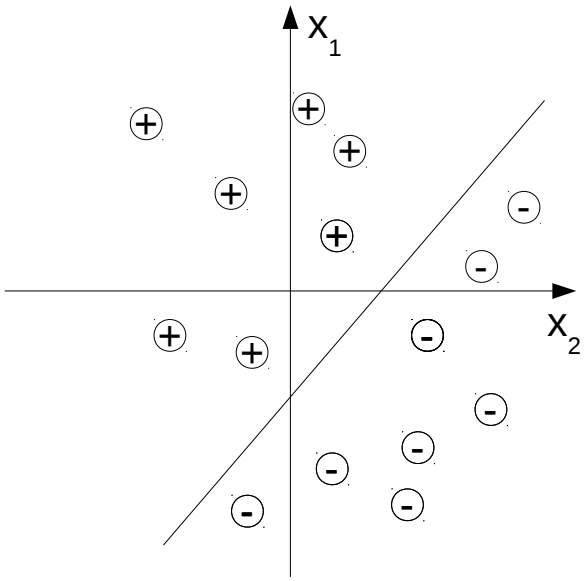


Figure 2.1: Linear classification in two dimensions. Examples above the separation line are classified as positive, the others as negative.

(section 2.1.3) can be used.

Linear structured prediction

A more interesting class of problems involve outputs, and possibly inputs, which are complex data structures, such as a natural language sentence and its parse tree or a sentence and its translation in another language.

We can address these problem using a *feature extraction function* (or just *feature function*) $f : X \times Y \rightarrow \mathcal{R}^n$ that maps each pair of possible inputs $x \in X$ and outputs $y \in GEN(x)$ to a fixed-dimension vector of reals. The design of this function is task-specific and in general requires a significant engineering effort.

Once we specified the feature function, we can formulate the prediction problem using a linear model as:

$$\overset{*}{y} \equiv \underset{y \in GEN(x)}{\operatorname{argmax}} \; \theta^T \cdot f(x, y) \tag{2.9}$$

where $\theta \in \mathcal{R}^n$ is the parameter vector (note that there is no "bias" feature).

This optimization problem will be usually non-trivial, and will require a task-specific algorithm. In particular, computing an exact solution, or even

an approximated solution with guaranteed quality, to this problem may be unfeasible, and therefore the algorithm will produce a heuristic solution.

Therefore, training algorithms for linear structured prediction also need to be designed specifically for the task, although they often fall into general categories which are based on extensions of the training algorithms used for binary classification.
Common approaches include the *structured (averaged) perceptron* (Collins, 2002), *structured SVM* (Tsochantaridis et al., 2005; Finley and Joachims, 2008; Cherry and Foster, 2012) and *structured MIRA* Crammer and Singer (2003); McDonald et al. (2005); Watanabe et al. (2007); Cherry and Foster (2012).

Logistic regression can also be generalized to structured prediction by replacing the logistic sigmoid function with the logistic softmax function, resulting in a probabilistic model in the form of:

$$P(Y = y|X = x) = \frac{\exp\left(\theta^T \cdot f(x, y)\right)}{\sum_{y' \in GEN(x)} \exp\left(\theta^T \cdot f(x, y')\right)} \tag{2.10}$$

which is trained with the categorical cross-entropy loss, yielding a so-called *maximum entropy* model.
The optimization problem can be solved using unconstrained continuous optimization algorithms as in the logistic regression case, although the denominator of the formula above can't be feasibly computed in a direct way, hence some task-specific mathematical tricks [3] or approximations need to be used.

Multi-class classification, that is classification with a small number greater than two of output labels, can be considered as a simpler special case of structured prediction or it can be reduced to binary classification (bo Duan and Keerthi, 2005; Chang and Lin, 2011).

A notion of linear separability also exists for linear structured prediction, which implies that these models can't learn certain kinds of relations.
This is not necessarily always a severe issue since the feature function is specifically designed for the task and with sufficient feature engineering a problem with a non-linear decision boundary can be transformed into a lin-

[3] generally based on exploiting some way to factorize the probabilistic model

early separable problem. Feature engineering, however, is time-consuming and requires significant domain-specific expertise.

2.1.3 Neural networks

Some machine learning tasks involve inherently non-linear underlying processes and are difficult to solve with linear methods even after substantial feature engineering. In these cases, non-linear models may be more appropriate.

These non-linear models usually are usually composed by a non-linear feature transformation that maps the input features to a different, usually higher dimensional, transformed feature space, followed by a linear model.
Different methods are distinguished by the form of feature mapping they use, its parametrization (which can be adaptive or fixed) and the specific algorithms used to solve the training problem.

Popular choices include *SVM with non-linear kernels* (Aizerman et al., 1964; Boser et al., 1992), *decision tree ensemble* methods (such as *random forests* (Breiman, 2001) and *gradient tree boosting* (Friedman, 2001)) and *artificial neural networks*.

Artificial neural networks (ANN), or simply *neural networks* (NN), encompass a large variety of machine learning techniques, all characterized by the general approach of computing complicated, highly non-linear functions using networks of interconnected units with adjustable parameters.
Owing to their flexible architecture that can be customized for tasks with structured inputs and outputs and their ability to scale well to large number of input features and large training sets, neural networks have become popular in NLP applications in recent years.
Since the field is vast, even a survey would be beyond the scope of this document, thus, in the following sections we will describe some types of neural networks that were relevant for this thesis work.

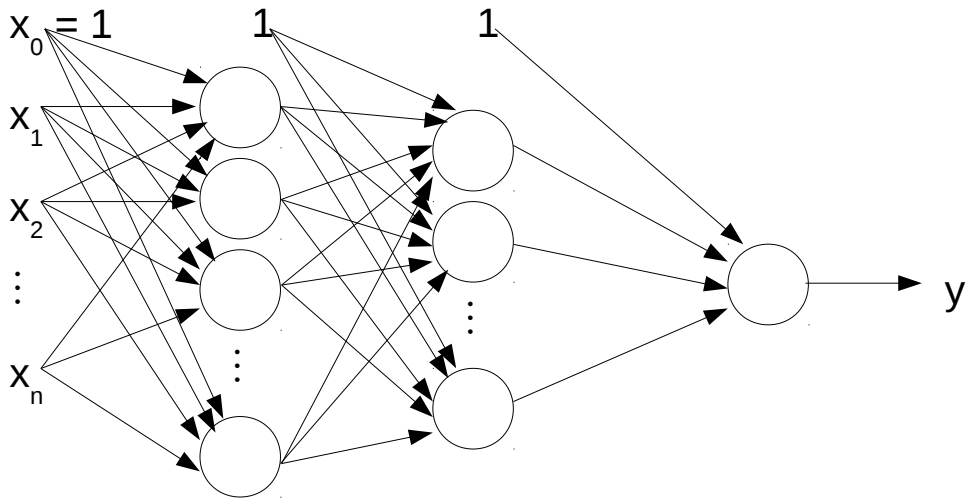


Figure 2.2: Multi-layer perceptron with two hidden layers and a single output unit. Each unit computes the weighted sum of its inputs and maps it through a squashing function such as the logistic sigmoid.

Multi-layer perceptron

Consider a logistic regression model whose inputs are themselves computed by s (non-thresholded) logistic regression models of the original features:

$$h_{regr}(x, \Theta^{(1)}, \theta^{(2)}) \equiv \mathrm{lgc}\left(\theta_0^{(2)} + \sum_{i=1}^{s} \theta_i^{(2)} \cdot \mathrm{lgc}\left(\Theta_{i,0}^{(1)} + \sum_{j=1}^{n} \Theta_{i,j}^{(1)} \cdot x_j\right)\right) \quad (2.11)$$

where the model parameters $\theta \equiv (\Theta^{(1)}, \theta^{(2)})$ are a matrix $\Theta^{(1)} \in \mathcal{R}^{s \times n+1}$ whose rows are the parameter vectors of the intermediate logistic models (the *hidden units*), and the parameter vector $\theta^{(2)} \in \mathcal{R}^{s+1}$ of the final logistic model (the *output unit*).
More concisely:

$$h_{regr}(x, \Theta^{(1)}, \theta^{(2)}) \equiv \mathrm{lgc}\left(\theta_0^{(2)} + \theta_{1:}^{(2)} \cdot \mathrm{lgc}(\Theta^{(1)} \cdot x)\right) \quad (2.12)$$

where $x_0 = 1$ and the inner logistic function is applied element-wise.

This model is known as *feed-forward neural network* or *multi-layer perceptron* (MLP) with a single *hidden layer* and logistic *activation*. Given a sufficient number s of hidden units, it can represent any arbitrary boolean function of n variables.

More generally the network architecture (the number of hidden layers and of units in each of them), and the logistic function in the hidden units can be replaced by any squashing [4] *activation function*, such as the *hyperbolic tangent* sigmoid:

$$\tanh(u) \equiv \frac{\exp(2u) - 1}{\exp(2u) + 1} \tag{2.13}$$

and the logistic function in the output layer can be replaced by any arbitrary function or removed. All these models, even with a single hidden layer, can approximate arbitrary well any Borel-measurable function with compact support and range within the range of the output activation function (Cybenko, 1989; Hornik et al., 1989). This makes multi-layer perceptron well suited for essentially arbitrary classification or regression tasks. Furthermore, the number of output units can be increased in order to perform multiclass classification or vector regression.

The general form of the multi-layer perceptron with l hidden layers is:

$$\begin{aligned} & h_{regr}(x, \Theta^{(1)}, \ldots, \Theta^{(L+1)}) \equiv a_{1:}^{(l+1)} \\ \text{where} \quad & a^{(0)} = x, \\ \text{and } \forall j = 1 \ldots l+1, \quad & a_0^{(j)} = 1, \\ & a_{1:}^{(j)} = \pi^{(j)} \left(\Theta^{(j)} \cdot a^{(j-1)} \right). \end{aligned} \tag{2.14}$$

MLP training In order to train a multi-layer perceptron, the main approach is to consider the number of hidden layers and the number of units in each hidden layer as hyper-parameters, choose a loss function and optionally a regularization function, and minimize the resulting cost function.

For instance, using the cross-entropy loss and L2-regularization, we obtain:

$$\begin{aligned} \overset{*}{\theta} = \underset{\theta}{\operatorname{argmin}} & -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \left(h_{regr}(x^{(i)}, \theta) \right) + \left(1 - y^{(i)} \right) \log \left(1 - h_{regr}(x^{(i)}, \theta) \right) \right] + \\ & + \frac{\lambda}{2} \left\| \theta_{1:} \right\|^2 \end{aligned} \tag{2.15}$$

[4] Any monotonically increasing function $\pi : \mathcal{R} \to \mathcal{R}$ such that $\lim_{u \to -\inf} \pi(u) = -c$ and $\lim_{u \to \inf} \pi(u) = c$, for some $c > 0$.

where h_{regr} is defined as in eq. 2.14 and the parameter vector $\theta \equiv \left(\Theta^{(1)}, \ldots, \Theta^{(l+1)}\right)$ is the concatenation of the parameter matrices for each layer of the network.

Computing the global optimum of this type of problems is NP-hard even for very simple network architectures (Blum and Rivest, 1992; Orponen, 1994), however, since the cost function is differentiable, local optima can be computed efficiently using gradient-based techniques, notably stochastic gradient descent, conjugate gradient or L-BFGS.
In order to apply these methods we need a procedure to compute gradient of the (non-regularized) per-example cost:

$$C^{(i)}(\theta) \equiv L(h_{regr}(x^{(i)}, \theta), y^{(i)}) \tag{2.16}$$

This can be accomplished using the *error back-propagation algorithm* (Bryson and Ho, 1969; Rummelhart, 1986). Adding to $\frac{d}{d\theta}C^{(i)}(\theta)$ the contribution of the regularization function (which is $\frac{2\lambda}{m}\theta$ for L2-regularization) yields the regularized per-example gradient, which can be used directly for stochastic gradient descent. Alternatively, these terms can be summed over a mini-batch of examples or over the whole training set to be used with arbitrary unconstrained optimization methods.

Recurrent neural networks

Standard multi-layer perceptrons compute functional mappings between fixed-dimensional input spaces to fixed-dimensional output spaces. For some tasks, the input, the output or both are sequences of events of arbitrary length and their relation can't be accurately decomposed to a mapping between individual events. Typical examples arise in NLP, since inputs are often sentences represented as sequence of words or sounds, and outputs are either global properties of the sentence (e.g. language model probability, topic, sentiment, etc.) or local properties that are defined for each word but nevertheless depend on a context (e.g. part-of-speech tags, named entities, syntactic chunks, etc.).
In some cases, it is possible to transform the task into a fixed-dimensional classification or regression problem by turning the input sequence into an

$\mathcal{R}^n$ vector using cleverly engineered feature functions for global property prediction, or considering only finite windows of events for local property prediction. However, this approach isn't necessarily the best.

In the general case, we want to model a stochastic process between sequences $p(Y(1),\dots,Y(k)|X(1),\dots,X(k))$, where the sequence length k varies for each example.
A key observation to be made is that many realistic processes can be assumed to have a *Markov property* w.r.t. a state $v(t)$ that evolves over time:

$$\begin{aligned}\forall 1 \le t \le k, \quad & p(Y(t) = y(t)|Y(1:t:k) = y(1:t:k), X(1:k) = x(1:k)) = \\ & = p(Y(t) = y(t)|V(t) = v(t)) \\ \forall 1 \le t \le k, \quad & p(V(t) = v(t)|V(1:t:k) = v(1:t:k), X(1:k) = x(1:k)) = \\ & = p(V(t) = v(t)|V(t-1) = v(t-1), X(t) = x(t)).\end{aligned} \tag{2.17}$$

that is, the output at any time depends only on the current state, which in turn depends only on the current input and the previous state.
If the state is finitely-dimensional, the original infinitely-dimensional model can be decomposed in the finitely-dimensional models for $p(Y(t)|V(t))$ and $p(V(t)|V(t-1), X(t))$ (and $p(V(0))$ if the initial state is not known a priori).

Recurrent neural networks (RNN) are networks similar to MLPs but with feedback connections between the units. These feedback connections maintain a state, which is initialized to some default value and then evolves over time.
A typical model of RNN has a single hidden layer with feedback connections limited to hidden units:

$$\begin{aligned}h_{regr}(t, x(1,\dots,t), \Theta^{(1)}, \theta^{(2)}, \Theta^{REC}) &\equiv \pi^{(2)}\left(\theta_0^{(2)} + \theta_{1:}^{(2)} \cdot v(t)\right) \\ \text{where } v(\tau) &= \begin{cases} 0^{\otimes s} & \text{if } \tau = 0 \\ \pi^{(1)}(\Theta^{(1)} \cdot x(\tau) + \Theta^{REC} \cdot v(\tau-1)) & \text{otherwise} \end{cases}\end{aligned} \tag{2.18}$$

where the $\Theta^{REC} \in \mathcal{R}^{s \times s}$ is the matrix of parameters for the feedback connections.
More generally, we can have multiple hidden layers and feedback connection may go from any non-input layer to the same layer or to any previous non-

input layer. Alternatively, we can view an RNN just as a directed graph of units, each with an associated state variable maintaining its activation value, without making a distinction between forward and feedback connections.

Due to an obvious mapping from sequential logic circuits, it's easy to show that RNNs can represent arbitrary finite-state machines.
RNNs are dynamical systems which can exhibit various regimes, including chaotic, periodic or "quasiperiodic" behaviors. Under free evolution [5] their phase space (state space) can have multiple attractors of various types.

In order to train an RNN we need a training set of examples of the form $(x^{(i)}(1), \ldots, x^{(i)}(k^{(i)}), y^{(i)}(1), \ldots, y^{(i)}(k^{(i)}))$ if we are doing local property prediction or $(x^{(i)}(1), \ldots, x^{(i)}(k^{(i)}), y^{(i)})$ if we are doing global property prediction [6], a loss function and an optional regularization function. From these elements, we can formulate the training problem as a usual unconstrained optimization problem.
For instance, for global property prediction with cross-entropy loss and L2-regularization:

$$\overset{*}{\theta} = \underset{\theta}{\operatorname{argmin}} -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h_{regr}(t, x^{(i)}(1,\ldots,k^{(i)}), \theta)\right) + \left(1-y^{(i)}\right)\log\left(1-h_{regr}(t, x^{(i)}, \theta)\right)\right] + \\ + \frac{\lambda}{2}\left\|\theta_{1:}\right\|^2 \tag{2.19}$$

Solving (to a local optimum) this problem with gradient-based algorithms such as stochastic gradient descent requires a procedure to compute the gradient.

Note that, for any choice of parameters θ and input sequence $x(1, \ldots, k^{(i)})$, we can compute the output $z(t)$ at time t by "unfolding" the RNN into a feed-forward MLP whose depth depends on t.
Specifically, when there is a single hidden layer with feedback connection

[5] the behavior under an input sequence of constant null vectors

[6] In the RNN literature, global property prediction and local property prediction are sometimes called *global transduction* and *local transduction*, respectively. In this document we do not use this terminology to avoid confusion with transductive learning.

confined to it, we can write:

$$\begin{aligned} & & z(t) &= \pi^{(2)}\left(\Theta^{(2)} \cdot a^{(t)}\right) \\ \text{where} & & a^{(0)} &= 0^{\otimes s}, \\ \text{and } \forall j = 1 \ldots t, & & a_0^{(j)} &= 1, \\ & & a_{1:}^{(j)} &= \pi^{(1)}\left(\Theta^{(1)} \cdot x(j) + \Theta^{REC} \cdot a^{(j-1)}\right). \end{aligned} \tag{2.20}$$

This is a MLP with t hidden layers, all with the same number of units and identical parameters, and with inputs that skip layers (the $x(t)$ input goes directly into the t-th hidden layer, skipping over the previous ones). Thus, we can leverage the error backpropagation algorithm for MLPs to compute the per-example gradient terms, and therefore the total gradient of the cost function. This technique is known as *backpropagation through time* (Rumelhart et al., 1985).
Other techniques based on the *extended Kalman filter* [7] also exist (Jaeger, 2002).

In practice, the applicability of these techniques is limited by the fact the cost functions tend to have very irregular landscapes, especially when the length of input sequences is large. Gradient based techniques may get stuck at bad local minima or saddle points (the *vanishing gradient* problem). Online algorithms such as stochastic gradient descent or the extended Kalman filter method may fail to converge or even diverge (the *exploding gradient* problem). For a detailed discussion of these issues, and approaches to mitigate them, see (Bengio et al., 1994; Jaeger, 2002; Pascanu et al., 2012).
Despite these issues, RNNs have been successfully applied in a variety of application domains, including natural language processing for tasks such as language model probability estimation (Mikolov and Zweig, 2012), semantic analysis (Mikolov et al., 2013b) and machine translation (Auli et al., 2013).

Echo state networks

The Echo state property Among the various regimes that RNNs can exhibit, one of particular interest is the regime where, as time passes, the system

[7] A general Bayesian method to estimate the hidden state of stochastic non-linear dynamical systems (Jazwinski, 1970; Sorenson, 1960).

state tends to become less and less dependent on the initial state. Given two different input sequences which share a suffix, the corresponding state trajectories, and thus outputs will become less and less distinguishable as the length of this shared suffix increases. Formally, let $x(1,\dots)$ and $x'(1,\dots)$ be two infinite input sequences such that $\exists t_s : \forall t \geq t_s, x(t) = x'(t)$. Then this "fading memory" regime is defined as:

$$\lim_{t\to\text{inf}} \left\| h_{reg}(t,x,\theta) - h_{reg}(t,x',\theta) \right\| = 0 \tag{2.21}$$

If this occurs for any possible choice of pairs of input sequences with a shared suffix, the RNN is said to have the *echo state property* (ESP) (Jaeger, 2001).
Equivalently, let $\hat{h}(x,v,\theta)$ be the function that computes the output of the network given the infinite sequence $x(1,\dots)$ and the initial state v, then the echo state property can be defined as:

$$\forall \text{sequences } x, \forall \text{states } v, v', \lim_{t\to\text{inf}} \left\| \hat{h}(x,v,\theta) - \hat{h}(x,v',\theta) \right\| = 0 \tag{2.22}$$

that is, if we fix an input sequence and start the system from an arbitrary initial state, it exhibit an attractor dynamic, with a basis of attraction encompassing all the state space. Furthermore, if the input sequence converges, the system state also converges.

While the ESP prevents the network from representing arbitrary computations, it is nevertheless a often desirable condition, because it models many realistic processes which tend to operate in a steady-state regime, where dependencies from the past eventually fade away, as it is typically the case in local property prediction tasks. Moreover, the ESP makes the network intrinsically stable, avoiding divergences and chaotic behavior during both prediction and training.

If an RNN has a single hidden layer which contains all feedback connections, and with hidden units with a scalar, differentiable squashing activation function $\hat{\pi} : \mathcal{R} \to \mathcal{R}$, the presence of the ESP depends on the feedback parameter matrix Θ^{REC}.
A *necessary* condition for the ESP is that the spectral radius [8] of Θ^{REC} is less

[8] the absolute value of the eigenvalue with the largest absolute value.

than the inverse of the Lipschitz constant [9] of the activation function:

$$\rho(\Theta^{REC}) < 1/C \tag{2.23}$$

where $\rho(\dot{})$ denotes the spectral radius and C the Lipschitz constant of $\hat{\pi}$. For the logistic sigmoid, $C = 1/4$, while for the hyperbolic tangent sigmoid $C = 1$.

By Banach fixed point theorem, a *sufficient* condition for the ESP is that any norm (Euclidean or otherwise) of the recurrent matrix is less than $1/C$:

$$\|\Theta^{REC}\|_* < 1/C \tag{2.24}$$

This condition also ensures that convergence to the attractor is exponentially fast:

$$\|v(t) - v'(t)\|_* < \sigma^t \cdot \|v(0) - v'(0)\|_* \tag{2.25}$$

where the *contractivity hyperparameter* $\sigma = C\|\Theta^{REC}\|_* < 1$ is the Lipschitz constant of the whole state transition function of the network. The smaller the value of σ, the faster the network memory fades, making its behavior more local. Conversely, the larger it is, the richer the dynamics of the network, making its behavior more global.

If the process we are trying to model has known a characteristic time scale τ, we may want to set $\sigma \approx \exp(-\frac{1}{\tau})$. In practice some rounds of model selection are usually performed to determine an appropriate value (Gallicchio and Micheli, 2011a).

Random initialization Once we have chosen the hyperparameters s and σ and the type of hidden activation function (usually tanh), a simple way to obtain an echo state network is to first randomly initialize Θ^{REC} with small values (e.g. by sampling them from the uniform distribution between $[-1, 1]$ or the standard normal distribution). Additional constraints, to enforce sparsity (which reduces computational complexity), or some special connectivity pattern, can be included.

[9] the smallest c such that $\forall u, u',\ \|\pi(u) - \pi(u')\| \leq c\|u - u'\|$. For a differentiable function $\mathcal{R} \to \mathcal{R}$, this corresponds to the supremum of the absolute value of the derivative.

Then Θ^{REC} is rescaled according to some norm (usually L2) to obtain the desired σ:

$$Theta^{REC} \leftarrow Theta^{REC} \cdot \frac{\sigma}{C\|\Theta^{REC}\|_*} \tag{2.26}$$

as long as $\sigma < 1$ this guarantees the ESP. Alternatively we may rescale according to the spectral radius, which, while yielding only a necessary condition on the ESP, often works well in practice for large values of σ.
The input-to-hidden $\Theta^{(1)}$ parameter matrix can be generated by randomly sampling small values (e.g. with standard deviation $\approx 0.1 - 0.01$ for properly normalized input variables). If needed, the hidden-to-output matrix $\Theta^{(2)}$ can be initialized in a similar way.

We can use these parameters as the starting point for the local search gradient-based algorithms described in the previous section. Combining this initialization with regularization (Pascanu et al., 2012), or with the explicit rescaling of the Θ^{REC} to preserve the ESP at each step of the training algorithm(Palangi et al., 2013), enables effective training with little or no probability of divergence or chaotic behavior.
Alternatively, we can use these parameters directly for the *reservoir computing* approach.

Echo state networks While the ESP helps RNN training algorithms by preventing the exploding gradient problem, convergence might be slow due to the vanishing gradient problem. Moreover, it has been observed that training algorithms mostly modify the parameters in the hidden-to-output layer matrix leaving the input-to-hidden and the recurrent matrix relatively unchanged (Schiller and Steil, 2005).
Therefore, training all the parameters of a RNN with the ESP, might not be necessarily worth its computational cost. In fact, restricting training to the parameters of the output layer can be often sufficient to obtain good, even state of the art in some cases, performances.

An *Echo State Network* (ESN) (Jaeger, 2002; Jaeger and Haas, 2004) is a RNN with one hidden layer, where only the hidden-to-output parameters are subject to training, while the input-to-hidden and the recurrent hidden-to-

hidden parameters are randomly generated at the beginning of the training process and left untouched.
The hidden units, with their randomly initialized recurrent connections and input connections are known as the *reservoir* of the network, since they contain a large amount of rich non-linear dynamics, which can be tapped into by the output layer (known as the *readout* of the network). This is effectively a generalized linear model applied on top a fixed, randomly generated, highly non-linear feature mapping of the input. Since generalized linear models can be trained very efficiently using the methods described in the previous section (e.g. perceptron, logistic regression, linear SVM, etc.) this approach is orders of magnitude faster than full RNN training.

Training is typically performed as following:

- Choose the type of hidden activation function, the number of hidden units s, the contraction coefficient σ and and input feature scaling factor η.
- Sample the elements of the input-to-hidden matrix $\Theta^{(1)}$ and the recurrent matrix Θ^{REC} from uniform distributions over $[-\eta, \eta]$ and over $[-1, 1]$, respectively.
- Scale the recurrent matrix Θ^{REC} by $\frac{\sigma}{C\|\Theta^{REC}\|_2}$.
- For each training sequence i, initialize the hidden units in the all-zero state and pass the input vectors $x^{(i)}(t)$ through the reservoir, recording the sequence of hidden state vectors (for local property prediction) or the last hidden state vector (for global property prediction).
- Compute the hidden-to-output matrix $\Theta^{(2)}$ by training a linear model on the recorded hidden state vectors and the original output reference labels. This step can be interleaved with the previous one if an online algorithm (e.g. stochastic gradient descent) is used.

It might be counterintuitive that an untrained random mapping can still yield useful features that the generalized linear output layer can exploit. The ESP property plays a role in this regard, by making sure that the reservoir

state always "follows" the input, and in particular by preventing chaotic dynamics which would quickly turn the state into useless pseudo-random noise.
ESN have been successfully applied to various prediction tasks with sequential inputs, including speech recognition (Triefenbach et al., 2010) and language modeling (Rachez and Hagiwara, 2012).

Graph echo state networks

The echo state network approach has been extended to structured tasks where the input data structures are trees (Gallicchio and Micheli, 2013) and arbitrary graphs (Gallicchio and Micheli, 2010; Gallicchio, 2011; Gallicchio and Micheli, 2011c). In this document we will describe the graph version of ESN which have been used in this thesis work.

In the simplest case of unlabeled, undirected graphs, let the input be:

$$x \equiv \left(V_x, E_x, x_{(1)}, \ldots, x_{(k_x)}\right) \tag{2.27}$$

where $V_x = 1, \ldots, k_x$ is a finite set of vertices, E_x is a set of edges and for each vertex j, $x_{(j)} \in \mathcal{R}^n$ is a fixed-dimensional vector of input features. In a global property prediction task we want to perform classification or regression computing an output label for the whole input graph, while in a local property prediction task we want to compute an output label for each of its vertices.

The model Consider a local (q-dimensional) property prediction task. If the graph topology E_x did not carry any significant information, we could just process each vertex independently using a fixed-dimensional machine learning model, such as a MLP with a single hidden layer of s units and q output units. We can view all these instances of the same MLP model as a total per-example MLP model with $k_x n$ input units, $k_x s$ hidden units and $k_x q$ output units.
Obviously, we are interested to the case where the graph topology does carry significant information, and we want to incorporate this information in the model. A way of doing so is to add feedback connections between the hidden

layers of the per-vertex neural networks that comprise the total per-example network. Specifically, for each pair of neighboring nodes $(j, j') \in E_x$, we add feedback connections from the hidden units for vertex j to those for vertex j' (and viceversa), parametrized by a matrix $\Theta^{EDGE} \in \mathcal{R}^{s \times s}$ independent of the particular edge.
The end result is a recurrent neural network customized for the specific example x, where the input-to-hidden $\widehat{\Theta^{(1)}} \in \mathcal{R}^{k_x(n+1) \times k_x s}$ and hidden-to-output $\widehat{\Theta^{(2)}} \in \mathcal{R}^{k_x s \times k_x q}$ parameter matrices are just those of the MLP vertically replicated k_x times, and the feedback connection matrix $\widehat{\Theta^{REC}} \in \mathcal{R}^{k_x s \times k_x s}$ is the block matrix:

$$\widehat{\Theta^{REC}} = A_x \otimes \Theta^{EDGE} \tag{2.28}$$

where A_x is the adjacency matrix of the graph and $\otimes$ denotes the Kronecker product.
Note that while the size of the RNN varies with the size of the input graph, the number of free parameters is fixed.

We are interested in computing a function, while in general an RNN is a dynamical system whose output varies over time. However, if we choose the parameters so that the network has the echo state property, then we can run it while keeping the input vectors fixed and it will converge, exponentially fast, to a stable state and hence a stable output which we can consider as the prediction of our model.

Due to the block structure of the recurrent matrix $\widehat{\Theta^{REC}}$, if for all the possible inputs the graph has a bounded maximum degree D, the echo state property can be characterized in terms of the properties of edge matrix Θ^{EDGE}.
Specifically, a *sufficient* condition(Gallicchio and Micheli, 2010) is $\sigma < 1$, where the contractivity hyper-parameter σ is defined as:

$$\sigma = C \cdot D \cdot \|\Theta^{EDGE}\|_* \tag{2.29}$$

for any norm $\|\cdot\|_*$, where C denotes the Lipschitz constant of the hidden unit activation function.

The type of processes that this model can represent can be thought as a generalization of the fading-memory processes represented by ESNs for sequences: In a sequence-ESN the output at any given time depends on the current and past inputs, with the dependence from the past fading away with time. In a graph-ESN for local property prediction, the output at any given vertex depends on its input and the inputs on the nearby vertices, with the dependence fading away with distance on the graph. For a more detailed discussion, see (Gallicchio and Micheli, 2010; Gallicchio, 2011; Gallicchio and Micheli, 2011c).

Training Similarly to sequence-RNNs, graph-RNNs can trained by adapting all the parameters, using gradient-based optimization methods, or in a reservoir computing approach, training only the output layer parameters.

The reservoir computing approach, GraphESN (Gallicchio and Micheli, 2010), performs the random initialization and then adapts the hidden-to-output parameters using any generalized linear model technique.

Alternatively, the parameters of the recurrent layer can be adapted using GraphNN: a backpropagation-based full training method is introduced in (Scarselli et al., 2009). The parameters are randomly initialized satisfying the ESP, and then they are trained using local search. Given a training example $(x^{(i)}, y^{(i)})$ and a current parameter vector θ, the model is run on $x^{(i)}$ until convergence, then the per-example gradient is computed as in backpropagation-through-time and combined with a regularization gradient term designed to penalize violations of the ESP. This gradient is used to perform stochastic gradient descent.
This approach has been successfully applied to a large variety of tasks, including NLP (Muratore et al., 2010; Chau et al., 2009).

Extensions So far we have only discussed local property prediction.
In some learning tasks, the input graph always have some distinguished "super-root" vertex of special significance. In these cases global property prediction can be performed just by considering the output at this distinguished vertex.

In other tasks, where no appropriate special vertex, global property prediction can be performed by combining the k_x per-vertex state vectors into a single, fixed-dimensional, vector which is then feed to the output layer. Combination can be performed simply by summing, averaging, or by more advanced adaptive methods (Gallicchio and Micheli, 2011b).

The approach can be also extended to other types of input graphs: in directed graphs, incoming and outgoing edges can be distinguished by using two different feedback parameter matrices: $\Theta^{EDGE_{IN}}$ and $\Theta^{EDGE_{OUT}}$. Similarly, we can further distinguish the edges according to discrete labels and/or positions using additional matrices.
All these feedback matrices are randomly initialized and the condition on the contractivity hyper-parameter sufficient for the ESP is guaranteed by constraining their maximum norm: $\sigma < 1$, where:

$$\sigma = C \cdot D \cdot \max_{\psi} \|\Theta^{EDGE_{\psi}}\|_{*} \tag{2.30}$$

2.1.4 Generative models

The machine learning methods described so far are used for training *discriminative models*: given a distinction between input variables X and output variables Y, they represent hypotheses in the form $h(x, \theta) = \text{argmax}_Y P(Y = y|X = x)$ or $h_{regr,y} = P(Y = y|X = x)$, where the parameter θ is learned from the training data.
Generative models, on the other hand, represent the joint probability distribution of all variables X without making a distinction between input and output: $h(x, \theta) = P(X = x)$.

In order to make the model computationally tractable, the joint probability distribution is assumed to be factorizable into a product of unconditional and conditional probability distributions, each involving one or a few variables and controlled by some parameters.
For instance:

$$P(X_1, X_2, X_3, X_4) = P(X_1) \cdot P(X_2) \cdot P(X_3|X_1) \cdot P(X_4|X_2, X_3) \tag{2.31}$$

Various classes of generative model exist, differing in the type of variables they admit, the choice of the elementary probability distribution and the way factorization is performed. Typical choices are *graphical models* (which represent the dependence and independence assumptions according to some graph) such as *Bayesian networks* (Pearl, 1985) (notably including *hidden Markov models* (Stratonovich, 1960)) and *Markov random fields* (Moussouris, 1974). A comprehensive discussion of these models is beyond the scope of this document. In this section we will briefly describe the general principles of generative training.

Assume that we have already chosen the general structure of the model representation, with its choice of elementary probability distributions, factorization and optional *hidden variables* (unobservable random variables) [10].
Given a training set $T = \left(x^{(1)}, \ldots, x^{(m)}\right)$ of i.i.d. examples, we condition the joint probability distribution on the parameters and maximize the empirical likelihood of the parameters:

$$\begin{aligned} \overset{*}{\theta} &= \operatorname*{argmax}_{\theta} \prod_{i=1}^{m} P(X = x^{(i)} | \Theta = \theta) = \\ &= \operatorname*{argmax}_{\theta} \sum_{i=1}^{m} \log(P(X = x^{(i)} | \Theta = \theta)) \end{aligned} \tag{2.32}$$

Alternatively we may perform a maximum a posteriori estimation of the parameters w.r.t. a prior distribution $P(\Theta)$.

If there are no hidden variables, performing this estimation is generally relatively easy: if the elementary distributions the model factors into are in the *exponential family*, the ML or MAP (w.r.t. conjugate priors) estimate of their parameters can be computed from sample statistics (counts, means, sample variances, etc.) of their variables.
Alternatively, if the optimization objective is continuous and differentiable w.r.t. the parameters, gradient-based optimization techniques can be used to compute a local optimum, which is also the global optimum if the optimization objective is concave.

[10] This structure can be learned as well from the data, at least in principle, however doing so is generally very computationally expensive and requires a large amount of data.

If the model include hidden variables Z in addition to the observable variables X, the ML estimation problem involves the maximization of the empirical *marginal likelihood* w.r.t. the hidden variables:

$$\begin{aligned} \overset{*}{\theta} &= \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{m} \sum_{z \in Z} P(X = x^{(i)}, Z = z | \Theta = \theta) = \\ &= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log\left(\sum_{z \in Z} P(X = x^{(i)}, Z = z | \Theta = \theta) \right) \end{aligned} \tag{2.33}$$

Unless the model has some special structure or the hidden variables take values in a small finite set, an explicit computation of the marginal per-example likelihoods $P(X = x^{(i)}, Z = z | \Theta = \theta)$ is unfeasible, hence some other approach is needed.

The *expectation-maximization* (EM) method (Dempster et al., 1977) is a class of iterative algorithms to efficiently solve this training problem without explicitly computing the marginal likelihoods.
Approximation techniques which make use of randomized sampling, in particular *Markov chain Monte Carlo* methods such as *Gibbs sampling* (Geman and Geman, 1984) and the *Metropolis-Hastings* algorithm (Hastings, 1970) can be used in alternative to the EM method (by directly sampling the marginal log-likelihoods) or in conjunction to it (by sampling the $Q^{(t)}(\theta)$ functions).

2.2 Natural language processing

Natural language processing is the discipline concerning techniques that allow computers to extract and manipulate information available from written or spoken natural human language.

Traditional information technology requires data to be presented to computer systems encoded in strictly syntactically precise formats: data that does not come from sensors performing physical measurements must be entered by human operators using specifically designed artificial languages or form-based interfaces. Even with extensive human intervention, representing general semantic information, such as the plot of a story or the sequence of events in a newspaper article, has proven to be very challenging.

From the beginning of the electronic computer era, in the 1950s, computer scientists sought to automate the processing of natural language, founding what would become one of the main research fields of artificial intelligence. In fact, one of the first definitions of artificial intelligence, the Turing Test, involved the ability of a machine to impersonate an human in a textual conversation.

Inspired by mathematical linguistics and early *cognitivist* psychology, which tended to view human thought as an essentially symbolic manipulation process driven by syntactic rules, the first attempts at natural language processing were aimed at modeling the syntax and semantics of natural languages in terms of formal grammars, and constructing rule-based expert systems to manipulate such symbolic information. In particular, research initially focused on automatic translation from Russian to English.

