Neural Conditional Random Fields for Natural Language Understanding

Marc-Antoine Rondeau Beauchamp

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

December 2016

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Doctor of Philosophy.

2016/12/12

Abstract

This thesis presents work on Neural Conditional Random Fields (NeuroCRFs), a combination of neural network and conditional random field, applied to chunking and named entities recognition (NER), two information extraction tasks. Information extraction is a subfield of natural language understanding (NLU), the study of the automatic processing of natural language utterances in order to obtain the information they contain in a form suitable for further automatic processing. NER is the recognition and classification of the named entities found in an utterance, while chunking is the syntactic segmentation of an utterance. In both cases, information contained in an utterance is extracted in the form of segments, composed of successive words, and an attached class. In this thesis, chunking and NER are approached through sequence labelling, the assignment of a label to each element in an input sequence. This transforms the natural language utterance into a structured sequence of labels that can easily be interpreted to extract the required information.

NeuroCRFs are models composed of a neural network (NN) used for feature extraction and a conditional random field (CRF), used to factorize the complex distribution of output labels conditioned by the input utterance into simpler factor functions that are based on the NN. CRFs rely on a set of features than can be extracted from the natural language utterance. Once the set of features is defined, machine learning algorithms can be used to learn the relation between those features and the correct output sequence. Feature engineering is the main challenge of CRF, and requires extensive work by a human expert. NeuroCRFs use the feature learning capability of NNs to reduce, and even remove, the need for feature engineering.

This thesis includes three major contributions. The first is an extension of NeuroCRFs, where the NNs is used to learn and extract features corresponding to transitions between label in the output sequence, instead of the usual emission features.

The second contribution is a continuation of this concept. NeuroCRFs use a NN to learn factor functions corresponding to events in the output sequence. Label emissions and label transitions are only one form of such events. We extended this concept to add factor functions shared by multiple events. This improved performance at the cost of reintroducing some feature engineering. We also improved performance by combining those shared features with a large margin model training algorithm. Performance was further improved by combining NNs obtained with different initializations into a single ensemble model.

Finally, the third contribution addresses the limitations of the feedforward NNs (FFNNs) used in the previous experiments. FFNNs are limited by their input, a sliding window over the natural language utterance. The model is forced to assume that labels are independent of the input outside of this limited window. Recurrent layers, such as long short term memory (LSTM) layers, do not have this limitation. LSTM based NeuroCRFs, a new addition to the NeuroCRFs family, significantly improved performance over FFNN based NeuroCRFs. Bi-directional LSTM layers were found to remove the need for the sliding window.

Sommaire

Cette thèse présentera des travaux portant sur les NeuroCRFs, une combinaison de réseaux de neurones et de champs markoviens conditionnels (CRF), dans le contexte du chunking et de la reconnaissance d'entités nommées (NER), deux tâches d'extraction d'information. L'extraction d'information est un sous-domaine de la compréhension du langage naturel (NLU), l'étude du traitement automatique de phrases en langage naturel afin d'obtenir l'information contenue dans cette phrase dans une forme structurée compatible avec un traitement automatique subséquent. La NER consiste à reconnaitre et classifier les entités nommées présentes dans une phrase. Le chunking est la segmentation sémantique d'une phrase. Dans les deux cas, l'information est extraite sous forme de segments, composés de mots consécutifs, auxquels est attaché une classe. Dans cette thèse, ces tâches sont approchées par l'étiquetage de séquence, où une étiquette est appliquée à chaque élément d'une séquence. Cela transforme la phrase en une séquence d'étiquettes structurée, qui peut être interprétée facilement afin d'extraire l'information désirée.

Les NeuroCRFs sont des modèles composés d'un réseau de neurones (NN), utilisé pour extraire des caractéristiques, et d'un CRF qui va factoriser une complexe distribution d'étiquettes, conditionnée par la phrase, en un produit de plus simple fonctions, qui sont obtenues à partir des sorties du NN. Les CRFs dépendent d'un ensemble de caractéristiques qui peuvent être extraites d'une phrase en langage naturel. Une fois que cet ensemble est défini, les algorithmes d'apprentissage automatique permettent d'apprendre la relation entre ces caractéristiques et la séquence d'étiquettes désirée. L'ingénierie des caractéristiques est la principale difficulté d'un CRF, et demande l'attention d'un expert humain. Les NeuroCRFs exploitent la capacité d'apprentissage de caractéristiques des NNs afin de réduire ce travail d'ingénierie.

Cette thèse inclue trois contributions majeures. La première est une extension des NeuroCRFs, où le NN est utilisé pour apprendre et extraire des caractéristiques correspondant aux transitions entre deux étiquettes, plutôt qu'à l'émission d'une seule étiquette.

La seconde contribution est un prolongement de ce concept. Les NeuroCRFs utilisent leur NN afin d'apprendre des fonctions correspondant à des événements dans la séquence d'étiquettes. Les transitions entre étiquettes et l'émission d'une étiquette ne sont que deux formes d'événements. Nous étendons ce concept en ajoutant des fonctions qui sont partagées par plusieurs événements. Ceci améliora les performances, au prix d'efforts supplémentaires

d'ingénierie des caractéristiques. Des améliorations supplémentaires ont été obtenues avec un algorithme d'apprentissage maximisant la marge de la séquence correcte, et en combinant des NNs obtenues avec différentes initialisations dans un large modèle-ensemble.

Finalement, la troisième contribution adresse les limitations des NNs utilisés dans les expériences précédentes. Ces NNs sont limités par leur entrée, une fenêtre glissée sur la phrase en langage naturel. Le modèle doit supposer que les étiquettes sont indépendantes de l'entrée en dehors de cette fenêtre. Les couches de neurones récurrentes, par exemple des couches à longue mémoire à court terme (LSTM), n'ont pas cette limitation. Des NeuroCRFs basés sur des couches LSTM, un nouveau membre de la famille des NeuroCRFs, ont des performances significativement améliorées comparées aux NeuroCRFs sans récursion. L'ajout d'une récursion bidirectionnelle peut même remplacer la fenêtre glissée sur la phrase en langage naturel.

Acknowledgements

First, I would like to thank Richard Rose and Yi Su, my supervisors. Their advice and feedback guided me during my PhD. Their efforts to refocus me on the central research plan, when I was distracted by other interesting directions, was invaluable. I also would like to thank Fabrice Labeau, Douglas O'Shaugnessy, and Yannis Psaromiligkos for their service in the various exams required in the course the PhD program. Similarly, I am thankful for the comments and feedback of Philippe Langlais, Jean Gotman, Roni Khazaka, and Milica Popovich. The support and feedback of Paul Vozila and Nate Bodenstab, during internships at Nuance Communications, was also appreciated.

I would also like to thank my labmates, Hoda Daou, Sina Hamidi, Aanchan Mohan, Atta Norouzian, Fabien Sacuto, Vikrant Tomar, and Shou-Chun Yin, made my time at McGill richer and enjoyable. Their willingness to be conscripted as sounding board was appreciated, and helped the work leading to this thesis.

Les services de relecture de Dimtri Achminov ont été fortement appréciés. Je voudrais finalement remercier ma famille pour leur soutien et encouragement. En particulier Line, Jocelyn, Alain et Jacqueline, qui m'ont toujours prêté oreille dans les moments difficiles.

Preface

The original contributions of this thesis are the full-rank NeuroCRFs, presented in Chapter 3, their extension with added shared parameters, presented in Chapter 4, and the combination of long short term memory layer with NeuroCRF, presented in Chapter 5.

The neural network based systems used in this thesis were developed using the theano toolkit, originally created by the LISA team of Université de Montréal. The continuous word representations were obtained with the word2vec tool developed by Tomas Mikolov. Large margin training is based on work by Gimpel and Smith.

Contents

List of Figures

List of Tables

List of Acronyms

ASR automatic speech recognition

BLSTM bi-directional long short term memory

CRF conditional random field

FFNN feed forward neural network

HMM hidden Markov model

LSTM long short term memory

MEMM maximum entropy Markov model

NER named entity recognition

NeuroCRF neural conditional random field

NLP natural language processing

NLU natural language understanding

NN neural network

RNN recurrent neural network

Chapter 1

Introduction

This chapter will introduce the field of natural language understanding, and its subfield information extraction. Sequence labelling, the approach to information extraction used in this thesis, will also be presented. This will be followed by a short description of chunking and named entities recognition, two information extraction tasks than can be solved with sequence labelling. The main topic of this thesis, neural conditional random fields (Neuro-CRFs), a recent model family used for sequence labelling, will be presented next. Finally, the outline of this thesis, with a list of its major contributions, will conclude this introductory chapter.

1.1 Natural Language Understanding

Natural language understanding (NLU) is the study of the automatic processing of natural language utterances in order to extract the information they contain in a form suitable for further processing by an automatic system. Automatic systems require information in a highly structured and specific format, while natural languages carry information in an unstructured and highly variable format. NLU tasks consist of bridging this gap between the natural form of an utterance and the structured form required by automatic systems.

1.1.1 Information Extraction

Information extraction is a sub-field in NLU consisting of the extraction and classification of segments from a natural language utterance. The words contained in a segment, as well

as its class, are a structured representation of some of the information that was present in an unstructured form in the original natural language utterance. The extracted information is used for further processing. The structure of the information extracted is determined by the requirements of this processing. The type of information to extract is restricted to the information required by the subsequent processing.

1.1.2 Sequence Labelling

Sequence labelling consists of assigning a label to each element in an input sequence. This can be used for information extraction, as well as for other tasks. Conversely, other approaches, such as rules based systems, can be used for information extraction.

The segmentation and classification steps of information extraction are performed jointly, by assigning a label to each word in the sentence. Those labels are interpreted to retrieve the segments and their class. A common labelling scheme is IOB2, where the label identify the position of the word in its segment, as well as the class of the segment. Words that are not part of a relevant segment are assigned a label indicating this.

This approach does not depend on the type of information that must be extracted. It is suitable for all tasks where specific information must be extracted from a sentence in the form of classified segments. The classes, and therefore labels, will of course be different depending on the task, but the overall process will be identical.

1.1.3 Example Task

A system used to provide directions to a user will be used illustrate how sequence labelling is used to extract information from natural language utterances. This system needs to know the origin, destination, and means of transportation, in order to query its database and retrieve the required set of directions. The information extraction step has to identify the segments corresponding to those three classes, if they are present.

Using IOB2 labelling, this task requires seven labels:

- B-origin and I-origin
- B-dest and I-dest
- B-mean and I-mean

- O.

"B" indicates the beginning of a segment; "I" indicates the inside of a segment; "O" indicates the outside of a segment. The second part of the "B-*" and "I-*" labels, such as "origin", indicates the class.

A user needing directions in order to get home might say the sentence "I want to take the bus to get home from 2344 random street". The destination is "home" and the word "home" is therefore given the label "B-dest". The origin is "2343 random street", with the labels "B-origin I-origin I-origin". The means of transportation is "bus", with the label "B-mean". The other words are labelled "O". Once the origin, destination and means of transportation are identified, the system queries its database of bus routes.

1.2 Chunking

Chunking[1] is one of the two information extraction tasks studied in this thesis. Chunking is the creation of a flattened form of a sentence's parse tree. The information extracted consists of syntacticly related words [2]. The resulting segments, along with an associated type, form "chunks". Chunks correspond to a sub-tree in the full parse tree, with the label coming from the syntactic head of the sub-tree. While the resulting parsing is not as rich as a full parse tree, it can be used as features for subsequent processing such as [3], and can be obtained using sequence labelling.

1.3 Named Entities Recognition

The second information extraction task studied in this thesis is named entities recognition (NER) [4], the extraction and classification of named entities from natural language sentences. Named entities are, as the name indicates, proper names. They includes place names, organizations, persons, events, book titles, etc. Segments consisting of a named entity are extracted from the input sentence, and classified. The exact classes varies according to the requirement of the tasks, but classes for places, organizations and persons are common. In some cases, the distinction can be finer, and the classes can also include the role of a named entity in a query. In the example above, origins and destinations are places, but the system must be able to distinguish between them.

[1] Sometimes called "shallow parsing".

1.4 Conditional Random Field

Conditional random fields (CRFs) [5] are discriminative models where the distribution of an output sequence conditioned by an input sequence is factorized into simpler factor functions. The factor functions are organized in a factor graph, connecting a subset of the variables in the output and input sequence with the factor functions. Usually, the factor functions are log linear weighted sums of features extracted from the sequences.

Linear chain CRFs are a subset of CRFs where the factor graph places the output variables in a linear chain. The factors are functions of two consecutive output variables, as well as some input variables. The feature weights of those factor functions are tied.

The main challenge of CRFs is the feature engineering required by the factor functions. This is task specific, and requires work by an human expert. Examples of feature engineering can be seen in [6, 7, 8, 9].

1.5 Neural Network

Neural Networks (NNs) are network of simple non-linear units. The input of those units is a weighted sum of the output of other units. Units are organized into successive layers, where the output of a layer's units are connected to the input of the next layer's unit. The first layer's input is the input of the NN. Similarly, the last layer's output is the output of the NN. NNs form non-linear mapping between their input and outputs, making them well suited for complex feature analysis. Layers learn a representation of their input, which is then used as the input by the following layer. This allows the NN to build more and more complex representation of its input. Ideally, the NN output should be a perfect representation of the desired mapping between its input and output. The weights connecting units are selected in order to minimize a loss function based on the difference between the output of the NN given a specific input and the desired output.

1.6 NeuroCRF

NeuroCRFs are combinations of neural networks (NNs) and conditional random fields (CRFs). The factor functions of a NeuroCRF are modelled through NNs. As the NN can learn to extract relevant features from its input, this reduces the feature engineering usually required

by CRFs. Meanwhile, CRFs provide a powerful framework to group simple functions into a complex distribution over a random sequence conditioned by an input sequence.

1.7 Continuous Word Representation

The natural representation of words, in a neural network, is a very large one-hot vector, where the element corresponding to a word is set to one, and all other elements are set to zeros. This results in a high-dimensional, but very sparse, input space. A continuous word representation projects this input space into a much lower dimensional space, where a word is represented by a continuous dense vector. Those continuous word representations were first introduced for language modelling tasks [10]. They have also been used for NLP and NLU tasks. Those continuous word representation address the issues related to large vocabulary by learning similar representations for similar words. The representation is pre-trained on large corpora of unlabelled data. If a word is not present in the corpus used to train the NLU system, but is present in this large corpora, words with similar representations will still have been used when training the NLU system.

1.8 Thesis Outline

This section will present the organization of this thesis, highlighting its contributions. The content of this thesis is organized into 6 chapters.

Background

Chapter 2 will present background information. It will start by an overview of information extraction from natural language. This will be followed by a presentation of conditional random fields, neural networks and word representations. This will be followed by a description of the performance measures used in the experimental studies included in this thesis. Finally, Chapter 2 will conclude with information about chunking and named entities recognition, the two information extraction tasks studied in this thesis, as well as the datasets used in the experimental studies.

NeuroCRF

Chapter 3 is based on work published in "Full-rank linear-chain neurocrf for sequence labeling" [11]. The main focus of Chapter 3 is two fundamental forms of NeuroCRFs, full and low-rank NeuroCRFs. Full-rank NeuroCRFs are a contribution of this thesis.

It will present the general form of NeuroCRF, and how full and low-rank NeuroCRFs can be obtained by constraining the parameters of this general form. Chapter 3 will also present how to estimate the model parameters, for a given training corpus.

Finally, it will present an experimental study showing improvement over a baseline CRF with similar features. We will discuss the improvements obtained with full-rank NeuroCRFs, compared to low-rank NeuroCRFs, as well as the factors enabling and limiting those improvements.

Three Improvements to NeuroCRFs

Chapter 4 is based on work published in "Recent improvements to NeuroCRFs for named entity recognition" [12]. It will present three ways to improve the performance of NeuroCRFs. One of those improvements, added shared parameters, is a contribution of this thesis.

Chapter 4 will first present a way to add shared NN outputs, which will be mapped to multiple factor functions corresponding to transitions between labels. This is done by grouping labels and transitions manually, based on their similarity. NN outputs are added for each groups, and the factor functions are now based on sum of NN outputs, depending on the relevant transitions.

Chapter 4 will also present an investigation of a form of large margin training in order to reduce overfitting. This modifies the parameter estimation procedure to maximize not only the log likelihood of the training corpus, but also the margin of the correct output compared to the best competing output. Finally, Chapter 4 will present a way to exploit the non-convexity of NNs, and therefore NeuroCRFs ,to produce complementary models. Those models are combined into a larger ensemble model, improving performance.

Recurrent NeuroCRFs

Chapter 5, based on work published in "LSTM-Based NeuroCRFs for Named Entity Recognition" [13], will present the last contribution of this thesis. It will address a limitation of feedforward NNs (FFNNs), where the NN output is only a function of the current input,

limiting support for long term dependencies between input and output. This problem is addressed by replacing the feedforward NN by a recurrent NN, where the NN output is a function of previous inputs.

Chapter 5 will focus on long short term memory (LSTM) layers, a form of recurrent NN, and their bi-directional variant. The bi-directional variant divides the LSTM layer in two. One half uses a causal recurrence, where the output is based on previous inputs. The other half uses an anti-causal recurrence, where the output is based on following inputs.

Chapter 5 will conclude with an experimental study justifying the use of LSTM instead of simpler recurrent NNs. LSTM-based NeuroCRFs will be combined with the improvements presented in Chapter 4. They will be compared to bi-directional LSTM (BLSTM). We also present results showing that BLSTM remove the need for the sliding window over the input that was used in the previous FFNN-based NeuroCRFs.

Conclusion

Chapter 6, the conclusion, will summarize the important contribution of this thesis, and list promising future work that could build on the content of this thesis.

Chapter 2

Background

This chapter will present some background material for the following chapters. The first section will present an overview of information extraction from natural language, of which chunking and named entities extraction are a subset. This will be followed by a presentation of conditional random fields (CRFs), and feed-forward and recurrent neural networks (NN). The major contributions of this thesis are based on a combination of CRF and neural networks. The input of the NN used are based on continuous word representations, the subject of the next section. Finally, this chapter will conclude with a description of the datasets and performance measures used in the experimental studies found in this thesis.

2.1 Information Extraction From Natural Language

The information carried by natural language needs to be extracted before it can be used in some automatic processes. Unlike artificial languages, natural language are highly irregular, and the information cannot, in general, be extracted by a *simple* set of rules. In the case of speech, those problems are compounded with the higher irregularity of spontaneous speech and the low but still non-zero error rates of modern automatic speech recognition systems.

The tasks used in the experimental studies included in this thesis are chunking and named entity recognition. Both consists of extracting and labelling segments from a natural language sentence. In both cases, it is assumed that the sentence is available in text form. In both cases, the output is expressed as a sequence of labels, which are then automatically and trivially combined with the original sequence of words to extract the relevant segments. As such, while the two tasks are different, the same general tools are applicable to both.

More details on the tasks themselves can be found in Section 2.7.

Named entry recognition is used to retrieve named entities, such as place and person names. Chunking, or shallow parsing, is used to create a flattened form of a sentence grammatical parse tree.

While other approaches, such as support vector machines [14, 15, 16] and decision trees [17, 18] have been used, this section will focus on three related approaches:

1. Hidden Markov models (HMMs)
2. Maximum entropy hidden models (MEMMs)
3. Conditional random fields (CRFs).

Those approaches have the same fundamental structure of states and transitions between states, where the states correspond to the labels, and where every label has an associated observation (i.e. the word). CRFs will be covered in Section 2.2.

