End-to-End Word Sequence Modelling with Neural Networks and Word Embeddings:

a Deep Learning Approach and its Applications for NLP Tasks

THIS IS A TEMPORARY TITLE PAGE
It will be replaced for the final print by a version
provided by the service academique.

Thèse n. 1234 2011
présenté le 12 Mars 2011
à la Faculté des Sciences de Base
laboratoire SuperScience
programme doctoral en SuperScience
École Polytechnique Fédérale de Lausanne

pour l'obtention du grade de Docteur ès Sciences
par

Paolino Paperino

acceptée sur proposition du jury:

Prof Name Surname, président du jury
Prof Name Surname, directeur de thèse
Prof Name Surname, rapporteur
Prof Name Surname, rapporteur
Prof Name Surname, rapporteur

Lausanne, EPFL, 2011

Wings are a constraint that makes
it possible to fly.
— Robert Bringhurst

To my parents. . .

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Lausanne, 12 Mars 2011 D. K.

Preface

A preface is not mandatory. It would typically be written by some other person (eg your thesis director).

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Lausanne, 12 Mars 2011 T. D.

p

Abstract

Historically, Natural Language Processing (NLP) problems have been mainly tackled using generative models with task specific and manually engineered features. Recently, there has been a resurgence of interest for neural networks in the machine learning community, yielding state-of-the-art results in various fields such as computer vision, speech processing and natural language processing. The central idea behind these approaches is to learn input features along with the model, in an end-to-end manner, making as few assumptions as possible. In NLP, this is reflected by the use of *word embeddings* which consists in mapping words in a dictionary with continuous low-dimensional vectors. Using word embeddings as the main input feature has proven to be very efficient for a large variety of tasks while carrying very few *a-priori* linguistic assumptions.

In this thesis, we investigate a neural network-based approach for word sequence processing using word embeddings as input. This approach first models words *in context* in form of continuous vector representations, which are then used to perform the task of interest. Our approach is validated on different natural language processing applications, both supervised and unsupervised, from simple word tagging tasks (named entity recognition, shallow parsing) to more challenging structure prediction tasks (constituency parsing).

We first evaluate our approach on the task of *bilingual word alignment*, which consists in finding word correspondences between two sentences that are translations of each other. This is done, by modeling the source and target words *in context*. The obtained representations are then used to compute alignment scores. Training is done in an unsupervised manner by taking advantage of an aggregation operation that summarizes the alignment scores for a given target word. We shows that our model yields better performance than a standard generative model.

The model is then enhanced to tackle the case where phrases rather than single words are tagged. This is usually done by considering this problem as a classic word tagging problem. The phrase predictions are then retrieved using a transition graph. The proposed model allows to directly consider and tag arbitrary sized chunks without going through the word tagging intermediate state. We demonstrate the viability of our model by evaluating it on two chunck-prediction tasks: *named entity recognition* and *shallow parsing*.

The last part of this thesis focuses on the task of *syntactic constituency parsing*. In contrast

with the previous tasks, syntactic parsing aims at producing a structured analysis (a tree), given a natural language input sentence. We propose to recursively use the model previously introduced to tackle this problem. This is done by following a greedy procedure, seeing the parsing problem as a succession of word sequence tagging.

The model is then enhanced by a compositional vector representation which allows to propagate, through the tree, information about the early stages of the procedure. On the Wall Street Journal corpus, our model performs on par with standard generative model while being much faster due to its greedy nature. We then extend our model to morphologically rich languages and show that our model outperforms the current state-of-the-art model for various languages.

Key words: Neural Networks, Deep Learning, Natural Language Processing, Bilingual Word Alignment, Syntactic Parsing

Contents

List of Figures

List of Tables

1 Introduction

Interacting with a computer in natural language is one the oldest problems of computer science. Despite many efforts from the computational linguistic community, no attempts to design an unique data structure capturing the meaning of a piece of text have been successful. Instead, this problem has been divided into several specific sub-tasks. Each of them consists in extracting information reflecting our knowledge about natural language. It can consist in extracting semantic information such as for "named entity recognition" or "word sense disambiguation" as well as syntactic information such as for "syntactic parsing" or "part-of-speech tagging". This knowledge can then be used to perform higher level natural language processing (NLP) tasks such as translation or information retrieval.

Historically, NLP tasks have been mainly tackled with generative models using task specific and manually engineered features or linear models such as support vector machines trained over very high-dimensional sparse feature vectors. In the other hand, recent works in NLP using neural networks have focused on learning dense input representations using as few *a-priori* knowledge as possible. By learning these representations, also called embeddings, on large unlabeled databases, these models have been shown to yield state-of-the art performance for various NLP tasks.

In this thesis, we investigate a neural network-based approach for word sequence modeling using word embeddings as input. This approach first models words in context in form of continuous vector representations, which are then used to perform the tasks of interest. Our approach is validated on different natural language processing applications, both supervised and unsupervised, from simple word tagging tasks (named entity recognition, shallow parsing) to more challenging structure prediction tasks (constituency parsing).

1.1 Motivations and Objectives

Generative models are at the core of a majority of NLP systems. In the context of classification, they model the joint distribution between the observations and labels (classes). This distribution is often easily inferred by counting the number of time events occur at the same time, normalized by a common denominator. For instance, language modeling, which is a crucial component in machine translation and information retrieval, still widely rely on generative models. This task consists in assigning a probability that a sentence is a legal string in a language. This probability is estimated based on the relative frequency of word sequences observed in a training corpus. Similarly, most of the syntactic parsing systems rely on context free grammar (CFG) and assign probabilities to these rules by counting their occurences in a training corpus [Magerman, 1995, Collins, 2003, Charniak, 2000]. Decoding is then done using chart algorithms. Another example is the machine translation task. The highly used IBM machine translation alignment model is generative and is based on a counting of word concurrences [Brown et al., 1990].

Despite their advantages, generative models are facing several shortcomings. First, since there are a combinatorial number of possible observations, many rare (but not impossible) combinations never occur in training and are thus assigned a zero probability. These models thus need to be carefully smoothed in order to deal with unseen events. Second, generative models make various independence assumptions to enable efficient estimation and decoding. This makes difficult to incorporate arbitrary features. Finally, they often rely on task specific features (such as bags of words for information retrieval), or hand crafted features carrying linguistic knowledge (such as the head words for syntactic parsing).

On the other hand, discriminative models directly learn the conditional probability distribution of the classes given the observations. As they do not model the data but rather focus on a particular task, they tend to have better accuracies [Klein and Manning, 2002]. Furthermore, they allow to incorporate arbitrary input features more easily as they do not make any independence assumptions. Nevertheless, it is worth noting that discriminative models are also facing several issues. They are much more subject to overfitting. They are often more complex that their generative counterparts and they require more data in order to obtain good accuracy.

Recent advances in Machine Learning have made possible systems that can be trained in an end-to-end manner, without prior knowledge. Collobert et al. [2011] presents a good illustration of this idea applied to NLP. In this study, the authors propose a deep neural network, which learns the word representation (the features) and produces tags discriminatively in an end-to-end manner. This architecture is applied to various NLP task, like part-of-speech tagging, name entity recognition or semantic role labeling, and achieves state-of-the-art performance in all of them. More recently, several other works have taken advantage of word embeddings in various NLP domain such as machine translation [Cho et al., 2014] and question answering [Bordes et al., 2014].

These techniques offer many advantages: (1) they use much less prior knowledge such as linguistic knowledge which can potentially lead to suboptimal solutions (2) they are trained in an end-to-end manner, including the features, by back-propagating the error gradient. This allows to learn relevant knowledge for the task of interest (3) they are smoothed by design: even if an example have not been seen in the training corpus, its representation is close to similar examples (4) these representations can be trained on large unlabeled corpora.

In this thesis, we investigate a neural network-based approach for word sequence modeling. This approach relies on word embeddings and models words in context in form of continuous vector representations. This thesis aims at investigating how to take advantage of such representations to perform various NLP tasks. We first evaluate our approach on the task of word alignment which consists in finding the word correspondences between two sentences that are translations of each other. For both sentences, we use the sequence modeling approach to computes in-context word representations which are used to produce word association scores, using a simple dot product.

Word sequence modeling approaches has been successfully applied to phrase prediction problems which consist in identifying and labeling phrases in a sentence.These tasks can be seen as a word classification problem by modeling words with a fixed-size context using a sliding window approach. The phrase predictions are then recovered using a transition graph. In this thesis, we propose a model which directly tags the phrases without going through the intermediate word tagging stage. This is done by modeling arbitrary-sized chunks into fixed size representations which are used to perform the classification. This approach is evaluated on two phrase prediction tasks: named entity recognition and shallow parsing.

While the sequence modeling approach performs well on simple tasks like part-of-speech tagging [Collobert et al., 2011], it seems not to be relevant for structure prediction, like syntactic parsing, were dependencies between constituents can be complex and distant. Syntactic parsing is the task of producing a syntactic analysis of a sentence in form of a tree. Nodes of the tree represent grammar rules expressing syntactic relation between syntactic constituent. We propose to address this problem by recursively applying a word sequence modeling approach. By successively merging the predicted nodes to build the new input, our approach allows to reduce the input size and thus enables the model to consider a larger context. The model is then enhanced with a compositional feature which allows to summarize sub-trees, both syntactically and semantically, in form of continuous representations.

1.2 Contributions

The contributions of this thesis can be summarized as follows:

- Introducing an **unsupervised discriminative word alignment** model which outperforms a standard generative baseline on English-Romanian, Romanian-English and Czech-English.

  Word alignment is the task of finding the correspondences between words in a pair of sentences that are translations of each other. In this thesis, we propose a neural network model that extracts context information from the source and target sentences and then computes simple dot products to estimate alignment links. The model can be easily trained on unlabeled data via a simple aggregation operation. The aggregation combines the scores of all source words for a particular target word. The network is trained using a soft-margin criterion, which promotes source words which are likely to be aligned with a given target word according to the knowledge the model has learned so far. At test time, the aggregation operation is removed and source words are aligned to target words by choosing the highest scoring candidates. Results on three different pairs of languages shows that our model significantly outperforms a standard generative model. This work is currently under review at WMT 2016.

- Investigating a novel **chunk-based approach** for **phrase-prediction**.

  Phrase prediction problems consist in identifying and labeling phrases in a sentence. These problems are often seen as word classification problem by prefixing every possible tag using the standard BIOES scheme (Begin, Intermediate, Other, End, Single). A coherent path of tags is then recovered using a constrained transition graph. In this thesis, we propose a novel architecture which models arbitrary-sized chunks into fixed-size representations. These representations are used to perform the chunk classification without going through the intermediate word tagging stage. The architecture is based on neural networks and is trained with an objective function similar to that of a Conditional Random Field. We evaluate our approach on two standard NLP tasks: named entity recognition and shallow parsing and compare our performance with a BIOES-based model. We show that the proposed approach performs on par with this state-of-the-art baseline.

- Introducing a **greedy discriminative constituency parser** which rely on words embeddings.

  In this thesis, we propose a bottom-up greedy and purely discriminative syntactic parsing approach that relies only on a few simple features. The core of the architecture is a word sequence modeling architecture which performs a phrase-prediction task (as described in Chapter 4). This is done by leveraging continuous word vector representations to model the conditional distributions of context-aware syntactic rules. The learned distribution rules are naturally smoothed, thanks to the continuous nature of

the input features and the model. The tree is built in a greedy manner by recursively applying the model over the new inputs, taking into account the previous node predictions. By successively merging the predicted node to build the new input, our approach allows to reduce the input size and thus enables the model to consider a larger context. Despite the greedy nature of our approach, generalization accuracy compares favorably to existing generative or discriminative non-reranking parsers, while being faster. This work has been published in Legrand and Collobert [2014].

- Introducing a new **compositional sub-tree representation** based on **recursive neural networks** which summarizes syntactically and semantically the contents of sub-trees in form of continuous vectors.

  We introduce a compositional procedure which outputs a vector representation summarizing sub-trees, both syntactically (parsing/POS tags) and semantically (words). This procedure is jointly trained, in an end-to-end manner, along with the greedy parser introduced in Chapter 5. The composition is achieved over continuous (word or tag) representations using recursive neural networks. This compositional representation allows to reach performance on par with well-known existing parsers, while having the advantage of speed, thanks to the greedy nature of the parser. We provide a fully functional implementation of this parser. This work has been published in Legrand and Collobert [2015].

- Extending the RNN-based greedy parser to **morphologically rich language** by composing morphological information. This approach outperforms the current State-of-the art results for 6 languages.

  Morphologically rich languages (MRL) are languages in which lot of the structural information is contained at a word-level, leading to a high level of word-form variation. Unlike English they can have complex word structure as well as flexible word order. We propose to enhance our model for syntactic parsing of MRL by learning morphological embeddings. We take advantage of a recursive composition procedure similar to the one used in Chapter 6 to propagate morphological information during the parsing process. We evaluate our approach on the SPMRL (Syntactic Parsing of MRL) Shared Task 2014. We show that integrating morphological features, allows to increase dramatically the average performance and outperforms the current state-of-the-art performance for a majority of languages. This work is under review at ACL 2016.

1.3 Organization

The reminder of the thesis is organized as follows:

- Chapter 2, *Background*, presents the machine learning background underlying the whole thesis and introduces the notation used in this work. We then introduce the neural network formalism, including feed-forward and recursive neural networks. Finally, we provide a review of the natural language processing literature using deep neural network techniques.

- Chapter 3, *Sequence Processing for Bilingual Word Alignment*, introduces an unsupervised discriminative word alignment model using a word sequence modeling approach. This approach models words in context, for both the source and target sentences and produces an association scores by computing dot products. Our approach is evaluated in terms of alignment error rate (AER) for three pairs of languages (English-French, Romanian-English and English-Czech). Additionally, we evaluate our approach on a full machine translation pipeline using the freely available software Moses, trained on the data provided by the workshop on statistical machine translation 2016 (WMT16).

- Chapter 4, *Phrase Prediction: chunk-Based Approach*, introduces a chunk-based sequence modeling approach for phrase prediction. This approach models arbitrary-sized chunks into fixed size representations which are used to perform the classification. Our approach is evaluated on two standard phrase prediction tasks: named-entity recognition and shallow parsing. We provide a comparison with the sliding window approach introduced in Collobert et al. [2011].

- Chapter 5, *Sequence Processing for Structural Inference: Syntactic Parsing*, introduces a greedy parser which recursively apply a word sequence modeling approach. The input size is progressively reduced during the process by merging the predicted node to build the new input. This enables the model to consider a larger context. This approach is evaluated on a standard syntactic parsing task using the Wall Street Journal (WSJ) corpus. We compare our performance with the several well known existing parser.

- Chapter 6, *RNN-Based Phrase Embeddings for Syntactic Parsing*, extends the model introduced in the previous chapter with a compositional sub-tree representation which summarizes the contents of a sub-tree in the form of a continuous vector. The new model is evaluated on the WSJ corpus and compared to the literature. We then provide and analysis of the content of the representations. Finally, this model is evaluated on the data provided by the syntactic parsing of morphologically rich language workshop for 9 languages.

- Chapter 7, *Conclusion*, concludes with possible directions for the future research.