Despite initial high hopes, such approaches proved essentially fruitless: while some toy systems, restricted to very small and unambiguous vocabulary and simple syntax, were successfully demonstrated, attempts to extend them to real world language use cases failed. It turned out that natural language, and the thought processes associated with its production, are too complex and nuanced to be captured by a simple set of hand-coded syntactic rules.

Progress in natural language processing remained limited until statistical approaches started to surface in the 1980s.

Built on both theoretical advances in machine learning techniques and increased computational resources that allow to manage large data sets, and also inspired by the shift in linguistics and psychology from strict symbolic *computationalism* towards *connectionism*, statistical natural language processing consists in analyzing large sets of real world language samples (known as linguistic corpora), in order to automatically derive probabilistic models suitable for various processing tasks.

Rather than modeling language and language manipulation tasks using "hard", discrete symbolic manipulation rules, statistical methods typically yield "soft" models involving probabilities or some other form multi-valued confidence measure. In particular, modern approaches tend to be based on

Bayesian inference, which has both strong theoretical underpinnings and extensive practical success records.

Natural language processing has a large number of applications, including:

- *Speech recognition*: transcribe spoken language into written text, or some other representation suitable for input to other NLP tasks. Speech is generally more difficult to process than text, because of phonetic issues (noise, voice differences between speakers) and the fact that spoken language is even less syntactically precise than written text (it may contain fillers, repetitions, aborted or "repaired" sentences, etc.)

- *Semantic analysis*: determine the "meaning" of a text or speech. Extract the high-level conceptual information, converting it into a form which can be used to automate or aid decision making. It involves a number of different tasks, such grouping sentences into speech acts and classifying them by function in the discourse (question, claim, command, etc.) and their relationship, determining the general topic of the discourse, identifying specific conceptual entities and classifying them by their semantic role and their relationship, and so on. Open-ended, domain-independent semantic analysis of is extremely difficult for a machine, as it would require it to possess human commonsense knowledge and to follow human thought processes. For this reason, the task is usually limited to application-specific domains, where the input text is assumed to loosely adhere to some known format and there are a limited number of well defined conceptual entities that can be recognized, usually given as a formal ontology.

- *Sentiment analysis*: determine the opinions of speakers on certain topics. Related to semantic analysis, though more focused on detecting feelings and emotions rather than conceptual information. It can be targeted at analyzing the sentiments of a single speaker in a single interaction (e.g. a phone call from a customer to a product support call center) or at summarizing the general public opinion on a product,

organization or political issue or candidate from user-generated content on blogs and online social networks.

- *Natural language generation*: report to the user the output of a computation (an NLP task, database query or some other process) as natural text or speech. Common applications involve data-to-text transformations, such as the generation of weather forecasts or summaries of financial data, or text-to-text transformations, such as automatic text summarization, syntactic simplification to make text more accessible to young or non-native speakers, or, more experimentally, changing the tone of the text, and automatically generating jokes and puns.

- *Natural language user interface*: allow the user to control the functionality of a complex computer system through spoken or textual commands. Rather than forcing the user to learn a specific set of commands and combinators, as when interacting with an UNIX shell or a traditional programming environment, the goal is to accept instructions in a form as "natural" as possible, ideally as if the user was interacting with another human. Often, the system is also required to produce output in natural language form, hence this task involves a combination of semantic analysis, natural language generation, and possibly speech recognition and speech generation.

- *Machine translation*: Translate the text or speech in a given natural language to a different natural language. The translation should be both *accurate* (faithfully preserve the information, sentiment and tone of the source text or speech) and *fluent* (appear syntactically correct and idiomatic to a native speaker of the target language). Machine translation of written text is the focus of this PhD thesis.

These applications are based on several common low-level processing techniques. In the next section, we will review these techniques, with particular focus on those that have been used in this work.

2.2.1 Syntactic segmentation

The initial stages of text processing usually involves breaking the incoming stream of characters into its basic constituents and classify them according to their syntactic function. While the exact number and type of these stages varies from system to system, a text pre-processing pipeline is typically organized according to the following structure:

Sentence splitting

Sentences are the smallest linguistic units that are essentially syntactically self-contained. Splitting the input text into sentences is usually the first stage of any NLP system, since, at least at syntactical level, they can be processed independently. In European languages, sentences are usually delimited by specific punctuation characters (dot, question mark, exclamation mark, etc.), with some exceptions. Some other languages lack punctuation, making sentence splitting more difficult. Typical state of the art modules for European languages, like the Punkt Tokenizer, are based on a technique involving unsupervised learning from an unannotated corpus (Strunk et al., 2006) the system attempts to detect whether a dot in the text is used to denote a sentence boundary or an abbreviation by using a learned a dictionary of words which are likely abbreviations (detected by their brevity, presence of internal dots, etc.), and also by taking in consideration the word on the right of the dot (whether it is capitalized, or it appears frequently after the word on the left in the training corpus).

Word tokenization, lemmatization, morpheme extraction

Sentences can be further decomposed in words, which are defined as the smallest units of meaning that can be uttered or written in isolation. Word segmentation is a necessary pre-processing step for most other stages, since they typically operate at word level. In languages written using spaces, or equivalent punctuation characters, word segmentation is relatively easy. However, in some languages, such as German, long compound words need

to be broken down, similarly, in languages such as Turkish, single words can be composed by a large number of morphemes (smallest units of meaning) that should be considered separately. By contrast, in languages like Italian, short sequences of words that appear in a fixed form and have a specific meaning, should be aggregated as single words. Detecting word boundaries in spoken language is much more difficult, as inter-word pauses and even entire syllables are routinely dropped in normal speech.

2.2.2 Tagging

In NLP applications, once the input text or speech has been broken down into sentences and words, we often want to classify each word by assigning it a *tag*: a label in a small, discrete set.
One of the most typical tagging task is *part of speech* (POS) tagging: a very basic form of syntactic analysis where each word is labeled according to coarse syntactic function, such as verb, noun, adjective, etc. POS tagging is typically used as a preprocessing step for more sophisticated syntactic analysis (parsing) or syntax-based machine translation.
Other tagging tasks include *named entity recognition* (the identification of names of people, places, organizations, or dates and numerical values, etc.), "shallow parsing" (breaking the sentence into syntactic "chunks": non-overlapping consecutive strings of words with a coherent syntactic function such as "noun phrase", "verb phrase", etc.), super-sense tagging (classifying words according to coarse semantic categories), and so on.

In general, in all tagging tasks, the tag of each word depends not only on the specific word but also on a context of nearby words, hence, in machine learning terminology, tagging is a *structured prediction* problem (section 2.1.2). However, compared to other structured prediction problems that occur in NLP, such as parsing and machine translation, tagging is often much simpler. By making reasonable independence assumptions, tagging can be reformulated as a *multi-class classification* problem over small windows of consecutive words (n-grams), or as a structured prediction problem over models with *optimal substructure* properties (Bellman, 1957), such as *hidden Markov models*

(Stratonovich, 1960) or *conditional random fields* (Lafferty et al., 2001) where both prediction and training can be performed efficiently.

2.3 Parsing

In addition to word-level part of speech tags and shallow syntactic chunks, natural language has a recursive syntax, which controls how words or phrases that be quite distant in a sentence correlate with each other and combine to convey meaning.
The task of discovering the syntactic relationships in a sentence is known as *parsing*. Parsing is usually performed after POS tagging and possibly syntactic chunking, and produces a *parse graph*, typically a tree or a forest.
The form of the output being produced depends on the syntactic model (*grammar*) that we use. The most popular types of grammar are *generative grammars*, which produce *constituency parses*, and *dependency grammars*, which produce *dependency parses*.

2.3.1 Constituency parsing

Phrase structure grammars

Generative grammars (Chomsky, 1957) attempt to model the thought processes of a speaker producing a sentence as the iterative application of symbolic computational rules.

Given a finite alphabet A of symbols, consisting of *terminals* W (the words in the lexicon of the language) and other symbols known as *non-terminals* V, each rule $lhs ::= rhs$ specifies a (non-empty) *precondition* string $lhs \in A^*$ and a *postcondition* string $rhs \in A^*$.
The alphabet A, a finite set of rules R and a distinguished non-terminal start symbol $S \in V$ define a phrase structure grammar G.

Probabilistic context-free grammars (PCFG), also known as *stochastic context-free grammars* (SCFG), are a type of phrase grammar where each rule has precondition consisting of exactly one non-terminal and has an associated probability, such that these probabilities add up to one over rules with the

same precondition.
Sentence generation can be modeled as a stochastic process, where, starting from a string consisting only of the start symbol, at each step a non-terminal is selected and it is expanded by randomly and independently sampling a rule with matching precondition according to its associated probability [11]. Therefore, given a sentence x and one of its admissible constituency trees y, their joint probability $P(y, x)$ according to the grammar G is just the product of the probabilities of all the rules that appear in y, each taken with the multiplicity of its instances.

In a PCFG, the *derivation* of a sentence from the start symbol can be represented as an ordered tree of symbols, where the leaf vertices are labeled by terminals and internal vertices are labeled by non-terminals, with the start symbol at the root. Each internal vertex represents one application of a rule which rewrote the non-terminal in its label with the labels of its children vertices.
Each subtree in a derivation tree covers a contiguous substring of words in the sentence, denoted as *syntactic phrase*. Therefore, the child-to-parent relation can be interpreted as a *constituency* relation between syntactic phrases: each child phrase *constitutes* a part of its parent phrase.
For this reason, derivation trees are known as *constituency trees*.

We can estimate a PCFG from an annotated corpus and use it to parse any sentence x by computing the admissible constituency $\overset{*}{y}$ which maximizes the joint probability $P(y, x)$. This enables us to avoid the complexity of manually designing a grammar and resolve the ambiguity of parsing by using probabilities.
Given a sentence x, the *statistical parsing* task consists of finding the maximum likelihood constituency tree y which describes the generation of x according to a context-free grammar G.

A discussion of model estimation techniques and the details of the various parsing algorithms is beyond the scope of this document. It suffices to note that, due to the independence assumptions in the generation process,

[11] we do not make stochastic assumptions about the order in which non-terminals are expanded, since it does not influence the outcome of the process.

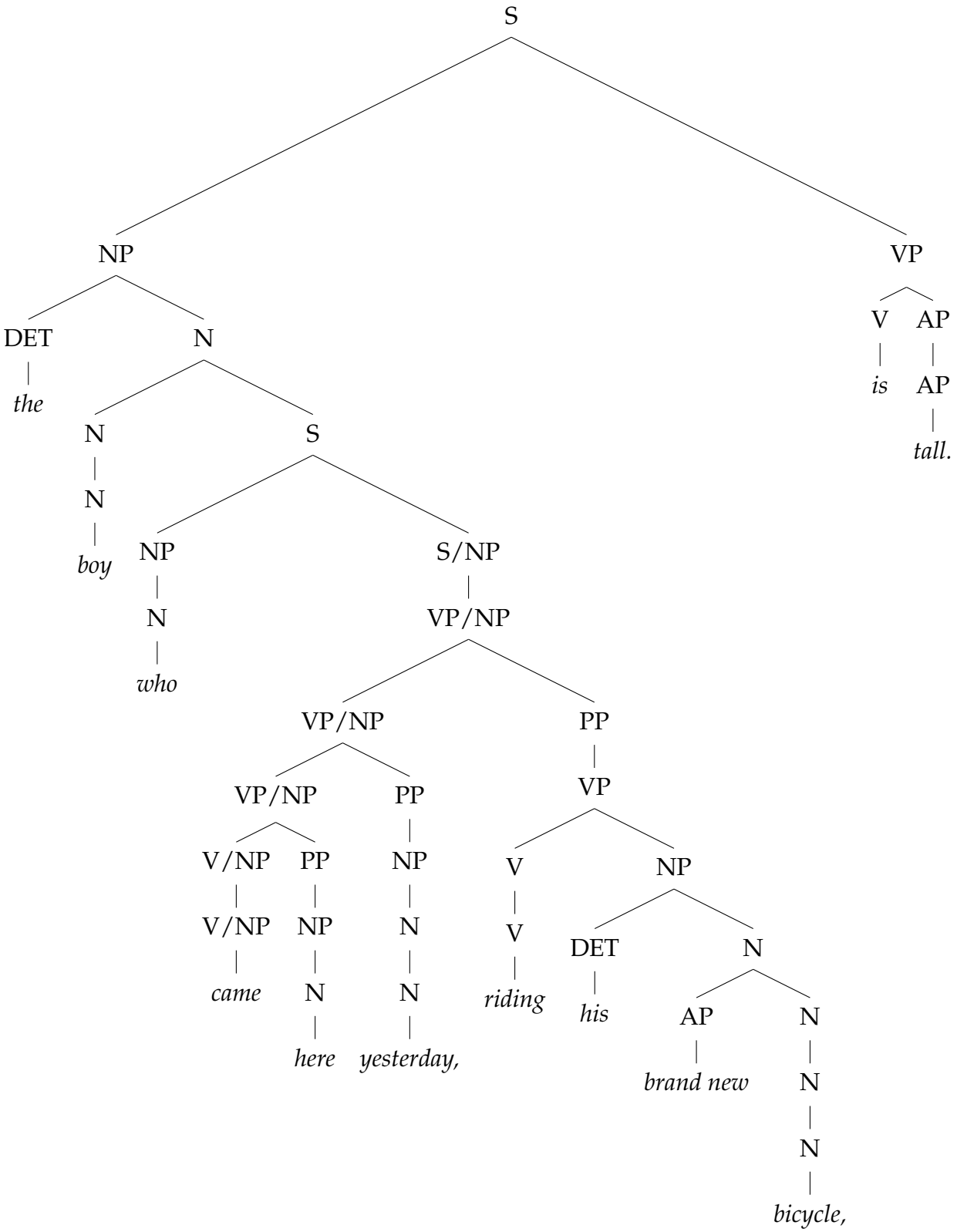


Figure 2.3: Constituency tree for the sentence "The boy who came here yesterday, reading his brand new bicycle, is tall.", statistically parsed with the DELPH-IN PET parser (http://moin.delph-in.net/PetTop).

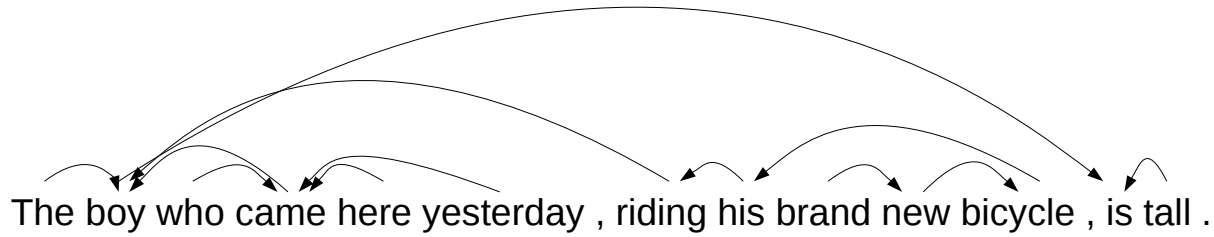


Figure 2.4: Dependency tree for the sentence "The boy who came here yesterday, reading his brand new bicycle, is tall.", labels not shown.

PCFG parsing retains the optimal substructure property of classical CFG parsing which enables dynamic programming. In particular chart parsing algorithms such as the *Cocke-Younger-Kasami* (CYK) algorithm (Cocke, 1969) and the *Earley* algorithm (Earley, 1970) can be adapted to perform this task, maintaining their asymptotic complexity (Charniak, 1997a,b; Stolcke, 1995).

While enabling efficient parsing, the stochastic independence assumptions of PCFGs can be often excessively strong, limiting the ability of the parsing algorithms to resolve ambiguity in a linguistically meaningful way. One way to relax this assumption is to *lexicalize* the grammar (Collins, 2003), by decorating each non-terminal in a constituency tree with an *head word*: the "most important" word on its underlying subtree.

2.3.2 Dependency parsing

Instead of modeling the syntax of a sentence in terms of syntactic constituents, *dependency grammars* (Tesnière, 1959; Mel'čuk, 1988) attempt to model the syntactic *dependency* relation between words.
Introduced by Lucien Tesnière in the 1950s, this theory of syntax views the main verb of a sentence as the *head*: the most important word of the sentence, with the main subject and object nouns or pronouns as its direct modifiers, their adjectives and subordinate verbs as their own modifiers and so on, resulting in a hierarchy of head-modifier dependency relationships between words.
Formally, this can be described as a directed graph on the words of a sentence, usually a tree rooted at the main verb (or a forest, if a sentence has

multiple main verbs). The edges are often labeled by a dependency relation tag (DEPREL) which describes the type of the syntactic relationship between the pair of words it connects. If each subtree in the dependency graph covers a contiguous substring of the sentence, the graph is defined as *projective*.

Dependency graphs represent information about a sentence which is in some ways related and in other ways orthogonal to the information represented by constituency trees. Non-projective dependency trees, in particular, may be more informative for languages, such as German, Italian or Czech, which allow significant freedom in the order of words in a sentence, while constituency trees, which are always projective, may be better suited for languages with a substantially fixed word order, such as English.
The syntactic dependency relation often carries more semantic content than the constituency relation, and for this reason it is widely used in semantic NLP tasks such as named entity recognition, semantic role labeling and text understanding.
Dependency relationships are also often relatively invariant between languages, which makes them interesting for syntax-based machine translation (Fox, 2002; Hwa et al., 2005).

Dependency parsing is often performed using statistical discriminative learning techniques.
Instead of building an explicit dependency grammar, the system directly estimates a conditional probability model $P(y|x)$ relating a sentence x to a dependency graph y, which is maximized during parsing using a dynamic programming, beam search or even greedy search approach.

In a global parser (Eisner, 1996; McDonald et al., 2005), a score $F(y, x)$ proportional to the logarithm of $P(y|x)$ is computed, typically as a linear combination of *features* $f(y, x)$:

$$F(y, x) \equiv \theta^T \cdot f(y, x) \tag{2.34}$$

where $\theta \in \mathcal{R}^n$ is a parameter vector estimated from an annotated training corpus and $f(y, x) \in \mathcal{R}^n$ is a feature vector depending on the sentence x and the candidate parse graph y.

Other approaches, known as *transition-based* methods (Yamada and Mat-

sumoto, 2003), model parsing as a formal automaton which recognizes the sentence x, producing a parse graph y as a side effect of its state transitions. By associating features to these transitions, linear, or even non-linear, scores can be computed, enabling us to define parsing as a score maximization problem.
In this thesis work, we extensively used the transition-based dependency parser DeSR (Attardi, 2006).

2.4 Language modeling

NLP applications that generate natural language, such as speech recognition or machine translation, have the requirement of *fluency*: the generated text must be syntactically correct and idiomatic.

If the output has a relatively fixed form, and is generated by a limited number of hand-coded rules, then acceptable fluency can be usually guaranteed by careful design of these rules. However, if the output is in free-form and the system is trained by statistical methods, as most state of the art speech recognition or machine translation system are, then there is the need of a method to evaluate the fluency of arbitrary sentences.

A *language model* is a stochastic model that estimates the probability that a speaker of some language will utter or write any given sentence. In terms of *Bayesian inference*(Box and Tiao, 1992; Jaynes, 2003), this probability can be interpreted as a *prior* over sentences. In addition to estimating the probability of complete sentences, these models are also designed to provide estimates for prefixes or arbitrary consecutive segments of sentences, in order to guide the language generating components as they incrementally build sentences by concatenating words or segments together.

The computation of a precise prior over sentences would require having an accurate model of all the speaker's mental processes that are involved with conscious thought and language production. Of course, this is way beyond the capabilities of any present machine, therefore actual systems use estimations statistically inferred from large amounts of training text. The rest of this section will describe the prevalent types of language models and

estimation techniques that are relevant to this work.

2.4.1 N-gram language models

Given a sentence x of L_x words $(x_1, \ldots, x_{L_x})$ we want a model which allows us to compute the unconditional probability $P(x)$. Usually, we also want to be able to compute the probability for arbitrary prefixes of a sentence $P(x_{:j})$, or, in applications such as syntax-based machine translation, the probability of contiguous substrings[12] $P(x_{j:j'})$.

It can be observed that in any natural language the frequency of different words varies greatly. For instance, in English, the most frequent word "**the**" makes up about 6-7% of a typical training corpus, while some words are very infrequent and may occur only once or twice even in very large training corpora. In general, word frequencies appear to be distributed according to a *heavy-tailed* probability distribution, known as *Zipf's law* or *Zeta probability distribution* (Zipf, 1935).

Since word frequencies vary so much, it makes sense to exploit them in a language model: if a sentence has many rare words, we can tell that its probability is likely low.
We can formalize this intuition by making a strong assumption of independence between individual words, and hence model the probability of a sentence (or a fragment of it) just as the product of its word probabilities:

$$P(x) = \prod_{j=1}^{L_x} P_1(x_j) \tag{2.35}$$

where $P_1(w)$ is a categorical probability distribution, estimated at maximum likelihood using the empirical frequencies of word occurrence in a monolingual training corpus:

$$\overset{*}{P_1}(w) = \frac{c(w)}{\sum_{w' in W} c(w')} \tag{2.36}$$

where W is the language lexicon and $c(w)$ is the empirical count of word w. This model is known as the *unigram* language model.

[12] assuming that sentences are always segmented at word boundaries

Of course, the independence assumption of the unigram language model is too strong: it would estimate the probability of all permutations of a given sentence the same, even if obviously most permutations of realistic sentences are gibberish that has virtually zero probability of being even produced by an actual speaker.

In a natural language, words are strongly correlated to each others. While some correlation occurs between distant words, and modeling it generally requires syntax-aware methods, the strongest correlations occur at short distance. We can model them by conditioning our word probability distributions for each word on a window of nearby words.

Specifically, for some small order $k > 2$, we condition the word probability on the $k-1$ previous words:

$$P(x) = \prod_{j=1}^{L_x+1} P_k(x_j|x_{j-k+1:j-1}) \tag{2.37}$$

(note that we consider the sentence augmented by one special terminator symbol "**</s>**" at index $L_x + 1$, and $k-1$ padding symbols "**<s>**" at non-positive indexes, which enable the model to take into account statistical regularities that occur at the start and the end of sentences.)

The conditional word probability distributions are assumed to be categorical probability distributions for each choice of the conditioning values.

This type of model belong to the class of *discrete-time Markov chains* (Damerau, 1971; Markov, 1971).

In principle, we could estimate the parameters of these conditional word probability distributions at maximum likelihood from the empirical frequencies:

$$P_k(w|\overset{*}{w_1},\dots,w_{k-1}) = \frac{c(w|w_1,\dots,w_{k-1})}{\sum_{w' in W} c(w'|w_1,\dots,w_{k-1})} \tag{2.38}$$

where $c(w|w_1,\dots,w_{k-1})$ is the number of times word w appears in the training corpus preceded by the sequence of words $w_1,\dots,w_{k-1}$.

However, the distribution of the frequencies of these k-grams (sentence fragments) is heavy-tailed much like the distribution of single words, and in fact, the higher the order k, the more extreme this phenomenon becomes (Egghe, 2000). There are $|W|^k$ k-th order possible n-grams, where the lexicon

size $|W|$ is in the order of $10^5 - 10^6$, and we have to estimate a numerical parameter for each of them. Even for $k = 3$, this exceeds the number of words of any realistic training corpus: many n-grams will appear only once or twice in the corpus, making the empirical frequency an unreliable estimator of the underlying probability. Many more n-grams will not appear at all in the training corpus, and therefore they will have a frequency equal to zero or even undefined (with an empirical count ratio of $0/0$).
In order to address this *sparsity* issue, we need more sophisticated models and estimation techniques to assign a reasonable probability to n-gram which occurred few times or none at all in the training corpus. A discussion of these so-called "smoothing" techniques is beyond the scope of this document. The Kneser-Ney (Kneser and Ney, 1995) and modified Kneser-Ney (Chen and Goodman, 1999) methods which have become the state of the art techniques for n-gram language model estimation.
In typical applications, n-gram orders between 3 and 5 are used, as anything above that is considered too computational expensive and prone to overfitting.

2.4.2 Neural language models

Although standard n-gram language models are very good at capturing very localized linguistic phenomena, they do not scale well enough to accurately model medium-distance ($5 - 7$ words) or long distance (sentence-wide) correlations between words. In order to overcome these limitations, we can use discriminative machine learning techniques with stronger generalization performances, such as *neural networks* (section 2.1.3).

Multi-layer perceptron language models Using the same k-th order Markov chain stochastic model of standard n-gram language models, we can modify the parametrization of the conditional word probability distribution $P_k(w|w_1, \dots, w_{k-1})$, replacing the conditional categorical distribution with a discriminative multiclass classification model with per-class probability outputs: a model that given a context of $k - 1$ consecutive words, computes a

probability distribution for the next word.
Specifically, the model will have $|W|$ real-valued outputs, one output y_w for each word w in the lexicon, and $|W| \cdot (k-1)$ binary valued inputs, corresponding to the context words each represented using the "one-hot" encoding:

$$\forall w' \in W, \forall j \in 1, \ldots, k-1,\ \hat{x}_{w',j} = \begin{cases} 1 & \text{if } w' = w_j \\ 0 & \text{otherwise} \end{cases} \tag{2.39}$$

Multi-layer perceptron neural networks are a natural choice for this type of problems, since they can be efficiently trained on large training sets (which are normally available for language model training, since they are just plain text corpora), they enable the many output units to share the same hidden units, limiting the model complexity, and, using the *multiple logistic softmax* (eq. 2.10) activation function in the output layer, they directly constrain the outputs to be normalized probabilities.

In the original model proposed by Bengio et. al (Bengio et al., 2006), there are two hidden layers, the first one with $d \cdot (k-1)$ linear units [13], the second one with s units with hyperbolic tangent activation, where d and s are hyperparameters controlling the capacity of the model.
The input-to-first-hidden parameters are constrained so that, for each position j in the context, the $|W|$ inputs corresponding that position feed only the d first-hidden units corresponding to the same position j, and the parameters of these connections are independent from the position j:

$$\Theta^{(1)}_{a,b} = \begin{cases} \Theta^{E}_{c,w_j} & \text{if } a = d \cdot j \text{ and } b = |W| \cdot j + c \\ 0 & \text{otherwise} \end{cases} \tag{2.40}$$

note that there is no bias feature for this layer.
The matrix $\Theta^E \in \mathcal{R}^{d \times |W|}$ is known as *word embeddings* matrix: for each word w in the lexicon, the corresponding column is a d-dimensional continuous representation of w.

[13] the linear hidden layer does not increase the effective *depth* of the network, but is nevertheless used because it allows an useful parametrization.

All the parameters of the model can be estimated using standard MLP training techniques, minimizing cross-entropy loss with L2-regularization using stochastic gradient descent.

In practice, for large lexicon size ($> 10^5$), prediction and especially training can be computationally expensive due to the large number of output units.
A number of techniques have been developed to reduce the time complexity of these model by performing approximations during training (Schwenk and Gauvain, 2005; Bengio and Senecal, 2008), or by modifying the output layer to use hierarchical encoding (Morin and Bengio, 2005), which groups words into hierarchical classes based on their frequencies, or by mapping the problem to binary classification task (Mnih and Teh, 2012), where the model takes both the target word w and its context $w_1, \ldots, w_{k-1}$ as inputs and computes the probability using a single output unit, estimating the parameters using the *noise contrastive method* (Gutmann and Hyvärinen, 2010).

Once they are trained MLP language models can be used in most applications as a drop-in replacement for standard n-gram language model or they can be used in combination with them, as they often have complementary strengths and weaknesses.
Moreover, the word embedding vectors, computed as a side effect of the training process, are themselves useful as features for various NLP tasks (Turian et al., 2010; Mikolov et al., 2013a).

Recurrent neural network language models While MLP language models exploit the generalization power of neural networks to overcome the overfitting and complexity issues of standard n-gram models, they are still limited by the fact that they can only model word correlations up to a distance equal to the order hyperparameter k.

Recurrent neural networks can be used overcome this limitation, allowing to model correlation over theoretically arbitrary distances.
Mikolov et al. (Mikolov et al., 2010, 2013b), introduced a RNN model with $|W|$ binary inputs in the input layer, one for each word in the lexicon, a

single hidden layer with feedback connections and $|W|$-dimensional output layer of probability estimates.
Starting from a default initial state $v(0)$, the network receives the words of the sentence one at time, updating its internal state and computing a probability distribution over the next word.
Training is performed using stochastic gradient descent with backpropagation-through-time. The column of the input-to-hidden parameters matrix can be interpreted as word embedding vectors.

RNN language models are usually more accurate in terms of empirical likelihood that MLP language models, since they can model medium-distance and long-distance correlations [14]. However, they can't be used as drop-in replacements of n-gram or MLP language models in applications such as speech recognition and machine translations, since their continuous valued internal state can't be easily processed by the dynamic programming algorithms that are commonly used for solving these structured prediction problems.
Specialized reranking algorithms which can make use of RNN language models have been developed, such as (Auli et al., 2013).

2.4.3 Syntactic language models

Neural language model can capture linguistic phenomena that entail medium distance and medium-long distance (up to about ten words) correlations. Due to the recursive syntactic structure of natural language sentences, however, this may not be enough to model all the relevant linguistic phenomena.
For instance, in the sentence "**The boy who came here yesterday, riding his brand new bicycle, is tall.**", the main verb "**is**" and its subject "**boy**" and subjective complement "**tall**" are obviously strongly correlated: changing one of them without appropriately changing the others can easily make

[14] but note that even when not explicitly enforced, these networks will typically have the echo state property, which will cause their sensitivity to word correlations to drop off exponentially with the distance.

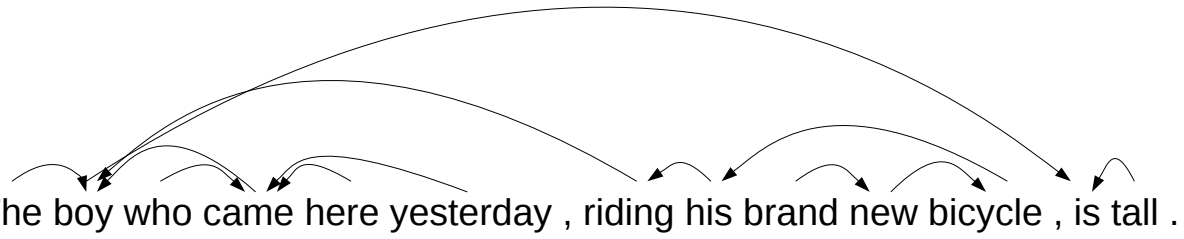


Figure 2.5: A sentence with a possible (unlabeled) dependency parse tree. Note that the main verb "**is**" and its subject "**boy**" are directly connected in the parse tree despite the fact that they occur far from each other in the sentence.

the sentence nonsensical. But the language models described so far would generally fail to appropriately model the long-distance correlation between "**boy**" and "**is tall**".

If we are able to parse the sentence according to some syntactical model, either a constituency grammar or a dependency grammar, these long-distance correlations become rather short when considering the distance on the parse graph. Syntactic analysis also produces a variety of other information such as *part-of-speech* (POS) tags, *dependency relation* (DEPREL) tags, lemmas, morphology, etc., which can be all used as features for a machine learning model.

Statistical parsers generally produce a model score, which can be often interpreted as a (log) probability. We could simply parse a sentence and use the parser score as a language model probability. This approach however, does not produce good results when applied to machine translation reranking tasks (Och et al., 2004; Post and Gildea, 2008; Carter and Monz, 2011), possibly because statistical parsers are trained on corpora consisting entirely of substantially grammatical sentences, while the outputs of a statistical machine translation system or a speech recognition system are often ungrammatical, and therefore fall outside the input space the parser was trained to process.

Instead of parsing the output of a generic SMT system with a conventional parser, it may be better to use a string-to-tree (Shen et al., 2008, 2010; Zollmann and Venugopal, 2006; Galley et al., 2006) or tree-to-tree (Nesson et al., 2006; DeNeefe and Knight, 2009; Smith and Eisner, 2006) SMT

system which directly generates a parse graph for the target sentence while computing it.
The language model probability computation can be formulated as a structured input probability regression problem, yielding a model which is either used for reranking, or is incorporated directly into the SMT decoder.

Shen et al. (Shen et al., 2008, 2010), use a generative syntactic language model for rooted dependency parse trees: the probability of a sentence is decomposed in Markovian fashion in a product of categorically-distributed word probabilities, each conditioned on a local context.
Collins et al. (Collins et al., 2005b) introduced a syntactic discriminative language model based on these principles for constituency parse trees, which is used for reranking in speech recognition tasks, which has been also adapted (Carter and Monz, 2011) to machine translation reranking (Shen et al., 2004).
Similar discriminative language models for dependency parse trees also exist: (Lambert et al., 2013; Popel and Mareček, 2010) (building on earlier work by Charniak (Charniak, 2001; Charniak et al., 2003)).

2.5 Statistical machine translation

Machine translation was one of the first research fields of computer science to be investigated. The first public demonstration occurred in 1954 with the Georgetown-IBM experiment (Hutchins, 2004), which consisted in the translation of about sixty sentences from Russian to English. However, the field really started to produce promising results only in the nineties, with the introduction of statistical methods pioneered by IBM (Brown et al., 1990, 1993) (although already suggested as early as 1949 by Weaver (Weaver, 1955)). Rather than using hand-coded rules, a translation model is learned from parallel corpora: large sets of texts with sentence-by-sentence reference translations. In the simplest approaches, source sentences and their reference translations are considered just as strings of words, and their stochastic relations are estimated using statistical methods, without any explicit modeling of syntax.

Statistical machine translation (SMT) can be considered a machine learning structured prediction problem: given a sentence in a source language, determine its best translation in a target language, according to a model learned from a parallel corpus.

Most SMT approaches consider the sentence as the fundamental unit of translation [15]. Given a sentence f in the source language, find its best translation e in the target language [16], according to a model learned from training data:

$$\overset{*}{e} \equiv \underset{e}{\operatorname{argmax}} P(e|f) \tag{2.41}$$

Applying Bayes' theorem, this formula can be decomposed as:

$$\overset{*}{e} = \underset{e}{\operatorname{argmax}} P(f|e) \cdot P(e) \tag{2.42}$$

where the prior probability $P(e)$ (the *language model* probability for the target language) provides fluency while the likelihood $P(f|e)$ provides accuracy.
This view lends itself to a theoretically useful, albeit fictitious, interpretation of the translation process known as the *noisy channel model*, in analogy to the noisy channel models used in information theory and electrical engineering: Assume we are translating from Italian to English. Under the noisy channel model, each Italian sentence is considered to be a "corrupted" version of an English sentence, originally generated by a random source of English sentences ($P(e)$) and then transmitted over a noisy channel ($P(f|e)$) which converts it to Italian. The translation task is therefore to undo the action of the channel, decoding the Italian sentence and recovering the "original" English. Although not realistic, this model allows us to leverage the framework and techniques originally developed for electrical engineering, and has been widely used in the early SMT approaches, although more recent approaches deviated from it.

[15] This makes an independence assumption between sentences which is not actually realistic. Translating at document level can yield superior performances, but it may be more computationally expensive, and it is a recent area of research (Hardmeier, 2012).

[16] Due to a historical convention, f stands for "foreign" or "French" and e stands for "English".

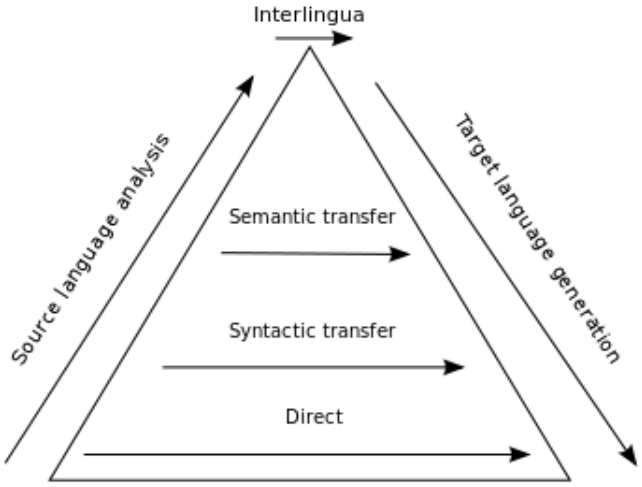


Figure 2.6: This diagram, known as "Vauquois triangle", represents the various ideal levels of machine translation approaches, from direct, word-based and phrase-based translation (bottom), to syntactic transfer, semantic transfer, up to translation by a hypothetical language-independent intermediate representation (interlingua). "Oblique" approaches, that use a different level of abstraction on the source language an the target language are also possible.

SMT approaches can be characterized by the level of representation they operate at.
The earliest methods operated at the level of single words. Although not competitive anymore as complete translation systems, some of these techniques are used in the training processes of other approaches.
Better performances can be obtained by methods which operate at the level of contiguous spans of words, known as *phrases*.
Other approaches exploit the recursive structure of natural language, both in a fully automatic form or by making use of expert-derived linguistic knowledge, which is incorporated in the models using "soft", probabilistic rules.
Ideally, we could hypothesize methods which abstract over syntax and operate at the level of meaning, possibly using a language-independent "interlingua" representation, however operating at a such high level is currently not possible and in fact may not necessarily be a good idea, since the higher the level of abstraction, the further the risk of losing some important information.