2.1.1 Hidden Markov Models

Hidden Markov models (HMMs) are a well known class of generative models, where an observation is generated by an hidden state. The observations $\mathbf{x} = \{x_1, \ldots, x_T\}$ depend only on the corresponding state y_t. A given state y_t depends only on the immediately previous state, and is independent of any other preceding states (i.e $p(y_t|y_{t-1}, y_{t-2}) = p(y_t|y_{t-1})$). The model is

$$p(\mathbf{y}, \mathbf{x}) = \prod_{t=1}^{T} p(x_t|y_t)p(y_t|y_{t-1}) = p(\mathbf{x}|\mathbf{y})p(\mathbf{y}). \tag{2.1}$$

The transition probabilities $p(y_t|y_{t-1})$ can be modelled by categorical distribution. The emission probability density $p(x_t|y_t)$ is more complex, and is specific to a given system.

HMM were used for NER in [19, 20, 21, 22]. HMMs were used for chunking in [23, 24]. While all those works use the same class of model, they rely on different features extracted from the words in order to obtain $p(x_t|y_t)$. This is feature engineering and will be discussed in more details in the context of CRFs, in Section 2.2.

2.1.2 Maximum Entropy Markov Models

Maximum entropy Markov models (MEMMs) [25] are a discriminative form of HMMs. They have the same structure of states, observations and transitions, but rather than focusing on

$p(x_t|y_t)$, the generative model of the observation given the state, they focus on $p(y_t|y_{t-1}, x_t)$, a discriminative model of the state given the previous state and current observation.

The model is

$$p(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} p(y_t|y_{t-1}, x_t), \tag{2.2}$$

$$p(y_t|y_{t-1}, x_t) = \frac{1}{Z(y_{t-1}, x_t)} \exp F(y_t, y_{t-1}, x_t), \tag{2.3}$$

$$Z(y_{t-1}, x_t) = \sum_{y} \exp F(y, y_{t-1}, x_t), \tag{2.4}$$

where $F(y, y_{t-1}, x_t)$ is a weighted sum of features extracted from (y, y_{t-1}, x_t), x_t is the current observation, y_t is the current state and y_{t-1} is the previous state. In MEMMs, $p(y_t|y_{t-1}, x_t)$ is a maximum entropy classifier.

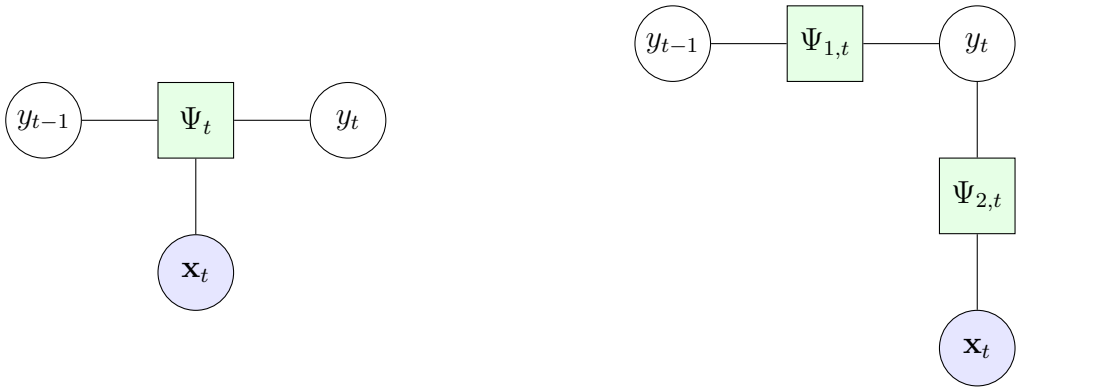
MEMMs were used for NER in [26] and [27]. They were also used for chunking in [28]. MEMMs were rapidly overshadowed by the similar but more powerful conditional random fields.

2.2 Conditional Random Fields

Conditional random fields (CRFs) are a class of graphical models [5, 29]. CRFs are an extension of MEMMs, intended to address the label bias problem where the model is biased towards states with few outgoing transitions. CRFs are used to model a conditional distribution $p(\mathbf{y}|\mathbf{x})$ over an input sequence $\mathbf{x}$ and an output sequence $\mathbf{y}$. CRFs are defined by a factor graph. The distribution $p(\mathbf{y}|\mathbf{x})$ is factorized according to this graph. The factor functions found in the factor graphs are similar to distributions over features extracted from the sequences $\mathbf{y}$ and $\mathbf{x}$. CRFs replace the global distribution over a large number of variables by many factor functions over a smaller number of variables.

CRFs are commonly used for chunking and NER, as well as for other sequence labelling taks. Passos et al. combined CRF and continuous word representation [9] for NER. In this particular case, the word representation training procedure is modified to better incorporate a lexicon of known named entities. CRFs with simpler features were applied to NER [6] and chunking [7] in 2003. CRF have been regularly used for information extraction since their introduction [30, 31, 32].

2.2.1 Definition



(a) General (b) Factorized into transition and emission

Fig. 2.1 Clique template of linear chain conditional random fields. Square boxes are factors, circles are variables. Subfigure (b) is similar to a HMM, replacing $p(x_t, y_t|y_{t-1}) = p(x_t|y_t)p(y_t|y_{t-1})$ with $\Psi_t(y_t, y_{t-1}, \mathbf{x}_t) = \Psi_{1,t}(y_t, y_{t-1})\Psi_{2,t}(y_t, \mathbf{x}_t)$.

CRFs are models of the form [5]

$$\log p(\mathbf{y}|\mathbf{x}) = \left(\sum_{\Psi_g \in \mathcal{G}} \log \Psi_g(\mathbf{x}, \mathbf{y}) \right) - \log Z(\mathbf{x}) \tag{2.5}$$

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{\Psi_g \in \mathcal{G}} \Psi_g(\mathbf{x}, \mathbf{y}), \tag{2.6}$$

where $\mathbf{y}$ and $\mathbf{x}$ are respectively the output and input sequence, Ψ_g is a factor function, $\mathcal{G}$ is the set of factor functions in the graph and $Z(\mathbf{x})$ is a normalization term. The factor functions can be grouped into clique templates $\mathcal{C}_p$.

$$\log p(\mathbf{y}|\mathbf{x}) = \left(\sum_{\mathcal{C}_p \in \mathcal{C}} \sum_{\Psi_c \in \mathcal{C}_p} \log \Psi_c(\mathbf{x}_c, \mathbf{y}_c) \right) - \log Z(\mathbf{x}) \tag{2.7}$$

$$\Psi_c(\mathbf{x}_c, \mathbf{y}_c) = \exp \sum_{\theta_{p,k} \in \theta_p} \theta_{p,k} f_{p,k}(\mathbf{x}_c, \mathbf{y}_c), \tag{2.8}$$

where $\mathbf{x}_c$ and $\mathbf{y}_c$ are the subsets of $\mathbf{x}$ and $\mathbf{y}$ relevant for the factor function Ψ_c belonging to the clique template $\mathcal{C}_p$, θ_p are the model parameters of the same clique template, $f_{p,k}$ is a

parameter less feature extraction function, and $\theta_{p,k}$ is the associated feature weight.

In practice, the linear-chain form turns out to be the most useful and popular. In this form, the distribution is factorized into T factor functions, where T is the length of the output and input sequences, with a single clique template, illustrated by Figure 2.1a. The factor functions are a log-linear combination of features extracted from a relevant segment of the input and output sequences. Linear-chain CRFs [29] are models of the form

$$\log p(\mathbf{y}|\mathbf{x}) = \left(\sum_{t=1}^{T} \log \Psi_t(y_t, y_{t-1}, \mathbf{x})\right) - \log Z(\mathbf{x}) = \left(\sum_{t=1}^{T} G_t(\mathbf{x}) F(y_t, y_{t-1})\right) - \log Z(\mathbf{x}) \quad (2.9)$$

$$\Psi_t(y_t, y_{t-1}, \mathbf{x}) = \exp G_t(\mathbf{x}) F(y_t, y_{t-1}) \quad (2.10)$$

$$G_t(\mathbf{x}) = C_t(\mathbf{x}) W + B \quad (2.11)$$

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \exp\left(\sum_{t=1}^{T} G_t(\mathbf{x}) F(y_t, y_{t-1})\right), \quad (2.12)$$

where $G_t(\mathbf{x})$ is a feature analysis matrix and $F(y_t, y_{t-1})$ is an output feature selection matrix. The feature analysis matrix $G_t(\mathbf{x})$ is composed of an input feature weights matrix W, a feature count matrix $C_t(\mathbf{x})$ and an output feature biases matrix B.

The feature count matrix $C_t(\mathbf{x})$ is a 1-by-N matrix, where N is the number of features. The columns contain the number of occurrences, in $\mathbf{x}$, of the corresponding feature around time t. The set of features used is specific to the goal of the system. An example would be that the t'th word x_t is "the", or that x_{t-1} is "a".

$C_t(\mathbf{x})$ is not a function of the output sequence $\mathbf{y}$. The output feature selection matrix $F(y_t, y_{t-1})$ has a similar role. It indicates the presence or absence of an output feature in the pair (y_t, y_{t-1}). Where $C_t(\mathbf{x})$ extracts feature counts from $\mathbf{x}$ around t, $F(y_t, y_{t-1})$ extracts feature occurrences from (y_t, y_{t-1}). An example of an output feature would be that the t'th label y_t is "True", or that there's a transition from "False" to "True". $F(y_t, y_{t-1})$ is a M-by-1 matrix, where M is the number of output features. A full feature is the occurrence of an output feature or the co-occurrence of an input and output feature.

Finally, the input feature weights matrix W is a N-by-M matrix. Its elements indicate the weight of an input feature co-occurring with an output feature. The output feature

biases matrix B is a 1-by-M matrix. Its elements indicate the weight of an output feature, independently of the input. The output feature weights are described as "biases" to unify the notation with the one used for neural networks.

Figure 2.1b shows the clique template of HMM-like linear-chain CRFs. In this particular case, the clique is further separated into transition factors $\Psi_{1,t}$ and emission factors $\Psi_{2,t}$. This structure mirrors the one of HMMs, where observations and transitions distributions are independent.

2.2.2 Feature engineering

While the feature weights matrix can be optimized automatically, identifying the relevant feature set requires work by an expert. This *feature engineering* is an essential step when designing a CRF-based system. The feature set is task-specific, although similar tasks will have similar feature sets. This decision is somewhat ad-hoc. Features are selected based on previous experiments and expert knowledge, but ultimately, only experimental results can determine if a feature set is suitable for a specific task. Among the possible features are

- Word identity,
- Word representation,
- Part-of-speech,
- Prefixes and suffixes,
- Presence in dictionaries.

It is also possible to include feature n-grams extracted from a window, or even input features that are independent of the position in $\mathbf{x}$.

The feature engineering process will determine the feature count matrix $C_t(\mathbf{x})$ and the output feature selection matrix $F(y_t, y_{t-1})$. In the feature weight matrix, every input feature is associated with all output features. It is common to fix the weights of some combination of input and output feature to 0, which reduces the overall number of features. This is part of feature engineering, and is done when a specific input and output feature are known to be independent. Feature pruning, discussed in Section 2.2.5, can be used to reduce the number of features.

2.2.3 Example

Feature engineering is the first step in the creation of a CRF model for the small example presented in Section 1.1.3 of the introduction chapter. A very simple feature set will be used to illustrate basic feature engineering.

The input features will be the presence of a word in a small sliding window around t, so that

$$C_t(\mathbf{x}) = \begin{bmatrix} V(x_{t-1}) & V(x_t) & V(x_{t+1}) \end{bmatrix} \tag{2.13}$$

$$V(x_t) = \begin{bmatrix} v_1(x_t) & v_2(x_t) & \dots & v_{|V|}(x_t) \end{bmatrix} \tag{2.14}$$

$$v_i(x) = \begin{cases} 1, & i = x \\ 0, & i \neq x \end{cases}, \tag{2.15}$$

where $|V|$ is the number of possible words and the words x_t are represented by an index between 1 and $|V|$. The output features are divided into two groups: emission and transition features. With 7 possible labels,

$$F(y_t, y_{t-1}) = \begin{bmatrix} e_1(y_t) & \cdots & e_7(y_t) & f_1(y_t, y_{t-1}) & \cdots & f_{49}(y_t, y_{t-1}) \end{bmatrix}^\top \tag{2.16}$$

$$e_i(y) = \begin{cases} 1, & i = y \\ 0, & i \neq y \end{cases} \tag{2.17}$$

$$f_i(y, y') = \begin{cases} 1, & i = 7y + y' \\ 0, & i \neq 7y + y' \end{cases}, \tag{2.18}$$

where the label y_t is represented by an index between 1 and 7, with 49 pairs of label (y_t, y_{t-1}), for a total of 56 output features. The feature weight matrix W is a $3|V|$-by-56 matrix, and the biases matrix B is a 1-by-56 matrix. The feature weights are in the form of $w_{x_{t-1}=1,y_t=1}$, $w_{x_{t-1}=1,y_t=1,y_{t-1}=1}$, $b_{y_t=1}$, and $b_{y_t=1,y_{t-1}=1}$. The feature weights correspond to the co-occurrence of an input feature and an output features, while the biases correspond to the occurrence of an output feature.

The resulting clique template follows Figure 2.1a. With $T = 3$, the full factor graphs is shown in Figure 2.2.

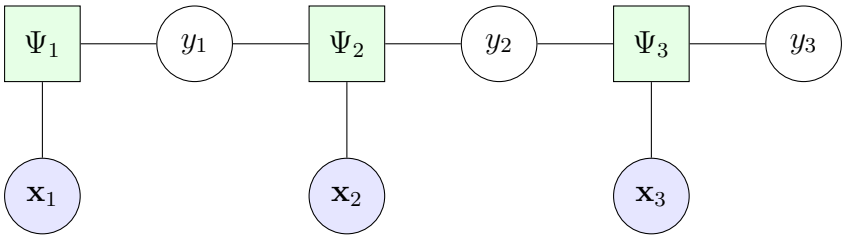


Fig. 2.2 Full linear chain factor graph with $T = 3$.

2.2.4 Parameters Estimation

Parameters are estimated using a large number of training examples. The parameters are selected to minimize the loss function [5, 29]

$$L(\text{train}) = \sum_{(\mathbf{y},\mathbf{x})\in\text{train}} -\log p(\mathbf{y}|\mathbf{x}), \tag{2.19}$$

where $(\mathbf{y}, \mathbf{x})$ is a training pair in the training corpus train.

Sutton and McCallum, in an excellent tutorial on CRF [29], showed that the loss function is minimized when the expected number of occurrences of a feature, according to the model, is equal to the number of occurrences of the feature in the training pair $(\mathbf{x}, \mathbf{y})$, so that

$$\sum_{t=1}^{T} \sum_{(y_t,y_{t-1})} p(y_t, y_{t-1}|\mathbf{x}) C_t^\top(\mathbf{x}) F^\top(y_t, y_{t-1}) = \sum_{t=1}^{T} C_t^\top(\mathbf{x}) F^\top(y_t, y_{t-1}) \tag{2.20}$$

$$\sum_{t=1}^{T} \sum_{(y_t,y_{t-1})} p(y_t, y_{t-1}|\mathbf{x}) F^\top(y_t, y_{t-1}) = \sum_{t=1}^{T} F^\top(y_t, y_{t-1}), \tag{2.21}$$

where the probability of a transition between label y_{t-1} and y_t at time t, $p(y_t, y_{t-1}|\mathbf{x})$, is computed using dynamic programming, using an algorithm similar to the Baum-Welch algorithm [33] used to estimate the parameters of HHMs.

In general, it is impossible to find an analytical solution [29]. Numerical techniques, such as gradient descent or the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) algorithm [34] are used. The convex nature of the loss function, being a sum of convex functions, simplifies this search, as convergence to a point close to the global minimum is assured if the learning rate is small enough.

2.2.5 Regularization and Feature Pruning

While CRF are convex models and training will converge to a point close to the global minimum, overfitting is still possible and will degrade performance on unseen data [29].

Norm regularization is usually applied to CRFs. Norm regularization is a prior applied to the model's parameters Θ, so that the loss function becomes

$$L(\text{train}; \{\sigma_i\}) = \sum_{(\mathbf{y},\mathbf{x})\in\text{train}} -\log p(\mathbf{y}, \Theta|\mathbf{x}), \tag{2.22}$$

$$p(\mathbf{y}, \Theta|\mathbf{x}) = p(\mathbf{y}|\mathbf{x})p(\Theta). \tag{2.23}$$

If a Gaussian prior with parameter specific standard deviation σ_i is applied to Θ, so that

$$\log p(\Theta) = \sum_{\theta_i\in\Theta} \frac{\theta_i^2}{2\sigma_i^2} - \log \sigma_i\sqrt{2\pi}, \tag{2.24}$$

the norm regularized loss function becomes

$$L(\text{train}; \{\sigma_i\}) = L(\text{train}) + \sum_{\theta_i\in\Theta} \frac{\theta_i^2}{2\sigma_i^2} - \log \sigma_i\sqrt{2\pi}, \tag{2.25}$$

$$\nabla_{\theta_i} L(\text{train}; \{\sigma_i\}) = \nabla_{\theta_i} L(\text{train}) + \frac{\theta_i}{\sigma_i^2}. \tag{2.26}$$

This prior will therefore penalize parameters with high absolute value and add a component pointing towards zero to the gradient. The magnitude of this component increases as the value of the parameters goes away from zero. The strength of this regularization term is controlled by the hyper-parameters σ_i^2.

Feature Pruning

Feature pruning has two related functions. First, it reduces the size of the model, reducing its memory and computational complexity. It also contributes to regularizing the model, by removing noisy features, in particular uncommon features.

Obviously, features whose weight is zero can be removed from the model without affecting performance. Regularization can be used to increase the number of feature where this applies. In particular, l_1 regularization can be used for feature pruning [35]. This can be combined

with occurrence count and feature weight ranking, to obtain a target feature count[36]. In this particular case, an initial feature set is created by ranking the features' occurrence count and selecting the top N. After training, the features with the smallest weights are removed and replaced by new features.

2.3 Neural Network

Conditional random fields are one of the two components of NeuroCRFs, the main topic of this thesis. NeuroCRFs also use a neural network, a combination of simple units called artificial neurones. Those units are organized into a larger network. While the individual units are simple, the whole network is complex, and can learn to approximate many functions [37].

Feed forward neural networks are organized in layers of units. The outputs of a layer are connected to the inputs of the next layer. They are no connection between units belonging to the same layer.

Artificial neurones are composed of three components. First, an activation function, typically non-linear. Second, a bias value. Finally, a weight vector, with a weight per connection to another unit.

In general, the neural network G is composed of units g_i so that

$$g_i(G) = \sigma_i(GW_i + b_i), \tag{2.27}$$

where σ_i is the unit activation function, W_i are the weights connecting unit i to the other units and b_i is the unit's bias. This fully connected form is obviously intractable. If a unit $g_{l,i}$ is associated with a layer G_l, this becomes

$$g_{l,i}(G_{l-1}) = \sigma_{l,i}(G_{l-1}W_{l,i} + b_{l,i}). \tag{2.28}$$

This restriction results in a feed forward neural network (FFNN). An example of FFNN is shown in Figure 2.3.

A set of units are assigned as output and a set is assigned as input. For a feed forward neural network, the input units are the lowest layer, and the output units are the top layer. The other layers are called "hidden layers", since they cannot be directly observed from outside the network. With L hidden layers containing N_l units each, the output layer

$G_{L+1}(\mathbf{x})$ is a composition of L layers $G_l(\mathbf{x})$:

$$G_{L+1}(\mathbf{x}) = \sigma_{L+1}(G_L(\mathbf{x})W_{L+1} + B_{L+1}) \tag{2.29}$$

$$G_l(\mathbf{x}) = \sigma_l(G_{l-1}(\mathbf{x})W_l + B_l) = \sigma_l(a_l(\mathbf{x})) \tag{2.30}$$

$$G_0(\mathbf{x}) = \mathbf{x}. \tag{2.31}$$

The dimensionality of the output layer $G_{L+1}(\mathbf{x})$ is fixed to the dimensionality of the function being approximated. The number of hidden layers L and the size of those layers $\{N_1, \ldots, N_L\}$ are not fixed. The parameters of this network are the $L+1$ weight matrices W_l and the $L+1$ biases matrices B_l.