2 Background

2.1 Machine learning

Machine learning is the field of study that aims at exploring algorithms which learn from experience to perform a given task, without being explicitly programmed. In such algorithms, the behavior is conditioned by a set of parameters, called the *model*. This model is modified according to some experience, by presenting realizations of the task contained in a *database* of training samples. The objective of machine learning is not to learn the training samples "by heart" but rather to *generalize*, that is to perform well on sample that has never been seen before. The following section introduces the machine learning mathematical formalism that will be used in the rest of the thesis.

2.1.1 Supervised learning

Supervised learning is a machine learning technique which aims at inferring a function from labeled training data. Let $\mathcal{X}$ and $\mathcal{Y}$ be the input and output spaces respectively. We define the set of N training sample $\{(x_i, y_i), \ldots, (x_N, y_N)\}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. Given a set of function, $\mathcal{F}: \mathcal{X} \to \mathcal{Y}$, we define a loss function (the "measure of performance"):

$$Q: \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$$

Using this notation, supervised learning consists in finding a function $f \in \mathcal{F}$ which minimize the empirical risk

$$R: \frac{1}{N} \sum_{n=1}^{N} Q(f(x_n), y_n)$$

We can distinguish two types of supervised learning:

Classification: in that case, $\mathcal{Y}$ is a finite set of categories (called *classes*). Training consist in finding a function f which assigns an input vector x to its corresponding category y. A simple loss function could be:

$$Q(f(x), y) = \begin{cases} 0 & \text{if } f(x) = y \\ 1 & \text{if } notherwise \end{cases}$$

Regression: in that case, $\mathcal{Y}$ is a real vector space. Training consist in finding a function f which minimize the distance $D(f(x), y)$, D being an arbitrary distance function. the most common distance function used is the Mean Square Error (MSE), which gives the following loss function:

$$Q(f(x), y) = ||f(x) - y||^2$$

2.1.2 Unsupervised learning

Unsupervised learning is the machine learning task which aims at discovering hidden structure from unlabeled data. Formally, given a set of N training samples, $\{(x_i), \ldots, (x_N)\}$, unsupervised learning consists in finding a function $f \in F : X \to Y$ which minimize the empirical risk

$$R : \frac{1}{N} \sum_{n=1}^{N} Q(f(x_n))$$

where $Q(f(x))$ is a loss function. Note that unlike for supervised learning, Q does not depend on any target label y but only on the input x.

Unsupervised learning techniques are used to discover intrinsic regularities in data for which no labels are available. They includes diverse techniques such as clustering (K-means) and dimensionality reduction (principal component analysis (PCA)) and can be trained in very different ways such as predicting the next input in a sequence or minimizing the reconstruction error.

2.1.3 Semi-supervised learning

Semi-supervised learning is the class of machine learning task which make use of unlabeled data along with labeled data. Typically, semi-supervised systems take advantage of the large amount of unsupervised data (which are available at low cost) to overcome the lack of labeled data (which require expensive human annotations).

2.2 Neural networks

In this thesis, we use the artificial neural network (ANN) family as set of functions F. The following section introduces the notation used in the rest of the thesis, for two types of ANN: the multi-layer perceptrons (MLP) and the recursive neural networks (RNN).

2.2.1 Feed forward neural networks

Perceptron: Neural network, as the name indicates, are computational mechanisms inspired by the neural cells in the brain. The perceptron, as first defined in Rosenblatt [1957], represents a brain computation unit. Considering a set of training samples $\{(x_i, y_i), \ldots, (x_N, y_N)\}$ where the label $y_i \in \{-1, 1\}$, the perceptron is a simple linear classifier defined as

$$f_\theta(x) = w.x + b$$

where $\theta = (w, b)$ are the parameters to train (w being a real-valued vector of weights and b real-valued bias scalar). The training is done by considering every training sample (x_i, y_i) successively and updating the weights for misclassified ones (if $y_i f_\theta(x_i) \leq 0$) using the following rules:

$$w \leftarrow w + y_i x_i \qquad b \leftarrow b + y_i$$

This update has for consequence to increase the score $y_i f_\theta(x_i)$. In the case of linearly separable classes, the convergence has been proven by Novikoff [1962].

Multi-Layer Perceptron: MLPs used in this thesis are combinations of perceptron units, organized in successive layer, followed by non-linearity operations (called transfer function). MLPs are known to be universal function approximators (Cybenko [1989], Hornik et al. [1989]). This means that given a finite number of training samples (x_i, y_i) and a target function $g(x)$, there exists an MLP that can approximate $g(x)$ as close as desired, for all (x_i, y_i). Figure 2.1 illustrates such a network.

While the neural inspiration of MLP is interesting to understand the motivations behind this model, it can be misleading and difficult to apprehend mathematically. To simplify the notation, a MLP can be seen as a stack of matrix-vector multiplications, followed by a point-wise non-linear operation. Formally, a MLP with L layers can be define as:

$$f_\theta(x) = \Phi_1(\Phi_2(\ldots \Phi_L(x)))$$

with

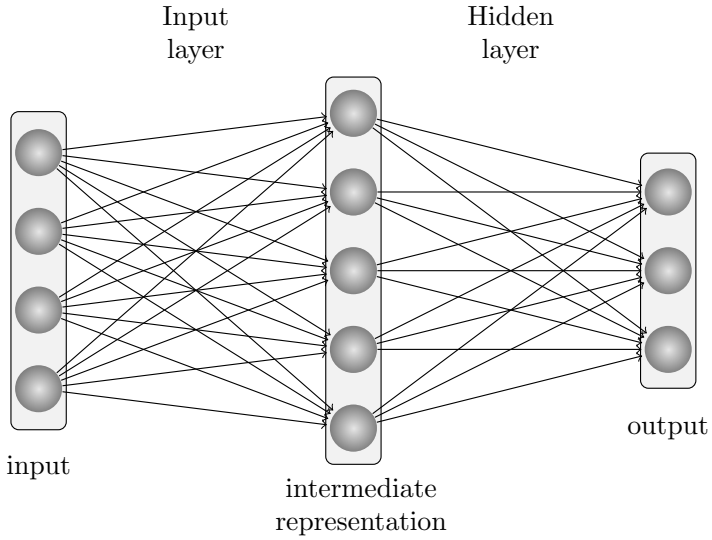

Figure 2.1: Illustratin of a 2-layer neural network.

$$\Phi_l(x) : \mathbb{R}^a \rightarrow \mathbb{R}^b = h(W_l.x + b_l)$$

where $x \in \mathbb{R}^a$ is an input vector, $W_l \in \mathbb{R}^{b \times a}$ a matrix of weight, $b_l \in \mathbb{R}^b$ a vector of bias and $h()$ a point-wise non-linear function such as the hyperbolic tangent or the sigmoid function.

Training a MLP: Many techniques exist to train a feed-forward neural network [Battiti, 1992]. Among them, the most popular remains the back-propagation. It was first applied to MLPs by Le Cun [1985] and Rumelhart et al. [1986] and consist in "back-propagating" the criterion (loss function Q) error, layer-by-layer, using chained derivatives. In this thesis, we use the stochastic gradient descent [Bottou, 1991] which has proven to be very effective in the case of large-scale machine learning problems [Bousquet and Bottou, 2008]. The idea of gradient descent is to update every parameter according to its influence on the criterion error. This is done by computing the partial derivative Δ_i of the criterion function according to every parameters θ_i:

$$\Delta_i(x) = \frac{\delta Q(f_\theta(x), y)}{\delta \theta_i}$$

Update is then done according to this error derivative:

$$\theta_i \leftarrow \theta_i - \lambda \Delta_i(x)$$

where λ is a learning coefficient.

2.2.2 Softmax layer

In the context of classification, it is often useful to obtain probability of a given class. If we denote $f_\theta^i(x)$ the i^{th} output of the network, we can interpret this score as a probability by adding a softmax operation over all the classes [Bridle, 1990]:

$$p(i|x,\theta) = \frac{e^{f_i(x)}}{\sum_j e^{f_j(x)}}$$

2.2.3 Recursive neural networks

Recursive neural networks (RNN) are neural networks in which the the same set of weights is applied recursively over a structure. They can be applied on arbitrary data structures, by allowing to compute fixed size representations of variable-length input elements. Note that recurrent neural network are recursive neural network for which the recursion is done over time.

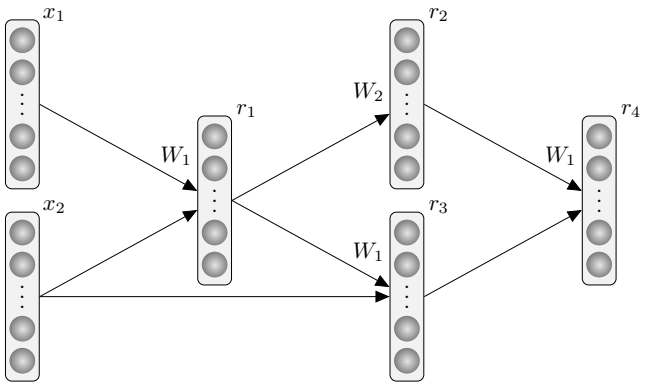

Figure 2.2: Example of recursive neural network for a directed acyclic graph.

Figure 2.2 presents an example of structure (a directed acyclic graph) on which we can apply a RNN. The output of the network is obtained by computing all the node representations, using their child's representations as input. This implies that the node computation order is imposed by the structure of the graph. If we define $[a\ b]$ the concatenation of vectors a and b, the output representation r_4 is computed as follows:

$$r_1 = h(W_1.[x_1\ x_2] + b_1)$$

$$r_2 = h(W_2.r_2 + b_2)$$

$$r_3 = h(W_1.[r_1\ x_2] + b_1)$$

$$r_4 = h(W_1.[r_2\ r_3] + b_1)$$

Training is done using the back-propagation through structure [Goller and Küchler, 1996], which is a modified version of the back propagation. The error gradient is back propagated through the structure, following the reversed forward node order.

2.3 Deep Learning for NLP

Deep learning is a branch of machine learning which aims at learning, successive levels of representation, using multiple levels of non-linear information processing (from abstract low-level to high-level ones), in order to model complex relationships. These representations are often trained on unlabeled data using unsupervised training techniques. In NLP, deep learning methods are typically based on neural network and rely on word embeddings (continuous vectors). This section first introduces the concept of word embeddings as well as different unsupervised techniques to produce such representations. We then review NLP applications taking advantage of words embedding.

2.3.1 Word embeddings

Historically, NLP problems has been mainly tackeled with linear models such as support vector machines or logistic regression, using high dimensional and very sparse feature vectors. As stated in Harris [1954], Firth [1957] and Wittgenstein [1953], words that occur in similar contexts tend to have similar meanings. One of the first approach to capture that knowledge in an fixed-sized low dimensional word embeddings was the Brown clustering algorithm [Brown et al., 1992]. This algorithm addresses the data sparsity problem inherent in NLP by grouping words into clusters assumed to be semantically related. A word is then represented by a low-dimensional binary vector representing a path in a binary tree.

Spectral methods: Several methods based on the spectral decomposition of the word coocurence matrix has been developed to produce words embeddings. Several work are based on the Singular Value Decomposition (SVD), including the LSA [Landauer and Dumais, 1997] and the ICA [Väyrynen and Honkela, 2004]. Lebret and Collobert [2014] proposed to perform a Principal Component Analysis (PCA) of the word cooccurence probability matrix while minimizing the reconstruction error according to the Hellinger distance (instead of the usual Euclidian distance). They evaluated the low dimensional word representations obtained on several NLP tasks, showing significant improvement over existing previous word embeddings.

Neural network methods: Bengio et al. [2003] proposed a method to learn dense word vector representations by training a neural network-based language model. This work inspired several authors: Collobert and Weston [2008] trained a language model jointly with different NLP tasks, showing generalization improvements for all them. Mnih and Hinton [2009] proposed a hierarchical distributed language model which is faster to train than the original neural language model while producing better word representations. Mikolov et al. [2013] proposed a neural architecture (called skip-gram model) for computing continuous vector

representations of words. Training is done by predicting surrounding words given an input word from a sentence. All these methods are trained on large unlabeled corpora using large neural networks.

2.3.2 Deep Learning for word sequence processing

Modeling natural language sequences using neural networks and continuous vectorial representations has a long history. Early work on distributed representations includes Hinton et al. [1986] and Elman [1990, 1991]. More recently, Bengio et al. [2000] was able to overcome the standard n-gram language model (in terms of perplexity) by training a neural network using continuous word vectors as input. This idea was then taken up in Collobert and Weston [2008] to learn word embeddings in an unsupervised manner. They showed that jointly leaning these embeddings (taking advantage of the large amount of unlabeled data), along with a bunch of supervised NLP task, allowed to improve the generalization of these tasks, yielding state-of-the-art results. In Turian et al. [2010], they showed that using unsupervised word representations as extra word features is a simple and general way to improve existing systems accuracy.

Deep learning techniques using words embeddings has been successfully used in various NLP domains such as sentiment classification [Kalchbrenner et al., 2014], paraphrase identification [Yin and Schütze, 2015] and question answering [Dong et al., 2015]. Recursive neural network using word embeddings were successfully applied to structure prediction tasks such as constituency parsing [Socher et al., 2013a], dependency parsing [Le and Zuidema, 2014, Zhu et al., 2015], sentiment classification [Socher et al., 2013b]. Recurrent neural networks using word embeddings were successfully applied to sequence modeling tasks such as language modeling [Mikolov et al., 2010], and machine translation [Sutskever et al., 2014, Cho et al., 2014].

3 Sequence Processing for Bilingual Word Alignment

3.1 Introduction

3.2 Background

3.2.1 Baseline

3.2.2 Evaluation

3.3 Unsupervised Discriminative Word Alingment Using Word Embeddings

3.4 Introduction

Word alignment is the task of finding the correspondence between foreign and English words in a pair of sentences that are translations of each other. Generative models for this task Brown et al. [1990], Och and Ney [2003], Vogel et al. [1996] still form the basis for many machine translation systems Koehn et al. [2003], Chiang [2007]. Yang et al. [2013] presented a feed-forward network-based model trained on alignments that were generated by a traditional generative model. This treats potentially erroneous alignments as supervision. Tamura et al. [2014] sidesteps this issue by negative sampling to train a recurrent-neural network on unlabeled data. They optimize a global loss that requires an expensive beam search to approximate the sum over all alignments.

In this paper we introduce a word alignment model that is simpler in structure and which relies on a more tractable training procedure. Our model is a neural network that extracts context information from the source and target sentences and then computes simple dot products to estimate alignment links. Our objective function is word-factored and does not require the expensive computation associated with global loss functions. The model can be easily trained on unlabeled data via a novel but simple *aggregation operation* which has been successfully applied in the computer vision literature Pinheiro and Collobert [2015]. The

aggregation combines the scores of all source words for a particular target word, and together with our soft-margin criterion, it promotes source words which are likely to be aligned with a given target word according to the knowledge the model has learned so far. At test time, the aggregation operation is removed and source words are aligned to target words by choosing the highest scoring candidates (§3.5, §3.6).

We evaluate several forms of our aggregation operation such as computing the sum, max and LogSumExp over alignment scores. Results on two standard tasks for English-French and English-Romanian alignment shows that our model is on par with Fast Align, a popular log-linear reparameterization of IBM Model 2 Dyer et al. [2013] on French-English data while significantly outperforming it on Romanian-English experiments (§3.7).