In the following sections we sections we will briefly introduce the principles behind the modern state of the art translation systems, such as *Moses* (Koehn et al., 2007), and the background techniques particularly related to this thesis work.

2.5.1 Word-based methods

Words are generally considered to represent the elementary units of meaning in a sentence, and usually they have some degree of correspondence between different languages, which enables us to compile bilingual dictionaries. Therefore, the simplest form of translation is to replace each word in the source sentence with its most likely translation in the target language.
For instance, the Italian sentence "**Mary è una ragazza.**" can be translated word by word into English as "**Mary is a girl.**".
Obviously, this approach is inadequate in many cases, as words are translated completely without context. For instance "**Mary legge il suo libro**

preferito." might be translated as "**Mary reads the his book favorite.**". Although the meaning is arguably preserved, the translation is ungrammatical: the *content words* (the two nouns, the verb and the adjective) are rendered correctly but they are not in the right order, pronoun gender is not in agreement with the gender of its subject and the determiner is spurious. In more complex examples, even the meaning could be lost.

We can improve this direct dictionary translation method by including a language model. The translation problem becomes:

$$\overset{*}{e} = \underset{e}{\operatorname{argmax}} \prod_{j=1}^{L_f} P_{LEX}(f_j|e_j) \cdot P(e) \tag{2.43}$$

where L_f is the length of the source sentence (and the target sentence as well, by construction), f_j and e_j are the j-th word in the source and target sentence, respectively and $P_{LEX}(f_j|e_j)$ is the *lexical translation probability* function: for each choice of target word, it is a categorical probability distribution over the set of source words.

For a n-gram language model, this is just a hidden Markov model decoding problem, which can be efficiently solved using standard dynamic programming techniques such as the *Viterbi algorithm* (Viterbi, 1967).

Given a training parallel corpus of sentence pairs, where each target sentence has the same number or words of the target sentence, the lexical translation probabilities can be estimated with maximum likelihood just from observed frequencies:

$$\overset{*}{P}_{LEX}(w_f|w_e) = \frac{c(w_f|w_e)}{c(w_e)} \tag{2.44}$$

where $c(w_e)$ is the number of times the target word w_e appears in the training set and $c(w_f|w_e)$ is the number of times the source work w_f appears in correspondence to the target word w_e.

Although a language model improves the quality of the translation by providing some context on the target side (for instance it would presumably solve the pronoun concordance issue of our previous example), this system is still quite limited. We would like our translation system to be able to swap, insert or drop words.

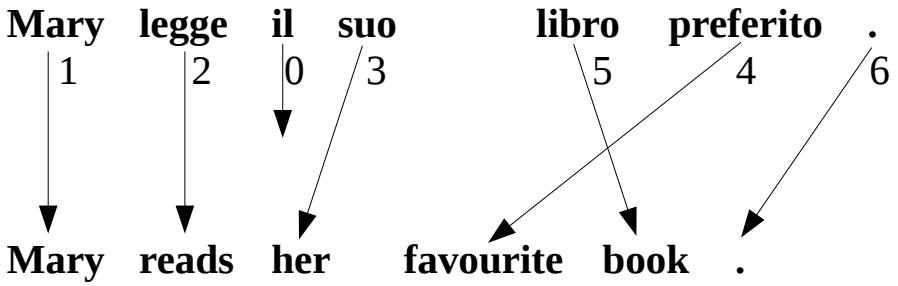


Figure 2.7: A sentence pair with a reasonable source-to-target word alignment.

Word alignments Even if the strict one-to-one, monotonic [17] correspondence assumption of our previous toy model turned out to be excessively strong, words in a sentence and its translation do happen to generally correspond to each other.

A key observation, which lead to the development of the first workable SMT systems (Brown et al., 1990), is that we can treat this correspondence, known as *word alignment* as a hidden variable in our stochastic model of the translation process, whose probability distribution relative to the source and target words can be estimated from the training corpus and which can be marginalized during decoding.

Today, word-based approaches are not used anymore as full translation system, but they are used to estimate the maximum likelihood word alignment of the training corpus and lexical translation probabilities which are then used in the training process of more advanced translation systems.

Given a source sentence f of length L_f and a target sentence e of (possibly different) length L_e, the source-to-target *alignment function*

$$a_{f,e} : 1, \ldots, L_f \rightarrow 0, \ldots, L_e \tag{2.45}$$

maps each position in the source sentence to a position in the target sentence, including a special NULL position at index 0 representing a lack of corresponding word. Each source word is therefore aligned to zero or one target word, while each target word can be aligned to multiple source words.

Making appropriate independence assumptions, the source sentence

[17] In the context of machine translation, "monotonic" means preserving the word order between the source and the target.

generation can be modeled by the following stochastic process: first the target sentence e is generated according to the language model probability $P(e)$. Then the length of the source sentence is sampled from some distribution $P(L_f|e)$, and one of the $(L_e+1)^{L_f}$ alignment functions is sampled from the alignment distribution $P(a|L_f,e)$. Finally, for each position j in the source sentence, a word is sampled according to the lexical translation probability of the aligned source word $P_{LEX}(f_j|e_{a(j)})$. Translation therefore consists in decoding e given f while marginalizing over the set of all possible alignments $A(L_f,L_e)$:

$$\overset{*}{e} = \underset{e}{\operatorname{argmax}} \sum_{a\in A(L_f,L_e)} \prod_{j=1}^{L_e} P_{LEX}(f_j|e_{a(j)}) \cdot P(a|L_f,e) \cdot P(L_f|e) \cdot P(e) \qquad (2.46)$$

In the general form, this decoding problem is computationally difficult: note that in principle, for any source sentence f, the set of possible translations $GEN(f)$ is infinite. Even if we constrain the length of the translations to be linear in the source length, there are still exponentially many possible translations, and for each of them, exponentially many possible alignments. In fact, it can be shown that this decoding problem is equivalent to the *traveling salesman problem* (TSP) (Knight, 1999; Zaslavskiy et al., 2009), a well known NP-complete, APX-complete combinatorial optimization problem.
Thus, in order to enable tractable decoding and training, the source length and the alignment probability distributions are restricted to have certain specific forms.

Training can be performed in a fully supervised way using a parallel training corpus of sentences with manually annotated word alignments, or more commonly, in an unsupervised way from an unannotated parallel corpus.
A discussion of word alignment models and their relative algorithms is beyond the scope of this document.

In practice, IBM model 4 (Brown et al., 1993) is often used to produced the (approximate) maximum likelihood alignment of a training corpus and estimate the lexical translation probabilities. This method, along with HMM alignment (Vogel et al., 1996) and the other IBM models introduced in (Brown

et al., 1993), is implemented in the *giza++* open source tool (Och and Ney, 2003).

2.5.2 Phrase-based methods

The main issue of word-based translation methods is that they do not strongly take into account correlations between source words. In fact, their main linguistic assumption that words represent the fundamental unit of meaning, and in particular that the meaning of a sentence simply decomposes over the meaning of its words, turned out to be too strong.
Therefore, in order to properly convey meaning between different languages, it can be useful to translate at the level of *phrases*: short contiguous substrings of words.
Despite the name, these are not necessarily syntactic phrases as defined in section 2.3.1: each of them is not necessarily covered by a single full subtree in a constituency or dependency parse graph of the sentence. In fact, in the basic phrase-based translation method, we do not explicitly model syntax. Hybrid or fully syntactic translation method exist and will be discussed in the next section.

Noisy channel model

We can introduce phrase-based translation by extending the theoretical noisy channel generation process of word-based translation:
First, a target language sentence e of length L_e is generated according to the language model probability $P(e)$. Then, this sentence is segmented into L_s non-overlapping, non-empty, contiguous phrases $\bar{e}_j$, according to the segmentation probability distribution $P_{SEG}(s|e)$, where s is a list of starting positions for each phrase. These phrases are then permuted according to an one-to-one phrase alignment sampled from another distribution $P_{ALIGN}(a|s,e)$, and finally each target phrase $\bar{e}_j$ is replaced with a source phrase $\bar{f}_j$ according to the phrase translation categorical probability distribution $P_{PHRASE}(\bar{f}_j|\bar{e}_j)$.
Note that here the main independence assumption is between the translation of phrases rather than between the translation of words.

The translation problem can be thus modeled as the decoding problem for that noisy channel:

$$\overset{*}{e} = \underset{e}{\operatorname{argmax}} \sum_{s \in S(e)} \sum_{a \in A(s,e)} \prod_{j=1}^{L_s} P_{PHRASE}(\bar{f}_j|\bar{e}_j) \cdot P_{ALIGN}(a|s,e) \cdot P_{SEG}(s|e) \cdot P(e) \tag{2.47}$$

Note that there are 2^{L_e-1} possible ways of segmenting a sentence e, and each segmentation s produces L_s phrases that can be permuted in $L_s!$ ways. We can reduce the number of segmentations by introducing a maximum phrase length M, and the number of permutations by a maximum distortion distance D [18], but the double sum in eq. 2.47 still insists over an approximately doubly-exponential in L_e number of terms.
Therefore, full parameter estimation and decoding in the noisy channel model are impractical.

Practical phrase-based translation models, developed between the late 1990s and the early 2000s by Franz Josef Och and others (Och et al., 1999; Och and Ney, 2002, 2004; Marcu and Wong, 2002; Koehn et al., 2003), manage to effectively tame this complexity by making further simplifying assumptions that enable efficient decomposition of the model probability:

- Instead of considering all the possible phrase segmentations of a sentence pair, they build a dictionary, known as *phrase table*, of phrase pairs up to a certain length seen during training. During decoding, they only consider segmentations which are consistent with these phrase pairs [19]. An uniform probability distribution, or some other simple prior, is considered over these segmentations.

- The distortion (phrase alignment) probability depends only on the *distortion distance* of each phrase pair: the distance between the endpoints of each pair of source phrases which are translated as consecutive target phrases.

[18] The distortion distance of a phrase in a permutation is the distance between the starting points of the phrase in the original string and in the permuted one.

[19] When a source sentence contains a new word not seen during training, the phrase table is temporarily with special phrase pair consisting just of that word translated as itself.

- The target sentence prior probability is computed by a k-th order n-gram language model.

We can further approximate by maximizing over possible segmentations and phrase alignments rather than summing over them, obtaining the following decoding problem:

$$\overset{*}{e} = \underset{e}{\operatorname{argmax}} \max_{s,a} \prod_{j=1}^{L_s} \left(P_{PHRASE}(\bar{f}_j|\bar{e}_j) \cdot P_D(|start(j) - end(j-1) - 1|) \right) \cdot P_{LM}(e) \tag{2.48}$$

where $start(j)$ and $end(j)$ are respectively the start and end position of the j-th source phrase in segmentation s and alignment a, with $end(0) = 0$. In this formula we assumed uniform probability over segmentations.
The distortion probability $P_D(d)$ can be modeled as a categorical probability distribution or as an exponential distribution $P_D(d) = \theta_D \exp(-\theta_D d)$, [20]

Discriminative models

Rather than using a generative model, with strong conditional independence assumptions between the various stages of the generative process, a discriminative, generalized linear model is used to directly estimate a translation score to be optimized.

Using a discriminative model has a number of benefits over a generative one: it avoids overly strong conditional independence assumptions, it enables the incorporation of a large number of additional features of various types, and it optimizes the model resources for accuracy in the space of the most high quality translation hypotheses:
Let $F(e, f, \theta)$ be the model score for a source sentence f and candidate translation e, under parameters θ. We can interpret $F(e, f, \theta)$ as the logarithm of the source-to-target probability $P(e|f)$, up to a normalization constant. However, since we are using this score in a maximization operation, we are only concerned that it accurately tracks the true probability only for candidate translations where this probability is high and have thus a chance

[20] In practice we can ignore normalization and define $P_D(d) = \exp(-\theta_D d)$.

of being selected as the one-best translation. We do not care if the estimation is inaccurate for low-probability candidate translations. By biasing the estimation in this way, we can therefore do better than naive maximum likelihood estimation.
Specifically, we define the model score as:

$$F(e, f, \theta) \equiv \theta^T \cdot h(e, f) \tag{2.49}$$

where $h(e, f) \in \mathcal{R}^n$ is the feature vector of the sentence pair.

The noisy channel model can be seen as a special case of the discriminative model, with feature functions corresponding to the logarithms of the factors in equation 2.48:

$$F_{NC}(e, f, \theta) \equiv \sum_{j=1}^{L_s} \left(\theta_{PI} \log(P_{PHRASE}(\bar{f}_j|\bar{e}_j)) - \theta_D|start(j) - end(j-1) - 1|\right) + \theta_{LM} P_{LM}(e) \tag{2.50}$$

where we assumed to use an exponential distortion distance probability distribution parametrized by θ_D while the other parameters θ_{PI} and θ_{LM} are fixed to one.
The discriminative model allows us to include additional features, and let us optimize all the parameters for maximum translation quality. A typical choice of features include:

- Direct (source-to-target) phrase translation probabilities $P_{PHRASE}(\bar{e}_j|\bar{f}_j)$.
- Direct and inverse lexical translation probabilities $P_{LEX}(e_i|f_i)$ and $P_{LEX}(f_i|e_i)$, defined as in word-based translation model. These features compensate the estimation error that phrase translation probabilities have for long rare phrases, due to data sparsity.
- The length L_e of the target sentence. Multiplied by its own parameter θ_W, usually greater than one, this feature compensates the bias of n-gram language models for short sentences.
- The number of phrase pairs L_s in the segmentation. Multiplied by its own parameter θ_S, this relaxes the assumption of uniform probability

over segmentations, enabling the model to express a preference for segmentations with shorter or longer phrases.

Training is typically performed in two phases. The training proper phase involves extracting the phrase table from a parallel corpus and estimating the categorical probability distributions used in the feature functions. Then the *tuning* phase then optimizes the linear model parameters θ against a separate tuning parallel corpus, using some automatic translation accuracy metric such as BLEU (Papineni et al., 2002) as the loss function to be optimized.

Training

Various approaches to phrase table extraction and estimation of the phrase translation probabilities have been proposed. The most commonly used is a heuristic method introduced by Och and Ney (Och et al., 1999; Och and Ney, 2003; Koehn et al., 2003) which extracts a phrase table making use of word-based alignments between sentence pairs of the parallel training corpus:

1. Train word-based alignment models (usually IBM model 4) on the training corpus both in the source-to-target and in the target-to-source directions and compute the maximum likelihood word alignments of the corpus.

2. Combine the two one-to-many, source-to-target and target-to-source word alignments into a single many-to-many symmetrical word alignment: start with alignment points in their *intersection* and iteratively add points in their *union* according to a heuristic (see (Koehn et al., 2003)).

3. Extract all the phrase pairs, up to a maximum length, which are *consistent* with the symmetric word alignment: A phrase pair is consistent with the alignment if at least one pair of its words are aligned to each other and none of its words are aligned to words outside the phrase pair.

4. Estimate the direct and inverse phrase translation probabilities $P_{PHRASE}(\bar{e}_j|\bar{f}_j)$ and $P_{PHRASE}(\bar{f}_j|\bar{e}_j)$ at maximum likelihood from

the observed frequency counts. Smoothing (e.g. Good-Turing) can be performed to compensate for data sparsity, just as in n-gram language model estimation. Categorically distributed relative distortion probabilities $P_D(d)$ or absolute distortion probabilities can be also estimated from observed frequency counts if needed.

Training the word alignment models also produces estimates for the lexical translation probabilities $P_{LEX}(w_e|w_f)$ and $P_{LEX}(w_f|w_e)$ which are used as features for the translation model.
This method has been adopted in state of the art phrase-based translation systems such as Moses (Koehn et al., 2007).

Tuning

Once the phrase table extraction and the estimation of relevant probability distributions has been performed, it is generally possible to guess an initial parameter vector $\theta^{(0)}$ which achieves good results. However, in order to maximize translation quality, the model parameters need to be tuned by machine learning methods.
This is performed using a separate parallel corpus, known as tuning set. We want our system to translate each source sentence in the tuning set as close as possible to the reference translation, as measured by a suitable loss function.

The most commonly used loss function (actually, a similarity function) is BLEU (Papineni et al., 2002), which is based on *precision* at n-gram level, with n-gram order between one and four. BLEU allows to use multiple reference translations for each source sentence, to take into account the fact that there is rarely one "true" translation for any given sentence.
BLUE was developed primarily to evaluate the quality of translation systems and using it to define a loss function suitable for machine learning presents some technical hurdles, as the BLEU score is defined at corpus level and does not decompose additively or multiplicatively over sentence pairs. Sentence-level variants of BLUE can be used, or the optimization algorithms can be adapted to deal with the non-decomposability.
Other loss function have been proposed, from *translation edit rate* (TER)

(Snover et al., 2006), which measures the editing effort needed to change the system output to a reference translation, to highly sophisticated measures such as MEANT (Lo et al., 2012) which take into account and semantic similarity. BLEU constitutes a good compromise between correlation with human quality judgment, performance and lack of dependence on external tools (syntactic parsers, semantic role labelers, etc.) which may be not available for many languages.

Optimizing the parameter vector θ to minimize the loss against a tuning set is a linear structured prediction problem (section 2.1.1).
Typically, iterative optimization algorithms which repeatedly invoke the decoder to compute n-best translation lists of the source sentences are used.

If the dimension of the parameter vector is small, as in the basic model described above, *minimum error rate training* (MERT) (Och, 2003), a method based on Powell's optimization algorithm (Powell, 1964), is typically used.
If the translation model is extended with additional features, particularly sparse features, the dimension of the parameter vector can easily range in the order of $10^5 - 10^6$. Since MERT running time tends to be approximately proportional to this dimension, it becomes unpractical to use in these cases. Margin-based machine learning methods, such as MIRA, batch MIRA (Watanabe et al., 2007; Chiang et al., 2009; Hasler et al., 2011) or structured SVM (Cherry and Foster, 2012) can be better suited for large dimensional tuning.

Decoding

The problem of computing the best scoring translation of a given source sentence in the phrase-based model is NP-hard and APX-complete, just as the translation problem for the word-based model (which is a special case of phrase-based translation). For this reason, decoding is performed using heuristic algorithms that compute a local optimum of the translation space.

The most commonly used decoding method is an incremental *beam search* algorithm (Koehn et al., 2003; Koehn, 2004a), similar to limited memory variants of A* (Hart et al., 1968):

We can define the decoding process in terms of a non-deterministic automaton: Starting from an initial state consisting of an empty target sentence, at each step non-deterministically choose a source phrase consisting only of yet unused source words and append one of its possible translations to the current target string. Repeat until all source words have been used. Output the best scoring translation of all final states.
Obviously this automaton has a number of possible execution traces exponential in the length of the source sentence, hence a naive implementation would be unfeasible. We can reduce this complexity using *hypothesis recombination* and *pruning*.

Hypothesis recombination exploits the fact that the score of each candidate translation e of a source sentence f, decomposes additively over the transitions of the automaton: At each state v we can compute a partial score, starting from zero at the initial state and adding at each step j the partial score of the previous state to a transition score which depends only on the identity of the current phrase-pair for the translation log-probability features, the last source word position $end(j-1)$ of the previous phrase pair for the distortion log-probability feature, the last $k-1$ target words for the k-th order language model feature, and the length of the current target string $L_{\bar{e}_{1:j-1}}$ and the number of phrase pairs used so far $j-1$ for their own features. Moreover set of descendant paths of state v only depends on the set of source words used so far U_f.
Therefore, we can define the set of used source words and all the properties the transition scores to its children depend upon as the *signature* $\sigma(v)$ of state v. States with the same signature can be combined together, keeping the one state with maximum partial score and discarding the rest.
We can view the signatures of the states as vertices in a graph, where edges represent the transitions, each weighted by its transition score. Finding the best translation then amounts to computing the maximum-weight path in this search graph, which can be done in polynomial time in the size of the graph (since it is directed acyclic). N-best list decoding consists in computing the N-maximum-weight paths.

Hypothesis recombination significantly reduces the size of the search

space, which however remains exponential in the source sentence size due to the fact that it can still arbitrary reorder the phrase pairs. Incremental decoding with hypothesis recombination is considered a dynamic programming algorithm due to the fact that it exploits the *optimal substructure property* of the decoding problem w.r.t. the local contexts defined by state signatures. However, since there are exponentially many state signatures [21], this is not enough to obtain polynomial time complexity. The complexity of the algorithm is: $O(L_f^2 \cdot 2^{L_f} \cdot |W_e|^{k-1} \cdot E)$, where E is the maximum number of translations of each source phrase in the phrase table, and $|W_e|$ is the size of the target lexicon.

In order to make decoding feasible, we need to *prune* unpromising hypotheses: remove from the search graph nodes which appear unlikely to be on the path of the best translation. This process is heuristic and therefore potentially introduces search errors which can prevent the decoding algorithm from computing truly maximum scoring translation.
In beam search, at each step, after we generate the new states and combine them according to their signatures, we rank them by their estimated total score, which is defined as the sum of their partial score and an estimated future score, computed by a heuristic. This is similar to the way A* and its derivatives estimate vertex scores for expansion and possibly pruning, although here we only use it for pruning (expansion always proceeds in a breath-first fashion) and since we are not interested in optimality we do not require the future score heuristic to be technically admissible [22].

In the phrase-based translation model, we estimate the future score of a state signature $\sigma(v)$ as an approximation of the score obtained by translating all the spans of currently unused source words ignoring the order of the phrase pairs. This can be quickly precomputed in polynomial time for all the possible source word spans before the decoding process starts (see (Koehn et al., 2003) for details).
This future score estimate is not very accurate in an absolute sense, but in

[21] specifically, the set of sets of used source words has cardinality 2^{L_f}

[22] In A*, a future score heuristic is admissible if it never underestimates the score of the best path from the current vertex to a goal vertex.

practice it is good enough to compare vertices with the same number of used source words. Therefore, in order to prune, we separate by the number of used source words in the state signatures in different bins, or *stacks* [23]. Within each stack, we remove all the worst vertices that exceed a maximum beam size B (histogram pruning) or have a score lower than c times the score of the best vertex in the stack (threshold pruning).

With pruning, the algorithm complexity drops to $O(L_f^2 \cdot B)$, which is quadratic in the length of the source sentence. We can further reduce the size of the search space by using a maximum distortion distance D, obtaining even linear complexity: $O(L_f \cdot D \cdot B)$. We went from exponential to linear complexity at the cost of sacrificing optimality.
In practice, the beam search algorithm is often very fast and relatively accurate, with the hyperparameters B, c and D trading off accuracy for speed.

Incremental beam search decoding can be used with more complex models with additional features. However, it is important to note that all these features need to be local, in the sense that they decompose additively over transitions, and, for each state they depend only on the current phrase pair and the signature of the previous state. In particular, this means that there exist a constant k' such that they can only depend on a context of $k' - 1$ words on the left of the current phrase pair (in the base model $k' = k$ is just the order of the n-gram language model). Increasing k' exponentially increases the complexity of the unpruned algorithm (which scales as $|W_e|^{k'-1}$). When pruning is enabled, for a fixed beam size B, increasing k' exponentially increases the number of discarded search vertices, which may significantly increase the number of search errors.
Therefore, there is a fundamental tradeoff in the design of features for a phrase-based translation model: more complex features can make the model better at scoring candidate translations, but if they necessitate an increase of the context length k', or more generally an increase of the amount of state information that must be included in the signatures, they may make the decoder actually worse at finding a high scoring translation, possibly

[23] which, despite the name, are just sets of state signatures, without any LIFO semantics.

decreasing the overall translation quality.
For this reason, some advanced machine translation systems tend to keep the proper decoder simple and add complexity around it, as a pre-processing step to *restructure* (typically *reorder*) the source sentence, or as a post-processing step to *rerank* a list of n-best candidate translation or a representation of the explicit search graph (known as word lattice).

Besides incremental beam search, other decoding algorithm have been proposed. Local search algorithm, starting from (Marcu and Wong, 2002), start by quickly generating an initial complete translation, either randomly or using beam search with a small beam size (possibly equal to 1 in the case of *greedy search*), and then they iteratively modify it until they hit a local maximum where they can't improve it for a given number of iterations. In principle, local search methods may be more accurate since they can use features that do not decompose locally without paying an exponential performance penalty of beam search. In practice, however, they may not be as efficient to compete with plain beam search.

Some variants of local search methods may be instead more efficient than beam search, at the cost of imposing additional constraints on the model features. For instance, phrase-based decoding can be directly reduced to an instance of the asymmetric generalized traveling salesman problem (AGTSP) (Zaslavskiy et al., 2009), which is then reduced to the usual symmetric TSP which can be solved at a local optimum using the Lin-Kernighan heuristic (Lin and Kernighan, 1973), a high performance local search algorithm.
This outperforms beam search when using bigram language models ($k = k' = 2$), but runs int practical complexity issues on when increasing the language model order even to trigrams.

Extensions

Since the phrase-based translation model was proposed in the early 2000s, a large number of variants that extend it with additional features have been proposed.
Most of these extension focus on the distortion component of the model

score which is pretty simplistic in the baseline model described so far, and proper modeling of distortion is believed to be highly beneficial to translation quality (Birch et al., 2008).

The lexicalized (phrasal) distortion model (Koehn et al., 2005; Tillmann, 2004) condition the distortion log-probabilities to the identity and position of the phrase pairs used in the translation.
Given the phrase pair $(\bar{f}_j, \bar{e}_j)$, with $\bar{e}_j$ occurring as the j-th target phrase in e, we characterize the orientation of $(\bar{f}_j, \bar{e}_j)$ with respect to the previous phrase pair $(\bar{f}_{j-1}, \bar{e}_{j-1})$ as:

- *left-monotone* if $\bar{f}_j$ occurs just after $\bar{f}_{j-1}$ in f (that is, if $start(j) = end(j) + 1$).
- *left-swap* if $\bar{f}_j$ occurs just before $\bar{f}_{j-1}$ in f.
- *left-discontinuous* otherwise

Similarly, we characterize its orientation with respect to the next phrase pair $(\bar{f}_{j+1}, \bar{e}_{j+1})$ as *right-monotone*, *right-swap* or *right-discontinuous*.
During phrase table extraction, for each phrase pair we estimate, based on the word alignments, the probability of it occurring in any of these configurations. During decoding, these (log-)probabilities are used as model features.
The "left" features require no additional information to be included in the state signatures, thus they do not affect hypothesis recombination. The "right" features require to include in each signature the identity of its last phrase pair. In practice both left and right features are often used.
Even though the type of ordering relations that it can represent are pretty coarse, lexicalized distortion modeling can provide a significant improvement in translation quality. In fact, it is enabled in the default configuration of the state of the art translation system Moses (Koehn et al., 2007).

More complex distortion models use features leaned from *reference reorderings* of the training corpus: each source sentence of the training corpus is permuted, based on the word alignments, in a target-like word order

(Al-Onaizan and Papineni, 2006; Visweswariah et al., 2011). Then a source-side n-gram language model can be trained on these reordered sentences (Feng et al., 2010), or reordered word adjacency features can be learned using a discriminative method (logistic regression) (Bisazza and Federico, 2013), or a Markov-chain operation sequence model over an ideal generative process can be learned, again using language model estimation techniques (Durrani et al., 2011) applied to a heuristically pre-processed training corpus. Syntactical features, specifically features for paths between source words in a dependency tree obtained by parsing the source sentence, can be used as well to model distortion (Chang et al., 2009).

Other extensions focus on syntactical phrase coherence (whether a source phrase spans a proper subtree of a source parse tree), tried without success with constituency parse trees (Koehn et al., 2003) but successfully with dependency parse trees (Cherry, 2008).
Various types of lexical sparse features have been used as well (Och et al., 2004; Chiang et al., 2009; Hasler et al., 2011).

Finally, phrase-based translation systems can be extended using additional target-side language models.
MLP neural language models can be incorporated straightforwardly in the model since they use fixed-size context of words just like standard n-gram language models. RNN language models, on the other hand, have continuous state vectors which do not play well with hypothesis recombination, and therefore they are not normally used in the decoder. Syntactical language models often require some modifications in the decoder.
See section 2.4 for a more detailed discussion.

2.5.3 Hierarchical and syntax-based methods

While phrase-based machine translation systems yield state of the art performances on many language pairs, their lack of explicit modeling of language syntax makes them prone to certain kinds of systematic errors.
For instance, when translating from English to French, a negative clause "**don't** X_f", should be translated using the split negation "**ne** X_e **pas**", but

if "X_f" is long, it is unlikely that a corresponding phrase pair exists in the phrase table, therefore the decoder will translate the clause using a concatenation of phrase pairs, which will typically results in failing to generate the "**pas**" particle. Similarly, long-distance concordance of grammatical gender and number are difficult to get right if they are not isomorphic between the two languages.
In practice, as long as we are translating between Romance languages and/or English, the impact of these problems is limited, because the overall clause structure is generally Subject-Verb-Object on both sides (although Romance languages use the Subject-Object-Verb structure in some constructions). Other common languages have different clause structure, such as Germanic languages other than English (SOV-V2 [24]), Japanese, Persian (SOV), or Russian (free form). Translating between languages with different clause structure involves long-distance reordering, and phrase-based systems, with their limited phrase length and distortion distance, often perform poorly at it.

Starting from the works on stochastic inversion transduction grammars by Dekai Wu in the late 1990s (Wu, 1997), numerous approaches have been proposed for exploiting the syntactic recursive structure of language in a statistical machine translation system.
Syntactic machine translation models can be characterized across various dimensions: by the type of grammatical formalism they use (*phrase structure grammars* or *dependency grammars*), by whether they use grammars based on expert linguistic knowledge, machine-learned from annotated parallel corpora [25], or grammars automatically constructed from unannotated corpora (*hierarchical systems*), or by which sides they use syntactic modeling on: *tree-to-tree* systems use syntax on both the source and the target languages, *tree-to-string* use it only on the source language and *string-to-tree* use it only on the target side.
An exhaustive description of all these approaches is beyond the scope of

[24] Subject-Object-Verb with finite verb in second position.

[25] typically themselves annotated using automatic statistical parsers, themselves trained on human-annotated corpora.

this document. In the rest of this section we will briefly describe the most common approaches focusing on those related to this thesis work.

Phrase structure grammar approaches

Phrase structure grammars (section 2.3.1) model the syntax of a language as a set of formal rewrite rules which generate its sentences. A typical way to use this formalism for syntactic machine translation, specifically, for tree-to-tree translation, is to extend the formalism to *synchronous phrase structure grammars*, which model the joint generation of a pair of sentences, one in the source language and the other in the target language.

Just as context-free grammars (CFG) are the most common type of phrase structure grammars used for parsing, *synchronous context-free grammars* (SCFG) (Lewis and Stearns, 1968) are commonly used for translation.
An SCFG models the generation of a pair of sentence according to the following ideal process:

- At each step t, the state of the process $(f^{(t)}, e^{(t)}, a^{(t)})$ consists of a pair of strings $f^{(t)}$ and $e^{(t)}$, made of terminals (words) and non-terminals, and a bijective (one-to-one) alignment relation $a^{(t)}$ between the non-terminals in $f^{(t)}$ and $e^{(t)}$, such that only identical non-terminals can be aligned to each other (this implies that the non-terminals in $f^{(t)}$ and $e^{(t)}$ are permutations of each others).

- Each rule is in the form $lhs ::= (rhs_f, rhs_e, rhs_a)$, where lhs is a single non-terminal, rhs_f, rhs_e is a pair of strings made of terminals and non-terminals, and rhs_a is the bijective alignment relation between their non-terminals, with the constraint that only identical non-terminals can be aligned to each other.

- Starting from the state consisting only of start non-terminals (S_1, S_1) (where the indices denote the alignment relation), choose at each step t a rule such that its left-hand side appears in the current state as a pair of aligned non-terminals and rewrite that pair with the right-hand side of the rule.

- Iterate until the string pair in the current state consists only of terminals.

Some variations of this formalism exist in the literature. For instance, it is possible to remove the constraint of alignment only between equal non-terminals, in which case the left-hand side of the rules consists of a pair of non-terminals. See (Satta, 2010) for a detailed description of the specific formalism and associated algorithms.

Rules in the grammar can be also weighted by probabilities $P_R((rhs_f, rhs_e, rhs_a)|lhs)$, obtaining a *probabilistic synchronous context-free grammars* (PSCFG).
More generally, multiple probabilities can be used, such as direct and inverse translation probabilities $P_{RD}((rhs_e, rhs_a)|rhs_f, lhs)$ and $P_{RI}((rhs_f, rhs_a)|rhs_e, lhs)$, or lexical translation probabilities (obtained when the rules are extracted from word-aligned parallel corpora). The logarithms of these probabilities can be used as features to be linearly combined according to a parameter vector θ to obtain a rule score.

The hierarchical translation system developed by David Chiang (Chiang, 2005, 2007) is the first practical PSCFG translation system, which learns a synchronous grammar from a parallel corpus without using linguistically motivated annotations.

The rule extraction procedure is an extension of the phrase table extraction procedure of phrase-based systems: given a word-aligned parallel corpus, it extracts a large number of simple PSCFG rules, with an alphabet of only two non-terminal symbols (the start symbol S and another generic non-terminal X) and at most two non-terminal instances per rule.

Although these automatically learned grammars are quite simple compared to those used by linguists, they can model various linguistic phenomena useful for translation, such as long-distance reordering, split translations and concordances.
For instance, when translating from English to French, the system can easily learn a rule for translating negations $X ::= (\textbf{do not}\ X_1, \textbf{ne}\ X_1 \textbf{pas})$.

Decoding. For each pair of source and target sentences (f, e) that the grammar generates, it also produces a joint derivation y, consisting of a pair of constituency trees whose internal (non-terminal) vertices are equal up to a permutation of their children. Since a typical grammar is ambiguous, there can be multiple derivations for each sentence pair.
Given a source sentence f, the translation problem consists in finding the most probable (highest scoring) compatible target sentence e according to the grammar:

$$\begin{aligned} \overset{*}{e} &= \underset{e}{\operatorname{argmax}}\, P(e|f) \\ &= \underset{e}{\operatorname{argmax}} \sum_{y \in GEN(e,f)} P(y, e|f) \end{aligned} \tag{2.51}$$

Since summing over all the possible derivations is unfeasible, we approximate this computation by considering only the best possible derivation, as we did in the word-based and phrase-based translation schemes:

$$\begin{aligned} \overset{*}{e} &\approx \underset{y \in GEN(e,f), e}{\operatorname{argmax}} \; P(y, e|f) \\ &= \underset{y \in GEN(e,f), e}{\operatorname{argmax}} \; P(e|y, f) \cdot P(y|f) \end{aligned} \tag{2.52}$$

But each derivation y deterministically (and trivially) defines a single target sentence e (that is, $P(e|y, f)$ is a degenerate probability distribution and $GEN(e, f) = GEN(f)$), therefore, the translation problem can be reduced to the problem of finding the best derivation of f:

$$\begin{aligned} \overset{*}{e} &\approx E(\overset{*}{y}) \\ \overset{*}{y} &= \underset{y \in GEN(f)}{\operatorname{argmax}}\, P(y|f) \end{aligned} \tag{2.53}$$

where the function $E(y)$ simply extracts the sentence e from the derivation y by returning the leaves of the target constituency tree.
Thus, we have reduced the translation problem to the constituency parsing problem of the source sentence.

In fact, a PCFG chart parser, such as the CYK algorithm, can be adapted to PSCFG decoding. Only the source right-hand side rhs_f of the rules is used during parsing, however, rules with the same source right-hand side but

different target right-hand side rhs_f or alignment relation rhs_a should still be distinguished, since they generate different derivations and hence possibly different translations.

Although the problem is formally straightforward, with a polynomial time complexity ($O(L_f^3 \cdot |G|)$), the size of a typical synchronous grammar is often much larger than the size of a monolingual grammar, and although the asymptotic complexity in the grammar size $|G|$ is only linear, exact parsing is not feasible in practical applications.
Therefore pruning is typically used. Specifically, for each cell in the parsing chart (the dynamic programming tableau), we store a limited number B of entries, and we perform histogram and threshold pruning, optionally augmenting the score with a heuristic "outside score" estimate in order to reduce the pruning errors.

In order to be competitive with phrase-based systems, PCFG systems need to include a language model feature in their models. The simplest way to do so is to exploit the fact that a n-gram language model is a weighted finite state automaton, and therefore it can be *intersected* with the PCFG yielding a larger PCFG with an additional weight that can be included in the log-linear score formula. Various systems, including the original hierarchical translation system (Chiang, 2005) use this approach, however, the grammars generated by this intersection operation tend to be huge, with a size exponential in the order k of the language model, limiting the usefulness of this approach.
Therefore, it is preferable to explicitly include the language model during decoding, by extending the decoding (usually CYK) algorithm, effectively performing this intersection online. *Cube pruning* (Chiang, 2007) is a heuristic variation of the CYK decoder specifically developed to efficiently incorporate a language model while limiting decoding errors. These techniques, included in the Joshua open-source translation system (Post et al., 2013), and in the late versions of Moses, are competitive with state of the art phrase-based methods.

While hierarchical systems do not use linguistically motivated grammars, various approaches have been developed to extract linguistically motivated

synchronous grammars from annotated (parsed) parallel corpora.
Rules can be extracted using a generative EM-trained model (Yamada and Knight, 2001), or heuristically from word-aligned corpora, for tree-to-tree (Shieber and Schabes, 1990; Zhang et al., 2008) and tree-to-string systems (Galley et al., 2004, 2006) [26], (Liu et al., 2006).
Tree-to-string models, in particular, in addition of being able to be decoded by a variation of the CYK decoder described above, also admit a specialized fast decoding algorithm based on a depth-first traversal of the source parse tree (Huang and Mi, 2010). This algorithm builds the target sentence left-to-right, hence it only uses a $k-1$ right-most context of target words for the language model, obtaining a decoding speed even greater than phrase-based decoding (Moses), with equal or higher translation quality on some language pairs.