The activation function of the last layer is usually a simple non-parametric function, chosen to accomplish a specific task. The softmax function is suitable for many classifications tasks.

$$\text{softmax}(a_{L+1}(\mathbf{x})) = \left[\frac{\exp a_{L+1,1}(\mathbf{x})}{Z(\mathbf{x})}, \frac{\exp a_{L+1,2}(\mathbf{x})}{Z(\mathbf{x})}, \ldots, \frac{\exp a_{L+1,N_{L+1}}(\mathbf{x})}{Z(\mathbf{x})}\right] \tag{2.32}$$

$$Z(\mathbf{x}) = \sum_{i=1}^{N_{L+1}} \exp a_{L+1,i}(\mathbf{x}) \tag{2.33}$$

With a softmax activation function, a NN can learn to approximate a posterior probability $P(y|\mathbf{x}) = \text{softmax}_y(a_{L+1}(\mathbf{x}))$. This can then be used for classification, so that $\hat{y} = \arg\max_y P(y|\mathbf{x})$.

Time Dimension

In the preceding description of NNs, we assumed that there is no time component. That is, we assumed that $\mathbf{x}$ and $G_l(\mathbf{x})$ are vectors. It is common to have a temporal component, where $\mathbf{x}$ and $G_l(\mathbf{x})$ are matrices, whose rows correspond to time indices.

This does not require any significant change for the NN itself. With T time indices, G_l becomes:

$$G_l(\mathbf{x}) = \left[G_{l,1}(\mathbf{x}_1)^\top, G_{l,2}(\mathbf{x}_2)^\top, \ldots, G_{l,T}(\mathbf{x}_T)^\top\right]^\top \tag{2.34}$$

$$G_{l,t}(\mathbf{x}_t) = \sigma_l(G_{l-1,t}(\mathbf{x}_t)W_l + B_l) \tag{2.35}$$

$$\mathbf{x} = \left[\mathbf{x}_1^\top, \mathbf{x}_2^\top, \ldots, \mathbf{x}_T^\top\right]^\top. \tag{2.36}$$

Its corresponding activation matrix is

$$\mathbf{a}_l = \left[\mathbf{a}_{l,1}^\top, \mathbf{a}_{l,2}^\top, \ldots, \mathbf{a}_{l,T}^\top\right]^\top, \tag{2.37}$$

$$\mathbf{a}_{l,t} = G_{l-1,t}(\mathbf{x}_t)W_l + B_l. \tag{2.38}$$

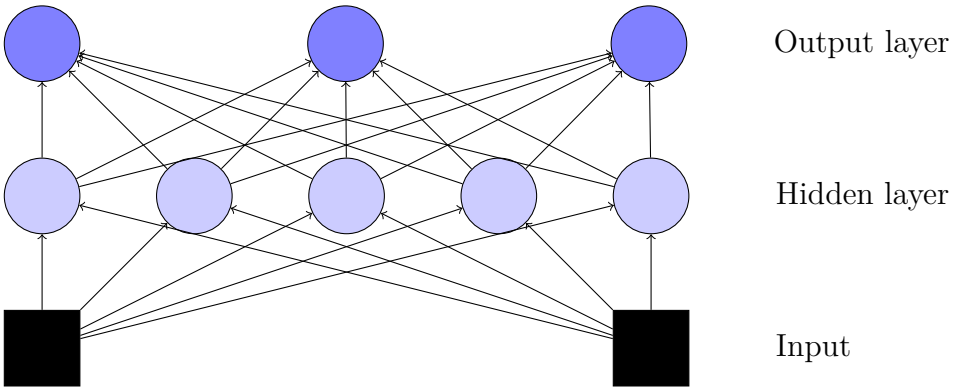


Fig. 2.3 Example of feed forward neural network with 3 output units, 5 hidden units and a bi-dimensional input.

2.3.1 Back Propagation

Numerical optimization is used to find good NN parameters In general, this is done by minimizing some loss function for a training corpus of relevant data. The loss function used varies depending on the task but, in general, minimizing or maximizing a function requires the gradient of its parameters.

With a training corpus "train" composed of training pairs $(\mathbf{y}, \mathbf{x})$, where $\mathbf{x}$ is an observation matrix and $\mathbf{y}$ is a target matrix, the loss function is usually a linear function of per-example losses, such as

$$L(\text{train}) = \sum_{(\mathbf{y},\mathbf{x})\in\text{train}} L(\mathbf{y}, \mathbf{x}), \tag{2.39}$$

where $L(\mathbf{y}, \mathbf{x})$ is the per-example loss. Because of this, obtaining the gradient of the parameters with respect to the per-example loss function is sufficient. It can then be used with an optimization algorithm such as stochastic gradient descent (SGD), presented in Section 3.3.1.

In order to do so, the gradient of the loss function with respect to the NN's output units, which can be obtained directly, is *back propagated* [38] from those units to the units connected to them. This process is repeated until the gradient of every unit is obtained. Fundamentally, back propagation is simply the systematic and recursive application of the chain rule.

The per-example loss is a function of the NN output matrix, such as

$$L(\mathbf{y}, \mathbf{x}) = f(G_{L+1,1}(\mathbf{x}_1), \ldots, G_{L+1,T}(\mathbf{x}_T)). \tag{2.40}$$

Because of this, the gradient of parameter θ_l of layer G_l is

$$\nabla_{\theta_l} L(\mathbf{y}, \mathbf{x}) = g\left(\frac{\partial G_{L+1,1}(\mathbf{x}_1)}{\partial \theta_l}, \ldots, \frac{G_{L+1,T}(\mathbf{x}_T)}{\partial \theta_l}\right). \tag{2.41}$$

A given unit i of layer G_l, at time step t, is connected to other units. When the derivative of the loss function with respect to those other units is know, it is back propagated to $G_{l,t,i}$. This process is repeated for all units at all time step. In the case of layers like Equation 2.30,

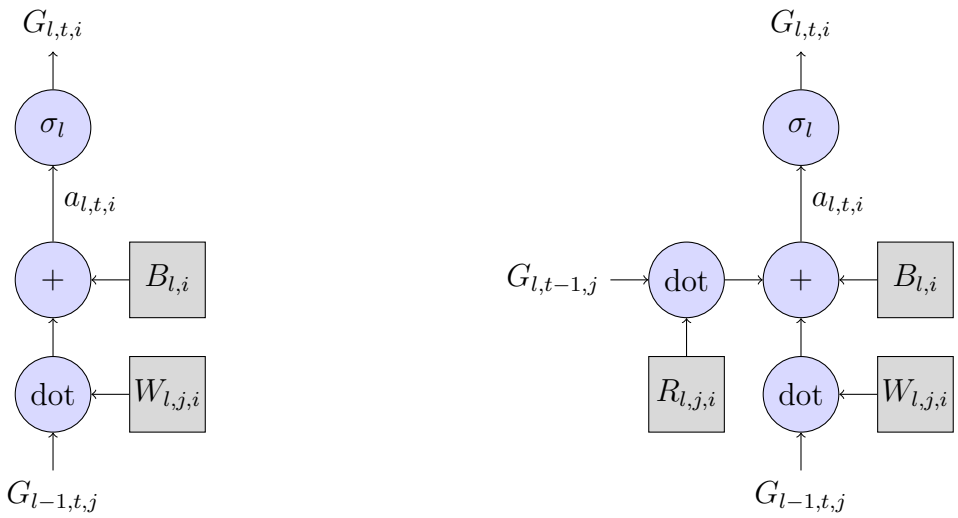
$$\frac{\partial L(\mathbf{y}, \mathbf{x})}{\partial G_{l,t,i}} = \sum_j \frac{\partial L(\mathbf{y}, \mathbf{x})}{G_{l+1,t,j}} \sigma'_{l+1}(a_{l+1,t,j}) W_{l+1,i,j}, \tag{2.42}$$

$$\sigma'_l(a_{l,t,i}) = \frac{\partial \sigma_l(a_{l,t,i})}{\partial a_{l,t,i}}, \tag{2.43}$$

where $\frac{\partial L(\mathbf{y}, \mathbf{x})}{G_{l+1,t,j}}$, the derivative of the loss function with respect to unit j of the layer G_{l+1} at time step t, is already known. Figure 2.4a shows the computation graph for a single unit. Back propagation through a unit follows this graph backward.

Parameters Gradient

Finally, once the gradient has been back propagated to all units and time steps in a layer G_l, the chain rule is then applied one more time, to get to the gradient of its parameters θ_l: In the case of Equation 2.30, whose parameters are $\theta_l = \{W_l, B_l\}$, the required derivatives



(a) Feed Forward (b) Recurrent

Fig. 2.4 Computation graph for unit i of layer l at time t. "dot" indicates matrix multiplication.

are

$$\frac{\partial G_{l,t,j}(\mathbf{x}_t)}{\partial W_{l,i,j}} = \sigma_l'(a_{l,t,j}) G_{l-1,t,i}(\mathbf{x}_t) \tag{2.44}$$

$$\frac{\partial G_{l,t,i}(\mathbf{x}_t)}{\partial B_{l,i}} = \sigma_l'(a_{l,t,i}). \tag{2.45}$$

Finally,

$$\frac{\partial L(\mathbf{y},\mathbf{x})}{\partial W_{l,i,j}} = \sum_t \frac{\partial L(\mathbf{y},\mathbf{x})}{\partial G_{l,t,j}(\mathbf{x}_t)} \frac{\partial G_{l,t,j}(\mathbf{x}_t)}{\partial W_{l,i,j}}, \tag{2.46}$$

$$\frac{\partial L(\mathbf{y},\mathbf{x})}{\partial B_{l,i}} = \sum_t \frac{\partial L(\mathbf{y},\mathbf{x})}{\partial G_{l,t,i}(\mathbf{x}_t)} \frac{\partial G_{l,t,i}(\mathbf{x}_t)}{\partial B_{l,i}}. \tag{2.47}$$

2.4 Recurrent Neural Network

Recurrent neural networks (RNNs) are NN containing one or more recurrent layers. Equation 2.30 defines feed-forward (FF) layers. Its recurrent counterpart, with added recurrent weight matrix R_l, is

$$G_{l,t}(\mathbf{x}) = \sigma_l(G_{l-1,t}(\mathbf{x}) W_l + G_{l,t-1}(\mathbf{x}) R_l + B_l) = \sigma_l(a_{l,t}), \tag{2.48}$$

$$G_{l,0}(\mathbf{x}) = 0. \tag{2.49}$$

The layer output is organized as a matrix whose rows correspond to time indices,

$$G_l(\mathbf{x}) = \left[G_{l,1}(\mathbf{x})^\top, G_{l,2}(\mathbf{x})^\top, \ldots, G_{l,T}(\mathbf{x})^\top\right]^\top. \tag{2.50}$$

Figure 2.4b shows the computation graph for a recurrent unit.

FF NNs assume that the observation at time t and the output at time $t' \neq t$ are independent. Since RNNs do not make this assumption, they can learn to extract features depending on previous observations. This is useful for many tasks where there is a dependency on previous inputs, such as language modelling [39, 40], acoustic modelling [41, 42], and machine translation [43]. While it is possible to use previous input without using recurrent layer, for example by using an input window containing surrounding observations, RNN have the advantage of being able to retain information indefinitely, as they are not limited by a finite window.

2.4.1 Back propagation Through Time

Section 2.3.1 explained back propagation for a FF NN. Back propagation needs to be modified, to incorporate the time recursion, in order to be applied to a RNN. This form of back propagation is known as back propagation through time (BPTT) [44].

As in Section 2.3.1, a given unit i of layer G_l, at time step t, is connected to other units. In the case of the FF NN of Section 2.3.1, those units are $G_{l+t,t,j}$, the units of the following layer at the same time step t. Those connections still applies to RNN. The recurrent connections, to units $G_{l,t+1,j}$, the units of the same layer at the following time step $t+1$, need to be added to Equation 2.30.

$$\frac{\partial L(\mathbf{y}, \mathbf{x})}{\partial G_{l,t,i}(\mathbf{x})} = \mathrm{FF}_{l,t,i} + \mathrm{Rec}_{l,t,i}, \tag{2.51}$$

$$\mathrm{FF}_{l,t,i} = \sum_j \left[\frac{\partial L(\mathbf{y}, \mathbf{x})}{\partial G_{l+1,t,j}(\mathbf{x})} \sigma'_{l+1}(a_{l+1,t,j}) W_{l+1,i,j} +\right], \tag{2.52}$$

$$\mathrm{Rec}_{l,t,i} = \sum_j \left[\frac{\partial L(\mathbf{y}, \mathbf{x})}{\partial G_{l,t+1,j}(\mathbf{x})} \sigma'_l(a_{l,t+1,j}) R_{l,i,j}\right], \tag{2.53}$$

$$\sigma'_l(a_{l,t,i}) = \frac{\partial \sigma_l(a_{l,t,i})}{\partial a_{l,t,i}}, \tag{2.54}$$

where $\frac{\partial L(\mathbf{y},\mathbf{x})}{\partial G_{l+1,t,j}(\mathbf{x})}$, the derivative of the loss function with respect to unit j of the layer G_{l+1} at time step t, and $\frac{\partial L(\mathbf{y},\mathbf{x})}{\partial G_{l,t+1,j}(\mathbf{x})}$, the derivative of the loss function with respect to unit j of the layer G_l at time step $t+1$, are already known. BPTT follows, backward, the computation graph for a recurrent unit, shown in Figure 2.4b.

Vanishing and Exploding Gradient

RNNs suffer from the vanishing and exploding gradient problem [45, 46], where the gradient tend to either vanish to zero or to increase exponentially during BPTT. One of the advantages of long-short term memory (LSTM) layers, which will be discussed in details in Chapter 5, is that they are more resilient to the vanishing gradient problem [47].

Assuming a simple layer, consisting of a single unit, with linear activation, the output of this unit, at time t is

$$y_t = x_t W + y_{t-1} R, \tag{2.55}$$

The contribution of the output y_{t-d} to the output y_t is multiplied by an exponential function of the recurrent weight. The gradient of y_t is back propagated to y_{t-d} by

$$\frac{\partial y_t}{\partial y_{t-d}} = R^d. \tag{2.56}$$

This simple example shows the fundamental nature of the problem. The repeated multiplication by the recurrent weights will tend to drive the gradient towards either zero or infinite values. The exact threshold depends on the activation function and the overall weight matrix [46].

Parameters Gradient

Once the gradient has been back propagated to all units and time steps in a layer G_l, the chain rule is then applied one more time, to get to the gradient of its parameters θ_l: In the case of Equation 2.48, whose parameters are $\theta_l = \{W_l, R_l, B_l\}$, the required derivatives are

$$\frac{\partial G_{l,t,j}(\mathbf{x}_t)}{\partial W_{l,i,j}} = \sigma'_l(a_{l,t,j}) G_{l-1,t,i}(\mathbf{x}), \tag{2.57}$$

$$\frac{\partial G_{l,t,j}(\mathbf{x})}{\partial R_{l,i,j}} = \sigma_l'(a_{l,t,j}) G_{l,t-1,i}(\mathbf{x}) \tag{2.58}$$

$$\frac{\partial G_{l,t,i}(\mathbf{x})}{\partial B_{l,i}} = \sigma_l'(a_{l,t,i}). \tag{2.59}$$

Finally,

$$\frac{\partial L(\mathbf{y},\mathbf{x})}{\partial W_{l,i,j}} = \sum_t \frac{\partial L(\mathbf{y},\mathbf{x})}{\partial G_{l,t,j}(\mathbf{x})} \frac{\partial G_{l,t,j}(\mathbf{x})}{\partial W_{l,i,j}}, \tag{2.60}$$

$$\frac{\partial L(\mathbf{y},\mathbf{x})}{\partial R_{l,i,j}} = \sum_t \frac{\partial L(\mathbf{y},\mathbf{x})}{\partial G_{l,t,j}(\mathbf{x})} \frac{\partial G_{l,t,j}(\mathbf{x})}{\partial R_{l,i,j}}, \tag{2.61}$$

$$\frac{\partial L(\mathbf{y},\mathbf{x})}{\partial B_{l,i}} = \sum_t \frac{\partial L(\mathbf{y},\mathbf{x})}{\partial G_{l,t,i}(\mathbf{x})} \frac{\partial G_{l,t,i}(\mathbf{x})}{\partial B_{l,i}}. \tag{2.62}$$

2.5 Continuous Word Representation

While NNs can learn useful features and perform feature analysis, their input must be in a vectorial form. For the NLU tasks considered in this thesis, the input is based on words. Each individual word can be represented by a one-hot vector. The dimensionality of this vector is equal to the size of the vocabulary. Most elements are set to 0, with only one exception: the element whose index is the index of the represented word in the vocabulary. In effect, the most basic representation of words is a very large, very sparse vector.

This kind of word representation is not practical. Similar words do not have similar vectors; the NN cannot generalize to words that were not seen during training. It is possible to create an alternative representation where similar words have similar vectors. In those cases, the very large, very sparse one-hot vector is replaced by a continuous vector of much smaller dimensionality. This representation is often called a "word embedding" or simply "embedding". It can be interpreted as a low dimensionality space embedded in the initial one-hot space. Word representations can be trained on unannotated data. This kind of representation was initially developed for language modelling [10]. They can be re-used for different NLU tasks.

Bengio et al. found that the combination of a continuous word representation and a neural network resulted in a significant perplexity reduction compared to similar n-gram language models. Le et al. used a similar representation in a class-based neural network

language model applied to a Mandarin Chinese ASR task [48]. Collobert presented two architectures combining input representation and neural networks [49, 3]. Turian et al. presented a comparison of the performances of various word representations, used as part of the feature set of a CRF for two NLU tasks [8]. Other approaches learn representation to predict surrounding words [50, 51], as well as presence or absence in lexicons [9].

While we settled on a specific word representation during the experimental studies, to control for the impact of the word representation [52], continuous word representation is an active area of research [53, 54, 55].

Similarity between words is ill-defined. Whether or not two words are similar to each other is more of an intuitive idea than something that can be reliably tested. While it is hard to define similarity in general, it is fairly easy to create pairs of similar words, such as city names, countries, verbs, stems, etc. In practise, works on word representations will usually present an informal and anecdotal demonstration that similar words have similar representation. This informal demonstration is followed by a more concrete and formal experimental study, where the representation is used for a real task.

Several approaches capable of generating reasonable word representations have been proposed. We will briefly introduce some in the following subsections.

2.5.1 Ranking approaches

Creating a representation using unannotated data is done by training a simple model on a task which does not require annotation. This is the case of language modelling, where the goal is simply to estimate the likelihood $p(w|h)$ that the word w follows the history h. When creating a representation, the goal is to learn features to represent words. The likelihood itself is of limited interest. Rather than train on likelihood, Collobert and Weston proposed the use of a ranking criterion [49, 3].

In this approach, a true window $\mathbf{w}$ is extracted from the unannotated data. The central word is replaced by another word, forming a "false" window $\mathbf{w}'$. A NN is trained to minimize the cost $c(\mathbf{w}, \mathbf{w}')$ for all pairs $(\mathbf{w}, \mathbf{w}')$

$$c(\mathbf{w}, \mathbf{w}') = \max\left(0, 1 - m(\mathbf{w}, \mathbf{w}')\right) \tag{2.63}$$

$$m(\mathbf{w}, \mathbf{w}') = \mathcal{N}(\mathbf{w}) - \mathcal{N}(\mathbf{w}'), \tag{2.64}$$

where $m(\mathbf{w}, \mathbf{w}')$ is the margin between the score $\mathcal{N}(\mathbf{w})$ of the true window and the score $\mathcal{N}(\mathbf{w}')$ of the false window. The representation learning network's input is window of word ids, indexing into a lookup table. This lookup table contains the continuous word representations. The entry in the lookup table corresponding to a word is tied and shared by all positions in the input window.