3.5 Aggregation Model

In the following, we consider a target-source sentence pair $(\mathbf{e}, \mathbf{f})$, with $\mathbf{e} = (e_1, \ldots, e_{|\mathbf{e}|})$ and $\mathbf{f} = (f_1, \ldots, f_{|\mathbf{f}|})$. Words are represented by f_j and e_i, which are indices in source and target dictionaries. For simplicity, we assume here that word indices are the only feature fed to our architecture. Given a source word f_j and a target word e_i, our architecture embeds a window (of size d^f_{win} and d^e_{win}, respectively) centered around each of these words into a d_{emb}-dimensional vector space. The embedding operation is performed with two distinct neural networks:

$$\text{net}_e([\mathbf{e}]_i^{d^e_{win}}) \in \mathbb{R}^{d_{emb}} \quad \text{and} \quad \text{net}_f([\mathbf{f}]_j^{d^f_{win}}) \in \mathbb{R}^{d_{emb}},$$

where we denote the window operator as

$$[\mathbf{x}]_i^d = (x_{i-d/2}, \ldots, x_{i+d/2}).$$

The matching score between a source word f_j and a target word e_i is then given by the dot-product:

$$s(i, j) = \text{net}_e([\mathbf{e}]_i^{d^e_{win}}) \cdot \text{net}_f([\mathbf{f}]_j^{d^f_{win}}). \tag{3.1}$$

If e_i is aligned to f_{a_i}, the score $s(i, a_i)$ should be high, while scores $s(i, j)\ \forall j \neq a_i$ should be low.

3.5.1 Unsupervised Training

In this paper, we consider an unsupervised setup where the alignment is not known at training time. We thus cannot minimize or maximize matching scores (3.1) in a direct manner. Instead,

given a target word e_i we consider the aggregated matching scores over the source sentence:

$$s_{aggr}(i, \mathbf{f}) = \operatorname*{Aggr}_{j=1}^{|\mathbf{f}|} s(i, j)\,, \tag{3.2}$$

where Aggr is an aggregation operator (§3.5.2). Consider a matching (positive) sentence pair $(\mathbf{e}^+, \mathbf{f})$ and a negative sentence pair $(\mathbf{e}^-, \mathbf{f})$. Given a word at index i^+ in the positive target sentence, we want to maximize the aggregated score $s_{aggr}(i^+, \mathbf{f})$ $(1 \leq i^+ \leq |\mathbf{e}^+|)$ because we know it should be aligned to at least one source word.[1] Conversely, given a word at index i^- in the negative target sentence, we want to minimize $s_{aggr}(i^-, \mathbf{f})$ $(1 \leq i^- \leq |\mathbf{e}^-|)$ because it is unlikely that the source sentence can explain the the negative target word. Following these principles, we consider a simple soft-margin loss:

$$\begin{aligned}\mathcal{L}(\mathbf{e}^+, \mathbf{e}^-, \mathbf{f}) = &\sum_{i^+=1}^{|\mathbf{e}^+|} \log(1 + e^{-s_{aggr}(i^+, \mathbf{f})}) \\ &+ \sum_{i^-=1}^{|\mathbf{e}^-|} \log(1 + e^{+s_{aggr}(i^-, \mathbf{f})})\,.\end{aligned} \tag{3.3}$$

Training is achieved by minimizing (3.3) and by sampling over triplets $(\mathbf{e}^+, \mathbf{e}^-, \mathbf{f})$ from the training data.

3.5.2 Choosing the Aggregation

The aggregation operation (3.2) is only present during training and it acts as a filter which aims to explain a given target word e_i by one or more source words. If we had the word alignments, then we would sum over the source words f_j aligned with e_i. In our setup, the alignment is not provided at training time, so we must rely on what the model has learned so far to filter the source words. We consider the following strategies:

- **Sum:** ignore the knowledge learned so far, and assign the same weight to all source words f_j to explain e_i.[2] In this case, we have

$$s_{aggr}(i, \mathbf{f}) = \sum_{j=1}^{|\mathbf{f}|} s(i, j)\,.$$

- **Max:** encourage the best aligned source word f_j, according to what the model has learned so far. In this case, the aggregation is written as:

$$s_{aggr}(i, \mathbf{f}) = \max_{j=1}^{|\mathbf{f}|} s(i, j)\,.$$

- **LSE:** give similar weights to source words with similar scores. This can be achieved with

[1] We discuss how we handle unaligned target words in §3.5.3.

[2] This can be seen by observing that the gradients for all source words are the same.

a Log-Sum-Exp aggregation operation (also called LogAdd), and defined with:

$$s_{aggr}(i, \mathbf{f}) = \frac{1}{r} \log \left(\sum_{j=1}^{|\mathbf{f}|} e^{r\, s(i,j)} \right), \tag{3.4}$$

where r is a positive scalar (to be chosen) controlling the smoothness of the aggregation. Note that if r is very small, the aggregation is equivalent to a sum, and if r is large, the aggregation acts as a max.

3.5.3 Decoding

At test time, we align each target word e_i with the source word f_j for which the matching score $s(i, j)$ in (3.1) is the highest. However, not every target word is aligned with a source word, so we consider only alignments with a matching score above a threshold:

$$s(i, j) > \mu^-(e_i) + \alpha\, \sigma^-(e_i), \tag{3.5}$$

where α is a tunable hyper-parameter, and

$$\mu^-(e_i) = \mathop{\mathbb{E}}_{\{\tilde{e}_k = e_i \in \tilde{\mathbf{e}},\, \tilde{f}_{j^-} \in \tilde{\mathbf{f}}^-\}} \left[s(k, j^-) \right]$$

is the expectation over all training sentences $\tilde{\mathbf{e}}$ containing the word e_i, and all words $\tilde{f}_j^-$ belonging to a corresponding negative source sentence $\tilde{\mathbf{f}}^-$, and $\sigma^-(e_i)$ is the respective variance.

3.6 Neural Network Architecture

Our model consists of two neural networks $\text{net}_e()$ and $\text{net}_f()$ as shown in (3.1). Both of them take the same form, so we detail only the target architecture. The discrete features $[\mathbf{e}]_i^{d^e_{win}}$ are embedded into a d^e_{emb}-dimensional vector space via a lookup-table operation (as first introduced in Bengio et al., 2000):

$$\begin{aligned} x^e &= \text{LT}_{W^e}([\mathbf{e}]_i^{d^e_{win}}) \\ &= (\text{LT}_{W^e}(e_{i-d^e_{win}/2}), \ldots, \text{LT}_{W^e}(e_{i+d^e_{win}/2})), \end{aligned}$$

where the lookup-table operation applied on index k returns the k^{th} column of the parameter matrix W^e:

$$\text{LT}_{W^e}(k) = W^e_{\bullet,k}.$$

The matrix W^e is of size $|\mathcal{V}^e| \times d^e_{emb}$, where $\mathcal{V}^e$ is the target vocabulary, and d^e_{emb} is the word embedding size for the target words. The word embeddings output by the lookup-table are concatenated and fed to a standard two-layer neural network architecture:

$$\text{net}_e(x^e) = M^{e,2} \tanh(M^{e,1}\, x^e) \tag{3.6}$$

where $M^{e,1} \in \mathbb{R}^{d^e_{hu} \times (d^e_{emb} d^e_{win})}$ (d^e_{hu} being a number of hidden units to be tuned) and $M^{e,2} \in \mathbb{R}^{d_{emb} \times d^e_{hu}}$ are the parameter matrices to be trained, and the $tanh(\cdot)$ operation is applied pointwise. All the parameters of the network architecture (W^e, $M^{e,1}$ and $M^{e,2}$) are trained by stochastic gradient, minimizing the loss (3.3) introduced in §3.5.1.

3.6.1 Additional Features

In addition to the raw word indices, we consider two additional discrete features which were handled in the same way as word features by introducing an additional lookup-table for each of them. The output of all lookup-tables was concatenated, and fed to the two-layer neural network architecture (3.6).

Distance to the diagonal. This feature can be computed for a target word e_i and a source word f_j:

$$diag(i, j) = \left| \frac{i}{|\mathbf{e}|} - \frac{j}{|\mathbf{f}|} \right|,$$

This feature allows the model to learn that aligned sentence pairs use roughly the same word order and alignment links remain close to the diagonal. We use this feature only for the source network.

Char n-gram. We consider unigram, bigram and trigram character features for which we have separate lookup-tables; we also use a special character marking word boundaries. For a given word, we extract all character n-grams of order n, then lookup their embeddings, and average them. This results in three averaged character embedding representations per word that are input to the source and target networks.

3.7 Experiments

Our models are evaluated on two datasets and we measure precision, recall, F-measure and Alignment Error Rate (AER). We use the English-French Hansards corpus as distributed by the NAACL 2003 shared task Mihalcea and Pedersen [2003]. This dataset contains 1.1M sentence pairs and the test set contains 447 examples. We also evaluate on the Romanian-English dataset of the ACL 20015 shared task Martin et al. [2005] comprising 95K sentence pairs for training and 448 instances for testing. We train models in each language direction and then symmetrize the resulting alignments using either the intersection or the grow-diag-final-and heuristic Och and Ney [2003], Koehn et al. [2003].

3.7.1 Setup

We tuned the hyper-parameters on the validation set of each dataset. The networks $\text{net}_e(\cdot)$ and $\text{net}_f(\cdot)$ are two-layers neural networks with $d^e_{hu} = d^f_{hu} = 256$ hidden units. The input window size of the source network is $d^e_{win} = 3$, while the target one is $d^f_{win} = 1$. The common

	Max	Sum	LSE
En-Fr	18.1	23.0	15.1
Fr-En	20.7	26.9	15.8
symmetrized	14.8	24.1	12.8
Ro-En	42.2	42.0	37.6
En-Ro	40.4	40.2	35.7
symmetrized	36.4	35.6	32.2

Table 3.1: Alignment error rates for different aggregators in each language direction and with grow-diag-final-and symmetrization.

embedding size of the networks is fixed to $d_{emb} = 256$. The source $\mathcal{V}_f$ and target $\mathcal{V}_e$ dictionaries are limited to the $20K$ most common words. Extra words are mapped to a unique *UNK* token. The word embedding sizes d^e_{emb} and d^f_{emb}, as well as the char-n-gram embedding size is 128. For LSE, we set $r = 1$ in (3.4). Word embeddings for all languages were pre-trained on Wikipedia data by performing a simple PCA on the matrix of word co-occurrences, following Lebret and Collobert [2014]. The Wikipedia corpora for English, French and Romanian comprise $1747M$, $444M$ and $61M$ tokens, respectively. For the ensembles we trained 10 systems, each with a different random seed, and averaged their scores. The alignment thresholding variables $\mu^-(e_i)$ and $\sigma^-(e_i)$ used for decoding (see §3.5.3) were estimated on 1000 training sentences, using for each of them 100 negative sentences. Words not appearing in this training subset were assigned $\mu^-(e_i) = \sigma^-(e_i) = 0$.

3.7.2 Results

We first explore different choices for the aggregation operator (§3.5.2) and then compare to the baseline. Table 3.1 shows that the Log-Sum-Exp aggregator performs best on both datasets for every direction as well as the symmetrization. Learning with the sum is most challenging on English-French whereas on Romanian-English both the sum and the max perform equally low. We use LSE for the remaining experiments.

On English-French data (Table 3.2) our model outperforms the baseline Dyer et al. [2013] in each individual language direction while being comparable to the baseline in the symmetrized setting. The choice of symmetrization heuristic greatly effects accuracy as the difference between the grow-diag-final-and heuristic and the intersection shows. For Romanian-English (Table 3.3) we add character n-gram features to better deal with the morphologically rich nature of Romanian which allows us to outperform the baseline. Adding ensembles further improves accuracy and leads to a significant improvement of 2.8 AER over the best baseline symmetrization (intersect).

	P	R	F1	AER
Baseline En-Fr	49.6	89.8	63.9	16.7
En-Fr	61.8	80.0	69.7	15.1
+ ensemble	60.4	85.1	70.6	**12.8**
Baseline Fr-En	52.9	88.4	66.2	16.2
Fr-En	65.5	78.0	71.2	15.8
+ ensemble	62.7	81.7	71.0	**14.9**
Baseline (intersect)	69.6	84.0	76.1	**11.4**
Baseline (gdfa)	47.7	92.6	62.9	15.3
intersect	75.8	64.2	70.6	20.3
grow-diag-final-and	60.9	84.4	70.8	12.8
+ ensemble	59.6	87.6	70.1	11.6

Table 3.2: English-French results on the test set in terms of precision (P), recall (R), F-score (F1) and AER; ensemble denotes a combination of ten systems and we use the intersection and grow-diag-final-and symmetrization (gdfa) heuristics.

	P	R	F1	AER
Baseline Ro-En	70.0	61.0	65.2	34.8
Ro-En	72.5	54.7	62.4	37.6
+ char n-gram	72.6	57.4	64.2	35.8
+ ensemble	75.2	60.3	66.9	**33.1**
Baseline En-Ro	71.3	60.8	65.6	34.4
En-Ro	78.0	54.7	64.3	35.7
+ char n-gram	80.0	55.6	65.6	34.4
+ ensemble	78.8	57.6	66.5	**33.5**
Baseline (intersect)	87.8	53.1	66.2	33.8
Baseline (gdfa)	69.5	66.5	68.0	32.0
intersect	91.6	41.0	56.6	43.4
grow-diag-final-and	79.1	59.3	67.8	32.2
+ char n-gram	76.4	62.0	68.4	31.6
+ ensemble	77.8	65.0	70.8	**29.2**

Table 3.3: Romanian-English results on the test set with char n-gram features (cf. Table 3.2).

3.8 Conclusion

In this paper, we presented a simple but very scalable neural network alignment model trained on unlabeled data. Our architecture computes alignment scores as dot products between representations of windows around target and source words. We apply an *aggregation operation* borrowed from the computer vision literature to make unsupervised training possible. The aggregation operation acts as a filter over alignment scores and allows us to determine which source words explain a given target word. We improve over a popular log-linear reparameterization of IBM Model 2 Dyer et al. [2013] by up to 2.8 AER on Romanian-English data while achieving comparable accuracy on English-French alignment.

content: ACL Paper (Under Review, https://publidiap.idiap.ch/downloads/internals/2016/Legrand_Idiap-Internal-RR-06-2016.pdf)

4 Phrase Prediction: chunk-Based Approach

4.1 Introduction

4.2 Background

4.3 BIOES Approach

4.4 Chunk-Based Approach

content: Unpublished work (results similar but chunk-based approach slower).

5 Sequence Processing for Structural Inference: Syntactic Parsing

5.1 Introduction

5.2 Syntactic Parsing

5.3 Baseline

5.4 Evaluation

5.5 Reccurent Greedy Parsing with Neural Networks

5.6 Introduction

While discriminative methods are at the core of most state-of-the-art approaches in Natural Language Processing (NLP), historically the task of syntactic parsing has been mainly solved with generative approaches. A major contribution in the parsing field is certainly probabilistic context-free grammar (PCFGs)-based parsers Magerman [1995], Collins [2003], Charniak [2000]. These types of parsers model the syntactic grammar by computing statistics of simple grammar rules occurring in a training corpus. Inference is then achieved with a simple bottom-up chart parser. These methods face a classical learning dilemma: on one hand PCFG rules have to be refined enough to avoid any ambiguities in the prediction. On the other hand, too much refinement in these rules implies lower occurrences in the training set, and thus a possible generalization issue. PCFGs-based parsers are thus judiciously composing with carefully chosen PCFG rules and clever regularization tricks.