Dependency grammars

The other major formalism for syntactic modeling, dependency grammars (section 2.3.2), has also receive substantial attention from a statistical machine translation perspective, due to the fact that dependency relations are often approximately preserved between different languages (Fox, 2002; Hwa et al., 2005).

Hiyan Alshawi proposed a first dependency statistical translation model in 2000, based on weighted *finite-state head transducers*, an extension of standard *finite-state transducers*, which are used to transform source dependency trees into target dependency trees.

Many other approaches use some extensions of the synchronous phrase structure grammars and the associated parsing/decoding machinery adapted to represent dependency rather than constituency relations. [27]

[26] this model actually use a grammar formalism more general than PSCFG, which can model complex relations between source trees and target sentences.

[27] This is relatively straightforward as long as the dependency trees are guaranteed to be projective. Non-projective dependency trees are more interesting from a linguistic point of view but are not easy to represent with extensions of phrase structure grammars, and thus they are not widely used in machine translation.

Chris Quirk et al. (Quirk et al., 2005) proposed an approach that operates at the level of *treelets*: connected fragments of a dependency tree.
This approach starts with a word-aligned parallel training corpus, with dependency and POS annotations on the source side (typically generated by a statistical parser). These dependency relations are *projected* across the word alignments, using a heuristic, generating a target parse tree, largely isomorphic to the source tree. Then, a treelet translation table is extracted: any source-side connected subgraph, up to a maximum number of vertices, whose top level consists in a single subroot or a sequence of siblings vertices is considered a valid source treelet, and if its node are aligned to a similarly structured target-side treelet, they are considered a valid treelet pair which is extracted.
During decoding the source sentence is parsed and matched against the treelets in the translation table are identified: a treelet matches a subgraph of the source tree if their topology and words are identical, and the match is considered *rooted* at the highest vertex of the subgraph, and associated with that vertex its target-side treelet as a candidate translation.
A bottom-up search, similar to CYK decoding, considers all the ways of partitioning the source tree into non-overlapping treelets that match against the translation table, and all the permutations of the corresponding candidate translations at each vertex, which are scored according to a distortion model based on source-side and target-side features, in addition to phrase-translation log-probabilities and a target-side language model log-probability. Independence assumptions in the scoring model can be exploited to apply dynamic programming.
This approach provides higher translation quality with respect to phrase-based approaches, at an higher computational cost (about 16 times higher).

If accurate dependency parsers are available for both the source and target languages, then linguistically motivated tree-to-tree approaches can be used.
Yuan Ding and Martha Palmer (Ding and Palmer, 2005) proposed *synchronous dependency insertion grammars* (SDIG), a type of tree transduction grammars which model the joint generation of a pair of trees in terms of

tree fragment rewrite operations (as opposed as string rewriting operations of conventional phrase structure grammars).
During training, tree fragment rewrite rules are extracted from a word-aligned parallel corpus parsed on both sides. Probabilities are estimated from empirical counts as in the hierarchical model.
During decoding, the source sentence is first parsed with a monolingual dependency parser and then the tree itself is parsed/decoded according to the synchronous grammar, using a variation of the CYK algorithm, then the target tree, and thus the target sentence, is extracted from the best derivation.

Other approaches are string-to-tree, using syntax only on the target side, mainly for the purpose of scoring candidate translations with a syntactic language model.
Libin Shen et al. (Shen et al., 2008, 2010) extend the hierarchical translation model to include in its rules target-side (projective) dependency relations, extracted from the training corpus.
Specifically, each rule (other than the glue rules) is in the form $X ::= (rhs_f, rhs_e, rhs_a, rhs_d)$, where rhs_f, rhs_e and rhs_a are defined as in a standard PSCFG and rhs_d is a dependency relation over the symbols in rhs_e, with the specific restriction that it must be a *well-formed dependency relation*.
A dependency relation between words in a contiguous span of a sentence is defined as well-formed if it is composed of a concatenation of complete subtrees with a common parent, which may be itself included in the relation (known as *fixed relation*) or not (*floating relation*). In other words, a well-formed dependency relation is a subtree up to the optional removal of some top branches at the sides and its root node.
During training, rule extraction is limited to rules whose dependency relations are well-formed w.r.t. the sentences they were extracted from.
During decoding, a variant of the CYK algorithm for hierarchical decoding is used: the source sentence is parsed using the source side of the rules in the grammar, with the additional restriction that on the target side the dependency relations must be combined in ways that preserve their well-formedness. This enables the system to efficiently score each subtree in the derivation (an thus each entry of the decoding chart) with a syntac-

tic language model specifically designed to decompose over well-formed dependency relations (section 2.4.3). An n-gram language model can be also incorporated using standard hierarchical decoding techniques such as cube pruning, therefore combining the strength of both methods, yielding a significant increase of translation quality.

Synchronous grammars are relatively limited in the type of bilingual relation they can efficiently represent. In order to relax some of these restrictions, David Smith and Jason Eisner proposed a framework usable both for string-to-tree and tree-to-tree dependency translation known as *quasi-synchronous grammars* (QSG) (Smith and Eisner, 2006), usable both for string-to-tree and tree-to-tree translation.
In this model, for each source sentence f, a specialized target-side monolingual grammar G_f is constructed, which generates only the strings that are candidate translations of f. This grammar is typically weighted (it is a PCFG or a variation thereof) in order that its generation probability distribution $P(e|G_f)$ approximates the translation probability distributions $P(e|f)$. This is obtained by decorating the rules with source-side information so that scores can be computed by a log-linear model using feature functions which take into account a context both on the source sentence (and its parse tree if available) and the target derivation.
Although the model is quite general and can be also used for phrase structure syntactic translation, the authors propose to use it for dependency syntactic translation. In their original work they run several experiment in using this model to improve the quality of word alignment, but they did not use it in a full translation system. However, this model is relevant since it inspired other approaches, such as the phrase dependency rescoring system of Kevin Gimpel and Noah Smith (Gimpel and Smith, 2013).

For further discussion of dependency statistical machine translation approaches, refer to (Ambati, 2008).

2.5.4 Neural network-based methods

In recent years a class of machine translation frameworks based on *recurrent neural networks* have been proposed (Sutskever et al., 2014; Cho et al., 2014b; Bahdanau et al., 2014).

These systems are similar to Mikolov et al. *recurrent neural language model* (Mikolov et al., 2010) (section 2.4.2), but they condition the target word probabilities on the source sentence: Instead of estimating the prior probability $P(e)$ of a target sentence e, they estimate the directly translation probability $P(e|f)$ of source sentence f to target sentence e.

Training is performed as in monolingual neural language model, except that a parallel corpus is used, minimizing the average per-word cross-entropy of the target training sentences conditional on the source training sentences.
Notably, these models don't require the computation of word alignments or other features.

Translation consists in maximum likelihood decoding, which is performed in an approximate fashion using beam search without hypothesis recombination (as recurrent neural network models aren't factorizable in a way exploitable by dynamic programming techniques).

The authors report performances comparable to state of the art phrase-based systems, at least for language pairs with a small amount of long-distance reordering such as English-French.

2.5.5 Pre-reordering

A competing approach to full syntax-based translation is to translate source sentences with a phrase-based system after a heuristic preprocessing step that restructures these sentences in ways designed to overcome the main difficulties of standard phrase-based models.
As we remarked before, the arguably main difficulty of phrase-based machine translation is long-distance reordering. In fact, the estimated amount of word order distortion between a language pair is the strongest known

predictive factor of the phrase-based translation quality for that language pair, for a given system and training corpus size (Birch et al., 2008). Therefore, we may want to pre-reorder the source sentences in a "target-like" word order, and then translate then with a phrase-based system itself trained on a corpus with reordered source sentences.

Pre-reordering does not address all the issues of phrase-based models, for instance, it does not help with split translations (e.g. "**don't** X_f" $\rightarrow$ "**ne** X_e **pas**"[28]) or some kinds of grammatical concordance, but in practice it performs competitively with both hierarchical and linguistically motivated syntax-based translation in terms of translation quality, and often better in terms of speed.

The simplest way of pre-reordering a source sentence is to syntactically parse it and then apply some hand-coded reordering rules which may permute the children of each vertex in the tree based on some local context. Since the linguistic "word-to-head" relation is mostly important for this kind of transformation, usually dependency parses or lexicalized constituency parses are used.

This approach has been successfully applied to various language pairs such as German-to-English (Collins et al., 2005a), Chinese-to-English (Wang et al., 2007), and even between English and fully SOV languages like Japanese, Korean, Hindi, Urdu and Turkish (Katz-Brown, 2008; Xu et al., 2009; Isozaki et al., 2010).

The main issue with this approach is that it requires a high quality parser for the source language, which may not be available for all languages, and substantial linguistic expertise to produce the reordering rules.

Therefore, it is interesting to consider approaches that preserve the "spirit" of statistical machine translation by automatically learning how to reorder source sentences from the training corpus.

[28] Although we can address some of these cases with a little bit of hand-coded pre-processing, e.g. by configuring the tokenizer to split at the apostrophe, thus splitting "**don't**" into "**don'**" and "**t**", "**doesn't**" into "**doesn'**" and "**t**", etc.

Syntax-free statistical reordering

Statistical reordering systems usually attempt to learn a reordering model from a *reference reordering* of the source side of the parallel training corpus, heuristically computed from the word alignments (just as the advanced in-decoder distortion models described in section 2.5.2).

Specifically, given a pair of training sentences (f, e), and a symmetric word alignment between them a, we assign to each source word f_j an index corresponding to its left-most aligned target word:

$$\text{idx}(j) \equiv \min_{(j,j') \in a} j' \tag{2.54}$$

then, we perform a stable sort of the source words according to this index to obtain the reference permutation f'.

In a given sentence, some source words may be not aligned to any target word. The heuristics proposed in the literature differ by how they deal with these unaligned words: they may attach them to the previous aligned source words (Al-Onaizan and Papineni, 2006), move them to the beginning of the sentence (Tromble and Eisner, 2009), or just drop them (Visweswariah et al., 2011).

We use the pairs (f, f') of source sentences and their reference permutations as a training set for the pre-reordering system, learning a model $h_{PERM}(\cdot, \theta)$. Then we apply this model to reorder all the source side of the training and tuning corpora, obtaining sentence pairs in the form (f'', e), where $f'' \approx f'$. We use these source-reordered parallel corpora to train (and tune) a conventional phrase-based decoder.

During decoding, we just apply the pre-reorderer and the decoder in sequence.

In principle, we may turn off the reordering capability of the downstream decoder, simplifying the search process, although in practice it is beneficial to leave it on, thus enabling the downstream decoder to have a chance at correcting some short-distance reordering errors of the upstream reordered.

Various types of reordering models, with corresponding training and prediction algorithms have proposed. In principle, all the advanced in-

decoder distortion models can be also used for pre-reordering, effectively employing a phrase-based model to "translate" between the source language and its reordered version (Costa-jussà and Fonollosa, 2006). Thus we use an upstream phrase-based system to translate f to f'' followed by a different downstream phrase-based system to translate f'' to e.
One may wonder what is the benefit of cascading two systems of the same kind. The main benefit is that since f and f'' must be permutations of each other, we can constrain the phrase table extraction of the upstream model to very short phrase pairs (even single-word pairs) and use specific heuristics which result in a much smaller phrase table. Thus, the upstream decoder can focus its search resources (beam space, essentially) on reordering alone, without having to juggle lots of translation options for each short span of words. It can therefore use a large, or even unlimited, maximum distortion distance.
The downstream decoder, on the other hand, can be optimized to handle a large number of long phrase pairs and perform little or no reordering. This separation of concerns can result in an increased overall translation quality.

In principle, we could apply a different kind of upstream translation model, such as the *hierarchical model* which is particularly well suited for long-distance reordering. However, due to the complexity of chart decoding, the computational cost of reordering may become the dominant w.r.t. the cost of downstream decoding, making this approach not competitive with a straight application of hierarchical translation.
Michel Galley and Christopher Manning proposed a variation of this approach, with hierarchical rules weighted by a score similar to the lexicalized distortion score, and a simpler shift-reduce parser instead of the CYK algorithm (Galley and Manning, 2008).

Other approaches use different types of reordering models based on classical combinatorial permutation problems such as the *traveling salesman problem* and the *linear order problem* (LOP).

Given a set of L distinguished elements $f_1, \ldots, f_L$, and a pair-wise preference scores matrix $B \in \mathcal{R}^{L \times L}$, the linear order problem consists in finding the permutation $\overset{*}{\pi} \in \Pi_L$ of the element indices such that the sum

of the pair-wise scores is maximized:

$$\overset{*}{\pi} \equiv \underset{\pi \in \Pi_L}{\operatorname{argmax}} \sum_{i=1}^{L} \sum_{j=1}^{L} B_{\pi(i),\pi(j)} \tag{2.55}$$

this is a NP-hard, APX-complete optimization problem related to TSP.
Roy Tromble and Jason Eisner use this problem as the basis of their pre-reordering system (Tromble and Eisner, 2009).
They define the scores between each pair of (POS tagged) source words as a linear combination of features of the two words and their local contexts:

$$B_{i,j}(\theta) \equiv \theta^t \cdot g(f, i, j) \tag{2.56}$$

where $\theta \in \mathcal{R}^n$ is the parameter vector and g is the local feature function. The linear ordering problem then becomes a maximization problem of a total score which is just the sum of these local pairwise scores:

$$\begin{aligned} \overset{*}{\pi} &= \underset{\pi \in \Pi_L}{\operatorname{argmax}}\, h_{REG}(\pi, f, \theta) \\ h_{REG}(\pi, f, \theta) &= \theta^T \cdot \sum_{i=1}^{L} \sum_{j=1}^{L} g(f, i, j) \end{aligned} \tag{2.57}$$

The authors propose to solve this optimization problem with a local search strategy, where each step of the local search involves the solution of a simpler, polynomial time, maximization problem using a dynamic programming algorithm[29].

Training of the parameter vector θ is preformed using the *averaged structured perceptron* algorithm, with a loss function that compares for each sentence f the reference permutation f' to the permutation produced by the model f'', with early stopping based on reordering BLEU score (f'' to f') on a validation set. The reference permutations are obtained by applying the heuristic described above to the automatically aligned (IBM Model 4 + Moses symmetrization heuristic) parallel training corpus.

A similar method, based on the traveling salesman problem, has been proposed by Karthik Visweswariah et al. (Visweswariah et al., 2011).

[29] which turns out to be an extension of monolingual CYK parsing over specially defined CFGs.

This model still uses pairwise scores between words computed by a linear model, but sums into the total score only the scores of word pairs which appear adjacent in the permutation π:

$$\begin{aligned} \overset{*}{\pi} &= \underset{\pi \in \Pi_L}{\operatorname{argmax}}\, h_{REG}(\pi, f, \theta) \\ h_{REG}(\pi, f, \theta) &= \theta^T \cdot \sum_{j=1}^{L} g(f, j, j-1) \end{aligned} \tag{2.58}$$

this is an asymmetric TSP which can be reduced to a symmetric TSP and solved at a local optimum using the Lin-Kernighan algorithm (Lin and Kernighan, 1973). Compared to the TSP phrase-based decoder (section 2.5.2) this is more efficient since the graph has only so many vertices as the number of words in the sentence, while the TSP phrase-based decoder generates large graphs whose size depends on the size of the phrase table and the order of the language model.

This reordering system is trained in a similar way to the LOP system. The main differences are that the authors used MIRA instead of the averaged perceptron and the reference permutations are generated from a small training set with manually annotated word alignments.

Syntax-based statistical reordering

The statistical reordering methods described in the previous section either do not use any linguistically motivated resource or use only a source-side POS tagger. This is beneficial for source languages where accurate parsers are not available. However, when accurate parsers are available, it may be beneficial to use methods which operate on parse trees of the source sentences, exploiting the linguistic information that they provide.

Chris Dyer and Philip Resnik proposed a reordering model based on a constituency parsing on the source sentence (Dyer and Resnik, 2010).
Given a source sentence f and its constituency parse tree y_f, we compute a specialized context-free grammar G_f which can generate only f and a large number its permutations [30].

[30]Note that this is a source-side sentence-specific grammar, which is different from the

Specifically, we relabel the non-terminals in y_f to make them unique, and for each of them we produce a set of CFG rules which can generate its children in the same order in which they appear in y_f and in many other orders (depending on a heuristic) as well.
Translation is not performed by cascading a reorder to a standard phrase-based decoder, but rather by representing the phrase-based model as a finite state transducer, intersecting it with the sentence-specific grammar G_f, and computing the best output using dynamic programming.
Training does not use word alignments, rather a custom discriminative structured learning algorithm for a generalized linear score model.

Dmitriy Genzel (Genzel, 2010) developed a pre-reordering system based on source dependency parsing. This model learns from a word-aligned parallel corpus tree modification rules similar to those used in the hand-coded reordering system.
Each rule has an *lhs* consisting of a vector of feature values relative to a vertex with an optional context of some of its children, and an *rhs* specifying a permutation of the children of the vertex. The rules are organized in a priority-ordered list $r_1, \ldots, r_k$.
During reordering, the rules are applied to the tree in their priority order: for each rule r_j the parse tree is visited top-down until a match between the *lhs* of the rule and a node is found or all the leaves are reached. If a match is found, the children of the matching node are permuted as specified by the *rhs* and the process skips to the next rule.

Training is performed by iterating over the sentence pairs of the parallel training set, and for each (f, e) of them consider a random sample of all the rules which can be obtained by permuting (with some heuristic limits) the children in all the vertices of the source parse tree y_f. These candidate rules are then collected for all the sentences and scored against a loss function based on the word-alignments (the number of crossed alignment links that the rule removes, possibly coupled with a translation BLEU score gain estimated with a word-based translation system operating on a small

target-side sentence-specific quasi-synchronous grammars of section 2.5.3

validation corpus). The best or the few best rules are included in the current rule list and the process is repeated for several iterations.

In addition of obtaining a significant gain of translation quality, this approach has also the advantage that the learned rules are easy to interpret by humans. In fact, they can be also manually edited or supplemented with linguistically motivated rules, enabling to build a reordering system which seamlessly combines expert knowledge and machine learned information.

2.5.6 Reranking

All the decoders described so far, both for phrase-based models and hierarchical/syntax-based models, rely on dynamic programming techniques to keep the size of their search space, and therefore their time complexity, manageable.

As they generate translations incrementally, left-to-right, bottom-up or top-down, they score partial translation hypotheses according to a model which must decompose over the decisions made by the decoder in a way such that the partial score at each decision point only depends on the current decision and a *signature* of the previous decisions for that hypotheses, thus enabling the decoder to recombine hypotheses with the same signature, dramatically reducing the search space. Since the decoding complexity is exponential in the size of these signatures, it is paramount that signatures are small.

This greatly limits the type of features that can be used in the model: in practice any feature can only depend on a fixed size k' context of previously generated target words, which is usually dominated by the order k of the n-gram language model. Therefore many types of accurate language models (high-order n-gram, high-order MLPLMs, RNNLMs and most discriminative syntactical LMs) and other interesting bilingual features (e.g. complex grammatical concordance features) are effectively ruled out.

In principle, local search decoders wouldn't have this issue, however, local search decoders aren't competitive with dynamic programming decoders and therefore they are not used in practice.

An effective workaround is to perform a *reranking* of candidate translations: the source sentence f is first decoded by a standard decoder using a conventional translation model, but instead of computing just the best candidate translation $\overset{*}{e}$, a set of promising candidate translations E_f is returned. Then a separate reranker scores these candidate translations according to a more sophisticated model and returns the new best translation.

Reranking models can be characterized by the representation of the set of candidate translation they use: *n-best lists* or *lattices*.

N-best reranking

An N-best list is just a list of the N highest scoring candidate translations accessible to the decoder. In general it is straightforward to adapt the dynamic programming decoders to generate N-best lists instead just the 1-best translation. Typically, the hyperparameter N ranges between 10^2 and 10^3.

In principle, translations in the N-best list can be just plain sentences, which would completely decouple the decoder from the reranker. However, it is common to also record the derivation of each translation and their total feature vectors, in order to include them in the reranking model.
Reranking just consists of evaluating the score of each translation in the N-best list according to the reranking model, typically a linear model over the original translation features and additional features, and returning the highest scoring translation.
Training of the parameter vector θ is usually performed using MERT or MIRA on a parallel corpus. For linear models, this is a discriminative learning problem effectively equivalent to parameter tuning for a standard translation model, and in fact the very same tools are typically used (although some specific margin-based learning algorithms have been proposed as well (Shen et al., 2004)).

Different N-best reranking approach differ by the type of feature they use. In sections 2.4.2 and 2.4.3 we have already discussed various types of neural and syntactic language models designed for reranking. They can be quite

beneficial in increasing translation quality, specifically in terms of *fluency* which is often relatively poor for standard translation systems.

Other approaches use features depending both on source and target words, exploiting the word or phrase alignment information produced by the upstream decoder for each candidate translation. For instance:
Och et al. (Och et al., 2004) describe various types of sparse lexical features and distortion features.
Philipp Kohen and Kevin Knight (Koehn and Knight, 2003) use syntactical features computed on phrase alignments between pairs of phrases which are guaranteed (by constraining the training of the upstream decoder) to be syntactical constituents.
Vassilina Nikoulina and Marc Dymetman (Nikoulina and Dymetman, 2008) dependency parse the source sentence f and the target sentences in the N-best list $e_1, \ldots, e_N$ and derive from the alignments some *dependency coupling features* which are used in the reranking model.

Lattice reranking

Reranking on N-best list is straightforward, but the quality improvements it can produce are limited by the fact that N-best lists of manageable size often contain little variation: most candidate translations often differs only by a few words. The combined decoding and reranking complexity is approximately linear in N, and in general increasing N above 10^3 is impractical.

A different approach is to represent the set of promising candidate translation E_f as a *translation lattice*: a representation of the dynamic programming tableau of the upstream decoder, which takes the form of a directed acyclic graph whose vertices represent the decision points with their state signatures and edges represent the decisions. Each directed path on this graph from a distinguished start vertex to the final vertices represent a candidate translation in E_f.
The translation lattice is a compact representation of a large number of promising candidate translations, in fact exponentially many in L_f, but still much less all possible translations $GEN(f)$ since most of them are pruned by

the upstream decoder.

Lattice reranking consists in finding the path with the highest score according to a reranking model. Compared to N-best reranking, this is a non-trivial structured prediction problem, requiring its own decoding algorithm, which is generally tightly coupled both to the reranking model and the representation of the lattice which depends on the internal structure of the upstream decoder.

Michael Auli et al. (Auli et al., 2013) propose a phrase-based lattice reranking approach using a recurrent neural network LM with additional source-side features:
In this approach, they augment the state signature of each vertex with a state vector for the pre-trained RNN and a partial score.
The start vertex has the default initial state vector and zero partial score. For each other vertex u_j we compute the partial RNNLM score for each of its parents $u_{j'} \in PARENTS(u_j)$, using the state vector of the parent $v(u'_j)$ and the string of target words specified by their shared edge. The largest of these scores becomes the partial score of u_j and the state vector of the RNN after computing it becomes the state vector of u_j. Once the final vertices have been scored, we compute the best path backwards by selecting the highest scoring final vertex, then we select its highest scoring parent, and so on upon reaching the start vertex. [31].
The RNNLM is trained separately from the reranker using source sentences and their reference translations.

Kevin Gimpel and Noah Smith (Gimpel and Smith, 2013) perform syntax-based lattice reordering using quasi-synchronous dependency grammars, which are an extension of the quasi-synchronous dependency grammars described in section 2.5.3: instead of modeling dependency relations between the words of target sentences which can be generated by a specific source sentence f, they model dependencies between *phrases*, defined as in the phrase-based translation model.
Reranking is performed by applying various pruning heuristics on the lattice

[31]This can be generalized by storing more than one state vectors at each vertex, but apparently this does not produce a substantial increase of translation quality.

E_f and then using it to generate the target-side context-free grammar G_f, where the terminal symbols are whole target phrases instead of single words. Rules of this grammar are weighted by a number of linguistically motivated features combined in a linear model. A dynamic programming algorithm is then used to compute the best sentence $\overset{*}{e}$ that G_f can generate, which is returned as the best translation of f.
The parameters for the linear models of the rules are estimated using MERT.

2.6 Summary

In this chapter we broadly described the theoretical underpinnings and a number of practical techniques of the field of statistical machine translation.

We introduced basic natural language processing concepts and techniques such as segmentation, tagging and parsing.

Then we described the general machine learning theory behind all statistical machine translation (and in general, statistical NLP) techniques: generative and discriminative learning models, regression, classification and structured prediction tasks, then we introduced various model representations and associated learning algorithms, starting from the simple linear models, to non-linear kernel models, to neural networks of different types.

We went further to describe statistical language models and finally we introduced statistical machine translation systems, starting from the simplest, but still relevant word-based models, to the modern phrase-based models, to the more complex and advanced models involving hierarchical and syntactical translation, pre-reordering and reranking.

The research field we explored are very vast and active. New techniques are constantly introduced pushing the envelope of performance even further. Therefore, this presentation is not intended to be exhaustive, neither in coverage nor in detail. We hope, however, to have captured the core aspects of the state of the art in the field and in particular to have provided the relevant background to understand both the technique upon the original contributions of this thesis work build upon and the context of research issues in which they have been developed.

Chapter 3

Characterization of German-to-English reordering as transitions on a dependency tree

In this chapter we study the word reordering phenomena involved with translation from German to English, from the perspective of transitions on a dependency parse of German sentences.
We imagine that a German sentence is reordered in an "English-like" word order by performing a "walk" on its dependency tree and incrementally emitting the words in the desired order. We quantitatively characterize these moves in relation to their impact on the translation quality of a phrase-based translation system, evaluating the impact of non-projectivity in the dependency tree and tree non-locality of the reordering movements.

We also assess the impact of non-projectivity and tree non-locality on the translation quality of a phrase-based system with hand-coded syntax-based pre-reordering rules.
In order to do so, we evaluate various hand-coded syntax-based reordering heuristics which differ by the type of parse tree they use and the tree locality of reordering operations that they can perform.

3.1 Motivation

Phrase-based statistical machine translation systems perform well when the typical word ordering of the source language and the target language are similar.
Short-distance reordering, such as the noun-adjective swap that occurs between Romance languages and English, can be resolved by the combination of the ability of the phrase table to store idiomatic phrase pairs (e.g. (**"Stati Uniti d'America"**,**"United States of America"**)), the language model (e.g. "**States United**" scores lower than "**United States**" in English) and the distortion models (chapter 2, section 2.5.2).
However, due to the context-size limits needed to make decoding efficient, phrase-based systems can't deal effectively with long-range reordering, which significantly limits their performance on language pairs with significantly different idiomatic word order (Birch et al., 2008), such as German and English.

As previously discussed in chapter 2, section 2.5.5, translation performance can be significantly increased, at least for some language pairs, by preprocessing the source sentences, permuting their words in a target-like order, and then feeding them to a phrase-based (or even hierarchical) translation system trained on a training corpus with a similarly permuted source side.

In fact, syntactic parse trees, particularly dependency parse trees, typically capture most linguistic information needed to perform proper reordering, as evidenced by reordering approaches that exploit hand-coded rules (Collins et al., 2005a; Wang et al., 2007; Katz-Brown, 2008; Xu et al., 2009; Isozaki et al., 2010).
Systems that use hand-coded rules, however, require substantial source language-specific linguistic expertise and engineering effort. Fully statistical systems would be preferable.

A few statistical syntax-based reordering systems have been proposed in the literature, e.g. Genzel (Genzel, 2010) and Dyer and Resnik (Dyer and Resnik, 2010).
These approaches learn reordering rules or reordering grammars from

source-parsed word-aligned corpora, and attain significant performance improvements over direct phrase-based translation, but they have some limitations which we attempted to overcome: they require constituency parse trees or projective dependency parse trees and limit the permutations they can generate to those that can be obtained by swaps between a parent vertex and its children or between sibling vertices in the parse tree.
As we discussed in the previous chapters, non-analytic languages with a relatively free word order have their syntax better represented by non-projective dependency parse trees, and the reordering operations needed to permute their words in a way suitable for translation to an analytic language with strict word order such as English can be more complex than tree-local swaps.

We hypothesized that these limitations in current syntax-based reordering systems imply that there may be some linguistic reordering phenomena relevant to machine translation that they may fail to exploit. Therefore, we intended to investigate the research question of whether source syntax non-projectivity and non-tree-local word swaps between idiomatic orders affect translation quality. Specifically, we analyzed the German-to-English language pair, as German displays a significant amount of non-projectivity when parsed with a suitable model and word alignment between German and English is known to typically include long-distance relations.

We evaluate our hypothesis against "pseudo-oracle" reference permutations of German into an English-like word order heuristically derived from IBM Model 4 word alignments and against hand-coded pre-reordering rules.

We didn't analyze the English-to-German language pair because English is intrinsically mostly projective, hence we hypothesized that using a non-projective source model wouldn't make much difference and because German is generally understood to have a freer word order than English, which makes the concepts of reference permutations and deterministic pre-reordering less justifiable. In fact, English-to-German pre-reordering seems to be a harder problem than German-to-English pre-reordering (Navratil et al., 2012).

3.2 German-to-English reordering

German and English are both Indo-European West Germanic languages which share various syntactic features, notably including the same basic subject-verb-object (SVO) main clause structure. However, due to isolation from other continental West Germanic languages and borrowings from other languages such as Old French and Old Norse, the English syntax significantly diverged from the syntax of other West Germanic languages such as German and Dutch.
This is particularly notable in the placement of verbs: English is quite strictly SVO in both main and relative clauses and places auxiliary verbs close to the main verbs (with optional adverbs in between), while German uses a SOV structure in relative clauses and splits auxiliaries and main verbs over the object in main clauses (SAuxOV). Other subtler differences in word placement exist. For an in-depth qualitative discussion, refer to Dryer and Haspelmath (2013); Bisazza (2013).

In sentences with a non-trivial structure which includes relative clauses, these syntactic differences can result in aligned words in a German-English sentence pair to be placed far from each other. This is relevant to machine translation, since phrase-based translation systems fail to properly address these phenomena when they occur at a distance exceeding the maximum distortion distance of the system (usually around 6 words). These issues make syntax-based translation or syntax-based pre-reordering followed by phrase-based translation particularly attractive for this language pair.

3.3 Reordering as a walk on a dependency tree

In order to quantitatively analyze the reordering phenomena in German-to-English in relation to the German dependency syntax, we introduce a reordering framework based on a *graph walk* of the dependency parse tree of the source sentence. This framework doesn't restrict the parse tree to be projective, and allows the generation of arbitrary permutations, therefore it allow us to quantify the frequency of non-projectivity and non-tree-locality

and their impact on translation quality.

Let $f \equiv (f_1, f_2, \ldots, f_{L_f})$ be a source sentence, annotated by a rooted dependency parse tree:

$$\forall j \in 1, \ldots, L_f, h_j \equiv PARENT(j) \tag{3.1}$$

We define a *walker* process that walks from word to word across the edges of the parse tree, and at each steps optionally *emits* the current word, with the constraint that each word must be eventually emitted exactly once.
Therefore, the final string of emitted words f' is a permutation of the original sentence f, and any permutation can be generated by a suitable walk on the parse tree.

3.3.1 Reordering automaton

We formalize the walker process as a non-deterministic finite-state automaton.
The state v of the automaton is the tuple:

$$v \equiv (j, E, a) \tag{3.2}$$

where $j \in 1, \ldots, L_f$ is the current vertex (word index), E is the set of emitted vertices, a is the last action taken by the automaton.
The initial state is:

$$v(0) \equiv (root_f, \{\}, null) \tag{3.3}$$

where $root_f$ is the root vertex of the parse tree [1].

At each step t, the automaton chooses one of the following actions:

- $EMIT$: emit the word f_j at the current vertex j. This action is enabled only if the current vertex has not been already emitted:

$$\frac{j \notin E}{(j, E, a) \overset{EMIT}{\rightarrow} (j, E \cup \{j\}, EMIT)} \tag{3.4}$$

[1] we can slightly reformulate the model to accommodate for dependency forests by including a dummy root vertex.

- UP: move to the parent of the current vertex. Enabled if there is a parent and we did not just come down from it:

$$\frac{h_j \neq null \wedge a \neq DOWN_j}{(j, E, a) \overset{UP}{\rightarrow} (h_j, E, UP_j)} \tag{3.5}$$

- $DOWN_{j'}$: move to the child j' of the current vertex. Enabled if the subtree rooted at j' contains vertices that have not been already emitted and if we did not just come up from it:

$$\frac{h_{j'} = j \wedge a \neq UP_{j'} \wedge \exists k \in SUBTREE(j') : k \notin E}{(j, E, a) \overset{DOWN_{j'}}{\rightarrow} (j', E, DOWN_{j'})} \tag{3.6}$$

the $DOWN_{j'}$ actions are parametrized by the index of the child they are descending to.
The UP action is not parametrized, but in the last action a part of the state it appears annotated by the current vertex, which we need in order to write the precondition of the $DOWN_{j'}$ actions.
The execution continues until all the vertices have been emitted.

We define the sequence of states of the walker automaton during one run as an *execution* $\bar{v} \in GEN(f)$. An execution also uniquely specifies the sequence of actions performed by the automation.

The preconditions make sure that all execution of the automaton always end generating a permutation of the source sentence. Furthermore, no cycles are possible: progress is made at every step, and it is not possible to enter in an execution that later turns out to be invalid.
Since the parse tree is a connected graph, every permutation of the source sentence can be generated by some execution. In fact, each permutation f' can be generated by exactly one execution, which we denote as $\bar{v}(f')$.

At the beginning of a run, the "walker" first needs to go down from the root r_1 to the vertex corresponding to the first word f'_1 in the permutation f' and then emits it. Then the "walker" climbs the tree up to the first vertex r_2 which is a parent of both the first word f'_1 and second word f'_2, then it goes down to f'_2 and emits it. The process continues until the last word f'_{L_f} in the permutation is emitted.

Therefore we can split the execution $\bar{v}(f')$ into a sequence of L_f *emission fragments* $\bar{v}_j(f')$, each ending with an $EMIT$ action.
The first fragment has zero or more $DOWN_*$ actions followed by one $EMIT$ action, while each other fragment has a non-empty sequence of UP and $DOWN_*$ actions (always zero or more UPs followed by zero or more DOWNs) followed by one $EMIT$ action. We define the highest vertex r_j in the tree visited in emission fragment $\bar{v}_j(f')$ as the *top vertex* of the fragment.

We define the *signed distance* of an emission fragment $\bar{v}_j(f')$ as the difference between the position of words f'_j and f'_{j-1} in the original sentence, that is:

$$\pi^{-1}_{f'}(j) - \pi^{-1}_{f'}(j-1) \tag{3.7}$$

where $\pi^{-1}_{f'}(j)$ maps the position of a word in the permuted sentence to its original position. For the first emission fragment, we assume that $\pi^{-1}_{f'}(0) = 0$.

We define the *excursion* of an emission fragment $\bar{v}_j(f')$ as the maximum of the unsigned distance in the original sentence between f'_j and r_j and between r_j and f'_{j-1}, again assuming that f'_0 stays at position 0.

Finally, we define an action in an execution as *forced* if it was the only action enabled at the step where it occurred.

3.3.2 Discussion

Suppose we perform reordering using a typical syntax-based system which processes source-side projective dependency parse trees and is limited to swaps between pair of vertices which are either in a parent-child relation or in a sibling relation [2].
It is easy to show that in our walk-based framework these permutations correspond to executions which describe depth-first visits of the parse tree. Specifically, the identity permutation corresponds to an inorder depth-first visit.

[2] systems which operate on constituency parse trees and only swap sibling vertices also produce these kind of permutations, due to an isomorphism between constituency parse trees and projective dependency parse trees.

Note also that in the execution of a depth-first visit the *UP* actions are always forced, since the "walker" process never leaves a subtree before all its vertices have been emitted.

Suppose instead that we could perform reordering according to an "oracle" which permuted the German words in an order that was as close as possible to the English word order [3].
The executions of our automaton corresponding to these permutations will in general contain *unforced UP* actions. We define these actions, and the execution fragments that exhibit them, as *non-tree-local*.

3.4 Characterization of German-to-English reordering against a reference permutation

If we actually had access to these "oracle" permutations we could characterize how much they differ from the *tree-local* permutations produced by typical syntax-based approaches by focusing on these *unforced UP* actions.

We could analyze the the frequency of non-tree-local execution fragments, the distributions of their lengths, and distances, how these distribution differ from tree-local execution fragments and, more importantly, what impact they have on translation quality.

In particular, we are interested in the *oracle performance gap*: the difference in translation quality between a phrase-based translation system augmented with oracle pre-reordering and the same phrase-based system without pre-reordering.
We hypothesize that sentences with large numbers of non-local execution fragments and in particular those with fragment distance above the phrase-based distortion limit have a high oracle performance gap and therefore

[3] clearly there is some ambiguity in the definition of what the English word order of a German sentence entails. We could make this definition more precise by specifying it as the permutation which when applied as a pre-processing step of a certain phrase-based translation system maximizes the expectation of a certain measure of translation quality (e.g. the BLEU score).

could potentially benefit from a reordering model that can appropriately handle non-tree-local reordering.