2.5.2 Skip-gram models

Skip-grams models [50, 51] are used to predict the words preceding and following a known word. This involves learning a word representation that can be used for other tasks. The skip-gram model is a NN using a single linear hidden layer, with a softmax output layer. Each word is represented by two continuous vectors of size H, $\mathbf{w}^{(i)}$ and $\mathbf{w}^{(o)}$. The first vector, $\mathbf{w}^{(i)}$, is equivalent to the weights between a one-hot input layer and the hidden layer. The second vector is equivalent to the weights between the hidden layer and the output layer. H is the size of the hidden layer. In both cases, those weights are stored in a lookup table and only the ones actually required are used. The input vector $\mathbf{w}^{(i)}$ is used as the representation.

Given a sequence of T words, a skip-gram model should maximize

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t), \tag{2.65}$$

where w_t is the word in position t and c is the size of the context, that is the number of word either preceding or following the central word that should be predicted.

$$p(w_{t+j}|w_t) = \frac{\exp\left(\mathbf{w}_t^{(i)} \cdot \mathbf{w}_{t+j}^{(o)}\right)}{\sum_{k=1}^{W}\exp\left(\mathbf{w}_t^{(i)} \cdot \mathbf{w}_k^{(o)}\right)}, \tag{2.66}$$

where W is the size of the vocabulary, $\mathbf{w}_t^{(i)}$ is the representation of the word at position t in the training sequence, $\mathbf{w}_k^{(o)}$ is the output representation of word w_k and $\mathbf{w}_{t+j}^{(o)}$ is the output representation of word w_{t+j} at position $t+j$ in the training sequence.

For non-trivial vocabulary size W, computing $\nabla p(w_{t+j}|w_t)$ is impractical. Negative sampling is an alternative, which optimizes a modified objective function

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log\sigma(\mathbf{w}_t^{(i)} \cdot \mathbf{w}_{t+j}^{(o)}) + \sum_{k=1}^{K} E_{\mathbf{w}^{(o)}\sim P_n(\mathbf{w})}\left[\log\sigma(-\mathbf{w}_t^{(i)} \cdot \mathbf{w}^{(o)})\right], \tag{2.67}$$

where $\sigma(x)$ is the logistic function, K is an hyper-parameter defining the number of negative samples to use, and $P_n(\mathbf{w})$ is a noise distribution.

2.5.3 Continuous Bag of Words models

Continuous bag of words (CBoW) [51] models are similar to skip-gram models. While skip-gram models use a central word to predict the surrounding words, CBoW models use the surrounding words to predict the central word. They use the same parameters, but combine them differently. The hidden layer of a skip-gram is the representation of the central word. The hidden layer of a CBoW is the sum of the representations of the surrounding words.

Given a sequence of $T = 2C+1$ words, a CBoW model should maximize $\log p(w_C|\{w_t|t \in T, t \neq C\})$.

$$H = \sum_{t \neq C} \mathbf{w}_t^{(i)}, \tag{2.68}$$

$$p(w_C|\{w_t|t \in T, t \neq C\}) = \frac{\exp\left(H \cdot \mathbf{w}_C^{(o)}\right)}{\sum_{k=1}^{W} \exp\left(H \cdot \mathbf{w}_k^{(o)}\right)}, \tag{2.69}$$

where H is the hidden layer, W is the size of the vocabulary, $\mathbf{w}_t^{(i)}$ is the representation of the word at position t in the training sequence, $\mathbf{w}_k^{(o)}$ is the output representation of word w_k and $\mathbf{w}_{t+j}^{(o)}$ is the output representation of word w_{t+j} at position $t+j$ in the training sequence.

Computing $\nabla p(w_C|\{w_t|t \in T, t \neq C\})$ has the same complexity as computing $\nabla p(w_{t+j}|w_t)$, its skip-gram equivalent, and is impractical for large vocabularies. Negative sampling is applicable to CBoW, resulting in the modified objective function

$$\log \sigma(H \cdot \mathbf{w}_C^{(o)}) + \sum_{k=1}^{K} E_{\mathbf{w}^{(o)} \sim P_n(\mathbf{w})} \left[\log \sigma(-H \cdot \mathbf{w}_k^{(o)})\right], \tag{2.70}$$

where $\sigma(x)$ is the logistic function, K is an hyper-parameter defining the number of negative samples to use, and $P_n(\mathbf{w})$ is a noise distribution.

2.6 Performance Measures

In this thesis, performance is evaluated in term of F_1, the harmonic mean of precision and recall. High F_1 indicate that a system was able to retrieve the segments that are present in

the reference, without also retrieving segments that are not in the reference. Recall is the percentage of segment found in the reference that were retrieved by the system. Precision is the percentage of segment retrieved by the system that are also in the reference.

$$F_1 = 2\frac{pr}{p+r} = 2\frac{c}{n+m}, \tag{2.71}$$

where c is the number of segment retrieved also present in the reference, n is the number of segment in the reference, m is the number of segment retrieved, $p = c/m$ is the precision, and $r = c/n$ is the recall. Usually, F_1 is multiplied by 100 and reported as a number between 0 and 100 rather than 0 and 1.

2.6.1 Classification Accuracy and Segmental F_1

It is also possible to factorize F_1 into two other related measures, so that

$$F_1 = A_c F_1^{(s)} = \frac{c}{c_s}\frac{2c_s}{n+m}, \tag{2.72}$$

where $A_c = c/c_s$ is the *classification accuracy*, c_s is the number of segment correctly retrieved, disregarding the class, and $F_1^{(s)}$ is the segmental F_1. $F_1^{(s)}$ is simply F_1 disregarding the class of the retrieved segments. High $F_1^{(s)}$ indicates that the system is capable of identifying segment boundaries. High classification accuracy A_c indicates that, given correct segmentation, the system is capable of classifying segments accurately.

Classification accuracy should be distinguished from label accuracy

$$A_l(\mathbf{y}', \mathbf{y}) = \frac{1}{T}\sum_{t=1}^{T}\begin{cases} 1, & y'_t = y_t \\ 0, & y'_t \neq y_t \end{cases},$$

which does not take into account segmentation, and is a label level performance measure. This measure is not usually a good performance measure, for example in cases where most words are assigned the same label. In those cases, a system that completely ignores its input and always assign this label will have a high label accuracy, despite its poor real performance.

2.7 Datasets

The sequence labelling techniques described in this thesis are evaluated on three datasets. Those datasets contain separate training and test corpora. The training corpora are used to learn the models' parameters. Some decisions, such as the various hyper-parameters controlling the machine learning algorithm used, are based on a small validation subset of the training corpus. This validation corpus is removed from the training data.

One dataset is used to evaluate performance on a chunking task. The two others are used to evaluate performance on named entity recognition tasks.

This section will present the details of those two tasks, as well as some statistics describing the datasets. It also contains a high level qualitative description of the three datasets.

2.7.1 Chunking

Chunking consists of segmenting a sentence into groups of syntactically related words. The resulting segmentation is dense, with very few words outside segments. In particular, the end point of a segment is usually the starting point of the next segment. This high density increases the importance of the segmentation, since any error will usually affects two segments.

Those experiments use the CoNLL-2000 shared task[56]. This dataset contains sentences extracted from the Wall Street Journal (WSJ). The reference segmentation is automatically derived from the Penn Treebank II corpus [57]. The segments are obtained by flattening the parse tree, using the syntactic category. Segments usually contain the syntactic head of a sub-tree and all the elements to its left. For example, a noun phrase will contain determiners and adjectives coming before the noun as well as the noun itself. The automatic flattening results in some inconsistent labels, due to some especially complex parse trees. Figure 2.5 shows the segmentation and labelling of an example sentence.

The segments are assigned to one of 11 classes. This results in 23 labels: two per class and one label to indicate that a word is not part of any segment. The CoNLL-2000 dataset does not contain a standard validation corpus. 1000 sentences were randomly selected and used to create the validation corpus used in the experiments presented in this thesis. Table 2.1 details the size, in words, sentences and segments, of the training, validation and test corpus. Table 2.2 shows the distribution of segment type for the training, validation and test corpus. The NP segments, corresponding to noun-phrases, form approximately half of the dataset.

Those distributions are virtually identical for the training, validation and test corpus.

Confectionery products sales	also	had	strong growth	in	the quarter	.
B-NP I-NP I-NP	B-ADVP	B-VP	B-NP I-NP	B-PP	B-NP I-NP	O
NP	ADVP	VP	NP	PP	NP	

Fig. 2.5 Segmented and labelled example sentence from CoNLL-2000

	Training	Validation	Test
Words	188,112	23,615	47,377
Sentences	7,936	1,000	2,012
Segments	95,020	11,958	23,852

Table 2.1 Size of the CoNLL-2000 dataset, in words, sentences and segments

	Training	Validation	Test
NP	48,947 (52%)	6,134 (51%)	12,422 (52%)
VP	19,027 (20%)	2,440 (20%)	4,658 (20%)
PP	18,907 (20%)	2,374 (20%)	4,811 (20%)
ADVP	3,774 (4%)	453 (4%)	866 (4%)
SBAR	1,940 (2%)	267 (2%)	535 (2%)
ADJP	1,843 (2%)	217 (2%)	438 (2%)
PRT	495 (1%)	61 (1%)	106 (0%)
CONJP	50 (0%)	6 (0%)	9 (0%)
INTJ	28 (0%)	3 (0%)	2 (0%)
LST	8 (0%)	2 (0%)	5 (0%)
UCP	1 (0%)	1 (0%)	0 (0%)

Table 2.2 Distribution of segment class in the CoNLL-2000 dataset

2.7.2 Named entity recognition

Named entity recognition (NER) consists of extracting and classifying phrases corresponding to named entities. Named entities are identifier for organizations, persons, places, etc. The named entities are a major component of the information contained in a sentence. In general, sentences usually consist of a few named entities, and very few named entities are immediately followed by another named entity, resulting in isolated segments, distributed sparsely in the sentence.

Two NER datasets are used in the experiments presented in this thesis. Both use the same classes of named entities, with similar definitions. Both are based on written text,

grammatically correct and written in a formal style. The named entities are classified as person name, organization name, location name and miscellaneous name. The organization class includes team names, including national teams, business names, and other groups of people. The miscellaneous class includes movie and book titles, as well as other named entities that are not part of the three other classes. Figure 2.6 shows a segmented and labelled example sentence.

The first dataset is CoNLL-2003[4]. It consists of manually annotated newswire text, extracted from the Reuters corpus. It is a small dataset, with a standard split into training, validation and test corpora. Table 2.3 details the size, in words, sentences and segments, of the training, validation and test corpus. Table 2.4 shows the distribution of segment type for the training, validation and test corpus. The distribution is not dominated by any class, but miscellaneous named entities are less frequent than the three other types. The distributions are similar for the training, validation and test corpus.

	Training	Validation	Test
Words	203,621	51,362	46,435
Sentences	14,041	3,250	3,453
Segments	23,499	5,942	5,648

Table 2.3 Size of the CoNLL-2003 dataset, in words, sentences and segments

	Training	Validation	Test
ORG	6,321 (27%)	1,341 (23%)	1,661 (29%)
PER	6,600 (28%)	1,842 (31%)	1,617 (29%)
LOC	7,140 (30%)	1,837 (31%)	1,668 (30%)
MISC	3,438 (15%)	922 (16%)	702 (12%)

Table 2.4 Distribution of segment class in the CoNLL-2003 dataset

The small size of CoNLL-2000 and CoNLL-2003 limits the models. A larger corpus is needed to truly take advantage of some of the models presented in this thesis. The second NER datasets, WikiNER[58] is used to address this issue. It is a much larger corpus of semi-automatically annotated data extracted from Wikipedia. The annotation process is summarized below. The dataset was randomly separated into training, validation and test corpora. The validation and test corpora contains 20% of the named entities, equally distributed between them. Table 2.5 details the size, in words, sentences and segments,

of the training, validation and test corpus. Comparing Table 2.5, Table 2.3 and Table 2.1 clearly show the significantly larger size of WikiNER. Table 2.6 shows the distribution of segment type for the training, validation and test corpus. The distribution are similar for the training, validation and test corpus. While the classes are the same, the distribution is not the same as the one observed with CoNLL-2003.

	Training	Validation	Test
Words	2,798,532	351,322	349,752
Sentences	113,812	14,178	14,163
Segments	244,368	30,546	30,546

Table 2.5 Size of the WikiNER dataset, in words, sentences and segments

	Training	Validation	Test
ORG	39,795 (16%)	4,912 (16%)	4,891 (16%)
PER	77,010 (32%)	9,594 (31%)	9,613 (31%)
LOC	68,737 (28%)	8,718 (29%)	8,580 (28%)
MISC	58,826 (24%)	7,322 (24%)	7,462 (24%)

Table 2.6 Distribution of segment class in the WikiNER dataset

It	retooled	its	Angus	Shops	in	Montreal	to	produce	Valentine	tanks	...
O	O	O	B-LOC	I-LOC	O	B-LOC	O	O	B-MISC	I-MISC	...
			LOC			LOC			Misc		

Fig. 2.6 Segmented and labelled example sentence from WikiNER

Automatic Annotation of Wikipedia

As described in [58], WikiNER is created using an automated process. This automation enables the creation of a large corpus without the expense of manual annotations. It is a trade-off between data and quality. The process is not fully automatic, and relies on implicit annotations directly created by the editors of Wikipedia. WikiNER uses a validation process to select sentences after annotation. This process rejects a large number of ambiguous sentences. Due to the large size of Wikipedia, the resulting dataset is larger than the manually annotated datasets.

Pages are assigned to categories by editors, with links within a page to its categories. It is possible to create a high precision system that automatically classify pages into one of the

4 classes of named entities used. Similarly, editors add links between named entities and the relevant page. In effect, editors manually segment the sentences and the segments are then automatically classified using the linked page.

The resulting annotations are not identical to true manual annotations. This is partly caused by errors in the page classification system, but it is also caused by the goals of editors. Linked are used to point the reader toward more information on a given topic. The link does not always include all the words in the named entities. Similarly, it is not infrequent for compound named entities to contains links pointing toward their components as well as a link pointing to the article describing the whole. This segmentation is different from the segmentation usually specified by annotation directives for NER. For example, in the sentence "Of those Tigers tanks lost against the *United States Army...*", "United States" is labelled as a MISC segment, while "Army" is labelled as a ORG segment. In the sentence "Another modern *United States Army* unit...", "United States Army" is labelled as a ORG segment. The later follows correct annotation directives, while the former example does not. This example illustrates the trade-off of WikiNER: we accept a slightly lower quality dataset in order to get access to a much larger dataset. Manually annotating a corpus of the size of WikiNER would be both prohibitively expensive and time consuming. For our purpose, which is to test algorithms and models, WikiNER is acceptable. The models have to learn complex dependencies between input and output sequence, and those dependencies are present in WikiNER.

Chapter 3

NeuroCRF

This chapter will describe NeuroCRFs, a combination of neural networks (NNs) and conditional random fields (CRFs).[1] This combination is intended to reduce the feature engineering required by CRFs. The feature learning capacities of NNs are the key to this reduction. Two forms of NeuroCRFs will be described. The first form, low-rank NeuroCRFs, is similar to existing models. The second form, full-rank NeuroCRFs, is intended to exploit the superior modelling capacity of NN, and is an original contribution of this thesis. Following chapters will show that this concept of rank is a special case of a more general notion. This will be followed by the description of the training procedure used to train NeuroCRFs, including the techniques used to regularize the models.

This chapter will conclude with an experimental study of NeuroCRFs, compared to a CRF baseline with an equivalent level of feature engineering. This study will also include a comparison of full and low-rank NeuroCRFs.

3.1 NeuroCRF

"NeuroCRF" refers to a large family of models. In general, any model composed of a NN and a linear chain CRF belongs to this family. Within this family, models can still be assigned to a more specific class. The NN is used to compute an output matrix, with one row per word in the utterance. The number and role of the columns, as well as the nature of the NN will determine the intra-familial type of a model. This chapter will present two classes of

[1] The content of this chapter is based on work published in "Full-rank linear-chain neurocrf for sequence labeling" [11].

NeuroCRF, both based on feed forward NNs. Those two classes are special case of the same general form, and can be obtained by adding and removing constraints on the parameters. The general form will be presented first, and will be used to describe the model training algorithm.

Equation 3.1, which is a restatement of Equation 2.9, is the general form of a linear chain NeuroCRF. The model includes a NN, with L hidden layers and whose output is $G_{L+1,t}(\mathbf{x})$. $G_{l,t}(\mathbf{x})$ is row t of the matrix G_l in Equation 2.30.

$$\log p(\mathbf{y}|\mathbf{x}) = \left(\sum_{t=1}^{T} G_{L+1,t}(\mathbf{x}) F(y_t, y_{t-1})\right) - \log Z(\mathbf{x}) \tag{3.1}$$

$$G_{L+1,t}(\mathbf{x}) = G_{L,t}(\mathbf{x}) W_{L+1} + B_{L+1} \tag{3.2}$$

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \exp\left(\sum_{t=1}^{T} G_{L+1,t}(\mathbf{x}) F(y_t, y_{t-1})\right), \tag{3.3}$$

The model's parameter are the weight and biases of the NN. The feature count function C_t of Equation 2.9 is replaced by the last hidden layer of the NN, $G_{L,t}$.

The nature of the NN will determine the constraints on most parameters, independently of the class of NeuroCRF. The constraints applied to W_{L+1} and B_{L+1}, as well as $F(y_t, y_{t-1})$, do not depend on the class of NN used. The combination of NN class, constraints and $F(y_t, y_{t-1})$ determines the intra-family type of a NeuroCRF.

CRFs factorize complex distributions into a set of simpler functions. The parameters of $F(y_t, y_{t-1})$ are a set of events, in this case the presence of y_t and y_{t-1} in $\mathbf{y}$ at indexes t and $t-1$ respectively. Those events are associated with feature learned by the NN; the event to features mapping is controlled by $F(\cdot)$. For any given event (y_t, y_{t-1}), $G_{L+1,t}(\mathbf{x})F(y_t, y_{t-1})$ is the potential of this event. This is related to the log likelihood $\log P(y_t, y_{t-1})$, with the key difference that the potentials are not constrained to sum to 0, and can be positive or negative.

The factor functions of this general linear chain NeuroCRF are

$$\Psi_t(y_t, y_{t-1}, \mathbf{x}) = \exp\left(G_{L+1,t}(\mathbf{x}) F(y_t, y_{t-1})\right). \tag{3.4}$$

3.1.1 Full-Rank NeuroCRF

In [11], we introduced full-rank NeuroCRFs, as well as the notion of NeuroCRF rank. This work extended an existing class of models, introduced by Collobert et al., which are low-rank NeuroCRFs. An overview of low-rank NeuroCRFs is presented in the following subsection. When the output features selection function of the form $F(y_t, y_{t-1})$, it is possible to reformulate Equation 3.1 as

$$\log p(\mathbf{y}|\mathbf{x}) = \left(\sum_{t=1}^{T} F^{\top}(y_{t-1}) \mathrm{rs}(G_{L+1,t}(\mathbf{x}), N, N) F(y_t) \right) - \log Z(\mathbf{x}), \tag{3.5}$$

where $\mathrm{rs}(v, r, c)$ is a function that transform the vector v with rc elements into a r-by-c matrix, $F(y)$ is an indicator matrix with one element per possible label, N is the number of labels, and $G_{L+1,t}(\mathbf{x})$, the NN output layer with N^2 units, is equivalent to a transition matrix, so that the only output features are label-to-label transitions. Most elements of $F(y)$ are zero, the exception being the element corresponding to y. This notation is useful to illustrate the notion NeuroCRF rank, but at the cost of flexibility.