Given the success of discriminative methods for various NLP tasks, similar methods have been attempted for the syntactic parsing task. One of the first successful discriminative parsers Ratnaparkhi [1999] was based on MaxEnt classifiers (trained over a large number of different features) and a greedy shift-reduce strategy. However, it did not perform on par with the best generative parsers of the time. Costa *et al.* Costa et al. [2002] introduced a

left-to-right incremental parser which used a recursive neural network to re-rank possible phrase attachments. They showed that RNN was able to capture enough information to make correct parsing decisions. Their system was, however, only tested on a subset of 2000 sentences. One had to wait a few more years before discriminative parsers could match Collins' parser performance. To this extent , Taskar *et al.* Taskar et al. [2004] proposed an approach which discriminates the entire space of parse trees, with a max margin criterion applied to Context Free Grammars. Other discriminative approaches Henderson [2004], Charniak and Johnson [2005] also outperformed standard PCFG-based generative parsers, but only by discriminatively re-ranking the K-best predicted trees coming from a generative parser.

Turian and Melamed Turian and Melamed [2006] later proposed a bottom-up greedy algorithm to construct the parse tree, using a feature boosting approach. The parsing is performed following a left-to-right or a right-to left strategy. The greedy decisions regarding the tree construction are made using decision tree classifiers. However, both of these parsers were limited to sentences of less than 15 words, due to a training time growing exponentially with the size of the input.

McClosky *et al.* McClosky et al. [2006] successfully leveraged unlabeled data to train a parser using a self-training technique. In this approach, a re-ranker is trained over a generative model. The re-ranker is used to generate "labels" over a large unlabeled corpus. These "labels" are then used to retrain the original generative model. This work is currently considered the state-of-the-art in syntactic parsing.

Most recent discriminative parsers Finkel et al. [2008], Petrov and Klein [2008] rely on Conditional Random Fields (CRFs) with PCFG-like features. Carreras *et al.* Carreras et al. [2008] used a global-linear model (instead of a CRF) with PCFGs and various new advanced features.

While PCFG-based parsers are widely used, other approaches do exist. In Collobert [2011], the proposed parser relies on continuous word vector representations, and a discriminative model to predict "levels" of the syntactic tree. Socher *et al.* Socher et al. [2013a] also relies on continuous word vector representations, which are "compressed" in a pairwise manner to form higher level chunk representations. Their approach is used as a re-ranker of the Stanford Parser Klein and Manning [2003].

Finally, it is worth noting that generative parsers are still evolving. PCFGs with latent-variables Matsuzaki et al. [2005] have been used in various ways to improve the performance of classical PCFG as in Cohen and Collins [2012].

In this paper, we propose a greedy and purely discriminative parsing approach. In contrast with most existing methods, it relies on a few simple features. The core of our architecture is a simple neural network which is fed with continuous word vector representations (as in Collobert and Weston [2008], Socher et al. [2013a]). It models the conditional distributions of *context-aware* syntactic rules. The learned distribution rules are naturally smoothed, thanks to the continuous nature of the input features.

Section 5.7 introduces our algorithm and relates it to PCFG-based parsers. Section 5.8 describes the classification model at the core of our architecture. Section 6.7 reports experimental comparisons with existing approaches. We conclude in Section 5.10.

5.7 A greedy discriminative parser

5.7.1 Smoothed Context Rule Learning

PCFG-based parsers rely on the statistical modeling of rules of the form $A \rightarrow B, C$, where A, B and C are tree nodes. The context-free grammar is always normalized in the Chomsky Normal Form (CNF) to make the global tree inference practical (with a dynamic programming like CYK or similar). In general a tree node is represented as several features, including for example its own parsing tags and head word (for non-terminal nodes) or word and Part Of Speech (POS) tag (for terminal nodes) Collins [2003]. State-of-the-art parsers rely on a judicious blending of carefully chosen features and regularization: adding features in PCFG rules might resolve some ambiguities, but at the cost of sparser occurrences of those rules. In that respect, the learned distributions must be carefully smoothed so that the model can generalize on unseen data. Some parsers also leverage other types of features (such as bigram or trigram dependencies between words Carreras et al. [2008]) to capture additional regularities in the data.

In contrast, our system models non-CNF rules of the form $A \rightarrow B_1, ..., B_K$. The score of each rule is determined by looking at a large context of tree nodes. More formally, we learn a classifier of the form:

$$f(C_{left}, B_1, ..., B_K, C_{right}) = (s_1, ..., s_{|\mathcal{T}|}) \tag{5.1}$$

where the B_k are either terminal or non-terminal nodes, K is the size of the right part of the rule, C_{left} and C_{right} are context terminals or non-terminals and s_t is the score for the parsing tag $t \in \mathcal{T}$. Each possible rule $A_i \rightarrow B_1, ..., B_K$ is thus assigned a score s_i by the classifier (with $A_i \in \mathcal{T}$). These scores can be interpreted as probabilities by performing a softmax operation. We used a Multi Layer Perceptron (MLP) as classifier. Formal details will be given in Section 5.8.2.

The only tree node features considered in our system are parsing tags (or POS tags for terminals), as well as the headword (or words for terminals). We overcome the problem of data sparsity which occurs in most classical parsers by leveraging *continuous vector representations* for all features associated to each tree node. In particular, word (or headword) representations are derived from recent distributed representations computed on large unlabeled corpora (such as Collobert and Weston [2008], Dhillon et al. [2011]). Thanks to this approach, our system can naturally generalize a rule like $NP \rightarrow a, clever, guy$ to a possibly unseen rule like $NP \rightarrow a, smart, guy$, as the vector representation of $smart$ and $clever$ are close to each

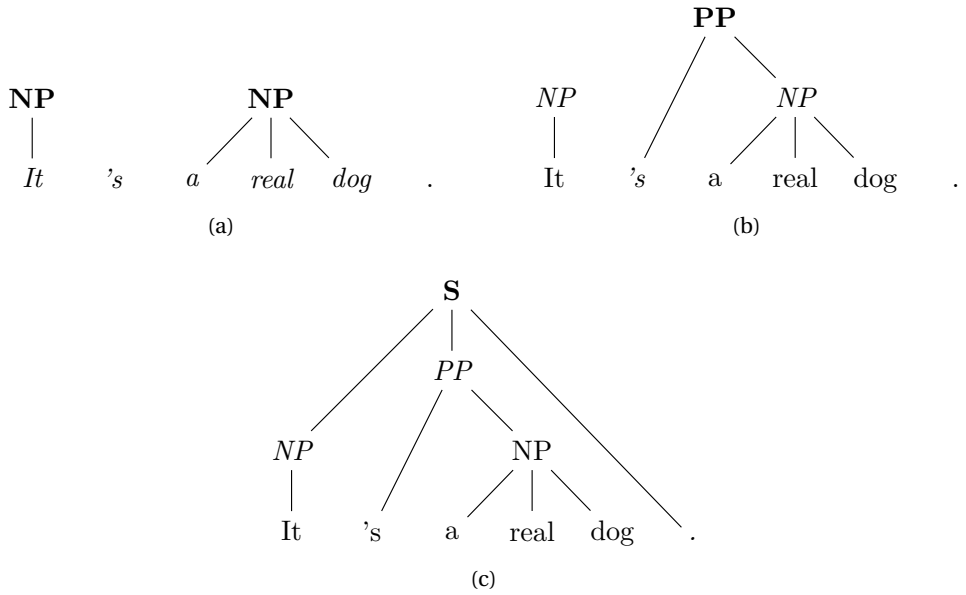

Figure 5.1: Illustration of our greedy algorithm: at each iteration (a)→(b)→(c), the classifier sees only the previous tree heads (ancestors), shown here in italics. It predicts new nodes (here in bold). New tree heads become the ancestors at the next iteration. All other previously discovered tree nodes (shown in regular black here) will remain unchanged and ignored in subsequent iterations.

other, given that they are semantically and syntactically related.

Several works leveraging continuous vector representations have been previously proposed for syntactic parsing. Collobert [2011] introduced a neural network-based approach, iteratively tagging "levels" of the parse tree where the full sentence was seen at each level. A complex pooling approach was introduced to capture long-range dependencies, and performance only matched early lexicalized parsers. Socher et al. [2011b] introduced a recursive approach, where representations are "compressed" two by two to form higher-level representations. However, the system was limited to bracketing, and did not produce parsing tags. The authors later proposed an improved version in Socher et al. [2013a], where their approach was used to re-rank the output of the Stanford Parser, approximately reaching state-of-the-art performance. In contrast, our approach does not rely on CNF grammars and does not re-rank an external generative parser.

5.7.2 Greedy Recurrent Algorithm

Our parser follows a bottom-up iterative approach: the tree is constructed starting from the terminal nodes (sentence words). Assuming that a part of the tree has been already predicted (see Figure 6.1), the next iteration of our algorithm looks for all possible new tree nodes

which combine ancestors (i.e., heads of the trees predicted so far). New nodes are found by maximizing the score of our context-rule classifier (5.1), constrained in such a way so that two new nodes cannot overlap, thanks to a dynamic programming approach. The system is recurrent, in the sense that new predicted parsing labels are used in the next iteration of our algorithm.

For each iteration, assuming N ancestors

$$X = [X_1, ..., X_N],$$

finding all possible new nodes with K ancestors would require to apply

$$f(C_{left}, B_1, ..., B_K, C_{right})$$

over all possible windows of K ancestors in X. One would also have to vary K from 1 to N, to discover new nodes of all possible sizes. Obviously, this could quickly become time consuming for large sentence sizes. This problem of finding nodes with a various number of ancestors can be viewed as the classical NLP problem of finding "chunks" of various sizes. This problem is typically transformed into a tagging task: finding the chunk with label A in the rule $A \rightarrow X_i, X_{i+1}, \ldots, X_j$ can simply be viewed as tagging the ancestors with B-A, I-A, $\ldots E$-A, where we use the standard BIOES label prefixing (B*egin*, I*ntermediate*, O*ther*, E*nd*, S*ingle*). See Table 5.1 for a concrete example. The classifier outputs the "O*ther*" tag, when the considered ancestors do not correspond to any possible rule.

In the end, our approach can be summarized as the following iterative algorithm:

1. Apply a sliding window over the current ancestors: the neural network classifier (5.1) is applied over all K consecutive ancestors $X_1, ..., X_N$, where K has to be carefully tuned.

2. Aggregate BIOES tags into chunks: a dynamic program (based on a CRF, as detailed in Section 5.8.3) finds the most likely sequence of BIOES parsing tags. The new nodes are then constructed by simply aggregating BIES tags

   $$B\text{-}A, I\text{-}A, \ldots E\text{-}A$$

   into A (for any label A).

3. Ancestors tagged as O, as well as newly found tree nodes are passed as ancestors to the next iteration.

The tree construction ends when there is only one ancestor remaining, or when the classifier did not find any new possible rule (everything is tagged as O).

Table 5.1: A simple example of a grammar rule extracted from the sentence "*It 's a real dog .*", and its corresponding BIOES grammar. In both cases, we include a left and right context of size 1. The middle column shows the required classifier evaluations. The right column shows the type of scores produced by the classifier.

GRAMMAR	CLASSIFIER EVALUATIONS	SCORES
NP → 'S **A REAL DOG .**	f('S, **A**, **REAL**, **DOG**, .)	$s_{NP}, ..., s_{VP}, s_O$
B-NP → 'S **A REAL**	f('S, **A**, **REAL**)	$s_{B\text{-}NP}, ..., s_{E\text{-}VP}, s_O$
I-NP → **A REAL DOG**	f(**A**, **REAL**, **DOG**)	
E-NP → **REAL DOG .**	f(**REAL**, **DOG**, .)	

5.8 Architecture

In this section, we formally introduce the classification architecture used to find new tree nodes at each iteration of our greedy recurrent approach. A simple two-layer neural network is at the core of the system. It leverages continuous vector word representations. In this respect, the network is clearly inspired by the work of Bengio et al. [2000] in the context of language modeling, and later re-introduced in Collobert et al. [2011] for various NLP tagging tasks.

Given an input sequence of N tree node ancestors $X_1, ..., X_N$ (as defined in Section 5.7.2), our model outputs a BIOES-prefixed parsing tag for each ancestor X_i, by applying a sliding window approach. These scores are then fed as input to a properly constrained graph on which we apply the Viterbi algorithm to infer the best sequence of parsing tags. The whole architecture (including transition scores in the graph) is trained in an end-to-end manner by maximizing the graph likelihood. The end-to-end neural network training approach was first introduced in Denker and Burges [1995]. The system can be also viewed as a particular Graph Transformer Network Bottou et al. [1997], or a particular *non-linear* Conditional Random Field (CRF) for sequences Lafferty et al. [2001]. Each layer of the architecture is presented in detail in the following paragraphs. The objective function will be introduced in Section 6.6.5.

5.8.1 Words Embeddings

Our system relies on *raw words*, following the idea of Collobert and Weston [2008]. Each word is mapped into a continuous vector space. For efficiency, words are fed into our architecture as indices taken from a finite dictionary $\mathcal{W}$. Word vector representations, as other network parameters, are trained by back-propagation.

More formally, given a sentence of N words, $w_1, w_2, ..., w_N$, each word $w_n \in \mathcal{W}$ is first embedded in a D-dimensional vector space by applying a lookup-table operation:

$$LT_W(w_n) = W_{w_n},$$

where the matrix $W \in \mathbb{R}^{D \times |\mathcal{W}|}$ represents the parameters to be trained in this lookup layer. Each column $W_n \in \mathbb{R}^D$ corresponds to the vector embedding of the n^{th} word in our dictionary $\mathcal{W}$.

These types of architectures allow us to take advantage of word vector representations trained on large unlabeled corpora, by simply initializing the word lookup table with a pre-trained representation Collobert and Weston [2008]. In this paper, we chose to use the representations from Lebret and Collobert [2014], obtained by a simple PCA on a matrix of word co-occurrences. As shown in Collobert [2011] for various NLP tasks, we will see that these representations can provide a great boost in parsing performance.

In practice, it is common to give several features (for each tree node) as input to the network. This can be easily done by adding a different lookup table for each discrete feature. The input becomes the concatenation of the outputs of all these lookup-tables:

$$\begin{aligned} LT_{W_1,...,W_K}(w_n) = &(LT_{W_1}(w_n))^T, \\ &..., \\ &(LT_{W_{|\mathcal{F}|}}(w_n))^T \end{aligned}$$

where $|\mathcal{F}|$ is the number of features. For simplicity, we consider only one lookup-table in the rest of the architecture description.

5.8.2 Sliding Window BIOES Tagger

The second module of our architecture is a simple neural network which applies a sliding window over the output of the lookup tables, as shown in Figure 5.2. The n^{th} window is defined as

$$u_n = [LT(X_{n-\frac{K-1}{2}}), ..., LT(X_n), ..., LT(X_{n+\frac{K-1}{2}})]\,,$$

where K is the size of window. The module outputs a vector of scores $s(u_n) = [s_1, ..., s_{|\mathcal{T}|}]$ (where s_t is the score of the BIOES-prefixed parsing tag $t \in \mathcal{T}$ for the ancestor X_n). The ancestors with indices exceeding the input boundaries ($n-(K-1)/2 < 1$ or $n+(K-1)/2 > N$) are mapped to a special padding vector (which is also learned). As any classical two-layer neural network, our architecture performs several matrix-vector operations on its inputs, interleaved with some non-linear transfer function $h(\cdot)$,

$$s(u_n) = M_2\, h(M_1\, u_n)\,,$$

where the matrices $M_1 \in \mathbb{R}^{H \times K|D|}$ and $M_1 \in \mathbb{R}^{|\mathcal{T}| \times H}$ are the trained parameters of the network. The number of hidden units H is a hyper-parameter to be tuned.