3.4.1 Heuristic pseudo-oracle

The oracle performance gap is by definition an upper bound on the improvement that the underlying translation system can obtain from pre-reordering. In practice, computing the oracle permutation of every sentence in the training, tuning and test corpora is too computationally expensive. Therefore, we approximate it using a heuristic that generates reference permutations from word alignments.

Following Al-Onaizan and Papineni (2006); Tromble and Eisner (2009); Visweswariah et al. (2011); Navratil et al. (2012), we concatenate the training, tuning, and test corpus the baseline translation system and we generate word alignments in both the source-to-target and the target-to-source directions using IBM model 4 as implemented in GIZA++ (Och et al., 1999) and then we combine them into a symmetrical word alignment using the "grow-diag-final-and" heuristic implemented in Moses (Koehn et al., 2007).

Given the symmetric word-aligned corpus, we assign to each source-side word an integer index corresponding to the position of the leftmost target-side word it is aligned to (attaching unaligned words to the following aligned word) and finally we perform a stable sort of source-side words according to this index.

On language pairs where IBM model 4 produces substantially accurate alignments (generally all European languages) this scheme generates a target-like reference reordering of the corpus.

3.4.2 Non-projective dependency parsing

For this set of experiments, we parsed the source side of the test corpus using the DeSR non-projective transition-based parser (Attardi, 2006).

As discussed in the previous section, the automaton execution corresponding to the identity permutation does not contain unforced UP actions if and only if the dependency tree is projective.

System	BLEU
baseline Moses	33.00
pseudo-oracle Moses	41.80

Figure 3.1: Translation quality of the baseline system and the system with pseudo-oracle reordering.

We define the execution fragments of the identity permutation which contain unforced *UP* actions as *non-projective*.

We can characterize the non-projectivity of a tree by computing statistics on these non-projective execution fragments. We can analyze their frequency and the distributions of their lengths, distances and excursions, how they relate with the non-tree-locality statistics of the (pseudo-)oracle permutation and to the translation quality and oracle performance gap.

3.4.3 Dataset and baseline system

The baseline phrase-based system was trained on the German-to-English corpus included in Europarl v7 (Koehn, 2005). We randomly split it in a 1,881,531 sentence pairs training set, a 2,000 sentence pairs development set (used for tuning) and a 2,000 sentence pairs test set. The English language model was trained on the English side of the parallel corpus augmented with a corpus of sentences from AP News, for a total of 22,891,001 sentences.

The baseline system is phrase-based Moses in a default configuration with maximum distortion distance equal to 6 and lexicalized reordering enabled. Maximum phrase size is equal to 7.
The language model is a 5-gram IRSTLM/KenLM.

The pseudo-oracle system was trained on the training and tuning corpus obtained by permuting the German source side using the heuristic described in section 3.4.1 and is otherwise equal to the baseline system.

As we can note in figure 3.1, there is a very large total performance gap of 8.8 BLEU points between the system with pseudo-oracle pre-reordering and the baseline system.

This implies that, even though the pseudo-oracle permutation does not yield the true quality upper bound of pre-reordering, it is good enough that it is worth considering how much actual pre-reordering systems (and in-decoder reordering models) can reproduce it.

3.4.4 Reordering example

Before we describe in detail the experiments that we performed, we will present an example to illustrate the linguistic phenomena we are interested in modeling.

Consider the German sentence from the Europarl corpus:
"**auf diese Weise wird die raffinierte Undurchsichtigkeit geschickt aufrechterhalten ; während der Euro " stark und stabil " sein sollte und die Währungsreserven anfangs lediglich zur Verteidigung während des Übergangszeitraums (falls notwendig) dienen sollten , erweist sich heute , daßweder die eine noch die andere dieser Behauptungen zutreffend waren und sich in Frankfurt überhaupt nichts tut !**"

It's full dependency parse tree is shown in figure 3.2, while its reference English translation is:
"**the issue therefore remains skilfully blurred ; while the euro was intended to be ' strong and stable ' and the reserve assets were originally intended to provide protection during the transitional period (should this prove necessary) , it now appears that neither of these expectations has been fulfilled and Frankfurt is totally deadlocked !**"

Using the heuristic described above, we compute the following pseudo-oracle permutation from the giza++ IBM Model 4 word alignment:
"**die raffinierte geschickt Undurchsichtigkeit aufrechterhalten ; während der Euro sollte auf sein " stark und stabil " und die Währungsreserven anfangs lediglich dienen sollten zur Verteidigung während des Übergangszeitraums (diese Weise wird falls notwendig) , erweist sich heute , daßweder die eine noch dieser Behauptungen zutreffend die andere waren und sich in Frankfurt überhaupt nichts tut !**"

Some of the word swaps (e.g. moving "**diese Weise wird**" inside the

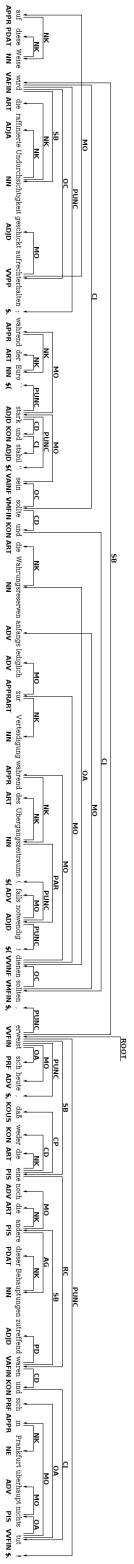


Figure 3.2: Dependency parse of "**auf diese Weise wird die raffinierte Undurchsichtigkeit geschickt aufrechterhalten ; während der Euro " stark und stabil " sein sollte und die Währungsreserven anfangs lediglich zur Verteidigung während des Übergangszeitraums (falls notwendig) dienen sollten , erweist sich heute , daßweder die eine noch die andere dieser Behauptungen zutreffend waren und sich in Frankfurt überhaupt nichts tut !**"

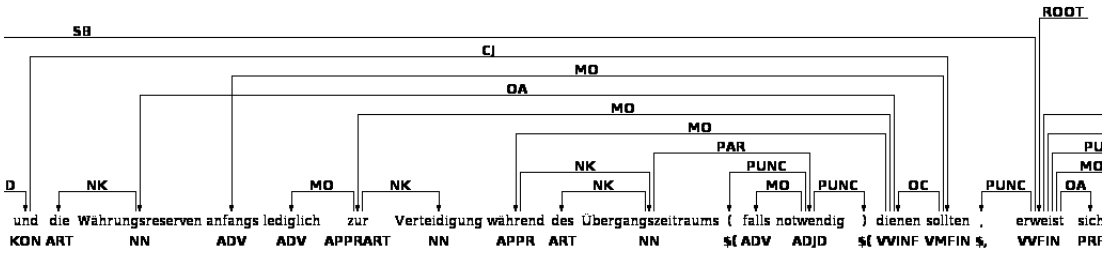


Figure 3.3: Dependency parse detail.

parentheses) in this permutation are artifacts that results from alignment errors, but the overall reordering is arguably sensible.

Consider specifically the reordered segment "**die Währungsreserven anfangs lediglich dienen sollten zur Verteidigung**" (fig. 3.3), corresponding to the English: "**the reserve assets were originally intended to provide protection**".

In order to compose this segment, the reordering automaton described in our framework must perform a complex sequence of moves on the parse tree:

- Starting from the modal verb "**sollten**", descend to the infinitive "**dienen**", then keep descending to "**Währungsreserven**" and finally to "**die**".

- Emit "**die**", then go up to "**Währungsreserven**", emit it and go up to "**dienen**" and up again to "**sollten**". Note that the first two UP actions are *forced* since they occur when no other actions are available, while the last one is *unforced* since "**dienen**" has not been emitted at that point and has also unemitted children. This unforced action indicates non-tree-local reordering.

- Go down to "**anfangs**". Note that the in the parse tree edge crosses another edge, indicating non-projectivity. Emit "**anfangs**" and go up (forced) back to "**sollten**".

- Go down to "**dienen**", down to "**zur**", down to "**lediglich**" and emit it.

- Go up (forced) to "**zur**", up (unforced) to "**dienen**", emit it, go up (unforced) to "**sollten**", emit it.
- Finally go down to "**dienen**", down to "**zur**" emit it, go down to "**Verteidigung**" and emit it.

Correctly reordering this segment would be difficult both for a phrase-based system (since the words are further apart than both the typical maximum distortion distance and maximum phrase length) and for a syntax-based system (due to the presence of non-projectivity and non-tree-locality).

3.4.5 Results

Our tests were performed on the test corpus described in the previous section consisting of 2,000 sentences, 52,711 words.

Non-tree-local reordering distribution

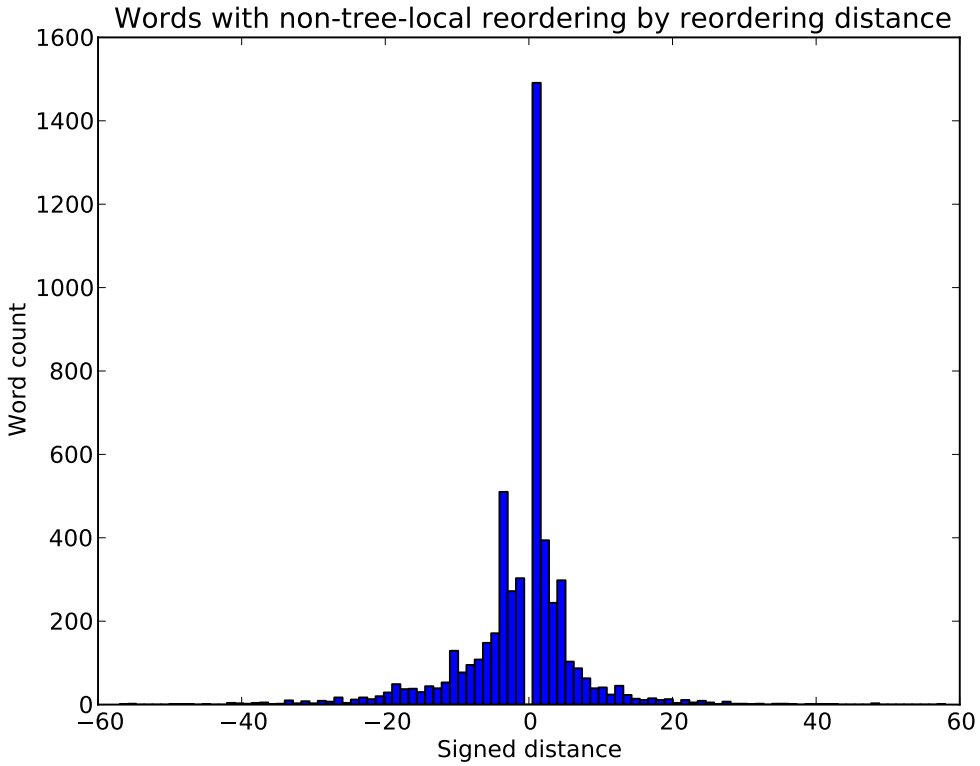
We estimated the distribution of non-tree-local reordering w.r.t. the pseudo-oracle permutation in the corpus.
We hypothesized that non-tree-local reordering is common and the distribution is heavy-tailed, and in particular that the non-tree-local fragments with a distance greater than the typical phrase-based maximum distortion distance are a non-negligible fraction.

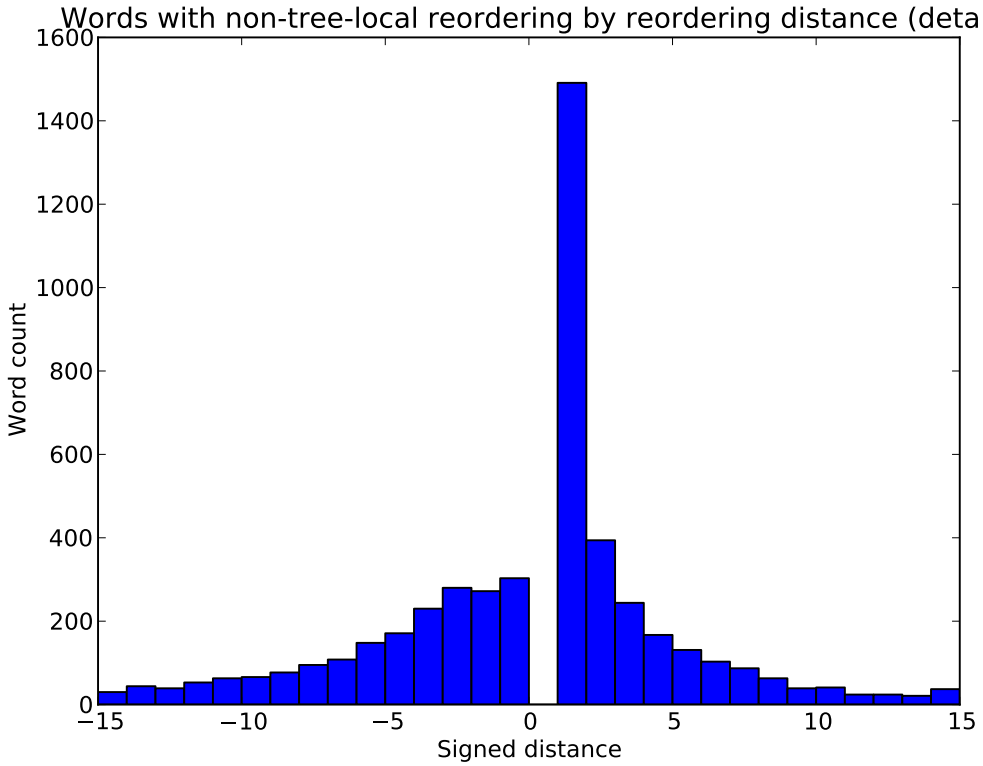
We found that:

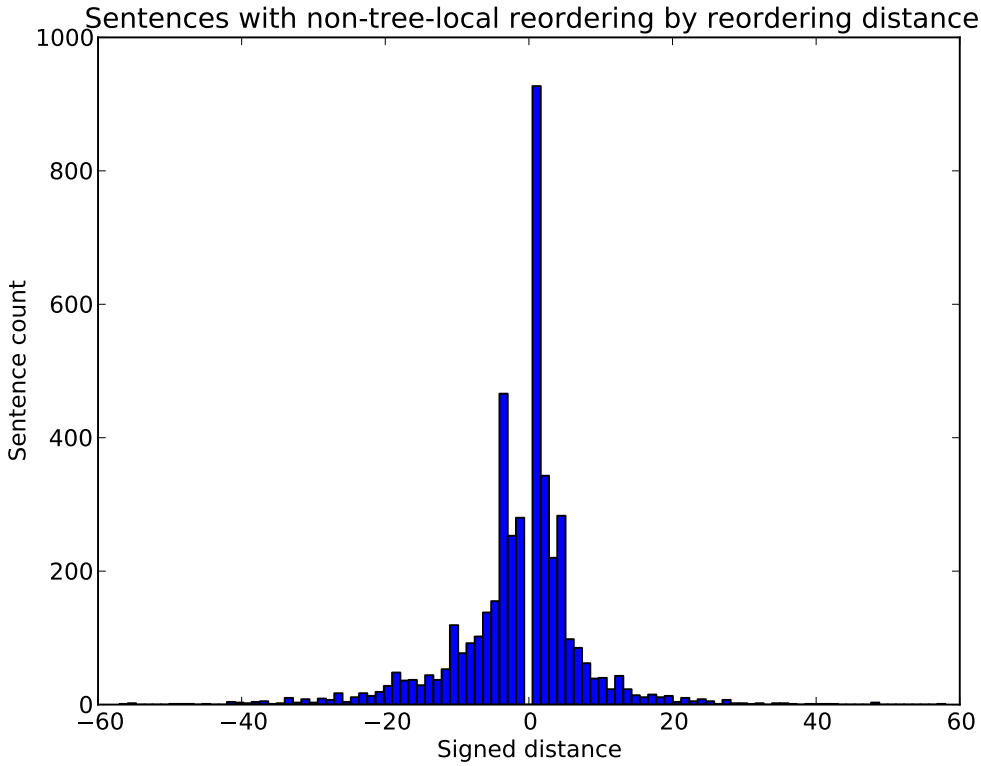
- 79% of sentences have at least one non-tree-local (NTL) fragment.
- There are 5,261 NTL fragments in the dataset, an average of one for each 10.02 words.
- 70% of all fragments are monotonic (that is, preserve the original ordering), while only 49% of NTL fragments are monotonic.
- 83% of all fragments have positive signed distance (18% excluding the monotonic ones).

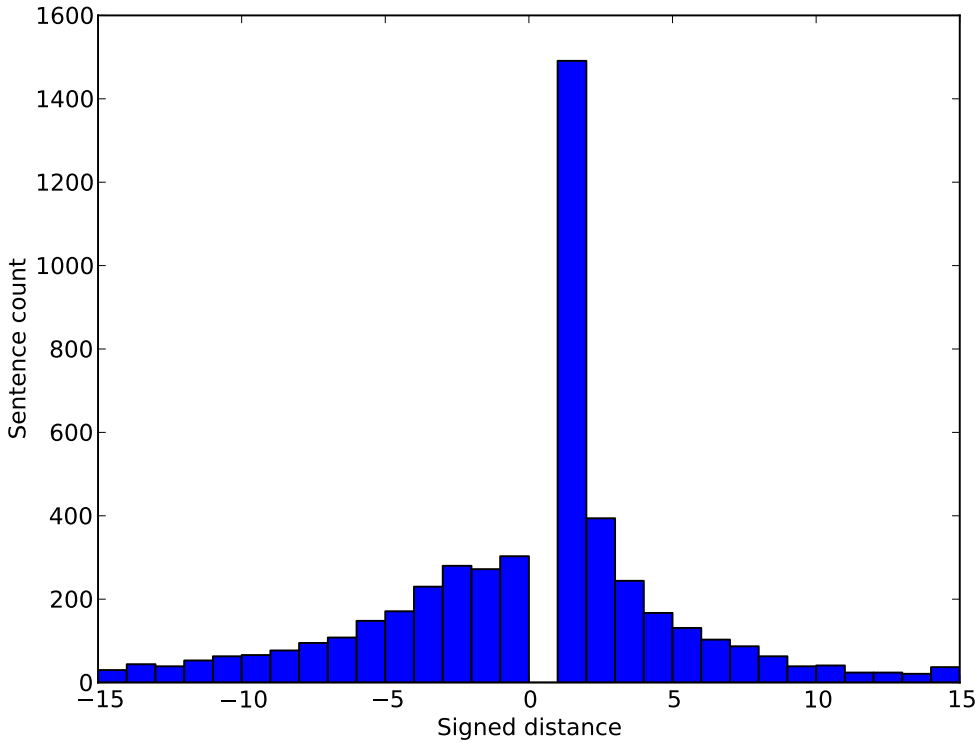
- 86% of NTL fragments have positive signed distance (22% excluding the monotonic ones).
- 25% of NTL fragments have an unsigned distance greater than 6 (the maximum distortion distance used in our configuration of Moses), 19% of NTL fragments have distance greater than 8.
- 29% of sentences with at least one NTL fragment have a NTL fragments with distance greater than 6, 21% with distance greater than 8.
- 23% of all sentences in the dataset have a NTL fragments with distance greater than 6, 17% with distance greater than 8.

Distribution plots:


Words with non-tree-local reordering by reordering distance
Word count
Signed distance
0
200
400
600
800
1000
1200
1400
1600
−60
−40
−20
0
20
40
60


Words with non-tree-local reordering by reordering distance (detail)
Word count
Signed distance
1600
1400
1200
1000
800
600
400
200
0
−15
−10
−5
0
5
10
15


Sentences with non-tree-local reordering by reordering distance
Sentence count
Signed distance
1000
800
600
400
200
0
−60
−40
−20
0
20
40
60


1600
1400
1200
1000
800
600
400
200
0
Sentence count
−15
−10
−5
0
5
10
15
Signed distance

We conclude that the hypothesis is strongly confirmed.

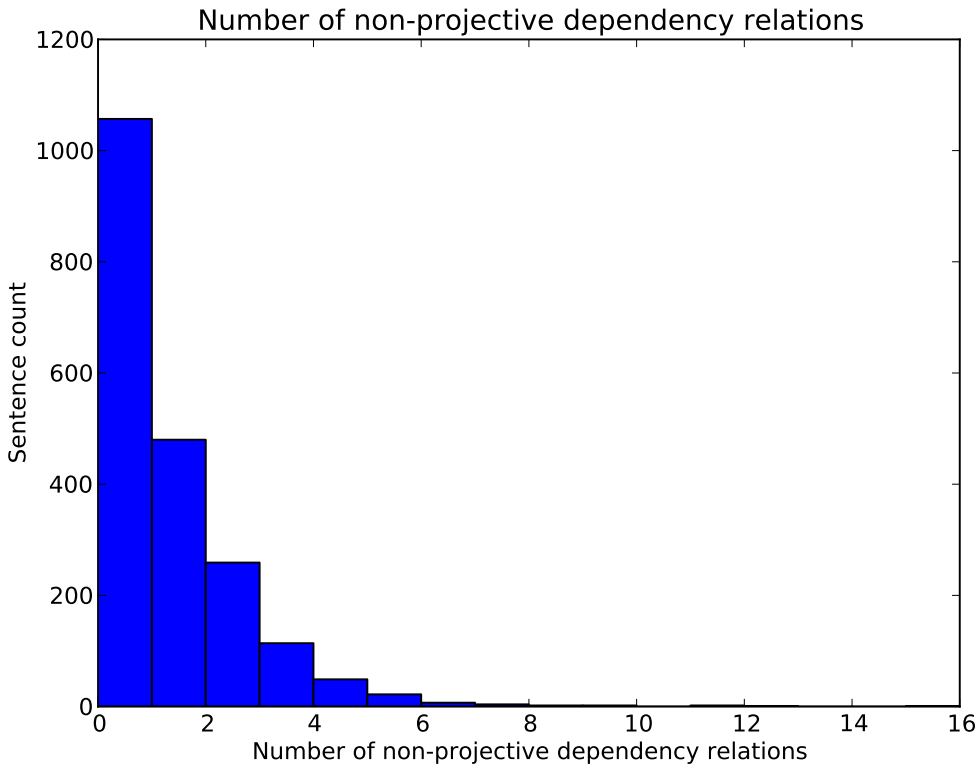
Non-projectivity distribution

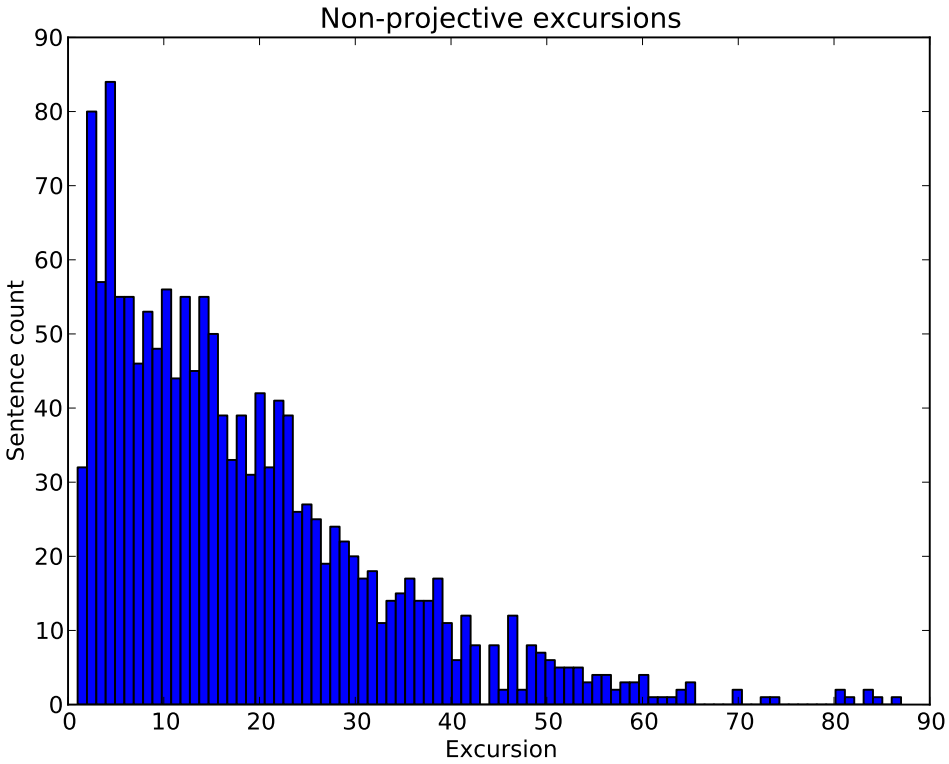
We estimated the distribution of non-projective relations in the German dependency parse trees of the dataset. We also estimate the distribution of the *excursion* of these non-projective dependency relations We hypothesized that non-projective relations are common and the excursion distribution is heavy-tailed.

We found that:

- 47% of the sentence have at least one non-projective relation, one each 29.3 words.
- Mean number of non-projective relations per sentence = 0.9, std = 1.35, median = 0.0, 95th percentile = 3.0.
- Mean excursion = 18.3, std = 14.5, median = 15, 5th percentile = 2.0, 95th percentile = 47.5.

Distribution plots:


Number of non-projective dependency relations
Sentence count
Number of non-projective dependency relations
0
200
400
600
800
1000
1200
0
2
4
6
8
10
12
14
16


Non-projective excursions
Sentence count
Excursion
0
10
20
30
40
50
60
70
80
90

We conclude that the hypothesis is strongly confirmed.

Non-tree-locality and translation quality

We analyzed the correlation between NTL reordering and baseline phrase-based translation quality, and between NTL reordering and pseudo-oracle quality gap.

Specifically for each sentence we considered the *maximum NTL fragment (unsigned) distance*[4] and the *cumulative NTL fragment distance* (the sum of the unsigned distance of each NTL fragment in the sentence).
Sentence-level translation quality can't be adequately measured using the standard BLEU score, hence we used the *sentence-level smoothed BLEU* (BLEU+1) metric (Lin and Och, 2004) and the *Translation Error Rate* (TER) metric[5] (Snover et al., 2006).

We hypothesized that NTL reordering correlates negatively with baseline phrase-based translation quality and positively with pseudo-oracle quality gap.

We found that:

- BLEU+1 correlates negatively with maximum NTL distance: Spearman's rank correlation coefficient = -0.246, p-value = 3.799e-29. Linear regression slope = 4.98e-3, intercept = 0.373, r-value = -0.214, p-value = 3.36e-22.

- TER correlates positively with maximum NTL distance: Spearman's rank correlation coefficient = 0.306, p-value = 1.16e-44. Linear regression slope = 6.96e-3, intercept = 0.509, r-value = 0.244, p-value = 1.50e-28.

- BLEU+1 pseudo-oracle gap correlates positively with maximum NTL distance: Spearman's rank correlation coefficient = 0.188, p-value =

[4]We assume that sentences without NTL fragments have a maximum NTL fragment distance equal to zero.

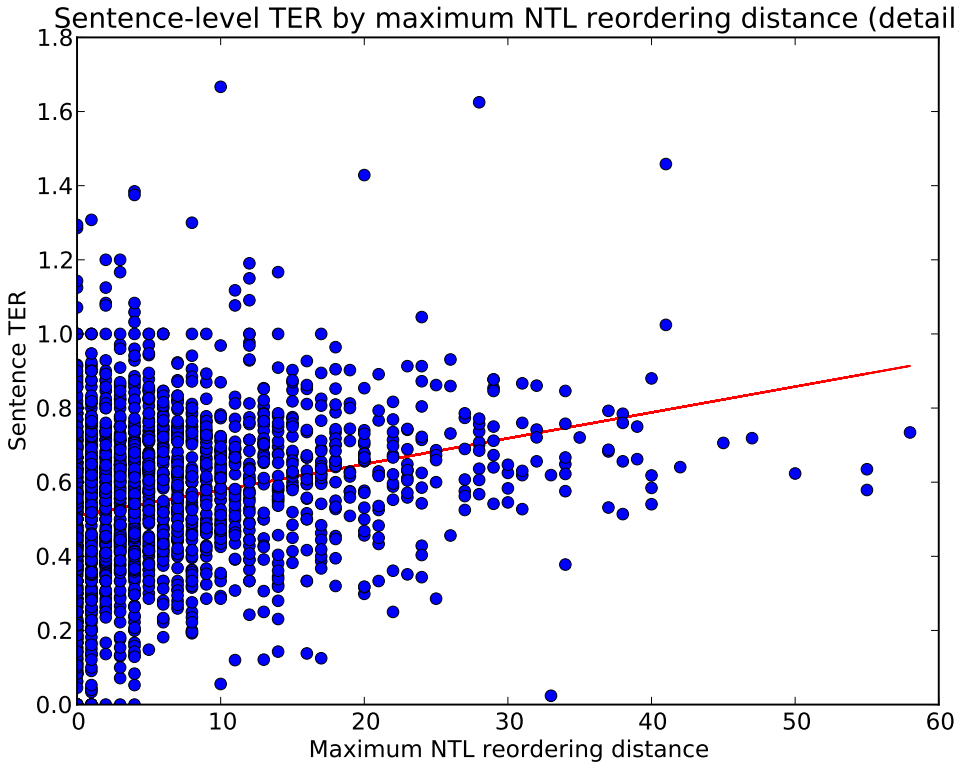
[5]note that TER is an error metric hence lower values denote a higher quality, in contrast to BLEU and BLEU+1 which are accuracy metrics

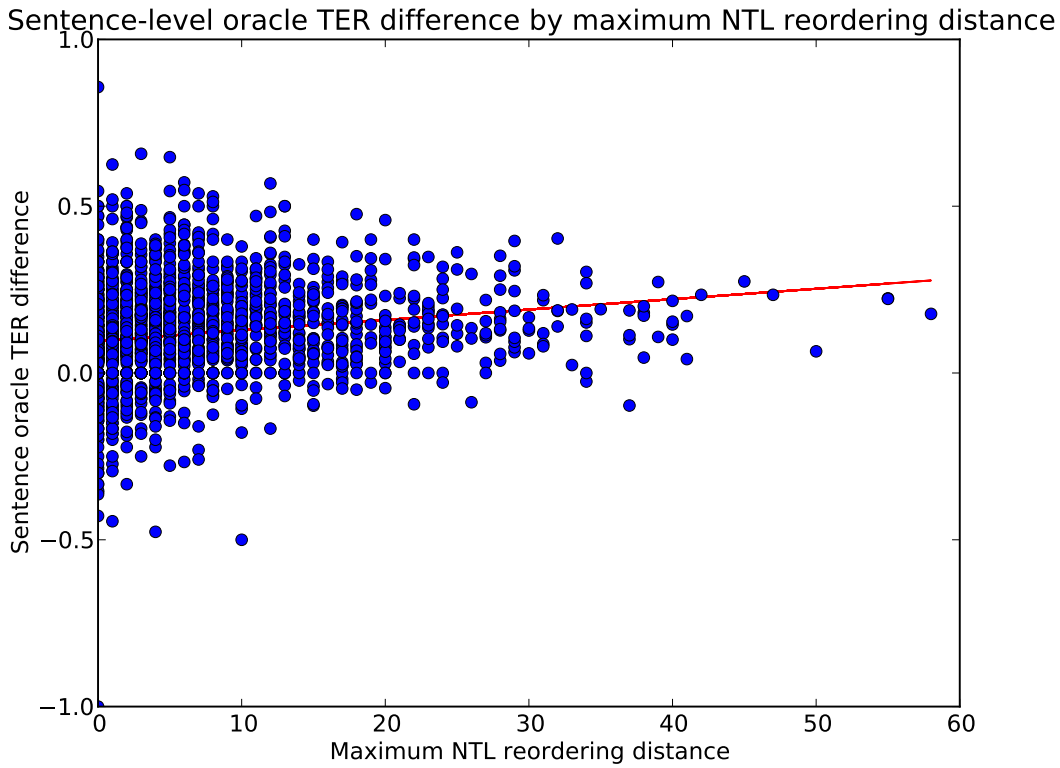
1.89e-17. Linear regression slope = 9.96e-4, intercept = -0.0730, r-value = -0.0660, p-value = 3.17e-3.

- TER pseudo-oracle gap[6] correlates positively with maximum NTL distance: Spearman's rank correlation coefficient = 0.268, p-value = 3.95e-34. Linear regression slope = 3.16e-3, intercept = 0.0967, r-value = 0.170, p-value = 1.98e-14.

Linear regression plots (TER only):

[6] defined as the TER of the baseline system minus the TER of the pseudo-oracle system, hence a positive value means that the pseudo-oracle system is better as for the BLEU+1 pseudo-oracle gap.


Sentence-level TER by maximum NTL reordering distance (detail)
Sentence TER
Maximum NTL reordering distance
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
0
10
20
30
40
50
60


Sentence-level oracle TER difference by maximum NTL reordering distance
Sentence oracle TER difference
Maximum NTL reordering distance
1.0
0.5
0.0
−0.5
−1.0
0
10
20
30
40
50
60

We conclude that the hypothesis is strongly confirmed.

Non-projectivity and translation quality

We analyzed the correlation between non-projectivity and translation quality and pseudo-oracle quality gap.

We hypothesized that the number of non-projective relations and maximum excursion of a sentence may correlate negatively with translation quality and positively with pseudo-oracle gap. However, since we are considering phrase-based system which don't make use of source dependency information, we don't expect these correlations to be very strong.

We found that:

- TER weakly correlates positively with the number of non-projective relations: Spearman's r = 0.126, p-value = 1.742e-8. Linear regression slope = 1.78e-2, intercept = 0.539, r-value = 0.105, p-value = 2.406e-6.
- BLEU+1 weakly correlates negatively with the number of non-projective relations: Spearman's r = -6.82e-2, p-value = 2.25e-3. Linear regression slope = -1.03e-2, intercept = 0.350, r-value = -7.47e-2, p-value = 8.24e-4.
- TER weakly correlates positively with maximum non-projective excursion: Spearman's r = 1.39e-1, p-value = 3.74e-10. Linear regression slope = 2.2e-3, intercept = 0.534, r-value = 1.36e-1, p-value = 1.005e-9.
- BLEU+1 weakly correlates negatively with maximum non-projective excursion: Spearman's r = -6.87e-2, p-value = 2.11e-3. Linear regression slope = -1.23e-3, intercept = 0.352, r-value = -9.24e-2, p-value = 3.48e-5
- BLEU+1 pseudo-oracle gap weakly correlates positively with the number of non-projective relations: Spearman's r = 1.31e-1, p-value = 8.24e-9. Linear regression slope = 1.06e-2, intercept = -7.17e-2, r-value = -7.91e-2, p-value = 5.42e-4.

- BLEU+1 pseudo-oracle gap weakly correlates positively with maximum non-projective excursion: Spearman's r = 1.38e-1, p-value = 5.91e-10. Linear regression slope = 5.38e-4, intercept = -7.46e-2, r-value = -6.23e-2, p-value = 5.35e-3.

- (similar TER results omitted for brevity)

We conclude that the hypothesis is confirmed in a weak form: the hypothesized correlations exist and the p-values are small enough to exclude that the effect is spurious, but these correlations are weak.

Non-projectivity and non-tree-local reordering

Since non-projectivity is not strongly correlated to phrase-based translation quality and pseudo-oracle quality gap, it could be argued that it might not be relevant to machine translation after all. Therefore, we checked if non-projectivity correlates with non-tree-local reordering, which is a relevant property.

We hypothesized that non-projectivity correlates with non-tree-locality.

We found that:

- For a word, being in a non-projective dependency relation correlates positively with being emitted in a NTL reordering fragment: Spearman's r = 2.28e-1, p-value = 0.0. Linear regression [7] slope = 4.03e-1, intercept = 8.79e-2, r-value = 2.27e-1, p-value = 0.0.

- The excursion of non-projective dependency relations correlates positively with being emitted in a NTL fragment: Spearman's r = 2.26e-1, p-value = 0.0. Linear regression slope = 1.17e-2, intercept = 9.35e-2, r-value = 1.55e-1, p-value = 5.30e-280.

We also found that the distance distributions of reordering fragments with and without non-projectivity are distinguishable (by a two-sample Kolmogorov-Smirnov test) only if the reordering fragments are non-tree-local:

[7] Considering non-projectivity and NTL as variables which take values in $0, 1$

- All words vs all non-projective words by unsigned distance, KS value = 2.12e-2, p-value = 5.03e-1 (not significant).
- All words vs all non-projective words by signed distance, KS value = 4.26e-2, p-value = 8.22e-3 (small).
- NTL words vs NTL non-projective words by unsigned distance, KS value = 4.21e-1, p-value = 1.37e-101.
- NTL words vs NTL non-projective words by unsigned distance, KS value = 4.06e-1, p-value = 1.04e-94.

We conclude that the hypothesis is confirmed: non-projectivity, especially with high excursion, correlates with non-tree-local reordering. Moreover, we argue that non-projective non-tree-local reordering tends to correspond to a specific class of non-tree-local reordering linguistic phenomena, as evidenced by the distinguishable distance distributions.

3.4.6 Discussion

We found that dependency non-tree-local reordering and dependency non-projectivity are both relevant to German-to-English machine translation. Both of them, especially non-tree-local reordering, negatively affect the translation quality of a standard phrase-based system, in particular when they involve long-distance relations.

Pre-reordering the German sentences in an English-like order can hugely improve the quality of a phrase-based system, and we found that this potential improvement correlates with dependency non-tree-local reordering and dependency non-projectivity.