In a full-rank NeuroCRFs, the NN is used to model label-to-label transitions. The NN learns features from the input that predict a transition from a specific label to another label at a given time. In contrast, the NN of low-rank NeuroCRFs, such as those described in [3], learns features that predict the emission of a specific label at a given time, ignoring the previous label. The equivalent of Equation 3.5 for a low-rank NeuroCRF is

$$\log p(\mathbf{y}|\mathbf{x}) = \left(\sum_{t=1}^{T} F^{\top}(y_{t-1}) \left(R G_{L+1,t}(\mathbf{x}) + A \right) F(y_t) \right) - \log Z(\mathbf{x}), \tag{3.6}$$

where R is a constant N-by-1 matrix of ones, with one element per label, and A is a transition weight matrix. The matrix multiplication of R and $G_{L+1,t}(\cdot)$ is the low-rank equivalent of $\mathrm{rs}(\cdot)$. Both transform the NN output in a N-by-N matrix. In low-rank NeuroCRFs, the effective NN output matrix $\hat{G} = R G_{L+1,t}$ is a square matrix of rank 1. The transition weight matrix A is removed in Equation 3.5, as it is redundant with the biases of the NN output layer.

NeuroCRF Rank

In general, this concept of rank is applicable to all NeuroCRFs, since the effective NN output matrix is

$$\hat{G}(\mathbf{x}) = \begin{bmatrix} \hat{G}_{1,1}(\mathbf{x}) & \hat{G}_{1,2}(\mathbf{x}) & \cdots & \hat{G}_{1,N}(\mathbf{x}) \\ \hat{G}_{2,1}(\mathbf{x}) & \hat{G}_{2,2}(\mathbf{x}) & \cdots & \hat{G}_{2,N}(\mathbf{x}) \\ & \vdots & & \\ \hat{G}_{N,1}(\mathbf{x}) & \hat{G}_{N,2}(\mathbf{x}) & \cdots & \hat{G}_{N,N}(\mathbf{x}) \end{bmatrix}, \tag{3.7}$$

where $\hat{G}_{y_{t-1},y_t}(\mathbf{x}) = G_{L+1,t}(\mathbf{x})F(y_t, y_{t-1})$. The rank of $\hat{G}$ is the rank of the NeuroCRF.

3.1.2 Low-Rank NeuroCRF

Low-rank NeuroCRFs are related to hidden Markov models (HMMs) and have a similar structure. In a HMM, the joint distribution over input and output $\log p(\mathbf{y}, \mathbf{x})$ is factorized into $\log p(\mathbf{x}|\mathbf{y})p(\mathbf{y})$, where

$$\log p(\mathbf{x}|\mathbf{y}) = \sum_{t=1}^{T} \log p(x_t|y_t), \tag{3.8}$$

$$\log p(\mathbf{y}) = \sum_{t=1}^{T} \log p(y_t|y_{t-1}). \tag{3.9}$$

The joint distribution is decomposed into a set of emission log-likelihoods $\log p(x|y)$ and transition log-likelihoods $\log p(y_t|y_{t-1})$. A similar factorization is possible for CRFs and NeuroCRFs. In the general form used in Equations 2.9 and 3.1, this factorization involves constraints on W_{L+1} and B_{L+1}.

Equation 3.1 can be rewritten in a less general form, to incorporate those constraints.

$$\log p(\mathbf{y}|\mathbf{x}) = \left(\sum_{t=1}^{T} G_{L+1,t}(\mathbf{x})F(y_t) + F^{\top}(y_{t-1})AF(y_t) \right) - \log Z(\mathbf{x}), \tag{3.10}$$

where A is a transition weight matrix, and $F(y)$ is an indicator matrix, with one element per possible label. The NN output $G_{L+1,t}(\cdot)$ also contains one element per possible label. Previous work on NeuroCRF, such as of [59, 3, 60, 61], used this factorization into emission and transition parameters.

Equivalence with Collobert's Model

The window approach with sentence-level log-likelihood from "Natural Language Processing (almost) from Scratch" [3] is equivalent to a low-rank NeuroCRF. Equations 12 and 13 of [3] are equivalent to Equation 3.10. Equation 12 of [3] can be restated as

$$s([x]_1^T, [i]_1^T) = \sum_t \left([A]_{[i]_{t-1},[i]_t} + [f_\theta]_{[i]_t,t} \right) = \sum_{t=1} G_{L+1,t}(\mathbf{x}) F(y_t) + F^\top(y_{t-1}) A F(y_t), \quad (3.11)$$

where

- The input sequence $[x]_1^T$ becomes $\mathbf{x}$;
- The output sequence $[i]_1^T$ becomes $\mathbf{y}$;
- The output at time t $[i]_t$ becomes y_t;
- The NN output $[f_\theta]_{[i]_t,t}$ becomes $G_{L+1,t}(\mathbf{x})F(y_t)$;
- The Transition weight $[A]_{[i]_{t-1},[i]_t}$ becomes $F^\top(y_{t-1})AF(y_t)$.

In Equation 13 of [3], the logadd term is a direct equivalent of the $\log Z$ term in Equation 3.10. Both are the log-sum of $s([x]_1^T, [i]_1^T) = s(\mathbf{x}, \mathbf{y})$ for all possible output sequences.

3.1.3 General Form of Full and Low-Rank NeuroCRF

Full and low-rank NeuroCRFs are special cases of a more general form of NeuroCRFs. The formulations used in the previous subsections incorporate constraints that would otherwise be applied to the weights and biases. Those formulations, by transforming the NN output into a matrix rather than the usual vector, allowed the use of rank in order to differentiate them. In this subsection, we will detail the constraints required to obtain full and low-rank NeuroCRFs using the general form. This will allow us to present the training procedure only for the general form.

The general form of full and low-rank NeuroCRFs will use an output feature selection function so that

$$F(y_t, y_{t-1}) = \left[\begin{array}{cccccc} e_1(y_t) & \cdots & e_N(y_t) & f_1(y_t, y_{t-1}) & \cdots & f_{N^2}(y_t, y_{t-1}) \end{array} \right]^\top \quad (3.12)$$

$$e_i(y) = \begin{cases} 1, & i = y \\ 0, & i \neq y \end{cases} \tag{3.13}$$

$$f_i(y, y') = \begin{cases} 1, & i = Ny + y' \\ 0, & i \neq Ny + y' \end{cases}, \tag{3.14}$$

where N is the number of label, and N^2 the number of transitions. This creates a $N^2 + N$ column vector, where the first N elements are emission and the remaining N^2 elements are transitions.

With an hidden layer with H elements, the parameters of the output layer $G_{L+1,t}$, W_{L+1} and B_{L+1} are a H-by-N^2+N matrix and a N^2+N vector, respectively. Different constraints are required for full and low-rank NeuroCRFs.

Using Equation 3.12, the general form is equivalent to the specific form of a *full-rank* NeuroCRF if

$$\hat{B}_{L+1,Ny_{t-1}+y_t} + G_{L,t}(\mathbf{x})\hat{W}_{L+1,Ny_{t-1}+y_t} = B_{L+1,N+Ny_{t-1}+y_t} + G_{L,t}(\mathbf{x})W_{L+1,N+Ny_{t-1}+y_t}, \tag{3.15}$$

where W_{L+1} and B_{L+1} are the constrained weights and biases of the general form, and $\hat{W}_{L+1}$ and $\hat{B}_{L+1}$ are the full-rank NeuroCRF parameters. In this case, the weights and biases are constrained, and must be expressible as

$$B_{L+1} = \left[0, 0, \ldots, 0, \hat{B}_{L+1}\right]. \tag{3.16}$$

$$W_{L+1} = \left[0, 0, \ldots, 0, \hat{W}_{L+1}\right], \tag{3.17}$$

both matrices with N columns of zeros, followed by the full-rank parameters. The columns fixed to zero are the ones assigned to the emission output features. In practice, they are simply removed from the model, so that the output feature selection matrix includes only entries for transitions.

The general form is equivalent to the specific form of a *low-rank* NeuroCRF if

$$O_l(y_t, y_{t-1}) = \left(G_{L,t}(\mathbf{x})\hat{W}_{L+1} + \hat{B}_{L+1}\right) F(y_t) + F^\top(y_{t-1})AF(y_t), \tag{3.18}$$

$$O_c(y_t, y_{t-1}) = (G_{L,t}(\mathbf{x})W_{L+1} + B_{L+1})\, F(y_t, y_{t-1}), \tag{3.19}$$

$$O_l(y_t, y_{t-1}) = O_c(y_t, y_{t-1}), \tag{3.20}$$

where W_{L+1} and B_{L+1} are the constrained weights and biases of the general form, and $\hat{W_{L+1}}$, $\hat{B}_{L+1}$ and A are the low-rank NeuroCRF parameters.

Using Equation 3.12, this expands into

$$\hat{B}_{L+1,y_t} + A_{y_{t-1},y_t} = B_{L+1,y_t} + B_{L+1,N+Ny_{t-1}+y_t}, \tag{3.21}$$

$$G_{L,t}(\mathbf{x})\hat{W}_{L+1,y_t} = G_{L,t}(\mathbf{x})W_{L+1,y_t}, \tag{3.22}$$

$$0 = G_{L,t}(\mathbf{x})W_{L+1,N+Ny_{t-1}+y_t}. \tag{3.23}$$

There are no constraints on B_{L+1}, and models can be converted between the two forms using

$$B_{L+1} = \left[\hat{B}_{L+1}, \mathrm{rs}(A, 1, N^2)\right]. \tag{3.24}$$

The weights are constrained and must be expressible as

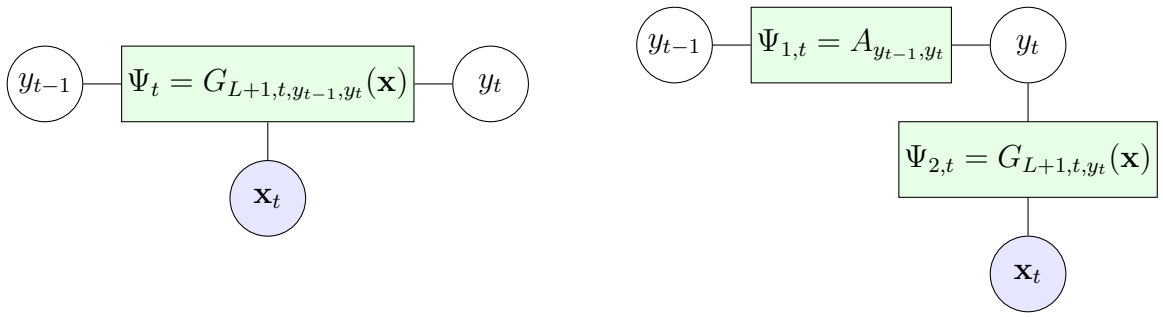
$$W_{L+1} = \left[\hat{W}_{L+1}, 0, 0, \ldots, 0\right], \tag{3.25}$$

a matrix where the first N columns are $\hat{W}_{L+1}$ and the remaining N^2 columns are fixed to zero.

While low and full-rank NeuroCRFs model the same set of events, their general forms show that the similarity is superficial. Low-rank NeuroCRFs use the NN to model emission only. A separate transition matrix is required to model transitions. Without this matrix, low-rank NeuroCRF would use a 0th order Markov assumption. Full-rank NeuroCRFs use the NN to model pair of label. This implicitly combines emission and transition, since the event (y_t, y_{t-1}) only occurs if the even y_t also occurs.

Figure 3.1a shows the clique template of full-rank NeuroCRFs. This is the NN-based counterpart of Figure 2.1a. Figure 3.1b shows the clique template of low-rank NeuroCRFs.

This is the NN-based counterpart of Figure 2.1b.



(a) Full-rank (b) Low-rank

Fig. 3.1 Clique templates of linear chain NeuroCRFs. Square boxes are factors, circles are variables.

3.1.4 Motivation

One of the key goal of NeuroCRFs is to replace the feature engineering, described in Section 2.2.2, of CRFs with the feature learning of neural networks. Both feature learning and feature engineering require work by an expert, and brute force approaches, such as random search for hyper-parameters and feature pruning, can be used to transfer some of the required work form the expert to a computer cluster.

One advantage of feature learning is that an expert on NN training can work on a wide variety of problems, in many domains such as image classification, automatic speech recognition and natural language understanding (NLU). All of those problems can be addressed using NNs. Similarly, while good feature sets have been developed for many NLU tasks, those feature sets are only available for some languages. Feature learning, on the other hand, does not rely on previous experience with the language; the only requirement is labeled data in a known format.

Finally, deep convolutional NNs are the state of the art in image processing. Those models have overtaken the previous state of the art, which was based on years of feature engineering. For example, the performance of [62], based on SIFT features [63], was improved upon by a deep convolutional NN in [64]. Similar improvements were observed for face verification [65] and other tasks [66].

While feature learning did not obtain the same dramatic improvement for NLU tasks, this

indicates that feature learning can not only equal but exceed the best engineered features[2].

3.1.5 Related Works

Conditional random fields have been used for sequence labeling since their introduction [5]. State of the art performance requires feature engineering, which leads to improvement from 84.04[6] to 90.90 [9] on CoNLL-2003 (NER).

Collobert et al. combined neural network and CRFs in [3] which resulted in a model similar to the low-rank NeuroCRF described below. Do et al. [59] used a similar combination[3]. NNs and CRFs have also been combined for joint intent detection and slot filling[67], once more using a low-rank NeuroCRF Similar combinations of NN and CRFs were also used in [68]. In [69], a similar approach was used, where the hidden layers of the NN were themselves equivalent to the output layer of a low-rank NeuroCRF.

Recent work combining neural network with rich features did outperform conventional CRF; this will be elaborated upon in Section 5.2. Finally, Sequence-to-Sequence models [70], discussed in more details in Section 5.2, have been used to create flatten parse trees [71], similar but richer to the output of the chunking task.

3.2 Dynamic Programming

The normalization term $Z(\mathbf{x})$ involves a summation over all possible output sequences. In general, the number of such sequence is an exponential function of T. Dynamic programming algorithms can be used to compute $Z(\mathbf{x})$ with a linear complexity, rather than the exponential complexity required by a naive summation [29]. The well known Viterbi algorithm [72] is also used to find the most likely output sequence for a given input sequence.

3.2.1 Forward Algorithm

The first algorithm is the *forward algorithm* [33]. It recursively computes a matrix α such that

$$\alpha_{t,y_t} = \sum_{\mathbf{y}_{\langle 1..t-1\rangle}} \exp\left(\sum_{t'=1}^{t} G_{L+1,t'}(\mathbf{x}) F(y_{t'}, y_{t'-1})\right). \tag{3.26}$$

[2]Recent works are starting to show improvements, as discussed in Section 5.2.
[3]We also coined "NeuroCRF" independently.

$Z(\mathbf{x})$ is the sum of all possible output sequences for their entire length and ending with any output label y_T. The value of α_{t,y_t} is the partial sum ending at time t and with a specific last output label y_t. Since $Z(\mathbf{x}) = \sum_{y_T} \alpha_{T,y_T}$, this matrix can be used to compute $Z(\mathbf{x})$.

The values $\alpha_{t,y}$ are computed with the recursion:

$$\alpha_{1,y_1} = \exp G_{L+1,1}(\mathbf{x}) F(y_1, \text{INIT}), \tag{3.27}$$

$$\alpha_{t,y_t} = \sum_{y_{t-1}} \alpha_{t-1,y_{t-1}} \exp G_{L+1,t}(\mathbf{x}) F(y_t, y_{t-1}), \tag{3.28}$$

where INIT indicates the initial state, and α_{1,y_1} initiates the recursion.

3.2.2 Backward Algorithm

The second algorithm is the *backward algorithm* [33]. It recursively computes a matrix β such that

$$\beta_{t,y_t} = \sum_{\mathbf{y}_{\langle t+1..T \rangle}} \exp \left(\sum_{t'=t+1}^{t} G_{L+1,T}(\mathbf{x}) F(y_{t'}, y_{t'-1}) \right). \tag{3.29}$$

The value of β_{t,y_t} is the partial sum starting at time $t+1$ and preceded by y_t. Like α, this matrix can be used to compute $Z(\mathbf{x})$:

$$Z(\mathbf{x}) = \beta_{0,\text{INIT}} = \sum_{y_1} \beta_{1,y_1} \exp G_{L+1,1}(\mathbf{x}) F(y_1, \text{INIT}). \tag{3.30}$$

The values $\beta_{t,y}$ are computed with the recursion:

$$\beta_{T,y_T} = 1 \tag{3.31}$$

$$\beta_{t,y_t} = \sum_{y_{t+1}} \beta_{t+1,y_{t+1}} \exp G_{L+1,t+1}(\mathbf{x}) F(y_t, y_{t-1}), \tag{3.32}$$

where β_{T,y_T} initiates the recursion.

3.2.3 Viterbi Algorithm

Finally, the Viterbi algorithm [72] can be used to find the most likely output sequence.

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \log p(\mathbf{y}|\mathbf{x}) = \arg\max_{\mathbf{y}} \sum_{t=1}^{T} G_{L+1,t}(\mathbf{x}) F(y_t, y_{t-1}) \tag{3.33}$$

The first step is the recursive computation of the δ matrix, whose elements δ_{t,y_t} are defined as

$$\delta_{t,y_t} = \max_{\mathbf{y}_{\langle 1..t-1 \rangle}} G_{L+1,t}(\mathbf{x}) F(y_t, y_{t-1}) + \sum_{t'=1}^{t-1} G_{L+1,t'}(\mathbf{x}) F(y_{t'}, y_{t'-1}). \tag{3.34}$$

This is analogous to α_{t,y_t}. While α_{t,y_t} is the sum for all $\mathbf{y}_{\langle 1..t \rangle}$ ending with y_t, δ_{t,y_t} is the maximum value. Like α, δ is computed recursively:

$$\delta_{1,y_1} = G_{L+1,1}(\mathbf{x}) F(y_1, \text{INIT}), \tag{3.35}$$

$$\delta_{t,y_t} = \max_{y_{t-1}} \delta_{t-1,y_{t-1}} + G_{L+1,t}(\mathbf{x}) F(y_t, y_{t-1}). \tag{3.36}$$

The second step is to backtrack recursively:

$$\hat{y}_T = \arg\max_{y_T} \delta_{T,y_T} \tag{3.37}$$

$$\hat{y}_t = \arg\max_{y_t} \delta_{t,y_t} + G_{L+1,t+1}(\mathbf{x}) F(\hat{y}_{t+1}, y_t) \tag{3.38}$$

3.3 Parameter Estimation

The values of the model parameters are estimated by maximizing the log-likelihood of a training corpus. This training corpus contains pairs of input and output sequences. This is equivalent to minimizing the loss function

$$L(\text{train}) = -\sum_{\substack{(\mathbf{x},\mathbf{y}) \in \\ \text{train}}} \log p(\mathbf{y}|\mathbf{x}) = \sum_{\substack{(\mathbf{x},\mathbf{y}) \in \\ \text{train}}} L(\mathbf{y}, \mathbf{x}), \tag{3.39}$$

where train is the training corpus and $L(\mathbf{y}, \mathbf{x}) = -\log p(\mathbf{y}, \mathbf{x})$ is the per-example loss function.