As transfer function, we chose in our experiments a (fast) "hard" version of the hyperbolic

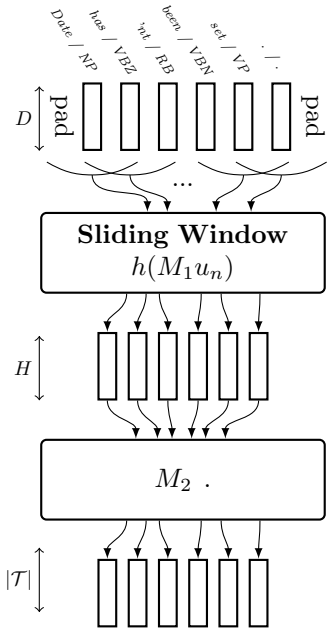

Figure 5.2: Sliding window tagger. Given the concatenated output of lookup tables (here the ancestor words/headwords and ancestor tags), the tagger outputs a BIOES-prefixed parsing tag for each ancestor node. The neural network itself is a standard two-layer neural network.

tangent:

$$h(x) = \begin{cases} -1 & \text{if} \quad x < -1 \\ x & \text{if} \quad -1 <= x <= 1 \\ 1 & \text{if} \quad x > 1 \end{cases} \tag{5.2}$$

5.8.3 Aggregating BIOES Predictions

The scores obtained from the previous module of our architecture are in BIOES format. The next module in our system aggregates these tags and finds the new tree nodes at each iteration of our greedy recurrent approach. We introduce a graph G of scores as shown in Figure 5.3: each node of the graph corresponds to a BIOES score produced for each ancestor by the neural network module. This graph is constrained in such a way that only feasible sequences of tags are possible (e.g. B-A tags can only be followed by I-A tags, for any parsing label A). Our graph also includes a duration model: on each edge, we add a transition score $A_{tt'}$ for jumping from tag $t \in \mathcal{T}$ to $t' \in \mathcal{T}$.

A score for a sequence of tags $[t]_1^N$ in the lattice G is obtained as the sum of scores along $[t]_1^N$

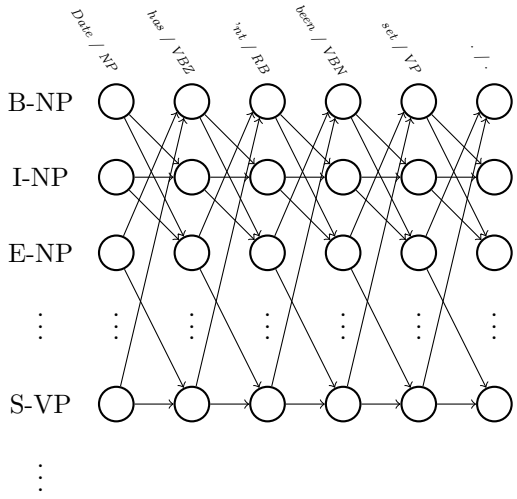

Figure 5.3: Constrained graph for tag inference. Only feasible sequences of tags are considered. The nodes of the graph are assigned a score from the tagger shown in Figure 5.2. Edges of the graph are assigned a transition score which is learned similarly to other parameters in the architecture.

in G:

$$S([t]_1^N, [u]_1^N, \theta) = \sum_{n=1}^{N} \left(A_{t_{n-1} t_n} + s(u_n)_{t_n} \right),$$

where θ represents all the trainable parameters of the complete architecture. The sequence of tags $[t^*]_1^N$ for the input sequence of tree node ancestors $X_1, \ldots, X_N$ is then inferred by finding the path which leads to the maximal score:

$$[t^*]_1^N = \underset{[t]_1^N \in \mathcal{T}^N}{\operatorname{argmax}} S([t]_1^N, [u]_1^N, \theta)$$

The Viterbi algorithm is the natural choice for this inference. From this optimal BIOES tag sequence, we extract sub-sequences B-A, …, E-A and S-A as new nodes for the tree. O tags are simply ignored. See Section 5.7.2 for more details.

5.8.4 Training Likelihood

Our architecture sees sequences of ancestor tree nodes, and outputs new possible syntactic tree nodes only from this history. Technically speaking, the training set can be prepared by iterating over each tree in the training corpus, removing all possible leaves in an iterative process so that all training rules are uncovered (see Figure 5.4).

The neural network is trained by maximizing a likelihood over the training data, using stochas-

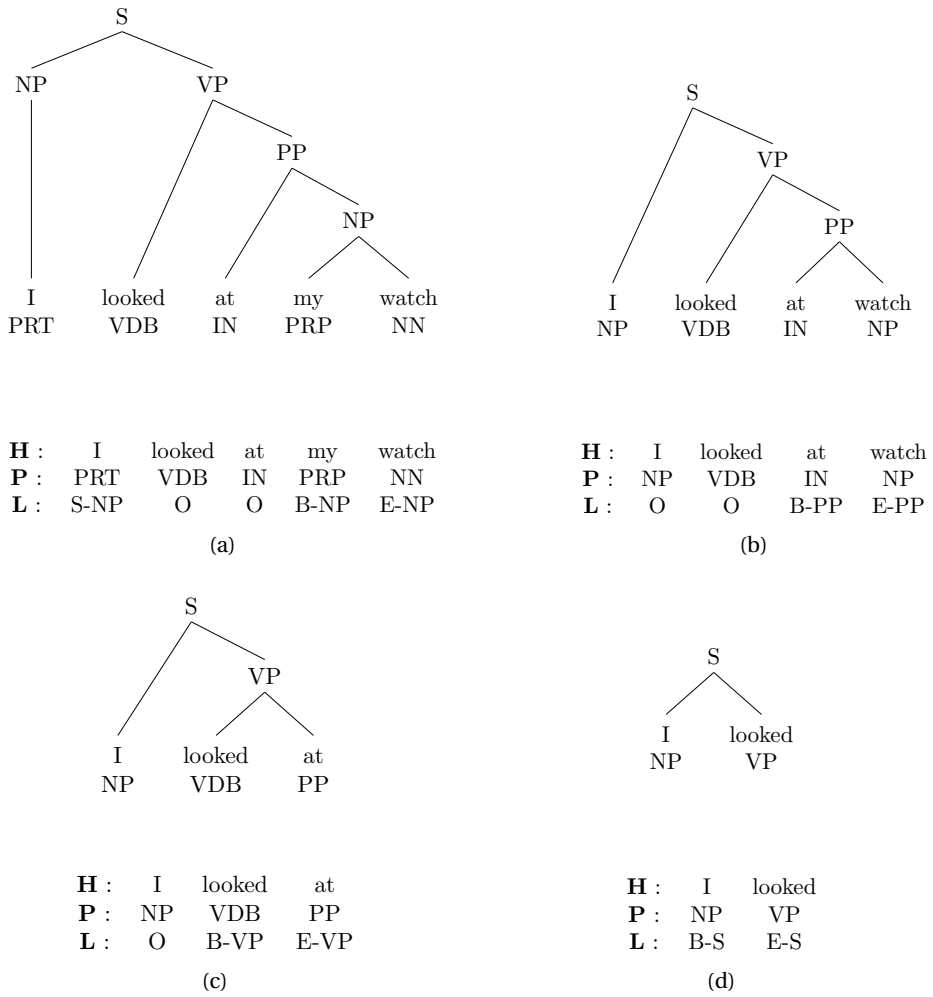

Figure 5.4: Iterative procedure (a)→(b)→(c)→(d) to generate the training data, which involves cutting out all tree leaves at each step. The data fed to our network architecture is then easily uncovered (H: ancestor headwords/words, P: ancestor POS/parsing tags, L: parsing labels to be predicted).

tic gradient ascent. The score for a path can be interpreted as a conditional probability over this path by exponentiating score (thus making it positive) and normalizing it with respect to all possible paths. We define $\mathcal{P}$ as the set of possible tag paths in the constrained graph G, as shown in Figure 5.3. The log-probability of a sequence of tags $[t]_1^N$ given the lookup table representations $[u]_1^N$ is given by:

$$\begin{aligned}\log P([t]_1^N|[u]_1^N,\theta) = & S([t]_1^N,[u]_1^N,\theta) \\ & - \underset{\forall [t']_1^N \in \mathcal{P}}{\text{logadd}}\, S([t']_1^N,[u]_1^N,\theta))\end{aligned} \tag{5.3}$$

where we adopt the notation $\text{logadd}_{z_n} = \log(\sum_i e^{z_i})$, as in Collobert et al. [2011].

Computing the log-likelihood efficiently is not straightforward, as the number of terms in the logadd grows exponentially with the length of the sentence. Fortunately, it can be computed in linear time with the Forward algorithm, which derives a recursion similar to the Viterbi algorithm (see Rabiner [1989]). The complete architecture is trained by simply backpropagating through this recursion, up to the lookup layers (for further details, see Collobert [2011]). Note that the likelihood (6.1) corresponds to a standard CRF model for sequences. The only difference here is that the underlying model is non-linear, while CRFs are often considered as linear models.

5.9 Experiments

5.9.1 Corpus

Experiments were performed using the standard English Penn Treebank data set (Marcus et al., 1993). We used the classical parsing setup, with sections 02-21 used to train our model, section 22 used as validation for choosing all our hyper-parameters, and section 23 used for testing. We applied only a small subset of the typical pre-processing set over the data: (1) functional labels, traces were removed, (2) the PRT label was replaced as ADVP Magerman [1995].

The Penn Treebank data set contains non-terminal tree nodes which only have one non-terminal child, as shown in Figure 5.5. To avoid possible looping issues in our parsing algorithm (e.g. a node being repetitively tagged with two different tags in our iterative process), we transformed the *training* corpus so that non-terminal nodes having only one non-terminal child were merged together, and take as tag the concatenation of all merged node tags (see Figure 5.5). This way, the system learns that a node must contain at least two ancestors. The iterative process is thus guaranteed to converge. We kept only concatenated labels which occurred at least 30 times (corresponding to the lowest number of occurrences of the less common original parsing tag), leading to 11 additional parsing tags. Added to the original 26 parsing tags, this resulted in 161 tags produced by our parser. At test time, the inverse operation is performed: concatenated tag nodes are simply expanded into their original form.

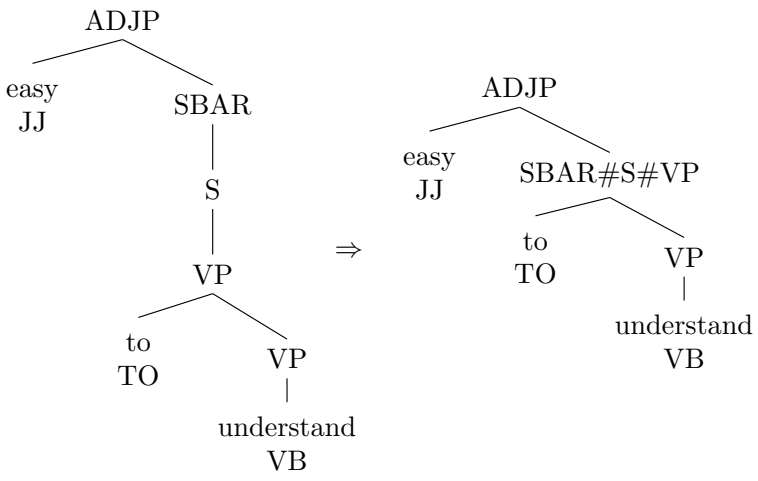

Figure 5.5: Training corpus pre-processing. Original Penn Treebank trees containing non-terminal nodes with only one non-terminal node (left), and after concatenating those nodes (right).

5.9.2 Features

We consider the following features to train our architecture:

- Words and headwords:
  - For terminal nodes, the word itself, in low caps[1]. As in Collins [2003], words occurring 5 times or less were mapped to an "UNKNOWN" word.
  - For non-terminal nodes: headwords, following the procedure described in Collins [2003].
- POS tags (for terminals) or parsing tags of the node's ancestors (for non-terminals). POS tags were produced with SENNA Collobert et al. [2011].
- POS tags of headwords.

5.9.3 Results

We train the network using stochastic gradient descent over the available training data, until convergence on the validation set. We chose the following hyper-parameters according to the validation. Lookup-table sizes for the words and tags (part-of-speech and parsing) are 100 and 20, respectively. The window size for the tagger is $K = 7$ (3 neighbors from each side). The size of the tagger's hidden layer is $H = 500$. We used the word embeddings obtained from

[1] Adding a capital feature had no impact on the performance of our parser. Note that POS tags were generated with the original caps in the sentence.

Table 5.2: Influence of different features. Results are given in terms of F1-score. POS = part-of-speech, hw = head-word, wi = word initialization from Lebret and Collobert [2014].

FEATURE	F1
WORD + POS	85.1
WORD + POS + HW	86.9
WORD + POS + WI	86.2
WORD + POS + HW + WI	88.3

Lebret and Collobert [2014] to initialize the word lookup-table. These embeddings were then fine-tuned during the training process. Finally, we fixed the learning rate to $\lambda = 0.025$ during the stochastic gradient procedure. The only "trick" used during training was to divide the learning rate by the input size of each linear layer Plaut and Hinton [1987].

Table 5.2 shows the importance of the different features we used. Even though the training procedure is non-convex, the variance of the F1 score over 20 different runs (for the architecture Word + POS + hw + wi) was only 0.01.

Since our architecture performs the decoding very quickly, we additionally performed a voting procedure using several models learned from different random initializations. We averaged all neural network classifiers (ignoring their own respective CRF decoding part) and trained a new CRF on top of it (without fine-tuning any of the neural network classifiers). The scores obtained with 10 classifiers are shown in Table 6.1.

Results in Table 6.1 are reported in terms of recall (R), precision (P) and F1 score. Scores were obtained using the Evalb implementation[2]. We compare our system with several other parsers. We chose to report the scores of the three main generative parsers, as well as those of known re-ranking parsers. We also considered two major purely discriminative parsers.

5.9.4 Rule Prediction Analysis

Figure 5.6 shows the output of the classifier (applied on every possible window of size 7) for the sentence "When the little guy gets frightened, the big guys hurt badly.". For this sentence, the expected rule are the following:

WHADVP →	When
NP →	the little guy
ADJP →	frightened
NP →	the big guys
ADVP →	badly

[2]Available at http://nlp.cs.nyu.edu/evalb/

Table 5.3: Results in terms of Precision (P), Recall (R), and F1 score. The reported time is the time to parse the full WSJ test corpus.

	Model	(R)	(P)	F1	(R)	(P)	F1	time
	Magerman (1995)	84.6	84.9	84.8				
Generative	Collins (1999)	88.5	88.7	88.6	88.1	88.3	88.2	1247
	Charniak (2000)	90.1	90.1	90.1	89.6	89.5	89.6	
Generative	Henderson (2004)				89.8	90.4	90.1	
with	Charniak and Johnson (2005)			92.0			91.1	
re-ranking	Socher et al (2013)			91.1			90.4	390
	McClosky et al (2006)						92.1	
	Petrov and Klein (2008)			90.0			89.4	
purely	Carreras et al. (2008)				90.7	91.4	91.1	
discriminative	Our Model	88.4	89.0	88.7	88.0	88.6	88.3	110
	Our Model (voting)	90.0	90.1	90.1	89.6	89.7	89.6	

It is interesting to see that the network alone is able to predict all the rules of the sentence. The CRF is however essential to produce a consistent output, by aggregating BIES prefixed chunks.