These gains that we measured obviously constitute a upper bound, since the "pseudo-oracle" pre-reordering heuristic that we used reorders the German words by looking at the English reference translation, hence it can't be used in an actual translation system.

Our results indicate that performing pre-reordering (or possibly in-decoder reordering) using models which are aware of dependency non-tree-local

reordering and dependency non-projectivity has the potential to bring significant improvements. Experiments on realizable translation systems are needed to determine how much they can actually affect translation quality. The next section and the next chapter will describe some of these experiments.

3.5 Effects of dependency non-projectivity on pre-reordering with hand-coded rules

In order to determine whether non-projectity and tree non-locality can be relevant to the translation quality of an actual translation system, we need to design a system with a reordering model that takes them into consideration.

Syntax-based pre-reordering with hand-coded rules yields good results for the German-to-English translation.

Performing non-tree-local reordering would probably require rules quite different than those known in the literature. The design of these rules would require a significant amount of specific linguistic expertise, and unfortunately the author of this work is not fluent in German and thus arguably lacks the required expertise.

Designing rules that exploit dependency non-projectivity, however, appears to be an easier task, since we can adapt reordering rules already known in the literature. Therefore, in this section we will experiment on German-to-English pre-reordering using non-projective dependency-based hand-coded rules.

The results will be likely less than ideal, as these rules aren't particularly optimized, but as long as they are positive they provide support to our research hypothesis.

In the next chapter, however, we will introduce several approaches that automatically learn how to perform syntax-based pre-reordering while exploiting both dependency non-projectivity and tree non-locality.

3.5.1 Legacy rules

As a starting point for our non-projective pre-reordering rules we consider two sets of rules known in the literature: Collins et al. (2005a) and Navratil et al. (2012).

Collins et al. rules

Collins et al. approach works on German sentences fully parsed to constituency trees where non-terminal nodes are annotated with *head* and *grammatical function* tags, which makes them isomorphic to labeled projective dependency trees. Generating these parse trees with sufficient accuracy requires an appropriate parser and is relatively time consuming, but the impact of this approach on translation quality is large enough that, despite its age, it still is used in state of the art translation systems (Durrani et al., 2014).
The original implementation of Collins et al. (2005a) is not publicly available and there is some ambiguity over some details in the original article, therefore in this work we refer to the open-source implementation provided by Howlett and Dras (2011a).

Given a parsed German sentence f, Collins et al. system applies a sequence of rules which modify both the parse tree and the sentence word order. Each rule is applied to all the locations of the parse tree that match its antecedent, until these locations are exhausted (this can be performed by a depth-first visit of the tree), then the system proceeds to the next rule in the sequence, until all rules have been applied.
The rules are:

- Rule 1 "*Verb initial*". Antecedent: internal vertex is a verb phrase ('VP') and has a head child ('-HD' grammatical function). Action: move the (leftmost) head child in the first position among its siblings.

- Rule 2 "*Verb second*". Antecedent: internal vertex is a clause ('S'), has a head child and a complementizer child ('KOUS', 'PRELS', 'PRELAT',

'PWS', or 'PWAV'). Action: move the (leftmost) head child after the (leftmost) complementizer child.

- Rule 3 "*Move subject*". Antecedent: internal vertex is a clause, has a head child and a subject child ('-SB' or 'PPER-EP'). Action: move the (leftmost) subject child before the (leftmost) head child.
- Rule 4 "*Particles*". Antecedent: internal vertex is clause, has a finite verb child ('VVFIN') and a particle child ('PTKVZ'). Action: move (leftmost) particle child before (leftmost) finite verb child.
- "*Remove VP*". Antecedent: internal vertex has is a verb phrase. Action: remove vertex and reattach its children below its parent.
- Rule 5 "*Infinitives*". Antecedent: internal vertex is a clause, has a finite verb child ('VVFIN', 'VAFIN', 'VMFIN'), one or more infinitive verb children ('VVINF', 'VVIZU', 'VAINF', 'VMINF', 'VZ') and an argument child ('-DA', '-OA', '-OA2', '-OG', '-PD', '-SB', '-SBP', or '-SP') in between. Action, move all the infinitive verb children after the (leftmost) finite verb child, preserving their relative order.
- Rule 6 "*Negation*". Antecedent: internal vertex is a clause, has a finite verb child ('VVFIN', 'VAFIN' or, 'VMFIN'), an infinite verb child ('VVINF', 'VVIZU', 'VAINF', or 'VMINF') and a negation particle child ('PTKNEG'). Action: move the negation particle child after the (leftmost) finite verb child.

The numbered rules swap a vertex either with its head (which would correspond to its parent in a dependency tree) or with a sibling. However, the "*Remove VP*" step modifies the topology of the tree. Therefore, Collins et al. approach is mostly, but not strictly, tree-local.

In our experiments we used the Howlett and Dras (2011a) implementation of Collins et al. (2005a) with the Berkeley parser (Petrov et al., 2006).

Navratil et al. rules

Navratil et al. (2012) introduced a set of simpler German-to-English pre-

reordering rules which only make use of part-of-speech (POS) tags and some additional grammatical function information.

The approach is briefly described in the original paper (since it serves as a baseline for an automatic system) and no reference implementation is provided. In our implementation we used a fast non-projective dependency parser (DeSR) to extract the grammatical function information (as the deprel tags) while ignoring the topology of the parse graph except subject to verb edges which are required in order to apply the last rule.
The rules are:

- Rule 1. Antecedent: an auxiliary verb ('VA*' or 'VM*') and a full verb ('V*') on its right, not separated by punctuation ('$*'). Action: move full verb after auxiliary verb.
- Rule 2. Antecedent: an auxiliary or modal verb and a negation particle ('PTKNEG') on its right, not separated by punctuation. Action: move negation particle after auxiliary verb.
- Rule 3. Antecedent: a sequence of auxiliary or full verbs and their subject ('-SB'), not separated by punctuation. Action: move sequence of verbs after the subject.

3.5.2 Proposed rules

We developed dependency-based versions of the rule sets described above. Each of these new rule sets can be used with in conjunction with either a projective or a non-projective dependency parser, enabling us to determine the impact of non-projectivity on translation quality.

In order to adapt Collins et al. (2005a) to be used with a dependency parse tree we modify it in the following ways:

- Rule 1 antecedent matches any verb and its action moves it in first position w.r.t. a subset of its children which are considered its verb phrase. This verb phrase is estimated using a heuristic[8].

[8] deprel not in 'PUNC', 'CD', 'NG', 'CP', 'OC', 'CJ', 'SB', 'PSEUDOROOT', 'CM', POS in 'V*', 'AV*' or 'APPR', 'NN', 'APPRART', 'PROAV', 'PTKZU'

- Rules 2, 3 and 4 match any verb.
- "*Remove VP*" iteratively reattaches vertices in a verb phrase other than the head to the head's parent.
- Rules 5 and 6 match any finite verb as the parent of the other elements.

When a non-head vertex is moved by a rule, all its subtree is moved as well, compacting any gap introduced by non-projectivity.

Navratil et al. (2012) rules were adapted by just enforcing the dependency relations as constraints: the rules only move a word with respect to another if they have an appropriate dependency relation.

3.5.3 Datasets and configurations

Howlett and Dras (2011a) discovered that the performance of Collins et al. (2005a) depend on the specific dataset and phrase-based system configuration[9].
Therefore, in addition to using the dataset described in section 3.4.3, we also use on the "Newstest 2009" test set from WMT 2009 for testing.

In addition to the constituency Berkeley parser (Petrov et al., 2006) for the original Collins et al. rules, we used three different dependency parsers for the dependency-based pre-reordering rules that we proposed: a projective parser (PCFG Stanford (Rafferty and Manning, 2008)) and the non-projective parsers (DeSR and graph-based Mate (Bohnet, 2010)).

3.5.4 Results

The results of our experiments are shown in figure 3.4. Significance was assessed using *paired bootstrap resampling* (Koehn, 2004b).

All pre-reordering strategies performed significantly better than the baseline phrase-based system, although Collins et al. rules and their dependency-based variants performed better than Navratil et al. rules.

[9]Pay attention that (Howlett and Dras, 2011a) has an erratum (Howlett and Dras, 2011b) that weakens the original strong claim.

System	BLEU	vs. baseline	vs. proj
baseline, Europarl	33.00		
baseline, Newstest	18.09		
Collins original, Europarl	33.52	+0.52**	
Collins original, Newstest	18.74	+0.65**	
Navratil original, Europarl	33.10	+0.10*	
Navratil original, Newstest	18.25	+0.16*	
DepCollins Stanford (proj), Europarl	33.31	+0.31**	
DepCollins Stanford (proj), Newstest	18.39	+0.30**	
DepCollins DeSR, Europarl	33.48	+0.48**	+0.17*
DepCollins DeSR, Newstest	18.38	+0.29**	-0.01
DepCollins Mate, Europarl	33.55	+0.55**	+0.34**
DepCollins Mate, Newstest	18.69	+0.60**	+0.30**
DepNavratil Stanford (proj), Europarl	33.13	+0.13*	
DepNavratil Stanford (proj), Newstest	18.33	+0.22*	
DepNavratil DeSR, Europarl	33.16	+0.16*	+0.03
DepNavratil DeSR, Newstest	18.35	+0.24*	+0.02
DepNavratil Mate, Europarl	33.16	+0.16*	+0.03
DepNavratil Mate, Newstest	18.37	+0.26**	+0.04

Figure 3.4: Translation quality of systems with hand-coded pre-reordering rules. *: significant at 5%. **: significant at 1%.

For the dependency version of Collins et al., we can observe that using a non-projective dependency parser generally yields a significantly better result than using a projective dependency parser. For Navratil et al. rules these differences consistent but they are too small to reach statistical significance.

3.5.5 Discussion

In these experiments we used dependency-based pre-reordering rules that had not been particularly optimized for the dependency parsing framework and yet we managed to significantly improve over the phrase-based baseline. In particular, our improvements were greater when we used non-projective dependency parser, even though the rules were not specifically designed to exploit non-projectivity.
We were not able to improve over the original Collins et al. rules, which are arguably very well designed, but we performed comparably.

We conclude that syntax-based reordering models which can process non-projective dependency parse trees are a viable approach for German-to-English machine translation.

3.6 Conclusions

In this chapter we characterized German-to-English reordering from the perspective of linguistic phenomena that involve the source-side dependency syntax. Specifically we considered dependency non-projectivity and reordering tree non-locality.

We introduced an automaton model which enables us to describe arbitrary permutations of a source sentences in terms of walks on its dependency tree.

We statistically analyzed German reordering into an English-like word order using a "pseudo-oracle" heuristic. We established significant correlations between non-projectivity and non-tree-locality with translation quality and a quality improvement upper bound.
We analyzed the effects of non-projectivity on actual pre-reordering systems

which use syntax-based hand-coded rules. We found significant effects.

We conclude that dependency non-projectivity and reordering tree non-locality are properties relevant to machine translation and a syntax-based reordering system can probably benefit from taking them into consideration.

Chapter 4

Discriminative non-tree-local dependency-based sentence pre-reordering

In this chapter we introduce several methods to improve the quality of machine translation by reordering the words of the source sentences in an order idiomatic to the target language. We used discriminative machine-learning techniques to train pre-reordering models from a source-parsed word-aligned parallel corpus. Specifically, we used non-projective dependency parses and our reordering models weren't constrained to perform only tree-local word swaps.

4.1 Motivation

In the previous chapter we introduced an automaton model based on walks on a generally non-projective dependency parse tree of a source sentence and we focused in particular on the concept of *non-tree-local reordering*.

We have shown that, at least for the German-to-English language pair, sentences with significant non-projectivity in their dependence parse tree and tree non-locality w.r.t. a "pseudo-oracle" permutation have the potential of gaining a large amount of translation quality when translated after being

appropriately pre-reordered.
In this chapter we will introduce several automatic pre-reordering methods which attempt to learn how to perform these kind of syntax-based reordering from a word-aligned training set using machine learning approaches.

We investigate the research question of whether it is possible to produce effective pre-reorderings using an automatically trained system which makes use of features derived from a non-projective dependency parse of the source sentences and can capture long-distance reordering phenomena that are non-local in the sentence both when viewed as a sequence and when viewed as a dependency tree.

Among the typical statistically-trained reordering models in the literature (both for pre-reordering and in-decoder reordering) the only ones that can generate non-tree-local permutations are those which don't make use of a syntax-tree at all:
These syntax-free approaches, such as Tromble and Eisner (2009) or Visweswariah et al. (2011), Navratil et al. (2012), define a global scoring model on the permutations of the source sentence based on features which depends on all pairs of words, each considered with a local context. This enables them to formulate the reordering problem as a classical combinatorial optimization problem such as the *linear order problem* (LOP) or the *traveling salesman problem* (TSP) which can be solved to a local optimum by specialized heuristic algorithms.
These approaches don't exploit syntactic information on the source sentence other than part-of-speech tags, which on one hand makes them flexible and suitable to use on resource-poor source languages, on the other hand prevents them from making use of valuable information on source languages for which high-quality parsers are available.

Therefore we devised several statistical reordering approaches which can, in principle, learn how to generate arbitrary permutations of the source sentences while exploiting the linguistic information provided by non-projective dependency parsing.

4.2 Pre-reordering using transition-based walks on dependency parse trees

The non-deterministic automaton model of the previous chapter (sec. 3.3) was primarily useful for analysis: given a permutation f' of a source sentence f, it allowed us to compute an execution $\bar{v}(f')$.
However, we can extend it to a *weighted* automaton model and use it to generate permutations given a source sentence f.
The reordering problem can be thus defined as the following optimization problem:

$$\begin{aligned} \overset{*}{f'} &\equiv out(\overset{*}{\bar{v}}) \\ \overset{*}{\bar{v}} &\equiv \underset{\bar{v} \in GEN(f)}{\text{argmax}}\ h(f, \tau, \theta) \end{aligned} \tag{4.1}$$

where $\bar{v} \in GEN(f)$ are the executions consistent with sentence f, $out(\bar{v})$ is the permutation generated by $\bar{v}$ and $h(f, \bar{v}, \theta)$ is a scoring model parametrized by θ.
Note that since our model has no ambiguity, this equation is exact. We don't need to sum over the many executions generating a single permutation since there is only one of them.

4.2.1 Scoring model

The scoring model estimates the log-probability (up to a normalization constant) of executing the execution $\bar{v}$ on sentence f or equivalently the log-probability of permuting f into $out(\bar{v})$.

In order to enable efficient reordering, the scoring model decomposes additively over the actions in the execution:

$$h(f, \bar{v}, \theta) \equiv \sum_{t=1}^{|\bar{v}|-1} \phi(f, v(t), v(t+1), \theta) \tag{4.2}$$

we further refine this as a linear model:

$$h(f, \bar{v}, \theta) \equiv \sum_{t=1}^{|\bar{v}|-1} \theta^T \cdot g(f, v(t), v(t+1)) \tag{4.3}$$

where $\theta \in \mathcal{R}^n$ is the parameter vector and $g(\dots)$ is the n-dimensional local feature function.

We can distinguish the local features between "stateless" features, which only depend on $j(t), E(t), a(t)$ and $a(t+1)$ and "stateful" features which also depend on the additional $s(t)$ component, which normally includes information such as the ordered string of the last k emitted vertices, the sequence of the k' last actions (other than a), visit counts for each vertex, and so on.
We will leave the full specification of s and the local features to the next sections.

4.2.2 Decoding

In principle we could perform decoding using a dynamic programming algorithm. Specifically, we could adapt the incremental beam search decoding algorithm introduced for phrase-based translation (chapter 2, sec. 2.5.2), where the states of the automaton take the role of *signatures* (vertices in the decoder search graph).
Unpruned time complexity is exponential in the sentence length (due to the presence of the emitted words set, which corresponds to the covered words set of phrase-based translation) and in the size of the additional stateful context s (which, like in phrase-based translation, disincentives features that depend on a rich state).
Applying threshold and histogram pruning according to a beam size B reduces the complexity to linear in the sentence length and beam size, trading off accuracy for speed.

In our experiments we attempted to use a simpler *greedy* reordering algorithm inspired by the success of greedy transition-based classifier-driven dependency parsers (Nivre and Scholz, 2004; Attardi, 2006; Attardi and Ciaramita, 2007).

Classifier-driven action selection

The most useful features in our model turned out to be highly stateful, depending on state properties such as vertex visit counts or emitted sub-vertices counts. This leaves few opportunities for hypothesis recombination, reducing the advantage of dynamic programming over simpler approaches. Beam search would still be useful, allowing the reorderer to explore various options at each decision point without committing to a single sequence of choices, but as a preliminary attempt, inspired by the success of greedy parsers such as DeSR, we tried a greedy reordering algorithm which at each step selects an action based on the output of a classifier applied to the local features of the state.

After various attempts, we settled for a two-stage classifier:

- The first stage is a three-class classifier, specifically a logistic classifier in one-vs-all configuration, which chooses between $EMIT$, UP or $DOWN$.
- If the first stage chose $DOWN$, the second stage is applied to choose the child j' to descend to, returning a proper action $DOWN_{j'}$. This is a 1-best ranking problem over the children of the current vertex j. We reduced this problem to a binary classification problem on pairs of children: For each pair $(k, k') : k < k'$ of children of j, a binary logistic classifier assigns a vote either to k or k'. The child that receives most votes is chosen as j'. This is similar to the max-wins, one-vs-one approach used in packages such as LIBSVM (Chang and Lin, 2011) to construct a M-class classifier from $M(M-1)/2$ binary classifiers, except that we use a single binary classifier acting on a vector of features extracted from the pair of children (k, k') and the node j, with their respective local contexts. We use logistic regression for this binary classifier.

Note that we are not strictly maximizing a global linear scoring function as as defined by equation 4.3, although this approach is closely related to that framework.
In fact, this approach is more general in that it allows the use of non-linear classifiers such as kernel-SVMs or MLPs.

4.2.3 Training

Dataset preparation

Dataset preparation is performed as in the previous chapter (sec. 3.4.1, 3.4.2): we use the Al-Onaizan and Papineni (2006) heuristic, to generate a source-side reference reordering from the symmetrized IBM Model 4 word alignments of the parallel corpus and we parse the source side using the DeSR parser (Attardi, 2006; Attardi and Ciaramita, 2007).

Reference executions generation

For each parsed source sentence f in the training corpus and its reference reordering f', we generate the unique execution $\bar{v}$ such that $f' = out(v)$.

Let j be the current vertex (starting at the root) and c be the emitted vertex count (starting at zero). At each step:

- if the current vertex is the next vertex to emit $f_j = f'_{c+1}$, then generate the $EMIT$ action and increment the emit count c by one.
- otherwise find the shortest path on the parse tree between j and the next vertex to emit [1] and output the corresponding sequence of UP and $DOWN_*$ actions.

this process continues until the vertices to emit have been exhausted.

Structured prediction model training

If we use the full model described in eq. 4.3, training can be performed with any linear structured prediction machine learning algorithm such as the structured perceptron, structured SVM or structured MIRA (chapter 2, section 2.1.2), using a loss function suitable for scoring permutations.
Tromble and Eisner (Tromble and Eisner, 2009) and Visweswariah et al. (Visweswariah et al., 2011) used this training strategy for their own models.

[1] since f' is properly defined as a permutation of f, the next vertex to emit at each step is unique even if the corresponding word is repeated.

In particular, online algorithms, such as perceptron or MIRA, which iteratively perform additive updates to the parameter vector, can exploit the additive decomposability of the scoring model in order to increase the training efficiency by performing early updating: as soon as the decoding process enters a state such that the correct reference permutation can no longer produced, it is stopped and the partial feature vector accumulated so far is used for the update.
It can be proved that this doesn't affect the convergence properties of the learning algorithms and empirically it tends to improve their speed (Collins and Roark, 2004; Daumé III and Marcu, 2005; Xu et al., 2007; Huang et al., 2012).

Classifier training

In our experiment we didn't use full structured prediction model training, instead we trained our classifiers in an offline fashion by generating appropriate training sets from the source corpus and the reference executions.

Let f be a parsed source sentence and $\bar{v}$ its reference execution, for each time step t in the execution v:

- generate a training example for the first stage classifier, mapping the local feature vector $g(f, v(t), v(t+1))$ to the corresponding $EMIT, UP$ or $DOWN$ action.

- if the current action is $DOWN_{j'}$ for some child j' of the current vertex j, then for each pair of children $(k, k') : k < k'$ of j, generate a positive example for the second stage classifier if $j' = k$, or a negative example if $j' = k'$.

Downstream translation system training and testing

Once the reordering system has been trained, we apply it on the source side of the whole training corpus and the tuning corpus.

For instance, if the parallel corpora are German-to-English, after the reordering step we obtain *German′*-to-English corpora, where *German′* is

German in an English-like word order. These reordered corpora are used to train and tune a standard phrase-based translation system such as Moses.

Finally, the reordering system is applied to source side of the test corpus, which is then translated with the downstream phrase-based system and the resulting translation is compared to the reference translation in order to obtain an accuracy measure.
We also evaluate the "monolingual" reordering accuracy of upstream reordering system by comparing its output on the source side of the test corpus to the reference reordering obtained from the alignment.

4.2.4 Experiments

We performed German-to-English and Italian-to-English reordering and translation experiments.

Data

The German-to-English corpus is the same described in the previous chapter 3.4.3. In addition to the test set extracted from Europarl, we also used a 3,000 sentence pairs "challenge" set of newspaper articles provided by the WMT 2013 translation task organizers.

The Italian-to-English corpus has been assembled by merging Europarl v7, JRC-ACQUIS v2.2 (Steinberger et al., 2006) and bilingual newspaper articles crawled from news websites such as Corriere.it and Asianews.it. It consists of a 3,075,777 sentence pairs training set, a 3,923 sentence pairs development set and a 2,000 sentence pairs test set.

For both language pairs, we trained a baseline Moses phrase-based translation system with the default configuration (including lexicalized reordering).

In order to keep the memory requirements and duration of classifier training manageable, we subsampled each training set to 40,000 sentences, while both the baseline and reordered Moses system are trained on the full training sets.

Features

After various experiments with feature selection, we settled for the following configuration for both German-to-English and Italian-to-English:

- First stage classifier: current vertex j stateful features (emitted?, left/right subtree emitted?, visit count), current vertex lexical and syntactical features (surface form f_j, additional annotations $lemma_j$, pos_i, $morph_i$, $deprel_i$, and pairwise combinations between lemma, POS and DEPREL), last two actions, last two visited vertices POS, DEPREL and visit count, last two emitted vertices POS and DEPREL, bigram and syntactical trigram features for the last two emitted vertices and the current vertex, all lexical, syntactical and stateful features for the neighborhood of the current vertex (left, right, parent, parent-left, parent-right, grandparent, left-child, right-child) and pairwise combination between syntactical features of these vertices.

- Second stage classifier: stateful features for the current vertex j and the the children pair (k, k'), lexical and syntactical features for each of the children and pairwise combinations of these features, visit count differences and signed distances between the two children and the current vertex, syntactical trigram features between all combinations of the two children, the current vertex, the parent h_j and the two last emitted vertices and the two last visited vertices, lexical and syntactical features for the two children left and right neighbors.

All features are encoded as binary one-of-n indicator functions.

Results

For both German-to-English and Italian-to-English experiments, we prepared the data as described above and we trained the classifiers on their subsampled training sets.
Both stages of the classifier are trained with the LIBLINEAR package (Fan et al., 2008), using the L2-regularized logistic regression method.
The regularization parameter C is chosen by two-fold cross-validation.

Src. language	reordered	BLEU	improvement
German	no	62.10	
German	yes	57.35	-4.75
Italian	no	70.24	
Italian	yes	68.78	-1.46

Figure 4.1: Walk-based SVM monolingual reordering scores.

In order to evaluate the classifiers accuracy in isolation from the rest of the system, we performed two-fold cross validation on the same training sets, which revealed an high accuracy: The first stage classifier obtains approximately 92% accuracy on both German and Italian, while the second stage classifier obtains approximately 89% accuracy on German and 92% on Italian.

We applied the reordering pre-processing system to the source side of the corpora and evaluated the monolingual BLEU and score of the test sets (extracted from Europarl) against their reference reordering computed from the alignment

To evaluate translation performance, we trained a Moses phrase-based system (max. phrase-table length: 7, max. distortion distance: 6, lexicalized reordering model enabled) on the reordered training and tuning corpora, and evaluated the BLEU of the (Europarl) test sets. As a baseline, we also trained and evaluated Moses system on the original corpora.
We also applied our baseline and reordered German-to-English systems to the WMT2013 translation task dataset.

4.2.5 Discussion and error analysis

Unfortunately we were generally unable to improve the translation quality over the baseline.

Quality on the WMT 2013 set is very low, both for the baseline system and for our system. We attribute this to the fact that it comes form a different domain than the training set.

Src. language	test set	system	BLEU	improvement
German	Europarl	baseline	33.00	
German	Europarl	reordered	32.42	-0.58
German	WMT2013	baseline	18.80	
German	WMT2013	reordered	18.13	-0.67
Italian	Europarl	baseline	29.17	
Italian	Europarl	reordered	28.84	-0.33

Figure 4.2: Walk-based SVM translation scores

Since the cross-validation accuracy of the classifiers measured on their training sets is very high, we speculate that the problem lies not in high bias in the classifiers but in the greedy approach which, while appropriate for the computationally easier problem of parsing, does not perform well on the hard combinatorial problem of permutation optimization, that can be easily NP-hard, APX-complete even for trivial choices of features.
In a greedy system, once the classifiers make an error they cannot recover from it, and since the training sets are only generated from perfect reference executions, the system now finds itself in a region of the state space, and therefore of the feature space, that was not covered during training, causing error accumulation.
Moreover, the classifiers are trained to minimize the cross-entropy classification on their own training sets, which does not necessarily minimize the reordering error.

We performed a semi-quantitative error analysis on the output of our system:
We found that our system has a bias towards emitting words according to an in-order depth-first visit of the dependency parse tree. When it makes a mistake starting from a correct sequence, 63% of times it errs towards in-order depth-first. After making a mistake, the system rarely recovers.

We considered using a proper scoring function rather than classifiers, which would enable decoding using beam search.
However, after some preliminary exploratory tests, we found that the transi-

tions which are enabled at each point of time often lead to states which have quite different scores, which would be a problem for beam search. States that occur immediately after word emission and have the same number of emitted words have comparable scores, but at each point of time during the search process states with different number of emitted words would be in the beam.
This issue could perhaps be solved using a more complex search structure, akin to the multiple "stacks" of the phrase-based decoders, but due to time constraints we didn't investigate this issue any further.

4.3 Dependency-based Recurrent Neural Network reordering models

Given the search difficulty issues we experienced with the previous model, we decided to investigate a different class of model which have the property that state transition happen only in correspondence with word emission. This enables us to leverage the technology of incremental *language models*.

Using language models for reordering is not something new (Feng et al., 2010; Durrani et al., 2011; Bisazza and Federico, 2013; Bisazza, 2013), but instead of using a more or less standard n-gram language model, we are going to base our model on *recurrent neural network language models* (Mikolov et al., 2010).

Neural networks give the designers a great freedom in how to incorporate features, including sparse features which commonly arise when processing syntactical information, while n-gram based models have troubles dealing with sparse features. Since we want to include syntactic information in our model, this issue is relevant to us.

Moreover, n-gram language models are biased when used for reordering, since they assign some positive probability mass to sequences which are not permutations of the original sentence. This can be partially compensated during decoding by implicitly or explicitly re-normalizing the probabilities, but the estimation techniques which are used to train these models are

oblivious to the fact that they are being applied to permutations, which implies that some model capacity is wasted in representing the probabilities of impossible sequences.
Neural networks, on the other hand, can be trained more specifically on the types of sequences that will occur during decoding, hence they can avoid wasting model space to represent the probabilities of non-permutations.

4.3.1 Base RNN-RM

Let $f \equiv (f_1, f_2, \ldots, f_{L_f})$ be a source sentence.
We want a model that, given a non-empty sequence $\bar{i}$ of distinct indices in $[1, L_f]$, predicts the probability of the last index $\bar{i}_t$ conditional on the other indices $\bar{i}_{t-1:}$. Chaining these probability yields a probability distribution over the permutations of integers in $[1, L_f]$ and therefore over the permutations $f' \in GEN(f)$ of sentence f.

We assume that this process is Markovian w.r.t. some state described by a vector of real numbers $v \in \mathcal{R}^s$ and furthermore we assume that state transitions are deterministic.
These are standard assumptions which allow us to apply the standard state transition function for a single hidden layer recurrent neural network (sec. 2.1.3):

$$v(t) = \begin{cases} v_{init} & \text{if } t = 0 \\ \tanh(\Theta^{(1)} \cdot x(t) + \Theta^{REC} \cdot v(t-1)) & \text{otherwise} \end{cases} \tag{4.4}$$

where $x(t) \in \mathcal{R}^n$ is a feature vector associated to the t-th word in a permutation f' and $v_{init}, \Theta^{(1)}$ and Θ^{REC} are parameters[2]. We use the hyperbolic tangent sigmoid as activation function.

If we know the first $t-1$ words of the permutation f' in order to compute the probability distribution of the t-th word we do the following:

- Iteratively compute the state $v(t-1)$ from the feature vectors $x(1), \ldots, x(t-1)$.

[2]we don't use a bias feature since it is redundant when the layer has input features encoded with the "one-hot" encoding

- For the all the indices of the words that haven't occurred in the permutation so far $j \in J(t) \equiv ([1, L_f] - \bar{i}_{t-1:})$, compute a score $h(j,t) \equiv h_o(v(t-1), x_o(j))$, where $x_o(\cdot)$ is the feature vector of the candidate target word. (possibly computed by a different feature extraction function than $x(\cdot)$).
- Normalize the scores using the logistic softmax function: $P(\bar{I}_t = j|f, \bar{i}_{t-1:}, t) = \frac{\exp(h(j,t))}{\sum_{j' \in J(t)} \exp(h(j',t))}$.

The scoring function $h_o(v(t-1), x_o(j))$ applies a feed-forward hidden layer to the feature inputs $x_o(j)$, and then takes a weighed inner product between the activation of this layer and the state $v(t-1)$. The result is then linearly combined to an additional feature equal to the logarithm of the remaining words in the permutation $(L_f - t)$[3] and to a bias feature:

$$h_o(v(t-1), x_o(j)) \equiv < \tanh(\Theta^{(o)} \cdot x_o(j)), \theta^{(2)} \odot v(t-1) > + \theta^{(\alpha)} \cdot \log(L_f - t) + \theta^{(bias)} \tag{4.5}$$

We can compute the probability of an entire permutation f' just by multiplying the probabilities for each word:

$$P(f'|f) = P(\bar{I} = \bar{i}|f) = \prod_{t=1}^{L_f} P(\bar{I}_t = \bar{i}_t|f, t) \tag{4.6}$$

Evaluation

Given a source sentence f and a reference permutation f' we can evaluate how much the model fits the reference using the *log-likelihood averaged by words*, which is defined as

$$L(f'|f) = \log_2(P(f'|f))/L_f \tag{4.7}$$

(the logarithm is in base two in order to obtain a result in bits, consistently with the language models literature.)

This measure negated and averaged over a dataset of pairs of sentences and reference permutations is the *empirical cross-entropy* CE of the model w.r.t. the dataset. The *perplexity* of the model is defined 2^{CE}.

[3] since we are then passing this score to a softmax of variable size $(L_f - t)$, this feature helps the model to keep the score already approximately scaled.

Training

Given a training set of pairs of sentences and reference permutations, the training problem is defined as finding the set of parameters $\theta \equiv (v_{init}, \Theta^{(1)}, \theta^{(2)}, \Theta^{REC}, \Theta^{(o)}, \theta^{(\alpha)}, \theta^{(bias)})$ which minimizes the empirical cross-entropy of the model w.r.t. the training set.
It is easy to show that this function is differentiable w.r.t. the parameters: gradients can be efficiently computed using *error backpropagation through time*. In principle any unconstrained gradient-based optimization algorithm could be used to solve this problem to a local minimum.

In practice we used the following training architecture:

- Stochastic gradient descent, with each training pair (f, f') considered as a single minibatch for updating purposes.
- Gradients computed using the automatic differentiation facilities of Theano (Bergstra et al., 2010) (which implements a generalized back-propagation). No truncation is used.
- L2-regularization ($\lambda = 10^{-4}$ per minibatch).
- Learning rates dynamically adjusted per scalar parameter using the *AdaDelta* heuristic (Zeiler, 2012).
- Gradient clipping heuristic to prevent the "exploding gradient" problem (Graves, 2013).
- Early stopping w.r.t. a validation set to prevent overfitting.
- Uniform random initialization for parameters other than the recurrent parameter matrix Θ^{REC}. Random initialization with *echo state property* for Θ^{REC}, with contraction coefficient $\sigma = 0.99$.

Training time complexity is $O(L_f^2)$ per sentence, which could be reduced to $O(L_f)$ using truncated backpropagation through time at the expense of update accuracy and hence convergence speed. Space complexity is $O(L_f)$ per sentence.

Decoding

In order to use the RNN-RM model for pre-reordering we need to compute the most likely permutation $\overset{*}{f'}$ of the source sentence f:

$$\overset{*}{f'} \equiv \underset{f' \in GEN(f)}{\text{argmax}} \; P(f'|f) \tag{4.8}$$

Solving this problem to the global optimum is computationally hard[4], hence we solve it to a local optimum using a *beam search* strategy.

We generate the permutation incrementally from left to right. Starting from an initial state consisting of an empty string and the initial state vector v_{init}, at each step we generate all possible successor states and retain the B-most probable of them (histogram pruning), according to the probability of the entire prefix of permutation they represent.

Since RNN state vectors do not decompose in a meaningful way, we don't use any hypothesis recombination.
At step t there are $L_f - t$ possible successor states, and the process always takes exactly L_f steps[5], therefore time complexity is $O(B \cdot L_f^2)$ and space complexity is $O(B)$.

Features

We use two different feature configurations: *unlexicalized* and *lexicalized*.

In the *unlexicalized* configuration, the state transition input feature function $x(j)$ is composed by the following features, all encoded using the "one-hot" encoding:

- Unigram: $POS(j)$, $DEPREL(j)$, $POS(j) * DEPREL(j)$.
- Left, right and parent unigram: $POS(k)$, $DEPREL(k)$, $POS(k) * DEPREL(k)$, where k is the index of respectively the word at the left (in the original sentence), at the right and the dependency parent of word j. Unique tags are used for padding.

[4] presumably at NP-hard

[5] actually, $L_f - 1$, since the last choice is forced

- Pair features: $POS(j) * POS(k)$, $POS(j) * DEPREL(k)$, $DEPREL(j) * POS(k)$, $DEPREL(j) * DEPREL(k)$, for k defined as above.
- Triple features $POS(j) * POS(left_j) * POS(right_j)$, $POS(j) * POS(left_j) * POS(parent_j)$, $POS(j) * POS(right_j) * POS(parent_j)$.
- Bigram: $POS(j) * POS(k)$, $POS(j) * DEPREL(k)$, $DEPREL(j) * POS(k)$ where k is the previous emitted word in the permutation.
- Topological features: three binary features which indicate whether word j and the previously emitted word are in a parent-child, child-parent or sibling-sibling relation, respectively.

The target word feature function $x_o(j)$ is the same of $x(j)$ except that each feature is also conjoined with a quantized signed distance[6] between word j and the previous emitted word.

The *lexicalized* configuration is equivalent to the unlexicalized one except that $x(j)$ and $x_o(j)$ also have the surface form of word j (not conjoined with the signed distance). Words that appear less than 10 times in the training set are replaced by a distinguished rare word tag.

4.3.2 Fragment RNN-RM

The *Base RNN-RM* described in the previous section includes dependency information, but not the full information of reordering fragments as defined by our automaton model.

In order to determine whether this rich information is relevant to machine translation pre-reordering, we propose an extension, denoted as *Fragment RNN-RM*, which includes reordering fragment features, at expense of a significant increase of time complexity.

We consider a hierarchical recurrent neural network. At top level, this is defined as the previous RNN. However, the $x(j)$ and $x_o(j)$ vectors, in addition to the feature vectors described as above now contain also the final

[6] values greater than 5 and smaller than 10 are quantized as 5, values greater or equal to 10 are quantized as 10. Negative values are treated similarly.

states of another recurrent neural network.
This internal RNN has a separate clock and a separate state vector. For each step t of the top-level RNN which transitions between word $f'(t-1)$ and $f'(t)$, the internal RNN is reinitialized to its own initial state and performs multiple internal steps, one for each action in the fragment of the execution that the walker automaton must perform to walk between words $f'(t-1)$ and $f'(t)$ in the dependency parse (with a special shortcut of length one if they are adjacent in f with monotonic relative order).

The state transition of the inner RNN is defined as:

$$v_r(t) = \begin{cases} v_{r_{init}} & \text{if } t_r = 0 \\ \tanh(\Theta^{(r_1)} \cdot x_r(t_r) + \Theta^{r_{REC}} \cdot v_r(t_r - 1)) & \text{otherwise} \end{cases} \tag{4.9}$$

where $x_r(t_r)$ is the feature function for the word traversed at inner time t_r in the execution fragment. $\Theta^{(r_1)}$ and $\Theta^{r_{REC}}$ are parameters.

Evaluation and decoding are performed essentially in the same was as in Base RNN-RM, except that the time complexity is now $O(L_f^3)$ since the length of execution fragments is $O(L_f)$.