The loss function is minimized if

$$\sum_{\substack{(\mathbf{x},\mathbf{y})\in \\ \text{train}}} \sum_{t=1}^{T} \nabla(G_{L+1,t}(\mathbf{x}))F(y_t, y_{t-1})) = \sum_{\substack{(\mathbf{x},\mathbf{y})\in \\ \text{train}}} \frac{\nabla Z(\mathbf{x})}{Z(\mathbf{x})}, \tag{3.40}$$

where $\nabla(G_{L+1,t}(\mathbf{x}))F(y_t, y_{t-1}))$ and $\nabla Z(\mathbf{x})$ are gradients with respect to the parameters. In general, this cannot be solved in closed form, forcing the use of a numerical approach, such as stochastic gradient descent (SGD).

3.3.1 Stochastic Gradient Descent

Gradient descent (GD) minimizes $L(\text{train})$ by updating the parameters $\Theta = \{W, B\}$ recursively. Starting with an initial Θ_0, randomly selected and some learning rate λ, the parameters are updated by following the gradient:

$$\Theta_i = \Theta_{i-1} - \lambda \nabla L(\text{train}). \tag{3.41}$$

Stochastic gradient descent (SGD) use an approximation of $\nabla L(\text{train})$ to minimize $L(\text{train})$. This approximation is simply $\nabla L(\mathbf{y}, \mathbf{x})$. A training pair $(\mathbf{y}, \mathbf{x})$ is sampled from train and the parameters are updated using $\nabla L(\mathbf{y}, \mathbf{x})$. This technique is used when the training set is large, and computing a gradient is computationally expensive. Using SGD, updating the model's parameters requires only the gradient for a training pair $(\mathbf{y}, \mathbf{x})$.

The learning rate is the key hyper-parameter of SGD. It is used to scale the gradient before updating the parameters. Large learning rates accelerates learning, and will reduce the number of updates required to achieve convergence. Small learning rates are required to reach a local minimum. It is common to scale the learning rate during training, to have the advantage of a large learning rate in the initial updates, when the parameters are not close to a local minimum, and to have the advantage of a small learning rate in the latter updates, when the parameters are close to a local minimum.

The algorithm used in the following experiments uses a validation set to identify the best hyper-parameters, including the learning rate. Models are trained as long as their performance, on the validation set, improves. If validation performance does not improves, or gets worse, the learning rate is reduced and training resumed with the previous best parameters.

Computing $\nabla L(\mathbf{y}, \mathbf{x})$ is the main difficulty of parameter estimation, and is the focus of the remainder of this section.

Gradient

The first step is to define the gradient in terms of $\frac{\partial}{\partial \Theta} G_{L+1,t}(\mathbf{x})$. It will then be possible to back-propagate this gradient into the NN. To simplify notation, Equation 3.1 is rewriten using factor functions

$$\Psi_t(y_t, y_{t-1}, \mathbf{x}) = \exp\left(G_{L+1,t}(\mathbf{x}) F(y_t, y_{t-1})\right), \tag{3.42}$$

$$L(\mathbf{y}, \mathbf{x}) = -\log p(\mathbf{y}|\mathbf{x}) = \log Z(\mathbf{x}) - \sum_{t=1}^{T} \log \Psi_t(y_t, y_{t-1}, \mathbf{x}), \tag{3.43}$$

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, \mathbf{x}). \tag{3.44}$$

Since

$$\nabla \log \Psi_t(y_t, y_{t-1}, \mathbf{x}) = \frac{\partial}{\partial \Theta} G_{L+1,t}(\mathbf{x}) F^{\top}(y_t, y_{t-1}), \tag{3.45}$$

$$\nabla \Psi_t(y_t, y_{t-1}, \mathbf{x}) = \Psi_t(y_t, y_{t-1}, \mathbf{x}) \nabla \log \Psi_t(y_t, y_{t-1}, \mathbf{x}), \tag{3.46}$$

expressing $\nabla L(\text{train})$ in term of $\nabla \log \Psi_t(y_t, y_{t-1}, \mathbf{x})$ and $\nabla \Psi_t(y_t, y_{t-1}, \mathbf{x})$ is sufficient to allow back propagation of the cost through the NN.

The gradient for the loss function is

$$\nabla L(\text{train}) = \sum_{\substack{(\mathbf{x},\mathbf{y}) \in \\ \text{train}}} \nabla L(\mathbf{y}, \mathbf{x}), \tag{3.47}$$

$$\nabla L(\mathbf{y}, \mathbf{x}) = \frac{\nabla Z(\mathbf{x})}{Z(\mathbf{x})} - \sum_{t=1}^{T} \nabla \log \Psi_t(y_t, y_{t-1}, \mathbf{x}). \tag{3.48}$$

The product rules for derivative is applied to Equation 3.44, resulting in

$$\nabla Z(\mathbf{x}) = \sum_{\mathbf{y}} \sum_{t=1}^{T} \left(\prod_{t' \neq t} \Psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}) \right) \nabla \Psi_t(y_t, y_{t-1}, \mathbf{x}). \tag{3.49}$$

For convenience, we reorder the summation so that the sum over time is first:

$$\nabla Z(\mathbf{x}) = \sum_{t=1}^{T} \sum_{\mathbf{y}} \left(\prod_{t' \neq t} \Psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}) \right) \nabla \Psi_t(y_t, y_{t-1}, \mathbf{x}). \tag{3.50}$$

Since $\mathbf{y} = [\mathbf{y}_{\langle 1..t-2 \rangle}, y_{t-1}, y_t, \mathbf{y}_{\langle t+1..T \rangle}]$, Equation 3.50 can be factorized into 3 terms:

$$\nabla Z(\mathbf{x}) = \sum_{t=1}^{T} \sum_{\mathbf{y}} f_\alpha(\mathbf{y}_{\langle 1..t-2 \rangle}, y_{t-1}) f_t(y_t, y_{t-1}) f_\beta(y_t, \mathbf{y}_{\langle t+1..T \rangle}) \tag{3.51}$$

$$f_\alpha(\mathbf{y}_{\langle 1..t-2 \rangle}, y_{t-1}) = \Psi_{t-1}(y_{t-1}, y_{t-2}, \mathbf{x}) \prod_{t'=1}^{t-1} \Psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}), \tag{3.52}$$

$$f_\beta(y_t, \mathbf{y}_{\langle t+1..T \rangle}) = \prod_{t'=t+1}^{T} \Psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}), \tag{3.53}$$

$$f_t(y_t, y_{t-1}) = \nabla \Psi_t(y_t, y_{t-1}, \mathbf{x}) = \Psi_t(y_t, y_{t-1}, \mathbf{x}) \nabla \log \Psi_t(y_t, y_{t-1}, \mathbf{x}). \tag{3.54}$$

Finally, the summation over $\mathbf{y}$ can be decomposed to that

$$\nabla Z(\mathbf{x}) = \sum_{t=1}^{T} \sum_{(y_t, y_{t-1})} f_t(y_t, y_{t-1}) \left[\sum_{\mathbf{y}_{\langle 1..t-2 \rangle}} f_\alpha(\mathbf{y}_{\langle 1..t-2 \rangle}, y_{t-1}) \right] \left[\sum_{\mathbf{y}_{\langle t+1..T \rangle}} f_\beta(y_t, \mathbf{y}_{\langle t+1..T \rangle}) \right]. \tag{3.55}$$

Equation 3.55 can be simplified by including that α and β matrices defined by Equation 3.26 and Equation 3.29.

It is possible to obtain $\nabla Z(\mathbf{x})$ using the dynamic programming algorithms described earlier.

$$\nabla Z(\mathbf{x}) = \sum_{t=1}^{T} \sum_{(y_t, y_{t-1})} \alpha_{t-1, y_{t-1}} \beta_{t, y_t} \nabla \Psi_t(y_t, y_{t-1}, \mathbf{x}). \tag{3.56}$$

This formulation of $\nabla Z(\mathbf{x})$ can be modified to illustrate a characteristic of the parameters minimizing the loss function. The likelihood of a transition from y_{t-1} to y_t, at time t is

$$p(y_t, y_{t-1} | \mathbf{x}) = \frac{\alpha_{t-1, y_{t-1}} \beta_{t, y_t} \Psi_t(y_t, y_{t-1}, \mathbf{x})}{Z(\mathbf{x})}. \tag{3.57}$$

This likelihood can be used in Equation 3.56, to that

$$\nabla L(\mathbf{y}, \mathbf{x}) = \sum_{t=1}^{T} \sum_{(y_t, y_{t-1})} p(y_t, y_{t-1}|\mathbf{x}) \nabla \log \Psi_t(y_t, y_{t-1}, \mathbf{x}) - \sum_{t=1}^{T} \nabla \log \Psi_t(y_t, y_{t-1}, \mathbf{x}). \quad (3.58)$$

This is minimized when

$$\sum_{t=1}^{T} \sum_{(y_t, y_{t-1})} p(y_t, y_{t-1}|\mathbf{x}) \nabla \log \Psi_t(y_t, y_{t-1}, \mathbf{x}) = \sum_{t=1}^{T} \nabla \log \Psi_t(y_t, y_{t-1}, \mathbf{x}). \quad (3.59)$$

Finally, starting with Equation 3.58, it is possible to back-propagate the gradient down to the parameters. The gradient $\nabla_l L(\mathbf{y}, \mathbf{x})$ of the loss function with respect to the parameters of layer l θ_l is

$$\nabla_l L(\mathbf{y}, \mathbf{x}) = \sum_{t=1}^{T} \sum_{(y_t, y_{t-1})} p(y_t, y_{t-1}|\mathbf{x}) \frac{\partial G_{L+1,t}(\mathbf{x})}{\partial \theta_l} F(y_t, y_{t-1})^\top - \sum_{t=1}^{T} \frac{\partial G_{L+1,t}(\mathbf{x})}{\partial \theta_l} F(y_t, y_{t-1})^\top. \quad (3.60)$$

As described in Section 2.3.1 of Chapter 2, the chain rule is applied recursively to obtain $\frac{\partial G_{L+1,t}(\mathbf{x})}{\partial \theta_l}$ for all θ_l.

Recursive formulation

It is also possible to use back-propagation through time to propagate the gradient down to the factor functions.

This is done recursively, starting with

$$\frac{\partial L(\mathbf{y}, \mathbf{x})}{\partial \alpha_{T,y}} = \frac{\partial L(\mathbf{y}, \mathbf{x})}{\partial Z(\mathbf{x})}. \quad (3.61)$$

Then, for all $t < T$,

$$\frac{\partial L(\mathbf{y}, \mathbf{x})}{\partial \alpha_{t,y}} = \sum_{y'} \Psi_{t+1}(y', y, \mathbf{x}) \frac{\partial L(\mathbf{y}, \mathbf{x})}{\partial \alpha_{t+1,y'}}, \quad (3.62)$$

which is then propagated to the factor functions with

$$\frac{\partial L(\mathbf{y}, \mathbf{x})}{\partial \Psi_t(y_t, y_{t-1}, \mathbf{x})} = \frac{\partial L(\mathbf{y}, \mathbf{x})}{\partial \alpha_{t,y_t}} \alpha_{t-1,y_{t-1}}. \quad (3.63)$$

Back propagation can then be used to get the parameters' gradients.

The main advantage of this approach, compared to the more detailed process described earlier, is that it is easily implemented by the automatic derivative computation tools developed for NNs, such as Theano [73]. Those tools rely on systematic application of the chain rule to obtain, from a computation graph of simple operations, the graph corresponding to gradients.

3.3.2 Regularization

Overfitting occurs when the performance improves on the training data, but decrease on other, unseen data. Neural networks, as well as CRFs, tend to overfit data, and this remains true for NeuroCRFs. Norm regularization, described in Section 2.2.5, is used to limit overfitting. The experiments described below used L_2 norm regularization for the NN weights and the biases of the output layer. The word representation is not regularized.

Dropout [64, 74] is another form of regularization for NN. Its main function is to prevent co-adaptation between units. Co-adapted units are group of units effectively acting as a single unit: they always have correlated values. This grouping is usually caused by a combination of random initialization, output and input. This will cause overfitting, since a specific unit in the group can rely on a part of the input that is only coincidentally correlated, in the training data, with the features detected by the other units. If this happens, this unit might have a low value when it should have a high value. The units of the following layer that rely on the group will also have a low value, and so on.

Dropout limits this by, during training, randomly removing units from the NN. A random binary mask is applied to the hidden layers. The dropout rate is the probability of unit being removed from the network by this mask. The weights leading into a masked unit are not updated, since the mask block gradient propagation. Similarly, the weights leading from the units are also not updated, since their gradient is zeroed by the mask. Units whose weights are not updated will tend to drift from their group, which will break the random correlation. Finally, the units that relied on the masked units will have seen examples where those units are deactivated. In general, dropout reduces, without guarantees, correlation between units in the same layer.

3.4 Experimental Study

The performance of full and low-rank NeuroCRFs was compared to a CRF baseline for all three tasks presented in Section 2.7. In the case of chunking and NER using CoNLL-2003, we also included the state of the art results. Both are CRF with extensive feature engineering. Our CRF baseline is comparatively simpler and only required minimal feature engineering.

3.4.1 Datasets and Performance Metrics

The data used in those experiments is presented in Section 2.7. The performance metrics used are presented in Section 2.6.

3.4.2 Model Configurations

The training procedure described in the next subsection depends on some hyper-parameters, such as the learning rate and the size of the hidden layer. A random search [75] is used to find potential hyper-parameters. One hundred sets of hyper-parameters are generated randomly and are used to train models. The validation set is used to pick the 10 best configurations. For each of those configurations, 10 models are trained, with 10 different random initializations.[4] The final results are based on the set of hyper-parameters with the highest average validation score. Algorithm 1 summarizes the model configuration procedure.

The experiments below uses a IOB2 encoding of segments into labels. All experiments used a word representation trained on wikipedia data, with 100 dimensions normalized to have zero-mean and unit variance. This pre-trained word representation is used as the initialization for all NeuroCRFs. The hyper-parameters includes the hidden layer size, learning rate, norm regularization, and dropout rate. The hidden layer size and dropout rate are sampled from a uniform distribution, while the learning rate and norm regularization are sampled from a log-uniform distribution. All hyper-parameters are assumed to be independent of each others.

The input $\mathbf{x}$ includes some simple features:

1. Word, using their continuous word representation,

2. Capitalization feature extracted from the words, and

[4] Including the initialization used during the random search.

3. Part-of-speech tags, included in the corpora.

The input vector is a sliding window over those features, containing the features representing the current word, and some preceding and following words. The context size C is an hyper-parameter determining the number of preceding and following words to include in the sliding window. The sliding window contains $2C + 1$ words.

Data: training and validation corpus
Result: model configuration
for *hp, init in sample(100)* **do**
  model = train(hp, init)
  candidates_{hp} = val(model)
end
candidates = best(candidates, 10) **for** *hp in candidates* **do**
  for *init in sample(9)* **do**
    model = train(hp, init);
    $\text{score}_{hp} = \text{score}_{hp} + \text{val(model)}$
  end
  $\text{score}_{hp} = \text{score}_{hp}/10$
end
$\text{hp} = \arg\max \text{score}_{hp}$

Algorithm 1: Model configuration algorithm. "hp" is a set of hyper-parameters, "init" is a random initialization of the model's parameter, "candidates" and "score" are lists of validation scores with an entry per set of hyper-parameters. "sample" is a function sampling hyper-parameters and initialization for the random search, and "best" is used to select the hyper-parameters with the highest validation score.

3.4.3 Training Procedure

Models are trained using a variant of early stopping. Models are trained until performance stops improving on the validation set, even if performance still improves on the training set. This limits overfitting. Since obtaining the validation score takes a non-negligible amount of time, validation performance is evaluated twice per epoch. Since there is some randomness in the validation performance, training is not stopped immediately after the first drop in validation performance. Instead, the model's parameters are updated until no improvement was seen during the last half of updates. That is, if the last improvement was during the 5th update, training will stop after the 10th update. The model's parameters are saved

whenever a new best result, on validation, is observed. Those saved parameters are the ones returned by the training procedure.

This procedure uses a patience counter. Training stops when the number of updates equals this counter. The patience counter is doubled when validation performance improves sufficiently, using a very small threshold of less than 0.1% relative improvement.

The learning rate is not constant during training. When validation performance drops sufficiently, the previous best model parameters are restored, and the learning rate is halved [76].

3.4.4 CRF Baseline and State of the art

The CRF baseline is a conventional CRF using standard features. Those features are not engineered to target a specific task. The feature engineering for this baseline is simple, and involves a level of effort similar to the work required for the NeuroCRFs.

CoNLL-2000 and CoNLL-2003 are well known corpora and extensive feature engineering was applied to those two corpora over the years. The performance of our NeuroCRFs are also compared to the performance of the state of the art system for those corpora. The CRF baseline rely on no external data, in particular gazetteers, but this restriction is not applied to the state of the art results.

The feature set of the CRF baseline is derived from the features available for the NeuroCRF:

- Word identities, capitalization feature and part-of-speech tag extracted from a sliding window,
- Bigrams and tri-grams of the above,
- Word representation for the central word.

Those features are extracted by $C_t(\mathbf{x})$, the feature count matrix of Equation 2.11. The output feature selection matrix $F(y_t, y_{t-1})$ completes the feature set by linking input feature to outputs. For this CRF baseline, the output feature are labels emissions and label-to-label transitions, like the one used in the example in Section 2.2.3.

The state of the art for the chunking task is [7]. For the NER task (CoNLL-2003), the state of the art is [9].

3.4.5 Results

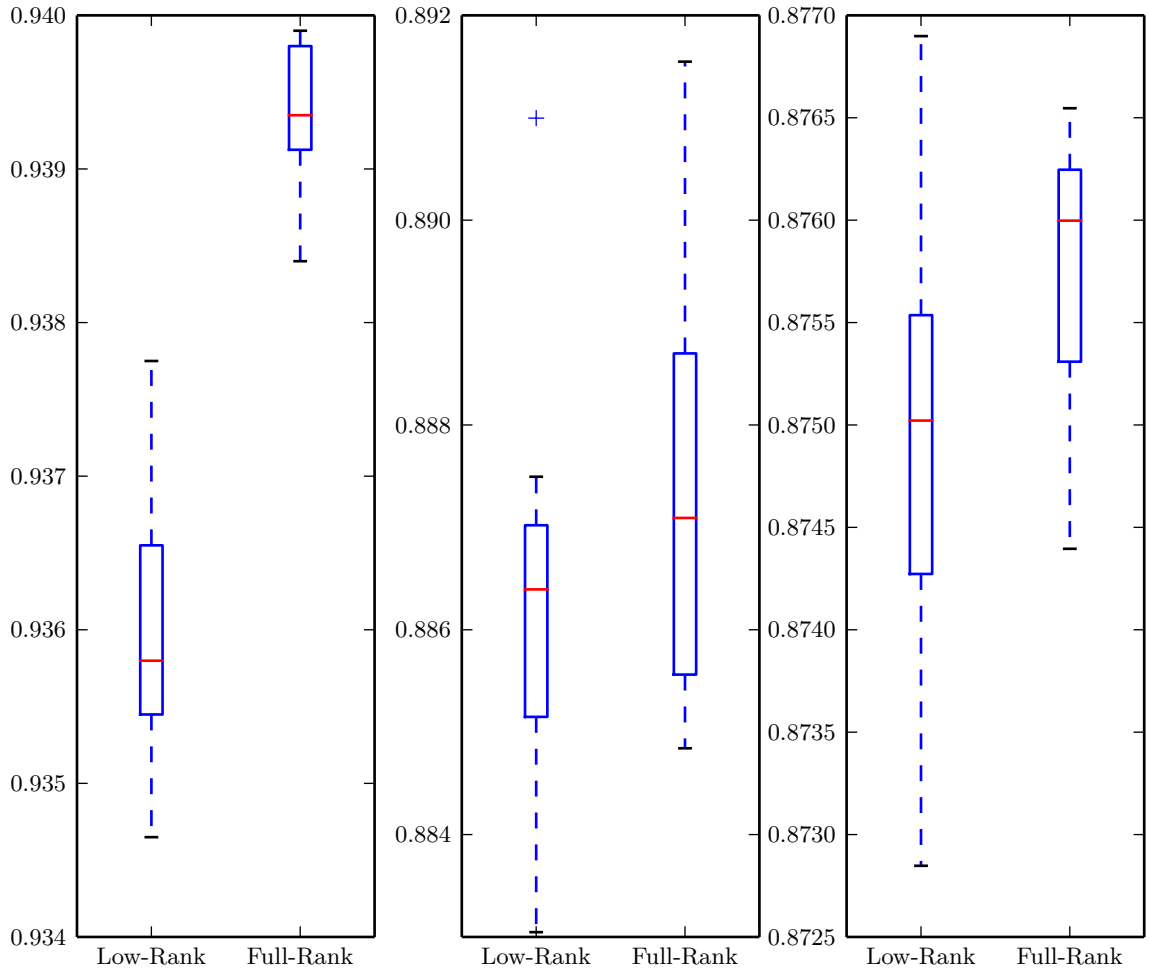
Table 3.1 shows the averages and standard deviations of 10 models trained with the same hyper-parameters and different random initializations, for the three tasks. Those results show that full-rank NeuroCRFs outperform low-rank NeuroCRFs on all three tasks.