5.10 Conclusion

We presented a very simple model that is able to learn syntactic grammar rules surprisingly well, considering the simple features employed. This parser achieves performance very close to state-of-the-art re-ranking systems and is almost the best among the purely generative parsers. Due to its simplicity, there are many possibilities for further improvement. In particular, the head-word procedure from Collins could be revisited, e.g. by learning a higher-level chunk representation in the same spirit as Socher et al. [2013a]. We could also investigate re-ranking approaches, as well as the use of unlabeled corpora.

Taskar et al. [2004]

Taskar et al. [2004]

[Taskar et al., 2004]

content: Legrand and Collobert [2014] (https://publidiap.idiap.ch/downloads/papers/2014/Legrand_ECML2014_2014.pdf)

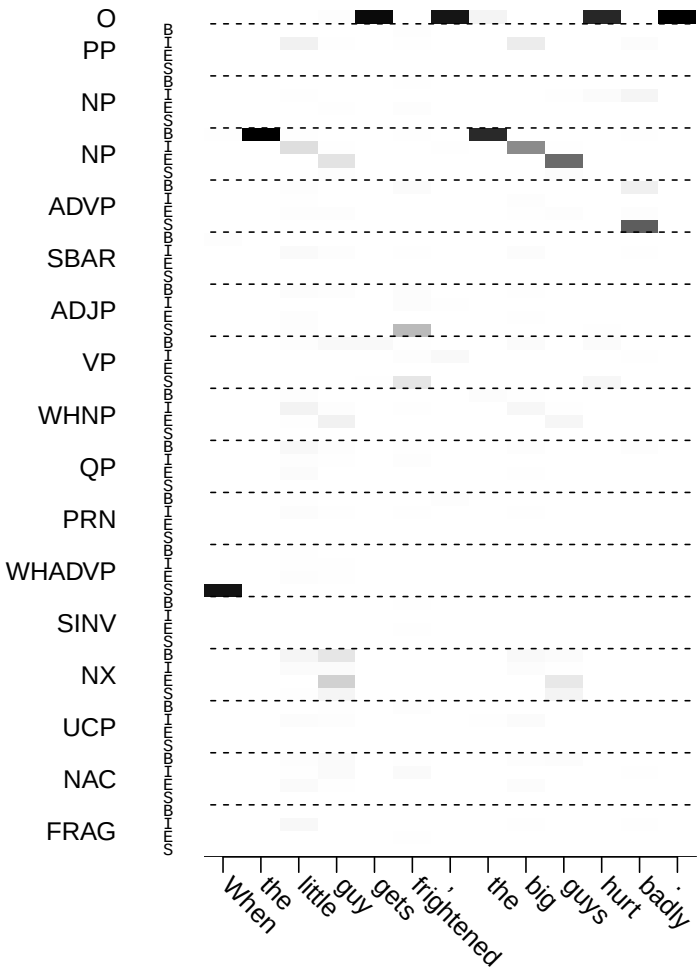

Figure 5.6: Normalized scores from the network classifier (black means high score) for the sentence "When the little guy gets frightened, the big guys hurt badly.". Each tag is in BIOES form (y axis). Each ancestor in the input is on the x axis.

6 RNN-Based Phrase Embeddings for Syntactic Parsing

6.1 Introduction

6.2 Related Work

6.3 Joint RNN-Based Greedy Parsing and Word Composition

6.4 Introduction

In Natural Language Processing (NLP), the parsing task aims at analysing the underlying syntactic structure of a natural language sequence of words (a sentence). The analysis is expressed as a tree of syntactic relations between sub-constituents of the sentence. In the linguistic world, Chomsky [1956] first introduced formally the parsing task, by defining the natural language syntax as a set of context-free grammar rules (a particular type of formal grammar), combined with transformations rules. Automated syntactic parsing became rapidly a key task in computational linguistic. A parse tree not only carries syntax information, but might also embed some semantic information (in the sense that it can disambiguate different interpretations of a given sentence). In that respect, parsing it has been widely used as an input feature for several other NLP tasks such as machine translation [Zollmann and Venugopal, 2006], information retrieval [Alonso et al., 2002], or Semantic Role Labeling [Punyakanok et al., 2008].

This paper introduces a greedy parser which leverages a new composition approach to keep an history of what has been predicted so far. The composition performs a syntactic and semantic summary of the contents of a sub-tree in the form of a vector representation. The composition is performed along the tree: bottom tree node representations are obtained by composing *continuous word vector representations*, and produces vector representations which are in turn composed together in subsequent nodes of the tree. The composition operation as well as tree node tagging and predictions are achieved with a Recurrent Neural Network (RNN). Both the composition and node prediction are trained *jointly*.

Section 6.5 presents several related approaches. Section 6.6 details our parsing architecture. An empirical evaluation of our models as well as our compositional vectors is given in Section 6.7.

6.5 Related Work

The first attempts to automatically parse natural language were mainly conducted using generative models. A wide range of parser were, and still are, based on Probabilistic context-free grammar (PCFGs) [Magerman, 1995, Collins, 2003, Charniak, 2000]. These types of parsers model the syntactic grammar by computing statistics of simple grammar rules (over parsing tags) occurring in a training corpus. However, many language ambiguities cannot be caught with simple tag-based PCFG rules. A key element in the success of PCFGs is to refine the rules with a word lexicon. This is usually achieved by attaching to PCFGs a lexical information called the *head-word*. Several head-word variants exist, but they all rely on a deterministic procedure which leverages clever linguistic knowledge. Parsing inference is mostly achieved using simple bottom-up chart parser [Kasami, 1965, Earley, 1970, Kay, 1986]. These methods face a classical learning dilemma: on one hand PCFG rules have to be refined enough to avoid any ambiguities in the prediction. On the other hand, too much refinement in these rules implies lower occurrences in the training set, and thus a possible generalization issue. PCFGs-based parsers are thus judiciously composed with carefully chosen PCFG rules and clever regularization tricks.

6.5.1 State-Of-The-Art

Discriminative approaches from Henderson [2004], Charniak and Johnson [2005] outperform standard PCFG-based generative parsers, but only by discriminatively *re-ranking* the K-best predicted trees coming out of a generative parser. To our knowledge, the state of the art in syntactic parsing is still held by McClosky et al. [2006], who leverages discriminative re-ranking, as well as self-training over unlabeled corpora: a re-ranker is trained over a generative model which is then used to label the unlabeled dataset. The original parser is then re-trained with this new "labeled" corpus. Petrov and Klein [2007] introduced a method to automaticaly refine PCFG rules by iteratively spliting them. This method leverages an efficient coarse-to-fine procedure to speed up the decoding process. More recently, Finkel et al. [2008], Petrov and Klein [2008] proposed PCFG-based *discriminative* parsers reaching the performance of their generative counterparts. Conditional Random Fields (CRFs) are at the core of such approaches. Carreras et al. [2008] currently holds the state-of-the-art among the (non-reranking) discriminative parsers. Their parser leverages a global-linear model (instead of a CRF) with PCFGs, together with various new advanced features. Huang et al. [2010] showed that jointly using multiple self-trained grammars can achieve higher accuracy than an individual grammar.

In contrast to these existing approaches, our parser does not rely on PCFGs, nor on refined features like head-words. Tagging nodes is achieved in a greedy manner, using only raw words and part-of-speech (POS) as features. Tree node history is maintained as a vector

representation obtained in a recurrent fashion, by composing past node representations and tag predictions.

6.5.2 Greedy Parsing

Many discriminative parsers follows a greedy strategy because of the lack (or the intractability) of a global tree score for an entire derivation path which would combine independent node decisions. Adopting a greedy strategy that maximize local scores for individual decisions is then a solution worth investigating. One of the first successful discriminative parsers [Ratnaparkhi, 1999] was based on MaxEnt classifiers (trained over a large number of different features) and powered a greedy shift-reduce strategy.

Henderson [2003] introduced a generative left-corner parser where the probability of a derivation given the derivation historic was approximated using a Simple Synchrony Networks (SNN), which is a neural network specifically designed for processing structures.

Turian and Melamed [2006] later proposed a bottom-up greedy algorithm following a left-to-right or a right-to left strategy and using using a feature boosting approach. In this approach, greedy decisions regarding the tree construction are made using decision tree classifiers. Their model was nevertheless limited to short length sentences.

Zhu et al. [2013] proposed a shift-reduce parser which achieves results comparable to their chart-based counterparts. This is done by leveraging several unsupervisely trained features (word Brown clustering, dependency relations, dependency language model) combined with a smart beam search strategy.

6.5.3 Parsing With Recurrent Neural Networks

Recurrent Neural Networks (RNNs) were seen very early [Elman, 1991] as a way to tackle the problem of parsing, as they can naturally recur along the parse tree. A first practical application of RNN on syntactic parsing were proposed by Costa et al. [2002]. Their approach was based on a left-to-right incremental parser, where a recursive neural network was used to re-rank possible phrase attachments. The goal of their contribution was, in their own terms, the assessment of a methodology rather than a fully functional system. They demonstrated that RNNs were able to capture enough information to make correct parsing decisions.

Collobert [2011] proposed a purely discriminative parser based on neural networks. This model leveraged continuous vector representations from Collobert and Weston [2008], and builds the full parsing tree in a bottom-up manner. To deal with the recursive structure inherent to syntactic parsing, a very simple history was given to the network as a new vector feature (corresponding to the nearest tag spanning the word being tagged).

Socher et al. [2011a] also leveraged continuous vectors from Collobert and Weston [2008], combining them to build a tree in a greedy manner. However, this work did not tackle the full parse tree problem, but was restricted to unlabeled bracketing. Socher et al. [2013a] introduced the Compositional Vector Grammar (CVG) which combines PCFGs with a Syntactically Untied Recursive Neural Network (SU-RNN). Composition is performed over a binary tree, then used to score the K-best trees coming out of a generative parser. For a given (parent) node of the tree, the authors apply a composition operation over its child nodes, conditioned with their syntax information. In contrast, we compose phrases (not limited to two words). Both the words and syntax information of the child nodes are fed to each composition operation, leading to a vector representation of each tree node carrying both some semantic and syntactic information. We also do not rely on any generative parser as our model *jointly* trains the task of node prediction, and the task of node composition.

Legrand and Collobert [2014] proposed a greedy RNN-based parser. The neural network was recurrent only in the sense it used previously predicted tags to produce next tree node tags. Contrary to Socher et al. [2013a], it did not involve composing sub-tree representations. Instead, head-words were used as a key feature. Our approach shares some similarities with [Legrand and Collobert, 2014], as it is also a greedy parser based on RNNs. However, instead of relying on head-words (which could be seen as a simplistic representations of sub-trees), we leverage compositional sub-tree vector representations trained jointly with the parser. This approach leads to much better parsing accuracy, while relying only on a few simple features (words and POS tags). Our model has also the ability of producing phrase embeddings, which may represent a valuable feature for other NLP tasks.

Chen and Manning [2014] proposed a greedy transition-based dependency parser based on neural networks, fed with dense word and tag vector representations. In contrast to our approach, it does not integrate a compositional procedure over sentence sub-trees. The network is only involved in predicting correct transitions at each step of the parsing process.

6.6 Greedy RNN Parsing

Our parser is based on a neural network tagger, and performs parsing in a greedy recurrent way. Our approach is a bottom-up iterative procedure: the tree is constructed starting from the terminal nodes (sentence words), as shown in Figure 6.1. At each iteration,

1. We look for all possible new tree nodes merging input constituents (i.e., heads of the trees predicted so far or leaves which have not been composed so far). For that purpose, we apply a neural network (see Figure 6.3) sliding window tagger over input constituents $X_1, \ldots, X_N$. Considering an arbitrary rule

$$A \rightarrow X_i, X_{i+1}, \ldots, X_j$$

   defining a new node with tag A, the tagger will produce prefixed tags B-A, I-A, $\ldots E$-A, re-

$\mathbf{I}_W$	:	Look	around	and	choose	your	own	ground	.
$\mathbf{I}_T$	:	VB	RP	CC	VB	PRP$	JJ	NN	.
$\mathbf{O}$	:	O	S-PRT	O	O	B-NP	I-NP	E-NP	O

$\mathbf{I}_W$	:	Look	*R1*	and	choose	*R2*	.
$\mathbf{I}_T$	:	VB	PRT	CC	VB	NP	.
$\mathbf{O}$	:	B-VP	E-VP	O	B-VP	E-VP	O

$\mathbf{I}_W$	:	*R3*	and	*R4*	.
$\mathbf{I}_T$	:	VP	CC	VP	.
$\mathbf{O}$	:	B-VP	I-VP	E-VP	O

$\mathbf{I}_W$	:	*R5*	.
$\mathbf{I}_T$	:	VP	.
$\mathbf{O}$	:	B-S	E-S

Figure 6.1: Greedy parsing algorithm, on the sentence "Look around and choose your own ground.". $\mathbf{I}_W$, $\mathbf{I}_T$ and $\mathbf{O}$ stand for input words (or composed word representations R_i), input syntactic tags (parsing or part-of-speech) and output tags (parsing), respectively. See Figure 6.2 and Section 6.6.2 for the word composition procedure. The tree produced after 4 greedy iterations (as shown here) can be reconstructed as the following: `(S (VP (VP (VB Look) (PRT (RP around))) (CC and) (VP (VB choose) (NP (PRP$ your) (JJ own) (NN ground)))) (. .)).`

spectively for constituents X_i, X_{i+1}, ..., X_j, following a classical BIOES prefixing scheme[1].

2. A simple dynamic programming is performed, only to insure the coherence of the tag prediction (e.g., a B-A can be followed only by a I-A or a E-A).

3. A (neural network) composition module computes vector representations of the new nodes, according to the representations of the merged constituents, as well as the tag predictions (see Figure 6.2).

4. New predicted nodes become input constituents and we go back to 1 (see Figure 6.1).

Our system is recurrent in two ways: newly predicted parsing node labels as well as vector representations obtained by composing these predicted nodes, are used in the next iteration of our algorithm.

We will detail our architecture in the following.

[1] B*egin*, I*ntermediate*, O*ther*, E*nd*, S*ingle*. This approach is very often used in NLP, when one wants to rewrite a chunk (here node) prediction problem into a word tagging problem.

6.6.1 Word Embeddings

Following the work from Collobert and Weston [2008] on various NLP tasks, our parser relies on *raw words*. Each word in a finite dictionary $\mathcal{W}$, is assigned a continuous vector representation. These representations as all parameters of our architecture are trained by back-propagation. More formally, given a sentence of N words, $w_1, w_2, ..., w_N$, each word $w_n \in \mathcal{W}$ is first embedded in a D-dimensional vector space by applying a lookup-table operation:

$$LT_W(w_n) = W_{w_n},$$

where the matrix $W \in \mathbb{R}^{D \times |\mathcal{W}|}$ represents the parameters to be trained in this lookup layer. Each column $W_n \in \mathbb{R}^D$ corresponds to the vector embedding of the n^{th} word in our dictionary $\mathcal{W}$.

In this work, two kind of features are used to feed the networks: words (or word compositions) and POS tags $\mathcal{T}$ (or parsing tags $\mathcal{P}$). As for words, a lookup-table associates each tag t in the finite set of tag $\mathcal{T} \cup \mathcal{P}$ with a continuous vector representation of size T. The output vectors of the different lookup-tables are simply concatenated to form the input of the next layer.