Training is also essentially performed in the same way, though gradient computation is much more involved since gradients propagate from the top-level RNN to the inner RNN.

In our Python/Theano implementation we just used two nested "scan" primitives to implement the RNNs and let Theano's automatic differentiation facilities compute the gradients. This may not be necessarily optimal.

Features

The *unlexicalized* features for the inner RNN input vector $x_r(t_r)$ depend on the current word in the execution fragment (at index t_r), the previous one and the action label: UP, $DOWN$ or $RIGHT$ (shortcut). $EMIT$ actions are not included as they always implicitly occur at the end of each fragment.
Specifically the features, encoded with the "one-hot" encoding are: $A * POS(t_r) * POS(t_r - 1)$, $A * POS(t_r) * DEPREL(t_r - 1)$, $A * DEPREL(t_r) * POS(t_r - 1)$, $A * DEPREL(t_r) * DEPREL(t_r - 1)$.

These features are also conjoined with the quantized signed distance (in the original sentence) between each pair of words.

The *lexicalized* features just include the surface form of each visited word at t_r.

4.3.3 Base GRU-RM

We also propose a variant of the Base RNN-RM where the standard recurrent hidden layer is replaced by a *Gated Recurrent Unit* layer, recently proposed by Cho et al. (2014a) for machine translation applications.
The Base GRU-RM is defined as the Base RNN-RM of sec. 4.3.1, except that the recurrence relation 4.4 is replaced by:

$$\begin{aligned} v_{rst}(t) &= \pi(\Theta^{(1)}_{rst} \cdot x(t) + \Theta^{REC}_{rst} \cdot v(t-1)) \\ v_{upd}(t) &= \pi(\Theta^{(1)}_{upd} \cdot x(t) + \Theta^{REC}_{upd} \cdot v(t-1)) \\ v_{raw}(t) &= \tanh(\Theta^{(1)} \cdot x(t) + \Theta^{REC} \cdot v(t-1) \odot v_{upd}(t)) \\ v(t) &= v_{rst}(t) \odot v(t-1) + (1 - v_{rst}(t)) \odot v_{raw}(t) \end{aligned} \tag{4.10}$$

where $v_{rst}(t)$ and $v_{upd}(t)$ are the activation vectors of the "reset" and "update" gates, respectively.

Features are the same of unlexicalized Base RNN-RM (we experienced difficulties training the Base GRU-RM with lexicalized features).
Training is also performed in the same way except that we found more beneficial to convergence speed to optimize using *Adam* Kingma and Ba (2014) [7] rather than AdaDelta.
In principle we could also extend the Fragment RNN-RM into a Fragment GRU-RM, but we did not investigate that model in this work.

4.3.4 Experiments

We performed German-to-English pre-reordering experiments with Base RNN-RM (both unlexicalized and lexicalized), Fragment RNN-RM (unlexicalized) and Base GRU-RM (unlexicalized). Due to time constraints and

[7] with learning rate $2 \cdot 10^{-5}$ and all the other hyperparameters equal to the default values in the article.

hardware limitations (namely, the lack of GPU-equipped workstations) we didn't perform Italian-to-English experiments.

Data

For our experiments with neural network models we used the same German-to-English datasets that were used for the previous set of experiments with the SVM classifiers model (sec. 4.2.4).
However, in the SVM models the size of the training set used to train the pre-reordering model was limited by RAM size. In the neural network models we use an online training algorithm that loads the training examples from the disk at the fly, hence we can use a much bigger dataset. We extract approximately 300,000 sentence pairs from the Moses training set based on a heuristic confidence measure of word-alignment quality (Huang, 2009; Navratil et al., 2012). We randomly removed 2,000 sentences from this filtered dataset to form a validation set for early stopping, the rest were used for training the pre-reordering model.

Results

The hidden state size s was set to 100 for the RNN models and 30 for the GRU model, validation was performed every 2,000 training examples. After 50 consecutive validation rounds without improvement, training was stopped and the set of training parameters that resulted in the lowest validation cross-entropy were saved.
Training took approximately 1.5 days for the unlexicalized Base RNN-RM, 2.5 days for the lexicalized Base RNN-RM and unlexicalized Base GRU-RM and 5 days for the unlexicalized Fragment RNN-RM on a 24-core machine without GPU (CPU load never rose to more than 400%).

Decoding was performed with a beam size of 4 (we found that any beam size above 1 doesn't affect accuracy by much). Decoding the whole corpus took about 1.0-1.2 days for all the models except Fragment RNN-RM for which it took about 3 days. We were able to parallelize decoding on multiple cores and multiple machines, hence it didn't take a significant wall-clock

reordering	BLEU	improvement
none	62.10	
unlex. Base RNN-RM	64.03	+1.93
lex. Base RNN-RM	63.99	+1.89
unlex. Fragment RNN-RM	64.43	+2.33
unlex. Base GRU-RM	64.78	+2.68

Figure 4.3: Recurrent neural network monolingual reordering scores. All improvements are significant at 1% level.

time, except in the case of the Fragment RNN-RM system.

Effects on monolingual reordering score are shown in fig. 4.3, effects on translation quality are shown in fig. 4.4.

We also measured correlations between tree non-locality (w.r.t. the "pseudo-oracle" permutation, as discussed in chapter 3) and translation quality improvement and between dependency non-projectivity and translation quality improvement.
We performed these tests w.r.t. our richest model (unlexicalized Fragment RNN-RM).
We found that:

- BLEU+1 improvement correlates positively with maximum NTL distance: Spearman's rank correlation coefficient = 0.210, p-value = 3.26e-3. Linear regression slope = 1.06e-4, intercept = -0.0480, r-value = -0.0660, p-value = 6.22e-2.

- BLEU+1 improvement correlates positively with the number of non-projective relations: Spearman's rank correlation coefficient = 0.185, p-value = 7.57e-3. Linear regression slope = -2.62e-4, intercept = 1.75e-16, r-value = -0.0211, p-value = 3.68e-1.

- Only Spearman's correlations are significant, suggesting a non-linear effect.

Test set	system	BLEU	improvement
Europarl	baseline	33.00	
Europarl	"oracle"	41.80	+8.80
Europarl	Collins	33.52	+0.52
Europarl	unlex. Base RNN-RM	33.41	+0.41
Europarl	lex. Base RNN-RM	33.38	+0.38
Europarl	unlex. Fragment RNN-RM	33.54	+0.54
Europarl	**unlex. Base GRU-RM**	34.15	**+1.15**
news2013	baseline	18.80	
news2013	Collins	NA	NA
news2013	unlex. Base RNN-RM	19.19	+0.39
news2013	lex. Base RNN-RM	19.01	+0.21
news2013	unlex. Fragment RNN-RM	19.27	+0.47
news2013	**unlex. Base GRU-RM**	19.28	**+0.48**
news2009	baseline	18.09	
news2009	**Collins**	18.74	**+0.65**
news2009	unlex. Base RNN-RM	18.50	+0.41
news2009	lex. Base RNN-RM	18.44	+0.35
news2009	unlex. Fragment RNN-RM	18.60	+0.51
news2009	unlex. Base GRU-RM	18.58	+0.49

Figure 4.4: Recurrent neural network translation scores. All improvements are significant at 1% level.

4.3.5 Discussion and analysis

All our models significantly improve over the phrase-based baseline, performing as well as or almost as well as Collins et al. (2005a) and improving over it in one case, which is an interesting result since our models doesn't require any specific linguistic expertise.

Surprisingly, the lexicalized version of Base RNN-RM performed worse than the unlexicalized one. This goes contrary to expectation as neural language models are usually lexicalized and in fact often use nothing but lexical features. We speculate that this may be due the relative small hidden state size that we used, as RNN models in the literature tend to use hidden state size in the range of 250 - 1,000. Due to time constraints and hardware limitations, we were not able to test our models in this regime.

The unlexicalized Fragment RNN-RM was quite accurate but very expensive both during training and decoding.
We believe that it should be considered more a proof of concept than a practical algorithm, at least in the short term.

The unlexicalized Base GRU-RM performed very well, especially on the Europarl dataset (where all the scores are much higher than the other datasets) and it never performed significantly worse than the unlexicalized Fragment RNN-RM which is much slower.

We also performed exploratory experiments with different feature sets (such as lexical-only features) but we couldn't obtain a good training error. Larger network sizes should increase model capacity and may possibly enable training on simpler feature sets.

4.4 Conclusions

We presented two classes of statistical syntax-based pre-reordering systems for machine translation.
Our systems processes source sentences parsed with non-projective dependency parsers and permutes them into a target-like word order, suitable for translation by an appropriately trained downstream phrase-based system.

The models we proposed are completely trained with machine learning approaches and is, in principle, capable of generating arbitrary permutations, without the hard constraints that are commonly present in other statistical syntax-based pre-reordering methods.
Practical constraints depend on the choice of features and are therefore quite flexible, allowing a tradeoff between accuracy and speed.

In our experiments with the SVM walk-based model we used a greedy reordering algorithm, inspired by transition-based dependency parsing, which is quite fast but has not turned out to be accurate enough to improve over the baseline.

In our experiments with the neural network models we managed to achieve translation quality improvements comparable or better to the best hand-coded pre-reordering rules.
Moreover, we have shown that reordering tree non-locality and dependency non-projectivity significantly affect the translation quality of a real syntax-based system, thereby confirming our research hypothesis.

Chapter 5

Translation reranking using source-side dependency syntax and graph echo state networks

In this chapter we introduce a class of syntax-aware and syntax-free reranking methods to select a most promising translation from a N-best list of candidate translations.

Specifically, we investigate the question of whether reranking on the basis of non-projective dependency features is feasible, although we also introduce and investigate some syntax-free models in the process.

We describe an approach that uses features computed by processing vector word embeddings of the source and translated sentences by a class of structural neural networks which can make use of both the dependency structure and the alignment structure of the sentence pair.

We also describe an approach which uses machine learning over hand-coded syntactic features derived from a dependency parse of the source sentences and from information (in particular, word-alignments) reported by the translation engine.

5.1 Reranking using graph echo state networks

Neural language models (section: 2.4.2) are powerful tools to model the linguistic phenomena producing medium-long range correlations between different words in a sentence or between source and target words in a sentence pair.
In this section we propose a class of monolingual and bilingual neural language models based on graph neural networks, particularly graph echo state networks (section: 2.1.3). These models are an extension of the monolingual RNN language models of Mikolov et al. (Mikolov et al., 2010) and its bilingual variants (Auli et al., 2013).
We will then describe the results of reraking experiments with these models on the Italian-to-English language pair.

5.1.1 Sequence graph monolingual language model

Given a sentence x of L_x words, plus a special terminator symbol "**</s>**", perhaps the simplest way to represent it as a graph (V_x, E_x) is to consider the words as vertexes linked by directed edges in a chain preserving their order:

$$V_x \equiv 1, \ldots, L_x,\ E_x \equiv (j, j+1), \forall j \in 1, \ldots, L_x \tag{5.1}$$

We can use this graph to compute the sentence probability as a global property regression task using a graph neural network: for each vertex, a fixed-size (s) reservoir of neural units is instantiated, which is fed with a vectorial representation of its corresponding word (e.g. a word embedding) and the state of its left neighboor and right neighboor, multiplied by suitable parameter matrices $\Theta^{EDGE_{LEFT}}$ and $\Theta^{EDGE_{RIGHT}}$, respectively.
Starting from the null initial state, the system is allowed to evolve for several time steps until convergence, then the states of all the vertex reservoirs are combined (by summing or averaging) and feed to a generalized linear model (e.g. a logistic softmax) which computes the sentence probability.

Formally, let $v(t) \in \mathcal{R}^{s \times L_x + 1}$ be the global state vector of the network at time t, which is the concatenation of per-vertex (per-word) state vectors

$v(t,j) \in \mathcal{R}^s$:

$$v(t,j) \equiv v_{1+s\cdot(j-1):s\cdot(j)}(t) \tag{5.2}$$

then the state evolution of the network is:

$$v(t,j) = \pi^{(1)}(\Theta^{(1)} \cdot \overline{x_j} + \Theta^{EDGE_{LEFT}} \cdot v(t-1,j-1) + \Theta^{EDGE_{RIGHT}} \cdot v(t-1,j+1)) \tag{5.3}$$

where $\pi^{(1)}(\cdot)$ is the activation function for the hidden layer, $\overline{x_j}$ is the vectorial word embedding of word x_j and assuming $\forall j, v(0,j) = 0^{\otimes s}$ and $\forall t, v(t,0) = v(t, L_x + 2) = 0^{\otimes s}$.

The network is run until convergence, which always occurs if it has the *echo state property* (Gallicchio and Micheli, 2011b). Then the per-vertex state vectors are summed or averaged[1], obtaining a fixed-dimensional global feature vector $\tilde{v} = \sum_{j=1}^{L_x+1} v(\infty, j)$ or $\tilde{v} = \frac{1}{L_x+1} \sum_{j=1}^{L_x+1} v(\infty, j)$, which is feed to the readout layer.

Note that this model is quite similar to an RNN language model, where we can consider the input words to be linked together in a chain graph with edges representing conditional dependence relations. However, the RNN model differs from our model in the way the computation is performed:

- In the RNN model, at each time step t, one input word is fed into the network, starting from a padding token, and the probability of the next word is computed. Once the input words have been exhausted, the sentence probability is computed just as the product of the individual word probabilities, according to a chain-rule decomposition.

- In our graph-based language model, all the input words are fed simultaneously in the network, and its state is left to evolve until convergence, then this state is reduced from a $s \cdot L_x + 1$-dimensional vector to a s-dimensional feature vector by the combination operation and a global estimation of the sentence probability is made by the readout layer.

[1] summing is theoretically more appropriate, since it enables the model to be sensitive to the sentence length. However, in a reranking setting, the sentence length is already represented by a separate feature, and averaging reduces the magnitude of the variance of the neural network features across the training examples, which may facilitate training.

Note that while in a RNN the estimated probability at each word is influenced only by its left context, in our model it is influenced both by the left and the right context.

Given a training corpus of sentences $x^{(1)}, \ldots, x^{(m)}$, we could fully train all the parameters in our model by converting the problem to a supervised learning task (using, for instance, noise-contrastive estimation (Gutmann and Hyvärinen, 2010)) and estimating the parameters using a backpropagation-based technique (Scarselli et al., 2009).
However, we can avoid this computationally expensive full parameter training by using the reservoir computing approach: we randomly initialize the edge parameter matrices while enforcing that they provide the *echo state property* which is required to avoid chaotic behavior. Since the degree of each vertex is always equal to one or two, this is accomplished by constraining their norm:

$$\max_{\psi \in LEFT,RIGHT} \left| \Theta^{EDGE_\psi} \right| < \frac{1}{2} \tag{5.4}$$

assuming a hyperbolic tangent activation function in the reservoir units.
We could also randomize the input matrix, or we can just chose it to provide s-dimensional word embedding vectors.
The only layer of the model which needs training is the output layer which computes the sentence probabilities from the combined state vectors.

Actually, if we are interested in performing reranking for SMT, we do not even have to train the readout layer as a separate component:
In a n-best reranking task, we are presented with a training set [2] of source sentences $f^{(1)}, \ldots, f^{(m)}$, each with its N-best translations $e^{(i,1)}, \ldots, e^{(i,N)}$. For each pair, we also have a fixed-dimensional vector of numeric features $\phi^{(i,k)} \in \mathcal{R}^{s'}$ used by the decoder to compute the model score of the sentence pair.
For each pair $f^{(i)}, e^{(i,k)}$, we can just compute the combined s-dimensional feature vector for $e^{(i,k)}$ and concatenate it with $\phi^{(i,k)}$. These extended feature vectors are then used to train a generalized linear model specific for reraking using an SMT tuning algorithm such as MERT or structured k-best MIRA.

[2] possibly corresponding to the tuning set of a statistical machine translation system

Figure 5.1: A graph neural network for a monolingual language model. Each word corresponds to vertex which shares a (directed) edge only with the vertexes of the adjacent words.
This is similar to a conventional RNN model.

We define this approach as the *Sequence GraphESN language model* (Sequence GESN-LM).

5.1.2 Bilingual graph language model

In a reranking task, for each sentence pair (f, e) we usually also get the bidirectional word alignments $WA_{f,e}$ between source and target words computed by the decoder. We can make use of these alignments to condition the language model on source words as well as target words.

We build sentence graphs as above for both the source and the target sentences. Then we include edges of a different type to represent word alignments. Note that since the graph is bipartite w.r.t. this type of edges, we do not have to use two different parameter matrices to distinguish the endpoints. A single matrix $\Theta^{EDGE_{ALIGN}}$ suffices.

Formally, denote the per-vertex state vectors as $v(t, j, \alpha)$, where $\alpha \in f, e$ and let $\neg f \equiv e$, $\neg e \equiv f$.
The state evolution of the neural network is:

$$v(t,j,\alpha) = \pi^{(1)}(\Theta^{(1)} \cdot \overline{x_j} + \Theta^{EDGE_{LEFT}} \cdot v(t-1, j-1, \alpha) + \Theta^{EDGE_{RIGHT}} \cdot v(t-1, j+1, \alpha) + \\ + \Theta^{EDGE_{ALIGN}} \sum_{j':(j,j') \in WA_{\alpha,\neg\alpha}} v(t-1, j', \neg\alpha)) \tag{5.5}$$

The combined feature vector is computed by summing or averaging only on the target-side vertex state vectors, since these are the ones which represent most of the relevant information: $\tilde{v} = \sum_{j=1}^{L_e+1} v(\infty, j, e)$ or $\tilde{v} = \frac{1}{L_e+1} \sum_{j=1}^{L_e+1} v(\infty, j, e)$.

In the reservoir approach we randomize the parameter matrix while enforcing the echo state condition on the norm.

Note that we have a problem here: in principle, each word on one side of the pair could be aligned with any arbitrary number of words on the other side. This means that the degree of the resulting graph is bounded by the sentence pair lengths, not by a constant hyperparameter as we would like. We can address this issue heuristically by just setting a maximum number of alignment edges per word d and drop supernumerary edges on the basis of their displacement distance: alignments between words at similar positions in the source and target sentences are preferred to those between far words. This yields the ESN condition:

$$\max_{\psi \in LEFT,RIGHT,ALIGN} \left| \Theta^{EDGE_\psi} \right| < \frac{1}{2+d} \tag{5.6}$$

Alternatively, instead of pruning the word alignment edges, we could average their contributions by modifying eq. 5.5:

$$\begin{aligned} v(t,j,\alpha) = \pi^{(1)}(\Theta^{(1)} \cdot \overline{x_j} + \Theta^{EDGE_{LEFT}} \cdot v(t-1,j-1,\alpha) + \Theta^{EDGE_{RIGHT}} \cdot v(t-1,j+1,\alpha) + \\ + \Theta^{EDGE_{ALIGN}} \cdot \frac{d}{|\{j' : (j,j') \in WA_{\alpha,\neg\alpha}\}|} \sum_{j':(j,j') \in WA_{\alpha,\neg\alpha}} v(t-1,j',\neg\alpha)) \end{aligned} \tag{5.7}$$

in this case d becomes a hyperparameter which controls a tradeoff between the influence of alignment edges and word adjacency edges.

This model, which we define as *Bilingual GraphESN language model* (Bilingual GESN-LM) bears some similarity with the bilingual RNN language model of Auli et al. (Auli et al., 2013), which extends a target-side Mikolov RNN language model with source-side features. Their model represents the source sentence either as a bag-of-words, discarding the position between its words, or, for each target word, as a set of fixed-size context windows around the respective aligned source words. Our model, takes into account an unbounded context, for both source and target words, while retaining positional information.

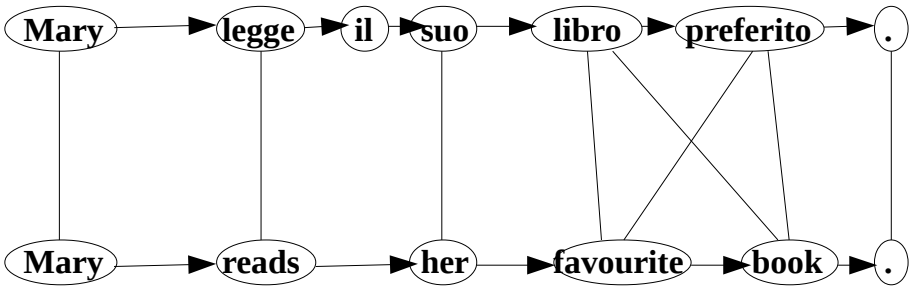


Figure 5.2: A graph neural network for sentence pair ranking. In addition to adjacency edges, there are other, undirected, edges of a different type, representing the bidirectional word alignment relations produced by the decoder.

5.1.3 Tree-to-string dependency bilingual graph language model

For a sentence pair (f, e), in addition to the word alignments $WA_{f,e}$ we may also have access to a *dependency parse graph* g_f of the source sentence: a tree (or forest) over the words of sentence f, whose edges represent syntactic dependency relations. For the most common languages, obtaining dependency parse graphs can be quite easy, as very accurate and fast parsers are available.

Incorporating source-side syntactic information in a reranking model can be useful, especially if the upstream translation system was a phrase-based decoder which did not make use of such information. Syntactic information can help to resolve complex long-distance reordering and concordance issues, which can't be captured by a fixed-size context language model and may be difficult even for a RNN or GraphESN language model due to the echo state property.

We can use include this syntactic information in our model as additional edges of a new type: For each pair of source words $(f_j, f_{j'})$ if f_j is the parent of $f_{j'}$ in g_f, we add a set of feedback connections from the reservoir of vertex f_j to the reservoir of vertex $f_{j'}$ parametrized by matrix $\Theta^{EDGE_{PARENT}}$. Since dependency edges are directed, we also need to add connections in the opposite way, from $f_{j'}$ to f_j, parametrized by another matrix $\Theta^{EDGE_{CHILD}}$.

Note that while each word has at most one parent, it may have in principle

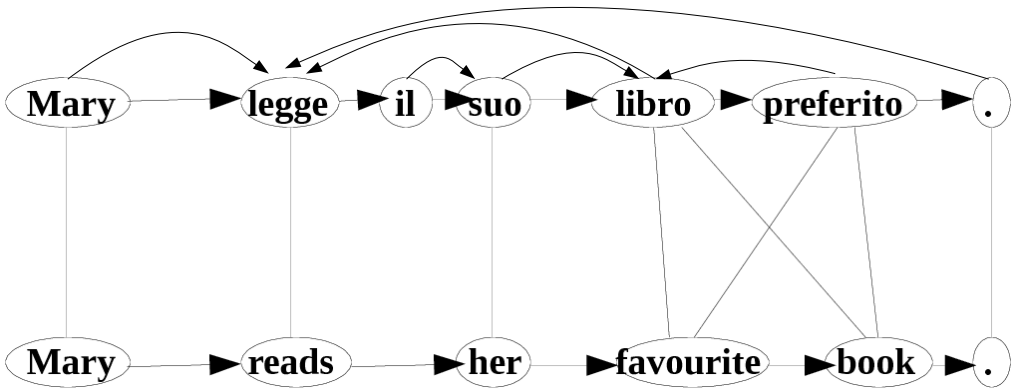


Figure 5.3: A graph neural network for sentence pair ranking with source-side dependency syntax. In addition to adjacency edges and word alignment edges, there are directed edges of yet another type, representing the syntactic dependency relations between the source words, produced by a stand-alone parser.

an arbitrary number of children. As in the case of alignment edges, we may prune these dependency edges at some maximum degree d_{DEP}, based on distance between parent and child, or we may average their contributions.
This yields the following ESP condition:

$$\max_{\psi \in LEFT,RIGHT,ALIGN,PARENT,CHILD} \left| \Theta^{EDGE_\psi} \right| < \frac{1}{3 + d_{ALIGN} + d_{DEP}} \tag{5.8}$$

This condition may be too much constraining for the target-side vertices, which do not have dependency edges. Thus, we may have separate versions of the word adjacency and alignment matrices for the source and target sides of the graph, splitting the ESN constraint in two inequalities:

$$\begin{aligned} \max_{\psi \in LEFT,RIGHT,ALIGN,PARENT,CHILD} \left| \Theta^{EDGE_{\psi,f}} \right| &< \frac{1}{3 + d_{ALIGN} + d_{DEP}} \\ \max_{\psi \in LEFT,RIGHT,ALIGN} \left| \Theta^{EDGE_{\psi,e}} \right| &< \frac{1}{2 + d_{ALIGN}} \end{aligned} \tag{5.9}$$

State evolution is performed until convergence according to equations analogue to (eq. 5.5 or eq. 5.7), then per-target-vertex state vectors are combined in a feature vector $\tilde{v}$ which is feed to the readout layer.
We define this model as *Dependency-Bilingual GraphESN language model* (DB-GESN-LM).

5.1.4 Experiments

Setup We performed experiments using the different versions of the graph-based language model (Sequence GESN-LM, Bilingual GESN-LM and Dependency-Bilingual GESN-LM) described above.

We set up an Italian-to-English phrase-based translation system using Moses (Koehn et al., 2007).

Due to time constraints, we didn't experiment on the German-to-English language pair. Our choice to use the Italian-to-English language pair instead of German-to-English was mainly driven by the consideration that since reranking operates downstream a phrase-based decoder, the long-distance distortions of German-to-English would be bottlenecked by the limited reordering distance of the decoder, and therefore the reranker would have to operate on candidate translation of limited quality. The Italian-to-English language pair has less long-distance distortions, presumably enabling more high quality candidate hypotheses to reach the n-best list that the reranker operates upon.

We trained the baseline phrase-based system using a parallel corpus assembled from Europarl v7 (Koehn, 2005), JRC-ACQUIS v2.2 (Steinberger et al., 2006) and additional bilingual articles crawled from online newspaper websites [3], totaling 3,081,700 sentence pairs, which were split into a 3,075,777 sp. phrase-table training corpus, a 3,923 sp. tuning corpus, and a 2,000 sp. test corpus.

We computed word alignments using Giza++ (Och and Ney, 2003) in standard configuration, and a KenLM 5-gram language model (Heafield, 2011) for English.

We trained and tuned Moses using two configurations: the default configuration and the configuration with sparse features (the "word translation" and "phrase translation" feature sets described in (Chiang et al., 2009)). For both configurations we performed model parameter tuning using k-best batch MIRA (Cherry and Foster, 2012).

This setup is similar to two of the configurations used in the WMT2013 state-

[3] Corriere.it and Asianews.it

of-the-art Edinburgh system (Durrani et al., 2013) and therefore constitutes a strong baseline.

For the Sequence GESN-LM language model, we used English word embeddings computed with an MLP language model (Collobert and Weston, 2008). Specifically, we used the 25-dimensional word embedding computed by Turian et al. (Turian et al., 2010). For the Bilingual GESN-LM and DB-GESN-LM we additionally used Italian word embeddings trained in the same way from a dump of the Italian Wikipedia.
The input-to-hidden matrices $\Theta^{(1)}$ was randomly initialized with values in the range $[-0.01, 0.01]$ and the feedback matrices Θ^{EDGE_ψ} were randomly initialized with a contractivity hyperparameter of 0.99. We used 30 units in each per-vertex reservoir. For the Bilingual GESN-LM and DB-GESN-LM, the echo state property on the feedback matrices were enforced by per-vertex averaging of the contributions of different edges of the same type.

Each GESN-LM reranker was trained on the 3,923 sentence pairs corpus used to tune the baseline model: we perform a 1000-best decoding of this corpus just as we were doing an additional round of tuning. For each source sentence $f^{(i)}$ in this corpus, and for each translation $e^{(i,k)}$ in the 1000-best list, we apply the reservoir of the GraphESN to the translation until the state converges, we combine the per-vertex state vectors by summing them and concatenate the resulting feature vector $\tilde{w}^{(i,k)}$ to the feature vector $\phi^{(i,k)}$ produced by the decoder[4]. Once we have computed these extended feature vectors for all the sentence pairs, we run another round of k-best batch MIRA to compute a new vectors of parameters over both these extended features.

In order to translate a new sentence, we first 1000-best decode it with the original decoder, compute for each translation the feature vector $\tilde{w}$, concatenate it with the decoder feature vector ϕ, and compute the dot product with the new parameter vector. The translation with the highest resulting score, is the 1-best output of the reranker system.

Significance was assessed using *paired bootstrap resampling* (Koehn, 2004b).

[4]there is no separate GraphESN readout layer

Configuration	BLEU-c	BLEU
Moses	28.77	29.58
Moses + sparse feats.	29.02	29.82
Moses + Sequence GESN-LM	29.14 (+0.37)	29.96 (+0.38)
Moses + sparse feats. + Sequence GESN-LM	29.19 (+ 0.17)	30.00 (+ 0.18)
Moses + Bilingual GESN-LM	29.28 (+0.51)	30.10 (+0.52)
Moses + sparse feats. + Bilingual GESN-LM	29.32 (+ 0.30)	30.12 (+ 0.30)
Moses + DB-GESN-LM	29.42 (+0.65)	30.46 (+0.64)
Moses + sparse feats. + DB-GESN-LM	29.41 (+ 0.39)	30.23 (+ 0.41)

Figure 5.4: Experimental results. BLEU and case-insensitive BLEU scores over a 2,000 sp. it-en test corpus. All improvements are significant at p-value < 0.01 except the Moses + sparse feats. + Sequence GESN-LM which are however significant at p-value < 0.05 level.

Results The results of these experiments are shown in fig. 5.4.
We obtain consistent improvements in BLEU scores (Papineni et al., 2002). The improvements are smaller in the configuration with sparse features: this may be due to the small size of the corpus used to train the GESN-LM language models, which may cause some overfitting.
Note however, that even if we used a small training (tuning) set and a small model capacity (25-dimensional word embeddings, 30 hidden units per word), we still obtain significant improvements [5].

From a performance point of view, the system is very fast: in our implementation written in Python-Theano (Bergstra et al., 2010), the overhead of the reranker is about 0.5 seconds per sentence during translation and 1.0 seconds per sentence during training, on a (24-core) [6] Intel(R) Xeon(R) CPU X5675 @ 3.07GHz machine with no GPU.

[5] Retraining under a different random initialization of the reservoir parameters yields a difference in BLEU scores in the order of 0.01 percent points.

[6] although the machine has 24 cores, CPU usage doesn't exceed 200%. We don't parallelize over sentences, although this is doable in principle.

5.1.5 Future work

Due to the fact that the GraphESN-LM formalism is very flexible, it is easy to conceive further variants of the models described above, incorporating almost any possible kind of additional information which may be available.
For instance, if reliable target-side dependency syntax is available, as in the case of using a string-to-tree or tree-to-tree dependency decoder as the upstream translations system, we can include this information as additional edges in the GraphESN-LM, obtaining a string-to-tree or a tree-to-tree neural language model. This can be considered a combination between neural networks and the syntactical language models described in section 2.4.3, which used generative or discriminative log-linear models, often requiring substantial feature engineering.

Adaptation of the reservoir parameters could be performed using noise-contrastive estimation, extending the approach used by Mnih and Teh (Mnih and Teh, 2012) for simplified MLP language models: given a monolingual corpus of sentences $x^{(i)}$ we build a supervised corpus where all the original sentences are included as positive examples, and negative examples are generated by sampling from a noise distribution: a simpler n-gram or MLP language model. Training this model using an appropriate, cross-entropy-bases loss, yields in the limits of a large training set, an estimator of the language model probability $P(x)$.

5.2 Reranking using source phrase dependency features

In addition to the GraphESNLM reranking model, we experimented with a linear reranking model based on manually engineered phrase-level source dependency features.
Dependency features have been used in the past for both direct translation and reranking, usually in a string-to-tree or a tree-to-tree configuration. These approaches generally require the decoder to be specifically designed to

produce suitable dependency structures on its output, or to use a specialized target-side parser capable of parsing potentially ungrammatical and unidiomatic sentences (such as the translation lattice phrase dependency parser of Gimpel and Smith (2013)).

In our work we investigated a tree-to-string N-best reranking model suitable for use with a standard phrase-based decoder (such as Moses) and a standard source-side dependency parser (such as DeSR (Attardi, 2006)). Using only off-the-shelf tools makes this system interesting for language pairs with limited target-side resources.

Source phrase dependency model

Dependency relations in a conventional dependency tree are syntactical relations between individual words. A phrase-based decoder, instead, operates in terms of phrase-pairs.

Each N-best candidate translation e_i of a source sentence f is defined by its derivation, which describes how f has been segmented into source phrases, how these source phrases have been reorederd and for each source phrase which corresponding target phrase has been chosen.
In our system we want to rerank the candidate translations according to a model of the quality of their derivations[7].
Specifically, we focus on the quality of phrase segmentation and reordering.

Segmentation features The source phrases produced by the segmentation performed by the decoder do not necessarily correspond to subtrees in the dependency parse tree (or forest) g_f of the sentence. And if the dependency parse is not projective, subtrees or even more weakly constrained structures (such as the "well-formed" dependency structures of Shen et al. (2008, 2010)) do not necessarily correspond to contiguous phrases in any possible segmentation.
There is a mismatch between the non-syntactic phrases the decoder operates

[7] A candidate target sentence may appear multiple times in the N-best list, each time with a different derivation. We consider these different candidate translations for our purposes.

upon and dependency structures defined in terms of their topological properties in the dependency graph.

This mismatch, however, is not necessarily detrimental to translation quality. In fact, restricting the decoder to operate only on syntactically reasonable phrases actually reduces the translation quality, suggesting that automatically segmented phrases based on a phrase table heuristically derived from a word-aligned parallel corpus and syntactically segmented phrases based on parse graph properties carry different kinds of information. It can be therefore beneficial to combine these different kinds of information using features in a machine learned representation, specifically a linear model with parameters tuned using standard techniques.

Cherry (2008) used global *cohesion* features which count the number of *interruptions* in the derivation of the translations, defined as the number of times the decoder chose a source phrase that was not contiguous in the source dependency tree up to a reordering between sibling vertices. They used their model directly in the decoder obtaining a significant increase of translation quality (BLUE score).

We propose a set of multiple features which operate strictly at source phrase level, inspired by the concept of *phrase dependency* relations of Gimpel and Smith (2013), but generalized to suit our purposes:
Given a source phrase $\bar{f}_j$ in a derivation, we define the set of its parent phrases $PARENTS(\bar{f}_j)$ as the set of other phrases in the same derivation which contain at least one word that is a parent of some word in $\bar{f}_j$:

$$\begin{aligned} PARENTS(\bar{f}_j) &\equiv \left\{ \bar{f}'_j | j' \neq j \wedge \exists k, k' : \bar{f}_{j',k'} = PARENT(\bar{f}_{j,k}) \right\} \\ CHILDREN(\bar{f}_j) &\equiv \left\{ \bar{f}'_j | \bar{f}_j \in PARENTS(\bar{f}'_j) \right\} \end{aligned} \tag{5.10}$$

where $\bar{f}_{j,k}$ is the k-th word of the j-th source phrase.
We also define the sets of left parents $PARENTS_L(\bar{f}_j)$, right parents $PARENTS_R(\bar{f}_j)$, left children $CHILDREN_L(\bar{f}_j)$ and right children $CHILDREN_R(\bar{f}_j)$. Note that only word dependency relations that cross the phrase boundaries are relevant to the definition of these phrase dependency relations.

We propose the following segmentation phrase feature functions:

- No parents $PARENTS(\bar{f}_j) = \emptyset$, no left parents $PARENTS_L(\bar{f}_j) = \emptyset$, no right parents $PARENTS_R(\bar{f}_j) = \emptyset$, one-sided parents $PARENTS_L(\bar{f}_j) = \emptyset \vee PARENTS_R(\bar{f}_j) = \emptyset$.
- Unambiguous (no more than one) parents $|PARENTS(\bar{f}_j)| \leq 1$, Unambiguous left parents $|PARENTS_L(\bar{f}_j)| \leq 1$, Unambiguous right parents $|PARENTS_L(\bar{f}_j)| \leq 1$.
- Unique parent $|PARENTS(\bar{f}_j)| = 1$.
- No children $CHILDREN(\bar{f}_j) = \emptyset$, no left children $CHILDREN_L(\bar{f}_j) = \emptyset$, no right children $CHILDREN_R(\bar{f}_j) = \emptyset$, one-sided children $CHILDREN_L(\bar{f}_j) = \emptyset \vee CHILDREN_R(\bar{f}_j) = \emptyset$.

The idea is that when phrase segmentation breaks the syntactic structures these features should be able to detect it, and the model will penalize (or possibly, reward) different types of breakages using parameters automatically learned during tuning.

Distortion features In addition to segmentation scoring features, we propose another set of features, still based on phrase dependency relations, which model the reordering between the source phrases of the derivation.

Specifically, we consider couples of source phrases which are aligned [8] to target phrases which are contiguous in target order.
Let $\tilde{f}_j \equiv (\bar{f}_{a(j-1)}, \bar{f}_{a(j)})$ be one of such couples. We define the following, mutually exclusive, feature functions:

- Unique parent-child $PARENTS(\bar{f}_{a(j)}) = \{\bar{f}_{a(j-1)}\}$.
- Unique child-parent $PARENTS(\bar{f}_{a(j-1)}) = \{\bar{f}_{a(j)}\}$.
- Siblings with unique parent $\exists j' : PARENTS(\bar{f}_{a(j)}) = PARENTS(\bar{f}_{a(j-1)}) = \bar{f}_{j'}$
- None of the above.