The boxes in Figure 3.2 shows the two central quartiles of those 10 models' F_1. Each box correspond to a group of 10 models, used for the indicated task. The vertical lines above and below the boxes indicate the range of the first and last quartile. The central horizontal lines, in the boxes, indicate the median. Figure 3.2a shows a clear separation between low and full-rank NeuroCRFs. Full-rank NeuroCRF clearly improved performance when applied to the chunking task. Figure 3.2b shows large variance caused by random initializations when models are trained and tested on the NER task. While, in general, full-rank models outperformed low-rank model, this large variance prevents us from reaching any firm conclusion for the NER (CoNLL-2003) task. Figure 3.2c shows a much smaller variance, caused by the larger training and test size of the WikiNER task. While the improvements are limited, this figure shows that the worse full-rank model outperforms the worst quartile of low-rank models. In general, full-rank models tend to outperform low-rank models. For the three tasks, those results show that properly regularized full-rank NeuroCRFs will usually perform at least as well as low-rank NeuroCRFs, and will often outperform them.

Table 3.1 shows that NeuroCRFs outperform the baseline CRFs. They do not outperform the state of the art results on those tasks, lacking the high level of feature engineering required to get this level of performance.

	Low-Rank		Full-Rank		CRF	State of the art
Task	μ	σ	μ	σ		
Chunking	93.59	0.0964	**93.94**	0.0507	93.28	94.38[7]
NER	88.63	0.2145	**88.75**	0.2305	87.82	90.90[9]
WikiNER	87.49	0.1107	**87.58**	0.0739	85.60	NA

Table 3.1 Comparison of low and full-rank NeuroCRF for the Chunking (CoNLL-2000), NER (CoNLL-2003), and WikiNER (Wikipedia) task. Results are the average μ and standard deviation σ of 10 models.



(a) Chunking (b) NER (c) WikiNER

Fig. 3.2 Boxplot comparing the performance of low and full-rank NeuroCRFs for the Chunking, NER and WikiNER task.

Chunking (CoNLL-2000)

Table 3.2 shows detailed results for the chunking task. Those results show improvement for all five measures. A two-tailed Student's T-tests shows that the change in F_1 is statistically significant ($p \leq 0.01\%$). In particular, both the classification accuracy and segmental F_1 improved for the full-rank NeuroCRF. The improved segmental F_1 indicates that modelling the transitions directly helps detecting the boundaries between segments. This is especially important for the chunking task, since the segments are usually adjacent to each other, and any error would tend to cascade into the adjoining segments. Similarly, the improved classification accuracy indicates that the model capacity to assign segments to class, by labelling words, was improved by modeling the label-to-label transitions. The classification accuracy A_c is high for both systems, but their $F_1^{(s)}$ is significantly lower. In the case of full-rank NeuroCRF, $A_c = 98.74\%$, while $F_1^{(s)} = 95.14$. Small improvement to $F_1^{(s)}$ would improve performance more than comparable small improvement to A_c. This indicates that the errors are caused by segmentation, and not by classification.

Figure 3.3 plot the precision vs recall of each model applied to the chunking task. This facilitate the comparison between the full and low-rank NeuroCRFs. The two configurations are clearly separated, with the full-rank NeuroCRFs occupying the top-right corner, which correspond to higher F_1. Figure 3.3 confirms that full-rank NeuroCRFs outperform low-rank NeuroCRFs for this task.

	Low-Rank		Full-Rank	
Measure	μ	σ	μ	σ
F_1	93.59	0.0964	**93.94**	0.0507
Precision	93.53%	0.1052	**93.94%**	0.0821
Recall	93.65%	0.1112	**93.93%**	0.0529
Class. Accuracy	98.63%	0.0353	**98.74%**	0.0284
$F_1^{(s)}$	94.89	0.0844	**95.14**	0.0534

Table 3.2 Detailed results comparing low and full-rank NeuroCRF for the Chunking task, showing the F_1, precision, recall, classification accuracy and segmental F_1. Results are the average μ and standard deviation σ of 10 models.

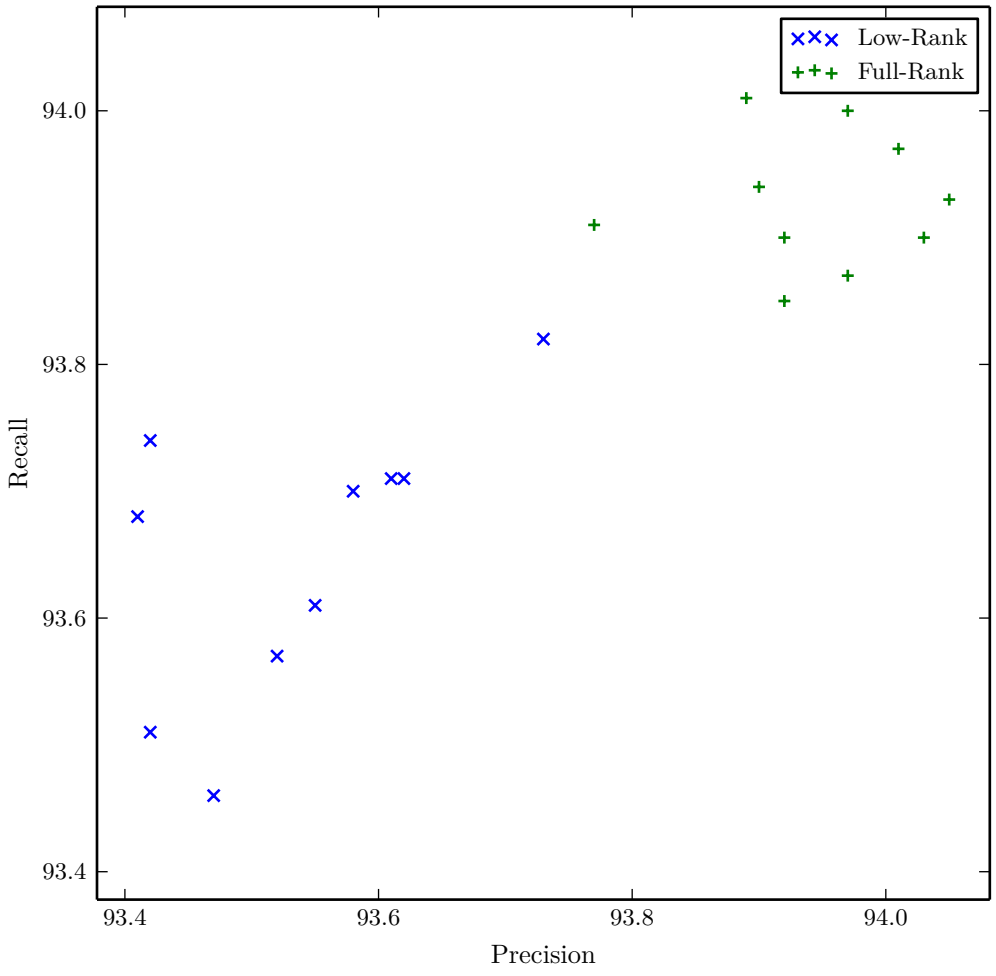


Fig. 3.3 Precision-Recall plot comparing the performance of low and full-rank NeuroCRF on the chunking task.

Named Entity Recognition (CoNLL-2003)

Table 3.3 shows detailed results on the NER task. Those results show improvements on most measures but they also show a large standard deviation. While the use of full-rank NeuroCRFs did improve performance, the improvement is limited, and it is not possible to separate this improvement from the variation caused by the random initialization combined with the non-convexity of NeuroCRFs. A two-tailed Student's T-tests shows that the change in F_1 is not statistically significant ($p \geq 20\%$). Nonetheless, Table 3.3 shows improvement for F_1, precision, recall and classification accuracy. Segmental F_1 was slightly reduced.

Named entities tend to be sparse in a sentences, with very few entities immediately following each others. This task is less sensitive to segmentation than the chunking task, since segmentation errors are unlikely to cascade. Table 3.3 shows that recall is lower than precision, indicating the presence of false negatives, for low and full-rank NeuroCRFs. This is confirmed by the drop in $F_1^{(s)}$ compared to Table 3.2. This drop indicates that the models are less able to segment their input accurately. Part of this can be attributed to the nature of those two tasks, and to the size of the training sets. While the sparseness prevent cascading errors, the sparseness of segment biases the NN toward missing segments. This (correct) bias can cause false negative when combined with atypical named entities, named entities appearing surrounded by atypical carrier phrases, or named entities appearing without carrier phrases.

This sparseness also has other undesired effects. First, it reduces the effective training size. While CoNLL-2003 contains more word and sentences than CoNLL-2000, it contains significantly less segments. This reduction in training size limit the capacity of the model to learn good classification, which explain, in part, the lower classification accuracy observed between Tables 3.2 and 3.3. Similarly, the smaller effective training set will limit the number of named entities and carrier phrases seen during training. NeuroCRFs are trained using example, and reducing the number of examples seen will reduce performance, and increases the sensitivity to initialization.

The second effect of the sparseness of named entities apply to the test set. The test set is smaller, containing only a few thousands segments. The performance of a model on smaller test sets will tend to be more variable, as a single mistake has more impact. This combines with the smaller effective training set to explain the larger standard deviation observed between Tables 3.2 and 3.3.

While the smaller effective training set explain in part the lower classification accuracy observed between Tables 3.2 and 3.3, it is not the only factor. The distributions of classes are not identical for those two tasks. In particular, the entropy of the class distribution, computed for the test set, is 0.37 bits for Chunking, while it is 1.93 bits for NER (CoNLL-2003). This results indicates that classification is harder for NER, where the entropy is close to the maximum, and combine with the smaller effective training set to explain the lower classification accuracy.

Figure 3.4 shows the precision vs recall plot for this task. As expected, it is not possible to clearly separate the two configurations. Full-rank models tend to have a higher precision, being more toward the left of the graph, but there is significant overlap. The same is true for recall, with the full-rank models being more toward the top of the graph. While this figure shows improved performance using a full-rank NeuroCRFs, the large spread and significant overlap prevents firm conclusions.

	Low-Rank		Full-Rank	
Measure	μ	σ	μ	σ
F_1	88.63	0.2145	**88.75**	0.2305
Precision	88.78%	0.2688	**88.93%**	0.2588
Recall	88.48%	0.2322	**88.57%**	0.3145
Class. Accuracy	94.51%	0.2012	**94.66%**	0.2095
$F_1^{(s)}$	**93.78**	0.1422	93.75	0.1648

Table 3.3 Detailed results comparing low and full-rank NeuroCRF for the NER (CoNLL-2003) task, showing the F_1, precision, recall, classification accuracy and segmental F_1. Results are the average μ and standard deviation σ of 10 models.

WikiNER

This task is similar to CoNLL-2003, and was included to compensate for the issues caused by the small size of CoNLL-2003. Table 3.4 shows the detailed results for this task. Those results show improvements on most measure, the exception being segmental F_1. As expected, the standard deviations are lower compared to Table 3.3. A two-tailed Student's T-tests shows that the change in F_1 is not statistically significant ($p \geq 6.00\%$).

Figure 3.5 shows the precision vs recall plot for this task. While there is some overlap, the full-rank NeuroCRFs are clearly clustered toward the upper-right corner, corresponding to

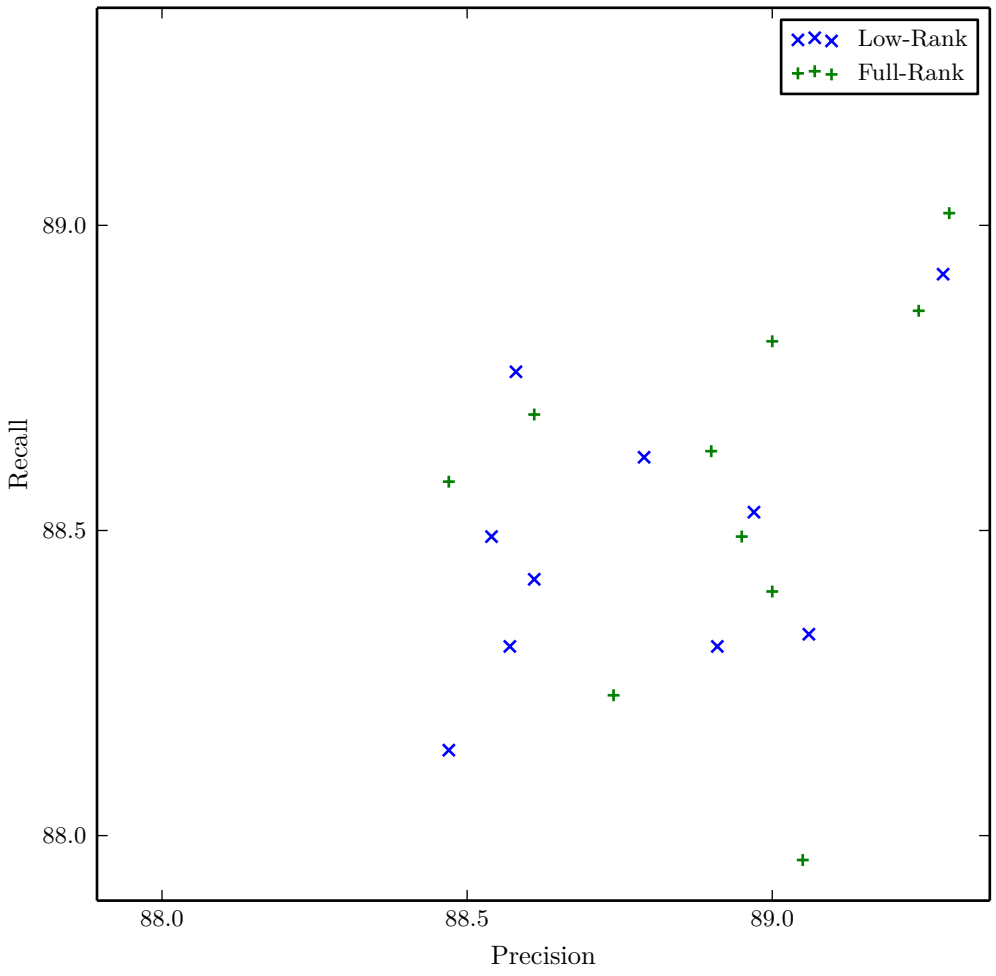


Fig. 3.4 Precision-Recall plot comparing the performance of low and full-rank NeuroCRF on the NER task.

their higher F_1. This is confirmed by the boxplot in Figure 3.2c, where the central quartiles of full-rank NeuroCRFs are above the median of low-rank NeuroCRFs. Both precision and recall tend to be higher using a full-rank NeuroCRF than when using a low-rank.

The low classification accuracy observed in Table 3.3 is observed in Table 3.4. This confirm the impact of the class entropy, which is 1.96 bits for the WikiNER task. Similarly, the sparseness of segments, biases the models toward no-segments, results in a lower recall, compared to precision, and a relatively low segmental F_1.

Measure	Low Rank		Full-Rank	
	μ	σ	μ	σ
F_1	87.49	0.1107	**87.58**	0.0739
Precision	87.65%	0.1092	**87.75%**	0.0738
Recall	87.34%	0.1233	**87.41%**	0.0797
Class. Accuracy	93.46%	0.0718	**93.61%**	0.0722
$F_1^{(s)}$	**93.62**	0.0658	93.56	0.0532

Table 3.4 Detailed results comparing low and full-rank NeuroCRF for the WikiNER task, showing the F_1, precision, recall, classification accuracy and segmental F_1. Results are the average μ and standard deviation σ of 10 models.

3.4.6 Impact of Mutual Information

The mutual information between two variables is a measure of how much is known about one if the other is known. If the two variables are independent, then the mutual information is zero. The mutual information between the y_t, the label emitted at time t, and the previous label y_{t-1} is

$$I(y_t; y_{t-1}) = \sum_{y \in \mathcal{Y}} \sum_{y' \in \hat{\mathcal{Y}}} P(y, y') \left[\log_2 P(y, y') - \log_2 P(y) - \log_2 P(y') \right], \tag{3.64}$$

where $\mathcal{Y}$ is the set of labels, and $\hat{\mathcal{Y}}$ is $\mathcal{Y}$ plus the initial state.

Table 3.5 shows the mutual information between the previous label y_{t-1} and the current label y_t. The motivation behind full-rank NeuroCRF is to use the NN to model the dependency between those two variables. The mutual information is significantly larger in the chunking task than in the two NER tasks. Since higher mutual information between two variables indicate a higher level of dependency, the low mutual information in the NER tasks

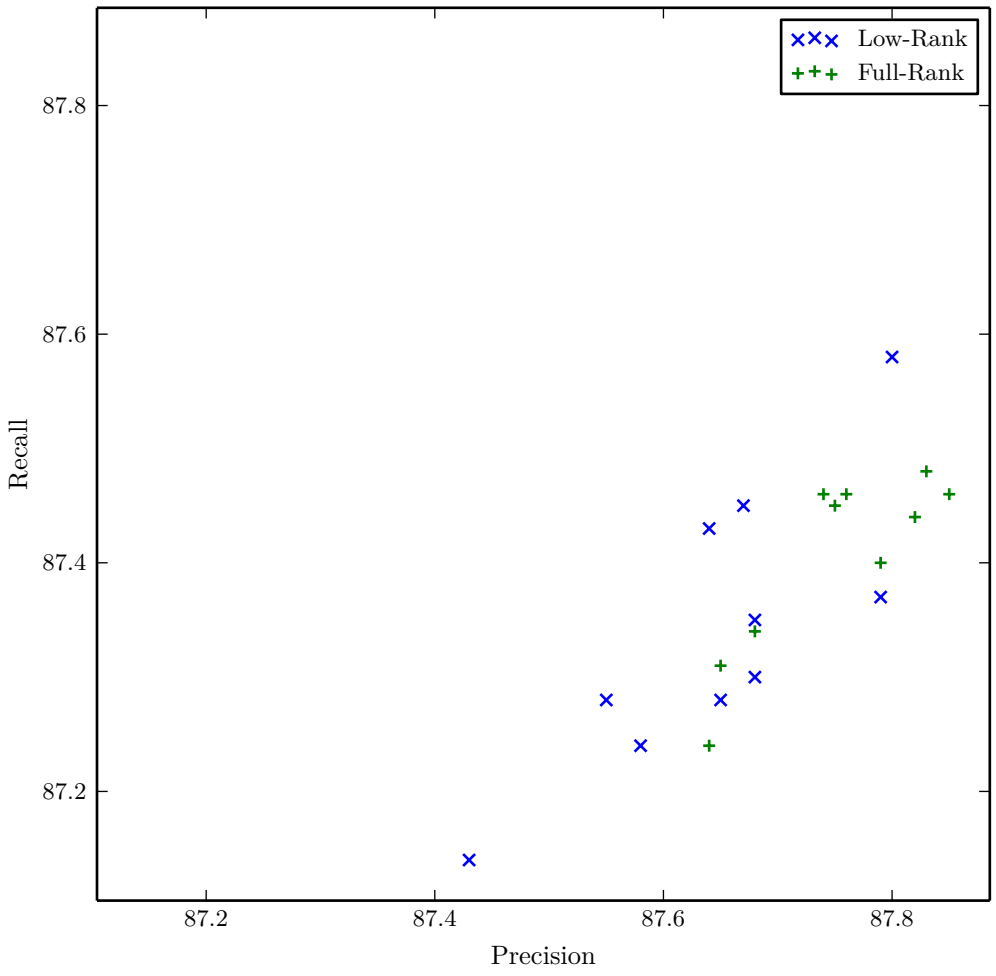


Fig. 3.5 Precision-Recall plot comparing the performance of low and full-rank NeuroCRF on the WikiNER task.

explains the lower gain when replacing a low-rank NeuroCRF with a full-rank NeuroCRF.