Using continuous word vectors as input allows us to take advantage of unsupervisely pretrained word embeddings. Lot of work on this domain has been done in recent year, including Collobert and Weston [2008], Mikolov et al. [2013]. In this paper, we chose to use the representations from Lebret and Collobert [2014], obtained by a simple PCA on a matrix of word co-occurrences.

6.6.2 Word-Tag Composition

At each step of the parsing procedure, we represents each node of the tree as a vector representation, which summarizes both the syntax (predicted POS or parsing tags) and the semantic (words) of the sub-tree corresponding to the given node. As shown in Figure 6.2, the vector representation is obtained by a simple recurrent procedure, which involves several components:

- Word vector representations *for the leaves* (coming out from a lookup table) (dimension D).
- Tag (POS for the leaves, predicted tags otherwise) vector representations (also coming out for another lookup table, as explained in Section 6.6.1) (dimension T).
- Compositional networks $C_k()$. Each of them can compress the representation of a chunk of size k into a D-dimensional vector.

Compositional networks take as input both the merged node representations and predicted tag representations. There is one different network C_k for each possible node with a number of

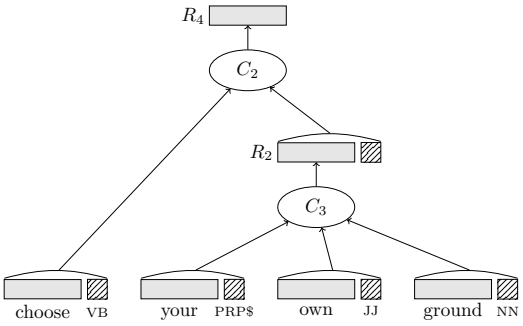

Figure 6.2: Recurrent composition of the sub-tree `(VP (VB choose) (NP (PRP$ your) (JJ own) (NN ground)))`. The representation R2 is first computed using the 3-inputs module C_3 with `your/PRP$ own/JJ ground/NN` as input. R4 is obtained by using the 2-inputs module C_2 with `choose/VB R1/NP` as input

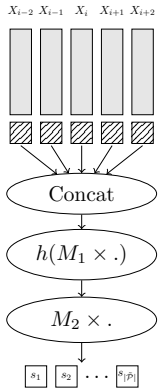

Figure 6.3: A constituent X_i is tagged by considering a fixed size context window of size K (here $K = 5$). The concatenated output of the compositional history and constituent tags is fed as input to the tagger. It outputs a score for each BIOES-prefixed parsing tag. The tagger is a standard two-layer neural network. Tags for the current sequence of constituents $X_1, \ldots, X_N$ is obtained by simply sliding this network over the sequence.

k merged constituent. In practice most tree nodes do not merge more than a few constituents[2]. In our case, denoting $z \in \mathbb{R}^{(D+T)\times k}$ the concatenation of the merged constituent representations (k vectors of tags and constituent representations), the compositional network is simply a matrix-vector operation followed by a non-linearity

$$C_k(z) = h(M^k z),$$

where $M^k \in \mathbb{R}^{D\times(k(D+T))}$ is a matrix of parameters to be trained, and $h()$ is a simple non-linearity such as a pointwise hyperbolic tangent.

Note that node and word representations are embedded in the same space. This way, the compositional networks C_k can compress indifferently information coming from leaves or sub-trees. Implementation-wise, one can store new node representations into the word lookup-table as the tree is created, such that subsequent composition or tagging operations can be achieved in an efficient manner.

[2] Taking $1 \le k \le 5$ covers already 98.6% of the nodes in the Wall Street Journal training corpus, and $1 \le k \le 7$ covers 99.8%.

6.6.3 Sliding Window BIOES Tagger

The tagging module of our architecture (see Figure 6.3) is a two-layer neural network which applies a sliding window over the input constituent representations (as computed in Section 6.6.2), as well as the input constituent tag representations. Considering N input constituents $X_1, \ldots, X_N$, if we assume their respective representations has been stored so far in lookup tables, the n^{th} window is defined as

$$u_n = [LT(X_{n-\frac{K-1}{2}}), ..., LT(X_n), ..., LT(X_{n+\frac{K-1}{2}})]\,,$$

where K is the size of window. Denoting $\tilde{\mathcal{P}}$ the set of BIOES-prefixed parsing tags from $\mathcal{P}$, the module outputs a vector of scores $s(u_n) = [s_1, ..., s_{|\tilde{\mathcal{P}}|}]$ (where s_t is the score of the BIOES-prefixed parsing tag $t \in \tilde{\mathcal{P}}$ for the constituent X_n). The constituent with indices exceeding the input boundaries ($n-(K-1)/2 < 1$ or $n+(K-1)/2 > N$) are mapped to a special padding vector (which is also learned). As any classical two-layer neural network, our architecture performs several matrix-vector operations on its inputs, interleaved with some non-linear transfer function $h(\cdot)$,

$$s(u_n) = M_2\, h(M_1\, u_n)\,,$$

where the matrices $M_1 \in \mathbb{R}^{H \times K|D|}$ and $M_1 \in \mathbb{R}^{|\tilde{\mathcal{P}}| \times H}$ are the trained parameters of the network, and $h()$ is a pointwise non-linear function such as the hyperbolic tangent. The number of hidden units H is a hyper-parameter to be tuned.

6.6.4 Coherent BIOES Predictions

The next module of our architecture aggregates the BIOES-prefixed parsing tags from our tagger module in a coherent manner. It is implemented as a Viterbi decoding algorithm over a constrained graph G, which encodes all the possible valid sequences of BIOES-prefixed tags over constituents: e.g. B-A tags can only be followed by I-A or E-A tags, for any parsing label A. Each node of the graph is assigned a score produced by the previous neural network module (score for each BIOES-prefixed tag, and for each word). The score $S([t]_1^N, [X]_1^N, \theta)$ for a sequence of tags $[t]_1^N$ in the lattice G is simply obtained by summing scores along the path ($[X]_1^N$ being the input sequence of constituents and θ all the parameters of the model). We followed the exact same approach as in [Legrand and Collobert, 2014], except that transition scores (edges on the graph) were all set to zero. Indeed, we observed in empirical experiments that adding transitions scores does not improve F1-score performance. This decoding is thus present only to insure coherence in the predicted sequence of tags.

6.6.5 Training Procedure

Both the composition network and tagging networks are trained by maximizing a likelihood over the training data using stochastic gradient ascent.

We performed all possible iterations, over all training sentences, of the greedy procedure presented in Figure 6.1 constrained with the provided labeled parse tree. This leads to our training set of sequences of tree nodes. While this procedure is similar to [Legrand and Collobert, 2014], it is worth mentioning that it implies the system is only trained on *correct* sequences of tree nodes. In that respect, it is not trained to recover from past mistakes it could have made during the recurrent process. For every tree node, the sub-trees (structure and tags) were also extracted during this procedure.

Training the system consists in repeating the following steps:

- Pick a random sequence of nodes extracted in the training set, as described above. Consider the associated sub-trees for each node which is not a leaf.
- Perform a forward pass of the word-tag composer (see Section 6.6.2) along these sub-trees.
- For all nodes in the sequence, perform a forward pass of the tagger according to word (or sub-tree) representations, as well as constituent tags.
- Compute a *likelihood of the right sequence of BIOS-prefixed tags* (see below), given the scores of the tagger.
- Backward gradient through the tagger up to the word (or sub-tree) and tag representations.
- Backward gradient through the word-tag composer up to the word and tag representation.
- Update all model parameters (from compositional networks C_i, tagger network, and lookup tables) with a fixed learning rate.

Details about the training likelihood can be found in [Legrand and Collobert, 2014]. The score $S([t]_1^N, [X]_1^N, \theta)$ of the true sequence of BIOS-prefixed tags $[t]_1^N$, given the input node sequence $[X]_1^N$ can be interpreted as a conditional probability by exponentiating this score (thus making it positive) and normalizing it with respect to all possible path scores. The log-probability of a sequence of tags $[t]_1^N$ for the input sequence of constituents $[X]_1^N$ is given by:

$$\log P([t]_1^N | [X]_1^N, \theta) = S([t]_1^N, [X]_1^N, \theta) - \log \left[\sum_{\forall [t']_1^N} \exp S([t']_1^N, [X]_1^N, \theta) \right] . \tag{6.1}$$

The second term of this equation (which correspond to the normalisation term) can be computed in linear time thanks to a recursion similar to the Viterbi algorithm [see Rabiner, 1989]. Similar training procedures have been proposed in the past for structured data [Denker and Burges, 1995, Bottou et al., 1997, Lafferty et al., 2001].

6.7 Experiments

6.7.1 Corpus

Experiments were conducted using the standard English Penn Treebank data set [Marcus et al., 1993]. We adopted the classical setup, with sections 02-21 for train, section 22 for validation, and section 23 for test. The validation corpus was used to select our hyper-parameters and best models.

We pre-processed the data only with a subset of operations which are achieved in standard parsers: (1) functional labels, traces were removed, (2) the PRT label was replaced by ADVP [Magerman, 1995]. (3) We tackled the unary chain issue - non-terminals with a single non-terminal child - by merging the nodes together and assigning as tag the concatenation of the merged node tags. This was done in order to avoid looping issues in the parsing algorithm (e.g. a node being repetitively tagged with two different tags in our iterative process) and ensure the convergence of the parsing process. Only concatenated labels which occurred at least 30 times (corresponding to the lowest number of occurrences of the less common original parsing tag) were kept, leading to 11 additional parsing tags. Added to the original 26 parsing tags, this resulted in 161 tags produced by our parser. At test time, the inverse operation is performed: concatenated tag nodes are simply expanded into their original form.

6.7.2 Detailed Setup

Our systems were trained using a stochastic gradient descent over the available training data until convergence on the validation set. Hyper-parameters were chosen according to the validation. Lookup-table sizes for the words and tags (part-of-speech and parsing) are 200 and 20, respectively. The window size for the tagger is $K = 7$ (3 neighbours from each side). The size of the tagger's hidden layer is $H = 500$. We used the word embeddings obtained from Lebret and Collobert [2014] to initialize the word lookup-table. These embeddings were then fine-tuned during the training process. We fixed the learning rate to $\lambda = 0.15$ during the stochastic gradient procedure. As suggested in Plaut and Hinton [1987], the learning rate was divided by the size of the input vector of each layer. The part-of-speech tags were obtained using the freely available software SENNA[3].

6.7.3 Word Embedding Dropout Regularization

We found that our system was easily subject to overfitting (training F1-score increasing while the validation curve was eventually decreasing as shown in Figure 6.4). As the capacity of our network mainly lies on the words and tag embeddings, we adopted a dropout regularization strategy [see Hinton et al., 2012] for the lookup tables. The key idea of the dropout regularization is to randomly drop units (along with their connections) from the neural network during

[3] http://ml.nec-labs.com/senna

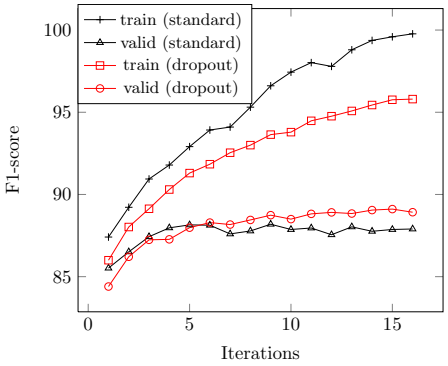

Figure 6.4: Train and validation F1-score, according to the number of training iterations, with and without the "dropout" procedure.

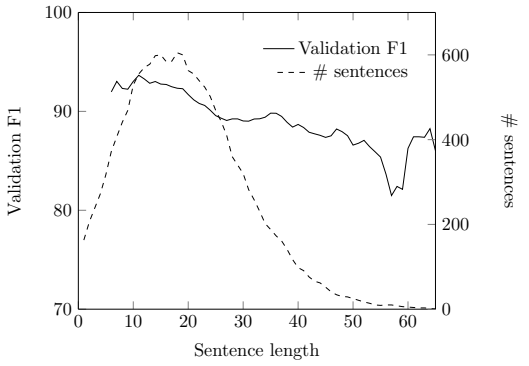

Figure 6.5: Validation F1 and number of sentences, according the the sentence length.

training. This prevents units from co-adapting too much. In our case, during the training phase, a "dropout mask" is applied to the output of the lookup-tables: each element of the output is set to 0 with a probability 0.25. At test time, no patch is applied but the output is re-weighted, scaling it by 0.75. We observed a good improvement in F1-score performance, as shown in Figure 6.4.

6.7.4 Performance comparison

F1 performance scores are reported in Table 6.1. Scores were obtained using the Evalb implementation[4]. We compared our system is compared with a range of different state-of-the-art parsers. In addition to the the four main generative parsers, we report the scores of well known re-ranking parsers (including the state-of-the-art from McClosky et al. [2006]) as well as for two major purely discriminative parsers. Detailed error analysis compared against a subset of these parsers is reported in Table 6.2, using the code provided by Kummerfeld et al. [2012]. Performance with respect to sentence length is reported in Figure 5.

We included a voting procedure using several models trained starting from different random initializations. The voting procedure is achieved in the following way: at each iteration of the greedy parsing procedure, given the input sequence of constituents, (1) node representations are computed for each model by composing the sub-tree representations corresponding to the given model and using its own compositional network (2) each model computes tag scores using its own tagger network (3) tag scores are averaged (4) a coherent path of tag is predicted using the Viterbi algorithm.

Finally, we report a brief quantitative evaluation of our compositional representations in Table 6.3. Random phrases were picked in the WSJ corpus, and closest neighbors (according

[4] Available at http://nlp.cs.nyu.edu/evalb

Table 6.1: Performance comparison of different state-of-the-art parsers, in terms of Precision (P), Recall (R), and F1 score, for sentences of size ≤ 40 words, and on the full WSJ test set. V_x denotes a voting procedure with x models. The reported time (in seconds) is the time to parse the full WSJ test corpus.

		< 40			FULL			
	MODEL	(R)	(P)	F1	(R)	(P)	F1	TIME
GENERATIVE	MAGERMAN (1995)	84.6	84.9	84.8				
	COLLINS (1999)	88.5	88.7	88.6	88.1	88.3	88.2	1247
	CHARNIAK (2000)	90.1	90.1	90.1	89.6	89.5	89.6	
	PETROV AND KLEIN [2007]	90.7	90.5	90.6	90.2	98.9	90.1	307
GENERATIVE WITH RE-RANKING	HENDERSON (2004)				89.8	90.4	90.1	
	CHARNIAK & JOHNSON (2005)			92.0			91.1	
	MCCLOSKY ET AL (2006)						**92.1**	
	SOCHER ET AL (2013)			91.1			90.4	390
DISCRIMINATIVE	CARRERAS ET AL. (2008)				90.7	91.4	91.1	
	LEGRAND & COLLOBERT (2014) (V_{10})	90.0	90.1	90.1	89.6	89.7	89.6	
	LEGRAND & COLLOBERT (2014) + DROPOUT (V_{10})	90.6	90.1	90.4	90.2	89.7	89.9	
	THIS WORK	88.8	89.1	89.0	88.2	88.6	88.4	
	THIS WORK + DROPOUT	89.7	90.3	90	89.1	89.9	89.5	30
	THIS WORK + DROPOUT (V_4)	90.5	90.8	90.7	90.1	90.4	90.3	120

to the Euclidean distance) with other phrases of the corpus are reported.