[8] phrase alignments returned by the decoder are always one-to-one

furthermore we define the inversion feature function $a(j-1) > a(j)$ which is included both as an individual feature and in logical conjunction with each of the feature functions defined above, resulting in a total of nine boolean distortion feature functions.

The rationale of these features is that they detect reordering operations which swap syntactic structures related by a dependency relation between themselves or with a shared parent structure.
This is similar to the kind reordering operations between structures allowed in the *synchronous dependency insertion grammar* model of Ding and Palmer (2005) or the aforementioned *well-formed structures* of Shen et al. (2010), but these approaches require target dependencies to be available and strictly constrain the reordering operation that can be performed, while our model only requires source dependencies and generates "soft" constraints whose strength is regulated by the automatically tuned parameters.

Scoring model The feature functions defined in the two previous paragraphs can be combined in a vector feature function $g(f, e, j)$ which computes the segmentation and distortion features of source phrase $\bar{f}_{a(j)}$ under derivation e (the distortion features of phrase $\bar{f}_{a(1)}$ are all defined equal to zero since this phrase has no previous phrase in target order).

We sum the values of these features phrase-wise to obtain a derivation feature vector:

$$\tilde{w}(f, e) \equiv \sum_{j=1}^{L_x(e)} g(f, e, j) \tag{5.11}$$

where $L_x(e)$ is the number of phrases in the segmentation of derivation e.

In order to perform reranking, this vector is then concatenated with the feature vector produced by the decoder $\phi(f, e)$ and multiplied by the parameter vector θ to obtain the final reranking score:

$$h(f, e) \equiv \theta^T \cdot (\tilde{w}(f, e), \phi(f, e)) \tag{5.12}$$

where the parameter vector θ is trained using a standard machine translation tuning technique such as k-best batch MIRA (Cherry and Foster, 2012).

Configuration	BLEU-c	BLEU
Moses + sparse feats.	29.02	29.82
Moses + sparse feats. + dep. feats.	29.17 (+ 0.15)	29.97 (+ 0.15)

Figure 5.5: Experimental results. BLEU and case-insensitive BLEU scores over a 2,000 sp. it-en test corpus. Improvements are significant at the p-value < 0.05 .

Experiments

Setup We tested our model in a Italian-to-English 1000-best translation reranking task.
The experimental setup is the same used to test the GraphESNLM models (section 5.1.4), although we only used the Moses configuration with sparse features.
Non-projective dependency parse trees (actually, forests) for the Italian source sentences have been computed using the transition-based DeSR parser (Attardi, 2006) in the state-of-the-art tree revision configuration (Attardi and Ciaramita, 2007).

Results The results of these experiments are shown in fig. 5.5.
We obtain a small but significant BLUE score improvement, similar to the improvement obtained by the Sequence GraphESNLM approach, which is based on completely different principles.
We also performed other experiments with slightly different feature function configurations but we obtained lower scores, although never lower than the baseline score of the decoder.

From a speed point of view, the reranker adds a negligible overhead the the runtime of the decoder, even in our unoptimized Python implementation.

5.2.1 Future work

Just like any discriminative system based on hand-coded feature functions, feature engineering is crucial. Different configurations of the feature functions could be tried.

In particular, in our model we did not use dependency type (deprel) tags, POS tags and morphological information even though they were available. It may be interesting to investigate features that combine this type of information with the purely topological relations that we used.
These combined features, however, would be sparse, and when concatenated with the already sparse decoder feature could generate a model with a large number of parameters which would be at risk of overfitting without a sufficiently large training (tuning) corpus. Due to time constraints, we used a small corpus of 3,923 sentence pairs, which would have been unsuitable for training a model with a very large number of parameters.

It can be observed that our model decomposes in terms of couples of target-consecutive phrases. This means that it could be used directly for decoding rather than reranking, by integrating in a phrase-based translation system without increasing the size of the state signatures (compared to the state signatures used by the lexicalized distortion model which is integrated in the standard configuration of Moses).
Using this model for decoding rather than reranking could possibly further increase translation quality (by decreasing the search error caused by pruning) and reduce the runtime (by eliminating the need to generate the 1000-best list).

Finally, this model could be combined with the GraphESNLM models to exploit the benefits of both human-engineered features and non-linear pseudo-random reservoir features. Hopefully, these features would be largely uncorrelated (specifically in the case of the simplest target-only GraphESNLM), giving to the final linear model a better ability to separate good translations from bad ones.

Chapter 6

Conclusions and future work

In this thesis we investigated the problem of statistical machine translation from various angles. We experimented with pre- and post-processing techniques for word reordering and candidate translation reranking, respectively.

We focused our research on syntax-based approaches which make use of *non-projective dependency grammars*, which are believed to be well suited to represent the syntactic structure of non-analytic languages such as German, Italian, Czech or Bulgarian.
We also made significant use of a variety of *discriminative machine learning* techniques, in order to leverage the success that these approaches had in recent years and develop flexible systems capable of being deployed for multiple language pairs without the need of significant language-specific engineering.

6.1 Characterization of German-to-English reordering as transitions on a dependency tree

We investigated preprocessing schemes that permute the words of the source sentences attempting to put them into an order which is idiomatic for the target language.
These approaches aim at facilitating the job of the main translation engine

by reliving it of the hard combinatorial problem of long-distance reordering, which is difficult for a phrase-based decoder, allowing it to focus its computational resources on target word selection.

We introduced an automaton model which enables us to describe the generation of arbitrary permutations of dependency-parsed source sentences in terms of walks on the dependency tree.
This model allows us to statistically characterize these permutations in terms of properties of the executions of the automaton that generates them.

We hypothesized that for certain language pairs such as German-to-English, where the syntax of the source language is naturally modeled using non-projective dependency relations and where word alignments often occur at long distance, an explicit modeling of non-projectivity and tree non-local reordering can benefit syntax-based reordering and therefore machine translation.

We statistically analyzed the occurrence of these linguistic phenomena in the German-to-English language pair, how they correlate with themselves, with phrase-based translation quality and with the "pseudo-oracle performance gap", a heuristic quasi-upper bound to the improvement of translation quality that be achieved by performing pre-reordering.

We also specifically analyzed the impact of non-projective relations using some dependency-based variations of hand-coded pre-reordering rules.

We concluded that designing a syntax-based system which can process non-tree-local reordering and dependency non-projectivity is likely to be a promising approach.

6.2 Pre-reordering for machine translation using transition-based walks on dependency parse trees

We proposed two classes of novel syntax-based reordering approaches which can process non-projective dependency trees and generate non-tree-local permutations.

The first approach is based on a direct instantiation of the tree walker automaton model introduced in the previous chapter.
We implemented a greedy classifier-driven transition-based reorderer, inspired by the success of greedy classifier-driven transition-based dependency parsers such as DeSR.
We found that the accuracy of our system is not sufficient to improve the full translation quality over the baseline phrase-based Moses decoder.
We attributed this failure to reordering being an inherently harder combinatorial problem than parsing, making greedy approaches unsuitable. While more advanced search techniques might succeed where the greedy failed, the evidence that we observed from an error analysis suggest us that the optimization problems posed by this approach might be intrinsically quite difficult.

The second approach is a novel class of recurrent neural network architectures similar to recurrent neural language models. These models can incrementally score, and therefore generate, arbitrary permutation of source sentences based on generally non-projective dependency features.
We applied these models to German-to-English pre-reordering and subsequent phrase-based translation, obtaining significant performance improvements over the baseline Moses decoder.
We were able to obtain performance improvements comparable to, or in some cases even better than those of the best hand-coded linguistically-informed pre-reordering rules.
Moreover, we verified that these gains correlate with non-projectivity and tree non-locality. These results, building upon those of the previous chapter, confirm our overarching research hypothesis that these neglected linguistic phenomena are relevant to translation quality, at lest for the German-to-English language pair.

6.3 Translation reranking using source-side dependency syntax and graph echo state networks

The main decoder of a translation system has to tradeoff accuracy of its scoring model for efficiency of its search process. For this reason, it is often possible to improve the overall translation quality by making the decoder generate a candidate list of candidate translations of a sentence that score high according to its coarse model, and then reranking them with a more accurate downstream scoring model.

In this chapter we wanted to investigate the viability of reranking based on the topological features of the source-side non-projective dependency parse graph.

We proposed a reranking model based on *phrase dependency* features on source-parsed sentence pairs: a source sentence is dependency-parsed, translated with a phrase-based decoder to a list of 1000 candidate translations. The source sentence and each candidate translation form a sentence pair that we score with our model.

We compute this score using features that take into account how the phrase segmentation operated by the decoder cuts the dependency parse of the source sentence and how these phrases are reordered in the translation w.r.t. their dependency relations, an approach similar to Gimpel and Smith (2013). The model also includes standard phrase-based features directly derived from the decoder and is trained using batch MIRA.

We obtained a significant quality improvement over the baseline decoder on the Italian-to-English language pair.

We also proposed a class of reranking models based on *graph echo state networks* (GraphESN), a type of *reservoir computing neural network* for prediction problems with a graph input (Gallicchio and Micheli, 2010).

Each candidate sentence pair is mapped to a graph which is applied to a GraphESN model, obtaining a per-vertex state vector, which is reduced to a fixed-dimensional per-sentence-pair vector, which is then used as a feature

vector, together with standard phrase-based features, for a linear scoring model, trained with batch MIRA.

The specifics of these *graph echo state networks language models* (GESN-LMs) depend on the way the graph is obtained from a sentence pair.
We proposed a *monolingual* approach which only considers target words as a sequence graph, a *bilingual* model which also includes the sequence of source words with source-target word-alignment edges, and finally a *tree-to-string* bilingual model which additionally includes edges representing dependency relations on the source sentence, obtained from a parser. We tested these obtaining a significant quality improvement over the baseline decoder on the Italian-to-English language pair, specifically the more complex models which take into account bilingual and syntactic relations yield better performances.

GraphESNs, being a reservoir computing approach, use untrained, randomly initialized recurrent neural networks. This greatly reduces training complexity compared to conventional recurrent neural networks, and in particular allowed us to keep the adaptable part of our scoring model linear, which enabled training with batch MIRA.
However, it may be possible that adapting the recurrent connections parameters yields a significant accuracy improvement. Devising a suitable training scheme for such model would be an interesting line of research.

We consider the success of our approach as a confirmation of our hypothesis that reranking based on non-projective topological features of the source sentences is viable.

6.4 Final considerations

The reranking that we introduced models yielded good results in terms of translation quality with a small computational cost. They are also the methods that required less engineering efforts, and are arguably the most innovative techniques used in this thesis.
Therefore, we believe that in the short-term these methods, specifically the

GESN-LMs, are the most promising line of research stemming from this thesis, in terms of expected return on effort.

In the medium or long-term, we expect the quality improvements of *n-best-list* reranking approaches (in general, not just those derived from our research) to level off, because these methods are fundamentally limited by the quality of the translations that the upstream decoder is able to place into the candidate translations list.

Word-lattice reraking schemes may remain viable for a longer time, as they rerank over exponentially larger parts of the translation space, however some empirical studies failed to notice a significant difference with n-best-list reranking in terms of translation quality (Auli et al., 2013).

Our reordering methods, specifically the one based on recurrent neural networks, are also very effective and innovative. Specifically, Fragment-RNN is able to fully exploit the dependency reordering automaton model although at a considerable computational cost. Base-GRU is simpler but performs better and is more efficient, and may be probably be considered mature enough for practical applications. Further development of these models appears to be a promising short-term/medium-term goal.

Eventually, performance of statistical reordering approaches is likely to become limited by the fact that they are trained on heuristic reference permutations, which, while reasonable, are not really theoretically principled, and the fact that reordering does not help with some forms of long-distance dependencies, such as grammatical agreement or split translations, that phrase-based downstream decoders have troubles dealing with.

Regarding the main translation engine, we believe that designing substantially new algorithms for decoding might be probably not be worth the effort at this point, at least unless significantly novel discrete optimization ideas are imported into the field currently dominated by beam search dynamic programming phrase-based or parser-derived algorithms.

However, research on the scoring models used for decoding seems an highly promising research area for the medium and long-term.

Including new features to translation model requires careful engineering due to the model quality vs. hypothesis recombination opportunities tradeoffs.

Therefore, reordering and reraking approaches can be considered as "test beds" for new features to include in translation models: if a feature performs well in a reordering or reranking approach it may be worth to try to adapt it for a use in the main decoder, thus restricting the tradeoff engineering effort to the most promising features.

More specifically, our recurrent neural network reordering models don't appear to be easily integrable in a standard phrase-based decoder due to the fact that the state of recurrent neural networks doesn't decompose over sequences. However, in the future we may investigate other neural network architectures that replace state recurrence with fixed-size context windows, which could be used in a phrase-based decoder without disrupting hypothesis recombination. These models also have the advantage of being less computationally expensive, both during training and during decoding.
We haven't investigated these approaches yet because we were more interesting in addressing the research question of whether dependency non-projectivity and non-tree-local reordering were relevant to an actual statistical syntax-based system, hence we went straight to recurrent neural networks because they are known to deliver state of the art performance on sequence prediction problems. Now that the question is arguably settled, the investigation of context-window approaches can be certain a promising approach.
In a similar vein, echo state networks and GraphESN could be also plausible research approaches to be investigated in this aspect.

In conclusion, in this thesis we presented a variety of statistical machine translation methods focused on exploiting syntactical information represented as source-side non-projective dependency parse trees, and making use of several discriminative machine learning techniques. We believe that these approaches may have the potential to be expanded into future lines of research and to be applied to practical machine translation systems.

Bibliography

Aizerman, A., Braverman, E. M., and Rozoner, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, 25:821–837.

Al-Onaizan, Y. and Papineni, K. (2006). Distortion models for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, ACL-44, pages 529–536, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ambati, V. (2008). Dependency structure trees in syntax based machine translation. Adv. MT Seminar Course Report.

Attardi, G. (2006). Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 166–170, Stroudsburg, PA, USA. Association for Computational Linguistics.

Attardi, G., Chanev, A., and Miceli Barone, A. V. (2011). A dependency based statistical translation model. In *Proceedings of the Fifth Workshop on Syntax, Semantics and Structure in Statistical Translation*, SSST-5, pages 79–87, Stroudsburg, PA, USA. Association for Computational Linguistics.

Attardi, G. and Ciaramita, M. (2007). Tree revision learning for dependency parsing. In Sidner, C. L., Schultz, T., Stone, M., and Zhai, C., editors, *HLT-NAACL*, pages 388–395. The Association for Computational Linguistics.

Auli, M., Galley, M., Quirk, C., and Zweig, G. (2013). Joint language and translation modeling with recurrent neural networks. EMNLP.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition.

Bengio, Y., Schwenk, H., Senécal, J.-S., Morin, F., and Gauvain, J.-L. (2006). Neural probabilistic language models. In *Innovations in Machine Learning*, pages 137–186. Springer.

Bengio, Y. and Senecal, J.-S. (2008). Adaptive importance sampling to accelerate training of a neural probabilistic language model. *Neural Networks, IEEE Transactions on*, 19(4):713–722.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

Birch, A., Osborne, M., and Koehn, P. (2008). Predicting success in machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 745–754, Stroudsburg, PA, USA. Association for Computational Linguistics.

Bisazza, A. (2013). *Linguistically Motivated Reordering Modeling for Phrase-Based Statistical Machine Translation*. PhD thesis, University of Trento.

Bisazza, A. and Federico, M. (2013). Efficient solutions for word reordering in German-English phrase-based statistical machine translation. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 440–451, Sofia, Bulgaria. Association for Computational Linguistics.

Blum, A. L. and Rivest, R. L. (1992). Training a 3-node neural network is np-complete. *Neural Networks*, 5(1):117–127.

bo Duan, K. and Keerthi, S. S. (2005). Which is the best multiclass svm method? an empirical study. In *Proceedings of the Sixth International Workshop on Multiple Classifier Systems*, pages 278–285.

Bohnet, B. (2010). Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 89–97. Association for Computational Linguistics.

Bosco, C. and Lombardo, V. (2004). Dependency and relational structure in treebank annotation. In *COLING 2004 Recent Advances in Dependency Grammar*, pages 1–8.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM.

Box, G. E. P. and Tiao, G. C. (1992). *Bayesian Inference in Statistical Analysis (Wiley Classics Library)*. Wiley-Interscience, reprint edition.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Brown, P. F., Cocke, J., Pietra, S. A. D., Pietra, V. J. D., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Roossin, P. S. (1990). A statistical approach to machine translation. *Comput. Linguist.*, 16(2):79–85.

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Comput. Linguist.*, 19(2):263–311.

Bryson, A. and Ho, Y.-C. (1969). Applied optimal control. *Blaisdell, Waltham, Mass*, 8.

Carter, S. and Monz, C. (2011). Syntactic discriminative language model rerankers for statistical machine translation. *Machine translation*, 25(4):317–339.

Chanev, A. (2005). Portability of dependency parsing algorithms–an application for italian. In *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories (TLT)*, pages 29–40.

Chang, C.-C. and Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27.

Chang, P.-C., Tseng, H., Jurafsky, D., and Manning, C. D. (2009). Discriminative reordering with chinese grammatical relations features. In *Proceedings of the Third Workshop on Syntax and Structure in Statistical Translation*, pages 51–59. Association for Computational Linguistics.

Charniak, E. (1997a). Statistical parsing with a context-free grammar and word statistics. *AAAI/IAAI*, 2005:598–603.

Charniak, E. (1997b). Statistical techniques for natural language parsing. *AI magazine*, 18(4):33.

Charniak, E. (2001). Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 124–131. Association for Computational Linguistics.

Charniak, E., Knight, K., and Yamada, K. (2003). Syntax-based language models for statistical machine translation. In *Proceedings of MT Summit IX*, pages 40–46. Citeseer.

Chau, R., Tsoi, A. C., Hagenbuchner, M., and Lee, V. (2009). A conceptlink graph for text structure mining. In *Proceedings of the Thirty-Second Australasian Conference on Computer Science-Volume 91*, pages 141–150. Australian Computer Society, Inc.

Chen, S. F. and Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393.

Cherry, C. (2008). Cohesive phrase-based decoding for statistical machine translation. In *In Proceedings of ACL-08: HLT*, pages 72–80.

Cherry, C. and Foster, G. (2012). Batch tuning strategies for statistical machine translation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 427–436. Association for Computational Linguistics.

Chiang, D. (2005). A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 263–270, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chiang, D. (2007). Hierarchical phrase-based translation. *computational linguistics*, 33(2):201–228.

Chiang, D., Knight, K., and Wang, W. (2009). 11,001 new features for statistical machine translation. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 218–226. Association for Computational Linguistics.

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

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chomsky, N. (1956). Three models for the description of language. *Information Theory, IRE Transactions on*, 2(3):113–124.

Chomsky, N. (1957). *Syntactic structures*. The Hague.

Cocke, J. (1969). *Programming Languages and Their Compilers: Preliminary Notes*. Courant Institute of Mathematical Sciences, New York University.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Collins, M. (2003). Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637.

Collins, M., Koehn, P., and Kučerová, I. (2005a). Clause restructuring for statistical machine translation. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 531–540. Association for Computational Linguistics.

Collins, M. and Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111. Association for Computational Linguistics.

Collins, M., Roark, B., and Saraclar, M. (2005b). Discriminative syntactic language modeling for speech recognition. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 507–514. Association for Computational Linguistics.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning, ICML*.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Costa-jussà, M. R. and Fonollosa, J. A. R. (2006). Statistical machine reordering. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pages 70–76, Stroudsburg, PA, USA. Association for Computational Linguistics.

Crammer, K. and Singer, Y. (2002). On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292.

Crammer, K. and Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. *J. Mach. Learn. Res.*, 3:951–991.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Damerau, F. J. (1971). *Markov models and linguistic theory: an experimental study of a model for English*. Mouton The Hague,, The Netherlands.

Daumé III, H. and Marcu, D. (2005). Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 169–176. ACM.

Dempster, A. P., Laird, N. M., Rubin, D. B., et al. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal statistical Society*, 39(1):1–38.

DeNeefe, S. and Knight, K. (2009). Synchronous tree adjoining machine translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 727–736. Association for Computational Linguistics.

Ding, Y. and Palmer, M. (2005). Machine translation using probabilistic synchronous dependency insertion grammars. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 541–548, Stroudsburg, PA, USA. Association for Computational Linguistics.

Dryer, M. S. and Haspelmath, M., editors (2013). *WALS Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Durrani, N., Haddow, B., Heafield, K., and Koehn, P. (2013). EdinburghâĂŹs machine translation systems for european language pairs. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 112–119.

Durrani, N., Haddow, B., Koehn, P., and Heafield, K. (2014). EdinburghâĂŹs phrase-based machine translation systems for wmt-14.

Durrani, N., Schmid, H., and Fraser, A. (2011). A joint sequence translation model with integrated reordering. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1045–1054. Association for Computational Linguistics.

Dyer, C. and Resnik, P. (2010). Context-free reordering, finite-state translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 858–866, Stroudsburg, PA, USA. Association for Computational Linguistics.

Earley, J. (1970). An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102.

Egghe, L. (2000). The distribution of n-grams. *Scientometrics*, 47(2):237–252.

Eisner, J. M. (1996). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874.

Feng, M., Mauser, A., and Ney, H. (2010). A source-side decoding sequence model for statistical machine translation. In *Conference of the Association for Machine Translation in the Americas (AMTA)*.

Ferguson, T. S. (1982). An inconsistent maximum likelihood estimate. *Journal of the American Statistical Association*, 77(380):pp. 831–834.

Finley, T. and Joachims, T. (2008). Training structural svms when exact inference is intractable. In *Proceedings of the 25th international conference on Machine learning*, pages 304–311. ACM.

Fox, H. J. (2002). Phrasal cohesion and statistical machine translation. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 304–3111. Association for Computational Linguistics.

Freund, Y. and Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296.

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

Galley, M., Graehl, J., Knight, K., Marcu, D., DeNeefe, S., Wang, W., and Thayer, I. (2006). Scalable inference and training of context-rich syntactic translation models. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 961–968, Sydney, Australia. Association for Computational Linguistics.

Galley, M., Hopkins, M., Knight, K., and Marcu, D. (2004). What's in a translation rule? In *HLT-NAACL*, pages 273–280.

Galley, M. and Manning, C. D. (2008). A simple and effective hierarchical phrase reordering model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 848–856, Stroudsburg, PA, USA. Association for Computational Linguistics.

Gallicchio, C. (2011). *Reservoir Computing for Learning in Structured Domains*. PhD thesis.

Gallicchio, C. and Micheli, A. (2010). Graph echo state networks. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–8. IEEE.

Gallicchio, C. and Micheli, A. (2011a). Architectural and markovian factors of echo state networks. *Neural Networks*, 24(5):440 – 456.

Gallicchio, C. and Micheli, A. (2011b). Exploiting vertices states in graphesn by weighted nearest neighbor. In *ESANN*. Citeseer.

Gallicchio, C. and Micheli, A. (2011c). Supervised state mapping of clustered graphesn states. In Apolloni, B., Bassis, S., Esposito, A., and Morabito, F. C., editors, *WIRN*, volume 234 of *Frontiers in Artificial Intelligence and Applications*, pages 28–35. IOS Press.

Gallicchio, C. and Micheli, A. (2013). Tree echo state networks. *Neurocomputing*, 101:319–337.

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):721–741.

Genzel, D. (2010). Automatically learning source-side reordering rules for large scale machine translation. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 376–384, Stroudsburg, PA, USA. Association for Computational Linguistics.

Gimpel, K. and Smith, N. A. (2013). Phrase dependency machine translation with quasi-synchronous tree-to-tree features.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *International Conference on Artificial Intelligence and Statistics*, pages 297–304.

Hardmeier, C. (2012). Discourse in statistical machine translation. a survey and a case study. *Discours. Revue de linguistique, psycholinguistique et informatique*, (11).

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107.

Hasler, E., Haddow, B., and Koehn, P. (2011). Margin infused relaxed algorithm for moses. *The Prague Bulletin of Mathematical Linguistics*, 96(1):69–78.

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109.

Heafield, K. (2011). Kenlm: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, WMT '11, pages 187–197, Stroudsburg, PA, USA. Association for Computational Linguistics.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Howlett, S. and Dras, M. (2011a). Clause restructuring for SMT not absolutely helpful. In *Proceedings of the 49th Annual Meeting of the Assocation for Computational Linguistics: Human Language Technologies*, pages 384–388.

Howlett, S. and Dras, M. (2011b). Erratum for clause restructuring for smt not absolutely helpful.

Huang, F. (2009). Confidence measure for word alignment. 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 932–940. Association for Computational Linguistics.

Huang, L., Fayong, S., and Guo, Y. (2012). Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL HLT '12, pages 142–151, Stroudsburg, PA, USA. Association for Computational Linguistics.

Huang, L. and Mi, H. (2010). Efficient incremental decoding for tree-to-string translation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 273–283. Association for Computational Linguistics.

Hutchins, W. J. (2004). The georgetown-ibm experiment demonstrated in january 1954. In *Machine Translation: From Real Users to Research*, pages 102–114. Springer.

Hwa, R., Resnik, P., Weinberg, A., Cabezas, C., and Kolak, O. (2005). Bootstrapping parsers via syntactic projection across parallel texts. *Nat. Lang. Eng.*, 11(3):311–325.

Isozaki, H., Sudoh, K., Tsukada, H., and Duh, K. (2010). Head finalization: A simple reordering rule for sov languages. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, WMT '10, pages 244–251, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jaeger, H. (2001). The âĂIJecho stateâĂİ approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34.

Jaeger, H. (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. GMD-Forschungszentrum Informationstechnik.

Jaeger, H. and Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80.

Jaynes, E. T. (2003). *Probability Theory: The Logic of Science*. Cambridge University Press.

Jazwinski, A. (1970). *Stochastic processes and filtering theory*. Number 64 in Mathematics in science and engineering. Acad. Press, New York, NY [u.a.].

Katz-Brown, J. E. (2008). *Dependency reordering features for Japanese-English phrase-based translation*. PhD thesis, Massachusetts Institute of Technology.

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

Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE.

Knight, K. (1999). Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4):607–615.

Koehn, P. (2004a). Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Machine translation: From real users to research*, pages 115–124. Springer.

Koehn, P. (2004b). Statistical significance tests for machine translation evaluation. In *EMNLP*, pages 388–395.

Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *Conference Proceedings: the tenth Machine Translation Summit*, pages 79–86, Phuket, Thailand. AAMT, AAMT.

Koehn, P., Axelrod, A., Mayne, A. B., Callison-Burch, C., Osborne, M., and Talbot, D. (2005). Edinburgh system description for the 2005 iwslt speech translation evaluation. In *International Workshop on Spoken Language Translation*.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Koehn, P. and Knight, K. (2003). Feature-rich statistical translation of noun phrases. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 311–318. Association for Computational Linguistics.

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics.

Lafferty, J., McCallum, A., and Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Lambert, B., Raj, B., and Singh, R. (2013). Discriminatively trained dependency language modeling for conversational speech recognition.

Lewis, II, P. M. and Stearns, R. E. (1968). Syntax-directed transduction. *J. ACM*, 15(3):465–488.

Lin, C.-Y. and Och, F. J. (2004). Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516.

Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45(3):503–528.

liu, l., Watanabe, T., Sumita, E., and Zhao, T. (2013). Additive neural networks for statistical machine translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 791–801, Sofia, Bulgaria. Association for Computational Linguistics.

Liu, Y., Liu, Q., and Lin, S. (2006). Tree-to-string alignment template for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, ACL-44, pages 609–616, Stroudsburg, PA, USA. Association for Computational Linguistics.

Lo, C.-k., Tumuluru, A. K., and Wu, D. (2012). Fully automatic semantic mt evaluation. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, WMT '12, pages 243–252, Stroudsburg, PA, USA. Association for Computational Linguistics.

Marcu, D. and Wong, W. (2002). A phrase-based, joint probability model for statistical machine translation. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 133–139, Stroudsburg, PA, USA. Association for Computational Linguistics.

Markov, A. (1971). Extension of the limit theorems of probability theory to a sum of variables connected in a chain.

McDonald, R., Crammer, K., and Pereira, F. (2005). Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 91–98, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mel'čuk, I. A. (1988). *Dependency syntax: theory and practice*. SUNY Press.

Miceli Barone, A. V. and Attardi, G. (2013). Pre-reordering for machine translation using transition-based walks on dependency parse trees. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 164–169, Sofia, Bulgaria. Association for Computational Linguistics.

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

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010).

Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

Mikolov, T., Yih, W.-t., and Zweig, G. (2013b). Linguistic regularities in continuous space word representations. In *Proceedings of NAACL-HLT*, pages 746–751.

Mikolov, T. and Zweig, G. (2012). Context dependent recurrent neural network language model. In *SLT*, pages 234–239.

Mnih, A. and Teh, Y. W. (2012). A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*.

Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pages 246–252.

Moussouris, J. (1974). Gibbs and markov random systems with constraints. *Journal of statistical physics*, 10(1):11–33.

Muratore, D., Hagenbuchner, M., Scarselli, F., and Tsoi, A. C. (2010). *Sentence extraction by graph neural networks*. Springer.

Navratil, J., Visweswariah, K., and Ramanathan, A. (2012). A comparison of syntactic reordering methods for english-german machine translation. In *COLING*, pages 2043–2058.

Nesson, R., Rush, A., and Shieber, S. (2006). Induction of probabilistic synchronous tree-insertion grammars for machine translation. Association for Machine Translation in the Americas.

Nikoulina, V. and Dymetman, M. (2008). Experiments in discriminating phrase-based translations on the basis of syntactic coupling features. In *Proceedings of the Second Workshop on Syntax and Structure in Statistical Translation*, SSST '08, pages 55–60, Stroudsburg, PA, USA. Association for Computational Linguistics.

Nivre, J. and Scholz, M. (2004). Deterministic dependency parsing of english text. In *Proceedings of the 20th International Conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Och, F. J. (2003). Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 160–167. Association for Computational Linguistics.

Och, F. J., Gildea, D., Khudanpur, S., Sarkar, A., Yamada, K., Fraser, A., Kumar, S., Shen, L., Smith, D., Eng, K., et al. (2004). A smorgasbord of features for statistical machine translation. In *HLT-NAACL*, pages 161–168.

Och, F. J. and Ney, H. (2002). Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 295–302. Association for Computational Linguistics.

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational linguistics*, 29(1):19–51.

Och, F. J. and Ney, H. (2004). The alignment template approach to statistical machine translation. *Computational linguistics*, 30(4):417–449.

Och, F. J., Tillmann, C., Ney, H., et al. (1999). Improved alignment models for statistical machine translation. In *Proc. of the Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 20–28.

Orponen, P. (1994). Computational complexity of neural networks: a survey. *Nordic Journal of Computing*, 1(1):94–110.

Palangi, H., Deng, L., and Ward, R. K. (2013). Learning input and recurrent weight matrices in echo state networks. *arXiv preprint arXiv:1311.2987*.

Papineni, K., Roukos, S., Ward, T., and jing Zhu, W. (2002). Bleu: a method for automatic evaluation of machine translation. pages 311–318.

Pascanu, R., Mikolov, T., and Bengio, Y. (2012). On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*.

Pearl, J. (1985). Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine*, pages 329–334.

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440. Association for Computational Linguistics.

Popel, M. and Mareček, D. (2010). Perplexity of n-gram and dependency language models. In *Text, Speech and Dialogue*, pages 173–180. Springer.

Post, M., Ganitkevitch, J., Orland, L., Weese, J., Cao, Y., and Callison-Burch, C. (2013). Joshua 5.0: Sparser, better, faster, server. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 204–210.

Post, M. and Gildea, D. (2008). Parsers as language models for statistical machine translation. In *Proceedings of the Eighth Conference of the Association for Machine Translation in the Americas*, pages 172–181.

Powell, M. J. D. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1988). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA.

Quirk, C., Menezes, A., and Cherry, C. (2005). Dependency treelet translation: Syntactically informed phrasal smt. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 271–279, Stroudsburg, PA, USA. Association for Computational Linguistics.

Rachez, A. and Hagiwara, M. (2012). Augmented echo state networks with a feature layer and a nonlinear readout. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE.

Rafferty, A. N. and Manning, C. D. (2008). Parsing three german treebanks: lexicalized and unlexicalized baselines. In *Proceedings of the Workshop on Parsing German*, pages 40–46. Association for Computational Linguistics.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, DTIC Document.

Rummelhart, D. (1986). Learning representations by back-propagating errors. *Nature*, 323(9):533–536.

Satta, G. (2010). Translation algorithms by means of language intersection. *Manuscript*.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *Neural Networks, IEEE Transactions on*, 20(1):61–80.

Schiller, U. D. and Steil, J. J. (2005). Analyzing the weight dynamics of recurrent learning algorithms. *Neurocomputing*, 63:5–23.

Schwenk, H. and Gauvain, J.-L. (2005). Training neural network language models on very large corpora. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 201–208. Association for Computational Linguistics.

Shen, L., Sarkar, A., and Och, F. J. (2004). Discriminative reranking for machine translation. In *HLT-NAACL*, pages 177–184.

Shen, L., Xu, J., and Weischedel, R. (2008). A new string-to-dependency machine translation algorithm with a target dependency language model. In *In Proc. of ACL*, pages 577–585.

Shen, L., Xu, J., and Weischedel, R. (2010). String-to-dependency statistical machine translation. *Comput. Linguist.*, 36(4):649–671.

Shieber, S. M. and Schabes, Y. (1990). Generation and synchronous tree-adjoining grammars. In *Proceedings of the Fifth International Workshop on Natural Language Generation*, pages 9–14.

Smith, D. A. and Eisner, J. (2006). Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In *Proceedings of the Workshop on Statistical Machine Translation*, pages 23–30. Association for Computational Linguistics.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas*, pages 223–231.

Sorenson, H. (1960). *Kalman Filtering: Theory and Application*. IEEE Press selected reprint series. IEEE Press.

Steinberger, R., Pouliquen, B., Widiger, A., Ignat, C., Erjavec, T., Tufis, D., and Varga, D. (2006). The jrc-acquis: A multilingual aligned parallel corpus with 20+ languages. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC'2006)*, Genoa, Italy.

Stolcke, A. (1995). An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational linguistics*, 21(2):165–201.

Stratonovich, R. (1960). Conditional markov processes. *Theory of Probability & Its Applications*, 5(2):156–178.

Strunk, J., Silla, CarlosN., J., and Kaestner, C. (2006). A comparative evaluation of a new unsupervised sentence boundary detection approach on documents in english and portuguese. In Gelbukh, A., editor, *Computational Linguistics and Intelligent Text Processing*, volume 3878 of *Lecture Notes in Computer Science*, pages 132–143. Springer Berlin Heidelberg.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215.

Tesnière, L. (1959). *Éléments de syntaxe structurale*. Éditions Klinksieck.

Tillmann, C. (2004). A unigram orientation model for statistical machine translation. In *Proceedings of HLT-NAACL 2004: Short Papers*, pages 101–104. Association for Computational Linguistics.

Triefenbach, F., Jalalvand, A., Schrauwen, B., and Martens, J.-P. (2010). Phoneme recognition with large hierarchical reservoirs. In *NIPS*, pages 2307–2315.

Tromble, R. and Eisner, J. (2009). Learning linear ordering problems for better translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, EMNLP '09, pages 1007–1016, Stroudsburg, PA, USA. Association for Computational Linguistics.

Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y., and Singer, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(9).

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

Vapnik, V. (1982). *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Visweswariah, K., Rajkumar, R., Gandhe, A., Ramanathan, A., and Navratil, J. (2011). A word reordering model for improved machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 486–496, Stroudsburg, PA, USA. Association for Computational Linguistics.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269.

Vogel, S., Ney, H., and Tillmann, C. (1996). Hmm-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 836–841. Association for Computational Linguistics.

Wang, C., Collins, M., and Koehn, P. (2007). Chinese syntactic reordering for statistical machine translation. In *EMNLP-CoNLL*, pages 737–745.

Watanabe, T., Suzuki, J., Tsukada, H., and Isozaki, H. (2007). Online large-margin training for statistical machine translation. In *In Proc. of EMNLP*. Citeseer.

Weaver, W. (1955). Translation. *Machine translation of languages*, 14:15–23.

Wu, D. (1997). Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational linguistics*, 23(3):377–403.

Xu, P., Kang, J., Ringgaard, M., and Och, F. (2009). Using a dependency parser to improve smt for subject-object-verb languages. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, pages 245–253, Stroudsburg, PA, USA. Association for Computational Linguistics.

Xu, Y., Fern, A., and Yoon, S. W. (2007). Discriminative learning of beam-search heuristics for planning. In *IJCAI*, pages 2041–2046.

Yamada, H. and Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In *The 8th International Workshop of Parsing Technologies (IWPT2003)*.

Yamada, K. and Knight, K. (2001). A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, ACL '01, pages 523–530, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zaslavskiy, M., Dymetman, M., and Cancedda, N. (2009). Phrase-based statistical machine translation as a traveling salesman problem. 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 1 - Volume 1*, ACL '09, pages 333–341, Stroudsburg, PA, USA. Association for Computational Linguistics.

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

Zhang, D., Sun, L., and Li, W. (2008). A structured prediction approach for statistical machine translation. In *IJCNLP*, pages 649–654.

Zipf, G. K. (1935). The psycho-biology of language.

Zollmann, A. and Venugopal, A. (2006). Syntax augmented machine translation via chart parsing. In *Proceedings of the Workshop on Statistical Machine Translation*, pages 138–141. Association for Computational Linguistics.