Table 6.2: Detailed parser comparison. We report the average number of bracket errors per sentence for different error categories.

	PP ATTACH	CLAUSE ATTACH	DIFF LABEL	MOD ATTACH	NP ATTACH	CO-ORD	1-WORD SPAN	UNARY	NP INT.	OTHER
MCCLOSKY ET AL (2006)	0.60	0.38	0.31	0.25	0.25	0.23	0.20	0.14	0.14	0.50
SOCHER ET AL (2013)	0.79	0.43	0.29	0.27	0.31	0.32	0.31	0.22	0.19	0.41
LEGRAND AND COLLOBERT (2014)	0.74	0.45	0.27	0.25	0.34	0.38	0.24	0.22	0.20	0.57
THIS WORK + DROPOUT	0.78	0.44	0.29	0.27	0.36	0.42	0.24	0.21	0.20	0.60
THIS WORK + DROPOUT (V_4)	0.71	0.43	0.25	0.24	0.35	0.38	0.23	0.21	0.19	0.56

Table 6.3: Nearest neighbors (in terms of vector representation Euclidean distance) for several phrases in the WSJ corpus. For every node in the corpus, the sub-tree representations were computed. Then, for the selected phrases, we computed all Euclidean distances. We report below the 5 top closest other phrases in WSJ.

brendan barba , chairman of the moonachie , n.j. , maker of plastic film products
edmund edelman , chairman of the los angeles county board of supervisors
esther dyson , editor of release 0.0 , an industry newsletter that spots new developments
michael slater , editor of the microprocessor report , an industry newsletter
bruce miller , president of art funding corp. , an art lender
jeffrey nichols , president of apms canada , toronto precious metals advisers ,
eli lilly & co. , indianapolis ,
john kinnard & co. , minneapolis ,
procter & gamble co. , cincinnati ,
anb investment management co. , chicago ,
scimed life systems inc. , minneapolis ,
rjr nabisco inc. 's french cracker subsidiary , belin ,
mr. engelken 's sister , martha , who was born two days before the home run ,
the company 's president , n.j . nicholas , who will eventually be co-chief executive of time warner alongside mr. ross ,
claudio 's sister , isabella , a novitiate in a convent ,
her daughter , elizabeth , an attorney who is vice chairman ,
his brother , parkhaji , whose head is swathed in a gorgeous crimson turban ,
mrs. coleman 's husband , joseph , a physician ,
chairman and chief executive officer
president and chief executive officer
president and chief operating officer
chairman and chief executive
executive vice president and chief financial officer
executive vice president and chief operating officer

6.8 Conclusion

In this paper, we introduced a new parsing architecture which leverages RNN-based compositional representation of parsing sub-trees, both encoding the syntactic (tags) and semantic (words) information. The parsing procedure is tightly integrated with the composition operation, and allows us to reach performance of very well-known parsers while (1) adopting a greedy and fast procedure (2) avoid standard refined features such as headwords.

content:

- Legrand and Collobert [2015]: https://publidiap.idiap.ch/downloads/papers/2015/Legrand_ICLR_2015.pdf
- (NAACL 2016 paper, under review, https://publidiap.idiap.ch/downloads/internals/2016/Legrand_Idiap-Internal-RR-05-2016.pdf)

7 Conclusion

A An appendix

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Bibliography

M. A. Alonso, J. Vilares, and V. M. Darriba. On the usefulness of extracting syntactic dependencies for text indexing. In *Artificial Intelligence and Cognitive Science*. 2002.

R. Battiti. First- and second-order methods for learning: Between steepest descent and newton's method. *Neural Comput.*, 1992.

Y. Bengio, R. Ducharme, and P. Vincent. A Neural Probabilistic Language Model. In *NIPS*, 2000.

Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 2003.

A. Bordes, S. Chopra, and J. Weston. Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2014.

L. Bottou. *Une Approche théorique de l'Apprentissage Connexionniste: Applications à la Reconnaissance de la Parole*. PhD thesis, 1991.

L. Bottou, Y. LeCun, and Y. Bengio. Global training of document processing systems using graph transformer networks. In *Proc. of Computer Vision and Pattern Recognition*, 1997.

O. Bousquet and L. Bottou. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems 20*. 2008.

John S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Advances in Neural Information Processing Systems 2*. 1990.

P. F. Brown, J. Cocke, S. Della Pietra, V. J. Della Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin. A Statistical Approach to Machine Translation. *Computational Linguistics*, 1990.

P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. Della Pietra, and J. C. Lai. Class-based n-gram models of natural language. *Comput. Linguist.*, 1992.

X. Carreras, M. Collins, and T. Koo. TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, 2008.

E. Charniak. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, 2000.

E. Charniak and M. Johnson. Coarse-to-fine N-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, 2005.

D. Chen and C. D. Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of EMNLP*, 2014.

D. Chiang. Hierarchical Phrase-Based Translation. *Computational Linguistics*, 2007.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 1956.

S. B. Cohen and M. Collins. Tensor decomposition for fast parsing with latent-variable PCFGs. In *Proceedings of NIPS*, 2012.

M. Collins. Head-driven statistical models for natural language parsing. *Comput. Linguist.*, 2003.

R. Collobert. Deep learning for efficient discriminative parsing. In *AISTATS*, 2011.

R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning, ICML*, 2008.

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

F. Costa, P. Frasconi, V. Lombardo, and G. Soda. Towards incremental parsing of natural language using recursive neural networks, 2002.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 1989.

J. S. Denker and C. J. C. Burges. Image segmentation and recognition. In *In The Mathematics of Induction*, 1995.

P. S. Dhillon, D. Foster, and L. Ungar. Multi-view learning of word embeddings via cca. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. Question answering over freebase with multi-column convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2015.

C. Dyer, V. Chahuneau, and N. A. Smith. A simple, fast, and effective reparameterization of IBM Model 2. In *Proc. of NAACL*, 2013.

J. Earley. An efficient context-free parsing algorithm. 1970.

J. L. Elman. Finding structure in time. *COGNITIVE SCIENCE*, 1990.

J. L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Mach. Learn.*, 1991.

J. R. Finkel, A. Kleeman, and C. D. Manning. Efficient, feature-based, conditional random field parsing. In *In Proc. ACL/HLT*, 2008.

J.R. Firth. A synopsis of linguistic theory 1930-1955. *Studies in linguistic analysis*, 1957.

Christoph Goller and Andreas Küchler. Learning task-dependent distributed representations by backpropagation through structure. In *In Proc. of the ICNN-96*, 1996.

Z. S. Harris. Distributional structure. *Word*, 1954.

J. Henderson. Inducing history representations for broad coverage statistical parsing. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, 2003.

J. Henderson. Discriminative training of a neural network statistical parser. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, 2004.

G. E. Hinton, J. L. McClelland, and D. E. Rumelhart. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Distributed Representations. 1986.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, 2012.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 1989.

Z. Huang, M. Harper, and S. Petrov. Self-training with products of latent variable grammars. In *Proceedings of EMNLP*, 2010.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2014.

T. Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical report, 1965.

M. Kay. Readings in natural language processing. chapter Algorithm Schemata and Data Structures in Syntactic Processing. 1986.

D. Klein and C. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, 2003.

Dan Klein and Christopher D. Manning. Conditional structure versus conditional estimation in nlp models. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, 2002.

P. Koehn, F. J. Och, and D. Marcu. Statistical Phrase-based Translation. In *Proc. of NAACL*, 2003.

J. K. Kummerfeld, D. Hall, J. R. Curran, and D. Klein. Parser showdown at the wall street corral: An empirical investigation of error types in parser output. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012.

J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Eighteenth International Conference on Machine Learning, ICML*, 2001.

T. K. Landauer and S. T. Dumais. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 1997.

Phong Le and Willem Zuidema. The inside-outside recursive neural network model for dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Y. Le Cun. A learning scheme for asymmetric threshold networks. In *Proceedings of Cognitiva 85*, 1985.

R. Lebret and R. Collobert. Word embeddings through hellinger PCA. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, 2014.

J. Legrand and R. Collobert. Recurrent greedy parsing with neural networks. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, 2014.

J. Legrand and R. Collobert. Joint RNN-based greedy parsing and word composition. In *International Conference on Learning Representations (ICLR)*, 2015.

D. M. Magerman. Statistical decision-tree models for parsing. In *In Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 1995.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 1993.

J. Martin, R. Mihalcea, and T. Pedersen. Word Alignment For Languages With Scarce Resources. In *Proc. of WPT*, 2005.

T. Matsuzaki, Y. Miyao, and J. Tsujii. Probabilistic CFG with latent annotations. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, 2005.

D. McClosky, E. Charniak, and M. Johnson. Effective self-training for parsing. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, 2006.

R. Mihalcea and T. Pedersen. An Evaluation Exercise for Word Alignment. In *Proc. of WPT*, 2003.

T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, 2010.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, 2013.

A. Mnih and G. E. Hinton. A scalable hierarchical distributed language model. In *Advances in Neural Information Processing Systems 21*. 2009.

A. B. J. Novikoff. On convergence proofs on perceptrons. 1962.

F. J. Och and H. Ney. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, 2003.

S. Petrov and D. Klein. Improved inference for unlexicalized parsing. In *Proceedings of HLT-NAACL 2007*, 2007.

S. Petrov and D. Klein. Sparse multi-scale grammars for discriminative latent variable parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2008.

P. O. Pinheiro and R. Collobert. From Image-level to Pixel-level Labeling with Convolutional Networks. In *Proc. of CVPR*, 2015.

D. C. Plaut and G. E. Hinton. Learning sets of filters using back-propagation. *Computer Speech and Language*, 1987.

V. Punyakanok, D. Roth, and W. Yih. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 2008.

L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

A. Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Mach. Learn.*, February 1999.

F. Rosenblatt. The perceptron: a perceiving and recognizing automaton. Technical report, 1957.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*. 1986.

R. Socher, C. C. Lin, A. Y. Ng, and C. D. Manning. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *Proceedings of ICML*, 2011a.

R. Socher, C.C.Y Lin, A. Y. Ng, and C. D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, 2011b.

R. Socher, J. Bauer, C. Manning, and A. Ng. Parsing With Compositional Vector Grammars. In *ACL*. 2013a.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013b.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 2014.

A. Tamura, T. Watanabe, and E. Sumita. Recurrent Neural Networks for Word Alignment Model. In *Proc. of ACL*, 2014.

B. Taskar, D. Klein, M. Collins, D. Koller, and C. D. Manning. Max-margin parsing. In *In Proceedings of EMNLP*, 2004.

J. Turian and I. Dan Melamed. Advances in discriminative parsing. In *In Proceedings of the Joint International Conference on Computational Linguistics and Association of Computational Linguistics (COLING/ACL*, 2006.

J. Turian, L. Ratinov, and Y. Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 2010.

J. Väyrynen and T. Honkela. Word category maps based on emergent features created by ICA. In *Proceedings of the STeP'2004 Cognition + Cybernetics Symposium*, 2004.

S. Vogel, H. Ney, and C. Tillmann. HMM-Based Word Alignment in Statistical Translation. In *Proc. of COLING*, 1996.

L. Wittgenstein. *Philosophical Investigations. (Translated by Anscombe, G.E.M.)*. 1953.

N. Yang, S. Liu, M. Li, M. Zhou, and N. Yu. Word Alignment Modeling with Context Dependent Deep Neural Network. In *Proc. of ACL*, 2013.

Wenpeng Yin and Hinrich Schütze. Convolutional neural network for paraphrase identification. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015.

Chenxi Zhu, Xipeng Qiu, Xinchi Chen, and Xuanjing Huang. A re-ranking model for dependency parser with recursive convolutional neural network. *CoRR*, 2015.

M. Zhu, Y. Zhang, W. Chen, M. Zhang, and J. Zhu. Fast and accurate shift-reduce constituent parsing. In *Proceedings of ACL*, 2013.

A. Zollmann and A. Venugopal. Syntax augmented machine translation via chart parsing. In *Proceedings of the Workshop on Statistical Machine Translation*, 2006.

Legrand Joël

Curriculum Vitae

Education

2012-present **Ph.D. in Electrical Engineering**, *École Polytechnique Fédérale de Lausanne*, Lausanne, Switzerland.
Under the supervision of Ronan Collobert

2011-2012 **M.Sc. in Fundamental Cognitive Science**, *Université Lyon 2*, Lyon, France.

2009-2011 **M.Sc. degree in Computer Science**, *Université de Lorraine*, Nancy, France.
Research Master "Recognition, Learning and Reasoning"

Master Thesis, *Developmental Reinforcement Learning for Robotics*.
Under the supervision of Alain Dutech

2005-2009 **Bachelor's degree in Mathematics and Computer Science**, *Université de Lorraine*, Nancy, France.

Experience

2015 **Research Intern at Facebook**, *Facebook AI Research*, Menlo Park (CA), USA,
Under the supervision of Ronan Collobert.
During my PhD, I did a three months internship a Facebook AI Research. My work focused on Machine Translation and more precisely on Bilingual Word Alignment using Convolutional Neural Networks.

2012–Present **Research Assistant**, *Idiap Research Institute*, Martigny, Switzerland,
Under the supervision of Ronan Collobert.
I am currently doing a PhD at the Idiap Research Institute. My research focuses on Natural Language Processing using Deep Neural Networks and more precisely on Syntactic Parsing and Sentence Representations.

2010–2011 **Research Intern**, *Laboratoire Lorrain de Recherche en Informatique et ses applications (LORIA)*, Nancy, France,
Under the supervision of Alain Dutech.
As a member of the team MAIA (Autonomous Intelligent MAchines), I had to explore a developmental approach to learning a robotics task where the perception and motor skills of the robot were growing in richness and complexity during learning. This was done using the Reinforcement Learning framework and tested using Khepera III robots for a simple orientation task in a maze.

Les Marais, 10 – 1922 Salvan, Switzerland
(0041) 76 703 99 77 • joel.legrand@idiap.ch
people.idiap.ch/jlegrand

2009–2010 **Research Intern**, *Laboratoire Lorrain de Recherche en Informatique et ses applications (LORIA)*, Nancy, France,
Under the supervision of Brigitte Wrobel-Dautcourt.
During this internship, I laid the foundation stone of artEoz, a didactic software intended for the first-year students in computer science. This sortware allows to visualize the memory when running a program written in Java or Python. It is freely available at this address: *http://arteoz.loria.fr/*.

Computer skills

Basic Matlab

Intermediate Python, Perl, java

Advanced Lua, Torch, C, GNU/Linux, Shell, LaTeX

Publications

2015 J. Legrand, R. Collobert, **Joint RNN-Based Greedy Parsing and Word Composition**, *In ICLR, 2015*

2014 J. Legrand, R. Collobert **Recurrent Greedy Parsing with Neural Networks**, *In ECML, 2014*

2013 R. Lebret, J. Legrand, R. Collobert **Is Deep Learning Really Necessary for Word Embeddings?**, *In NIPS Workshop on Deep Learning, 2013*

Languages

French **Mother tongue**

English **Fluent**

Spanish **Basic**

Interests

- Music (Guitar, Bouzouki, Accordion)
- Free Software

Les Marais, 10 – 1922 Salvan, Switzerland
(0041) 76 703 99 77 • joel.legrand@idiap.ch
people.idiap.ch/jlegrand