The lower mutual information, in the NER tasks, is caused by the predominance of the "O" label. Named entities tend to be sparse, and most words are therefore labelled with "O". Since named entities are also short and isolated, the preceding label also tend to be "O". The predominance of "O" labels in the data combines with the lower number of possible classes, compare to chunking, to explain the lower label entropies in the NER task.

Task	$H(y_t)$	$H(y_{t-1})$	$H(y_t, y_{t-1})$	$I(y_t; y_{t-1})$
Chunking	2.6538	2.7909	4.6157	0.8290
NER	1.1166	1.3896	2.2484	0.2578
WikiNER	1.0042	1.2030	1.9473	0.2599

Table 3.5 Comparison of mutual information in the training corpora of the chunking and NER tasks. $H(y_t)$ is the entropy of the emitted labels, $H(y_{t-1})$ is the entropy of the preceding labels, including the initial state, $H(y_t, y_{t-1})$ is the joint entropy and $I(y_t; y_{t-1})$ is the mutual information.

3.5 Summary

In this chapter, we described the training procedure for NeuroCRF. We presented a new member of this family of model, full-rank NeuroCRF, whose NN models transitions rather than the emissions modelled by low-rank NeuroCRF.

Our experimental study show significant improvements on a chunking task, CoNLL-2000. Improvements were also obtained on CoNLL-2003, a NER task, but the high variance caused by the random initialization prevent us from attributing those improvements to the use of a full-rank NeuroCRF. Similar experiments on WikiNER, a much larger NER task, also show improvement. The lower sensitivity of WikiNER to initialization confirms that the improvements are not caused by better initializations.

Chapter 4

Three Improvements to NeuroCRF

This chapter presents some improvement to the full-rank NeuroCRFs presented in Chapter 3.[1] The first improvement modifies the output features to share NN outputs, and therefore parameters, between related label emission and label-to-label transitions. The second is intended to increase the margin between the correct hypothesis and the incorrect hypothesis. Finally, the third improvement exploits the impact of a model's initialization by combining separately trained model into a single ensemble model. The impact of those improvement is investigated in an experimental study concluding this chapter.

4.1 Shared Parameters

As seen in Section 3.1.3, the full and low-rank NeuroCRFs are part of a larger family of models. Those two classes of NeuroCRF are characterized by constraints applied to their weights and biases, as well as by the form of their feature selection matrix $F(y_t, y_{t-1})$. Another class of NeuroCRF can be obtained by removing the constraints without altering $F(y_t, y_{t-1})$. This new class assigns a NN output to every label emission, as well as to every label-to-label transition, combining full and low-rank NeuroCRF into a single model. This increases the number of NN outputs required, adding more parameters to the model, but those outputs, and the corresponding parameters, are used by many transitions.

This is the basic form of a more general class of model. The core concept is to use $F(y_t, y_{t-1})$ to combine multiple NN outputs [12]. Label-to-label transitions are assigned to

[1] The content of this chapter is based on work published in "Recent improvements to NeuroCRFs for named entity recognition" [12]

groups of similar transitions. $F(y_t, y_{t-1})$ will be used to combine the NN outputs corresponding to the groups of (y_t, y_{t-1}). This approach requires feature engineering, in order to define which transitions are similar.

Similarly, this general approach, where the NN outputs no longer have a one-to-one correspondence with $F(y_t, y_{t-1})$ could be used to tie transitions together, reducing the number of parameters. Low-rank NeuroCRFs can be derived from full-rank NeuroCRFs by tying together all transitions having the same end label, although this does involve some other modifications to include the transition matrix A.

4.1.1 Generalized Events

By itself, the NN does not operate on the entire input sequence, nor does it attempts to predict the entire output. Instead, the NN outputs correspond to the immediate potential of specific events. As in Equation 3.24 and Equation 3.25, some of the weights and biases can be constrained to 0, so that some events are not a function of the input, and other events are not biased.

For full-rank NeuroCRFs, those events are label-to-label transitions. Low-rank NeuroCRFs includes two kind of events: label emissions and label-to-label transitions. The label emission potentials are functions of the input, while the label-to-label transition potentials are constant.

The shared parameter approach presented in this section requires feature engineering to create a set of generalized events. There will be one NN output per event; in this particular case, all events are a function of the input, and there are no zero constraints on the weights and biases matrices. Any given label-to-label transitions (y_t, y_{t-1}) must be associated with 1 or more events. The feature selection matrix $F(y_t, y_{t-1})$ is created using this mapping.

Our experimental study used a systematic approach to convert segment class, with the corresponding labels, into a set of generalized events. The procedure used is presented in the following subsection.

4.1.2 Transition Grouping Procedure

Given the improved performance obtained with full-rank NeuroCRFs, we decided to focus on an additive approach, where generalized events are added to the label-to-label transitions used in full-rank NeuroCRFs [12]. Label emissions were also added to the set of events,

resulting, before any grouping, in a hybrid low and full-rank NeuroCRF, where the NN is used to generate potentials for both emissions and transitions.

In this subsection, the feature engineering used to create generalized emission events based on label emission and label-to-label transitions is described. Other sets of generalized events are possible, and should be task-specific. The procedure described in this subsection is a decent starting point, and is applicable to all tasks. In general, the feature engineering required can be simplified by grouping label emissions, and extending this grouping to label-to-label transitions. The first step is to create groups of labels, corresponding to generalized emission events. This step requires feature engineering, but the following steps, where those groups are used to create generalized transition events, do not.

Our label grouping procedure creates $N_c + 4$ groups, where N_c is the number of classes. Group $\mathcal{E}_c$ contains all the labels used for segments belonging to class c. Group $\mathcal{E}_B$ contains all the labels used to indicate the beginning of a segment. Group $\mathcal{E}_I$ contains all the labels used for the rest of the segment. Group $\mathcal{E}_{SEG} = \mathcal{E}_B \cup \mathcal{E}_I$ contains the labels indicating that a word is part of a segment. Finally, group $\mathcal{E}_{NON}$ contains the label used to indicate than a word is not in a segment, as well as the initial state. Every group corresponds to a generalized emission event. Table 4.1 shows the results of this grouping for the NER tasks. This grouping assumes that all segment classes are equally related to each other, which is why they are all placed in the $\mathcal{E}_{SEG}$ group. Different relations between classes would require different groups.

The second step, which does not require feature engineering, is used to create the generalized transitions events from the generalized emission events. With a set of labels $\mathcal{L}$, the set of transitions is $\mathcal{T} = \mathcal{L} \times \mathcal{L}$, the cartesian product of $\mathcal{L}$. The generalized transition events are created by replacing $\mathcal{L}$ by

$$\mathcal{E} = \{\mathcal{E}_B, \mathcal{E}_I, \mathcal{E}_{SEG}, \mathcal{E}_{NON}, \mathcal{E}_1, \ldots, \mathcal{E}_{N_c}\} \cup \mathcal{L}, \tag{4.1}$$

where $\mathcal{E}$ is the union of the generalized emission events and label emission events. The set of generalized events $\mathcal{G} = (\mathcal{E} \times \mathcal{E}) \cup \mathcal{E}$ is a superset of $\mathcal{T}$ and $\mathcal{L}$, as generalized events are added to the existing emission and transition events.

Group	Labels
$\mathcal{E}_{LOC}$	B-LOC, I-LOC
$\mathcal{E}_{MISC}$	B-MISC, I-MISC
$\mathcal{E}_{ORG}$	B-ORG, I-ORG
$\mathcal{E}_{PER}$	B-PER, I-PER
$\mathcal{E}_{SEQ}$	B-LOC, B-MISC, B-ORG, B-PER, I-LOC, I-MISC, I-ORG, I-PER
$\mathcal{E}_{B}$	B-LOC, B-MISC, B-ORG, B-PER
$\mathcal{E}_{I}$	I-LOC, I-MISC, I-ORG, I-PER
$\mathcal{E}_{NON}$	O, init

Table 4.1 Grouping of label used to create generalized events for the NER tasks CoNLL-2003 and WikiNER

4.1.3 Feature Selection Matrix

The NN has one output per element in the set of generalized events $\mathcal{G}$. Using some arbitrary ordering, $G_{L+1,t,i}(\mathbf{x})$ is the output assigned to $\mathcal{G}_i$. The feature selection matrix $F(y_t, y_{t-1})$ is modified to use those generalized event outputs.

$$F(y_t, y_{t-1}) = \frac{1}{C(y_t, y_{t-1})} \left[\begin{array}{cccc} \text{in}((y_t, y_{t-1}), \mathcal{G}_1) & \text{in}((y_t, y_{t-1}), \mathcal{G}_2) & \cdots & \text{in}((y_t, y_{t-1}), \mathcal{G}_{|\mathcal{G}|}) \end{array} \right]^\top, \tag{4.2}$$

$$C(y_t, y_{t-1}) = \sum_{\mathcal{G}_i \in \mathcal{G}} \text{in}((y_t, y_{t-1}), \mathcal{G}_i), \tag{4.3}$$

Where $\text{in}((y_t, y_{t-1}), \mathcal{G}_i)$ is a binary indicator function that is equal to 1 when (y_t, y_{t-1}) is inside the generalized event $\mathcal{G}_i$. This modified feature selection matrix will average the $C(y_t, y_{t-1})$ NN outputs used by a transition from label y_{t-1} to label y_t. The indicator function $\text{in}((y_t, y_{t-1}), \mathcal{G}_i)$ is based on a reverse lookup, where every transitions has an associated set of generalized events $B(y_t, y_{t-1})$. Table 4.2 shows an example, for the transition $(\text{B} - \text{LOC}, \text{O})$.

4.2 Large Margin Training

Parameters are estimated by minimizing a loss function, which increases the log-likelihood of the correct output given the corresponding input. It is possible for this log-likelihood to be very close to the log-likelihood of the best competing hypothesis. Ideally, the parameters

$B(y_t, y_{t-1}) = B(\text{B} - \text{LOC}, \text{O})$				
Emission Events:				
$\mathcal{E}_B$	$\mathcal{E}_I$	$\mathcal{E}_{SEQ}$	$\mathcal{E}_{LOC}$	B-LOC
Transitions Events:				
$(\text{O}, \mathcal{E}_B)$	$(\text{O}, \mathcal{E}_I)$	$(\text{O}, \mathcal{E}_{SEQ})$	$(\text{O}, \mathcal{E}_{LOC})$	$(\text{O}, B - LOC)$
$(\mathcal{E}_{NON}, \mathcal{E}_B)$	$(\mathcal{E}_{NON}, \mathcal{E}_I)$	$(\mathcal{E}_{NON}, \mathcal{E}_{SEQ})$	$(\mathcal{E}_{NON}, \mathcal{E}_{LOC})$	$(\mathcal{E}_{NON}, B - LOC)$

Table 4.2 List of generalized events corresponding to the transition $(\text{B} - \text{LOC}, \text{O})$ for the NER tasks CoNLL-2003 and WikiNER

should be selected to maximize both the log-likelihood and the margin between the correct and best competing hypothesis. Large margin training has been applied to HMMs [77], and the same approach has been used with CRFs [1]. In this particular case, the loss function

$$L(\mathbf{y}, \mathbf{x}) = -\log p(\mathbf{y}|\mathbf{x}) = \log Z(\mathbf{x}) - \sum_{t=1}^{T} G_{L+1,t}(\mathbf{x}) F(y_t, y_{t-1})$$

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \exp\left(\sum_{t=1}^{T} G_{L+1,t}(\mathbf{x}) F(y_t, y_{t-1})\right),$$

is replaced by

$$L(\mathbf{y}, \mathbf{x}) = Z'(\mathbf{x}, \mathbf{y}) - \sum_{t=1}^{T} G_{L+1,t}(\mathbf{x}) F(y_t, y_{t-1}), \tag{4.4}$$

$$Z'(\mathbf{x}, \mathbf{y}) = \max_{\mathbf{y}'} \sum_{t=1}^{T} \left(G_{L+1,t}(\mathbf{x}) F(y'_t, y'_{t-1}) + \text{cost}(y_t, y'_t) \right), \tag{4.5}$$

$$\text{cost}(y, y') = \begin{cases} 0, & y = y' \\ 1, & y \neq y' \end{cases}, \tag{4.6}$$

where $\text{cost}(y_t, y'_t)$ is the contribution of y'_t to the margin and $Z'(\mathbf{x}, \mathbf{y})$ is the potential, to be minimized, of the best hypothesis competing with the correct decoding $\mathbf{y}$. This loss function will try to enforce a margin proportional to the difference between the correct hypothesis and the best competing hypothesis. The resulting loss function is not differentiable, but this optimization problem can be solved as explained in [77].

This loss function can be modified to use an upper bound of $Z'(\mathbf{x}, \mathbf{y})$ [78, 1]. Since,

$\log \sum_i \exp x_i \geq \max x_i$,

$$\log Z(\mathbf{x}, \mathbf{y}) = \log \sum_{\mathbf{y}'} \exp \left[\sum_{t=1}^{T} G_{L+1,t}(\mathbf{x}) F(y'_t, y'_{t-1}) + \text{cost}(y_t, y'_t) \right], \tag{4.7}$$

is an upper bound of $Z'(\mathbf{x}, \mathbf{y})$. The loss function becomes

$$L(\mathbf{y}, \mathbf{x}) = \log Z(\mathbf{x}, \mathbf{y}) - \sum_{t=1}^{T} G_{L+1,t}(\mathbf{x}) F(y_t, y_{t-1}), \tag{4.8}$$

which is differentiable. This is simply the loss function used to train NeuroCRFs, with an added cost term $\text{cost}(y_t, y'_t)$. This was applied to low-rank NeuroCRF in [59].

The cost term in Equation 4.7 increases the contribution of a possible $\mathbf{y}'$ to $Z(\mathbf{x}, \mathbf{y})$, which is minimized. This will push the NN outputs $G_{L+1,t,i}$ used by incorrect hypotheses towards $-\infty$. We obtained a similar effect by replacing $\text{cost}(y_t, y'_t)$ by

$$\text{cost}(y, y') = \begin{cases} -1, & y = y' \\ 0, & y \neq y' \end{cases}, \tag{4.9}$$

which decreases the contribution of a hypothesis $\mathbf{y}'$ to $Z(\mathbf{x}, \mathbf{y})$. Both cost terms are implemented by adding a constant term to the NN output matrix. The main advantage of Equation 4.9 over Equation 4.6 is that it is significantly more sparse.

The recursive formulation of $\nabla Z(\mathbf{x})$ from Section 3.3.1 becomes

$$\nabla Z(\mathbf{y}, \mathbf{x}) = \sum_{y'_T} \nabla \alpha_{T, y'_T} \tag{4.10}$$

$$\nabla \alpha_{1, y'_1} = \exp \text{cost}(y_1, y'_1) \nabla \Psi(y'_1, \text{INIT}, \mathbf{x}) \tag{4.11}$$

$$\nabla \alpha_{t, y'_t} = \sum_{y'_{t-1}} \exp \text{cost}(y_t, y'_t) \left(\nabla \alpha_{t-1, y'_{t-1}} \Psi(y'_t, y'_{t-1}, \mathbf{x}) + \alpha_{t-1, y'_{t-1}} \nabla \Psi(y'_t, y'_{t-1}, \mathbf{x}) \right), \tag{4.12}$$

where y_t is the correct label at time t and y'_t is a possible label. This shows that the contribution of $\nabla \Psi(y'_t, y'_{t-1}, \mathbf{x})$ is reduced when $y_t = y'_t$.

4.3 Ensemble Models

Ensemble learning uses the complementarity between models to improve performance. Models are combined into a single larger ensemble model. The errors of reasonably good models should be distributed randomly. Since the models will not make the same error at the same point, the ensemble model will be able to correct for the erroneous models. The ensemble model can be based on a voting system, such as ROVER [79], or on some other combination of systems, such as a product of experts.

The experimental studies used in this thesis report the results of multiple training runs, each using a different random initialization. This is done to compensate for the non-convexity of NNs. The different initializations will converge to different local minima, and will result in different performance. This provides us with complementary models. The ensemble model is a product of expert of those models.

Similar approaches, where models of the same class are ensemble, have also been developed for class of models where there is a known optimal model for a given training corpus. Those approaches will obtain complementary models by modifying the training data. Bagging [80] splits the training data into subset, used to train individual models. Adaboost [81] trains a series of models, and adapts the weight of training examples as a function of their error with the current model. We did not investigate these approaches due to the large training time required by NeuroCRF.

The ensemble model is a product of experts,

$$\hat{G}_{L+1,t}(\mathbf{x}) = \frac{1}{M}\sum_{i=1}^{M} G^{(i)}_{L+1,t}(\mathbf{x}) \tag{4.13}$$

$$L(\mathbf{y},\mathbf{x}) = -\log p(\mathbf{y}|\mathbf{x}) = \log Z(\mathbf{x}) - \sum_{t=1}^{T} \hat{G}_{L+1,t}(\mathbf{x}) F(y_t, y_{t-1}) \tag{4.14}$$

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \exp\left(\sum_{t=1}^{T} \hat{G}_{L+1,t}(\mathbf{x}) F(y_t, y_{t-1})\right), \tag{4.15}$$

where M models are combined, and $G^{(i)}_{L+1,t}(\mathbf{x})$ is the output of model i. As the NN outputs are used inside an exponential, the summation is equivalent to a product. The division by M is not strictly needed, but it helps prevent overflow.

This approach is equivalent to a single, larger network, where units are connected by block, so that

$$\hat{W}_l = \begin{bmatrix} W_l^{(1)} & 0 & \dots & 0 \\ 0 & W_l^{(2)} & \dots & 0 \\ & & \dots & \\ 0 & 0 & \dots & W_l^{(M)} \end{bmatrix}, \tag{4.16}$$

$$\hat{B}_l = \left[B_l^{(1)}, B_l^{(2)}, \cdots, B_l^{(M)}\right], \tag{4.17}$$

$$\hat{W}_{L+1} = \frac{1}{M}\left[W_{L+1}^{(1)}, W_{L+1}^{(2)}, \cdots, W_{L+1}^{(M)}\right] \tag{4.18}$$

$$\hat{B}_{L+1} = \frac{1}{M}\sum_{i=1}^{M} B_{L+1}^{(i)}. \tag{4.19}$$

4.4 Experimental Study

The performance of the improvement presented in this chapter were compared to full and low-rank NeuroCRF baselines. Those baseline results were the subject of Chapter 3, and were compared to CRF baselines in Section 3.4. The CRF baseline was not included in this chapter in order to reduce the number of results to compare.

4.4.1 Model Configuration and Training Procedure

The improvements presented in this chapter do not affect the model configuration and training procedure. As such, the model configuration procedure described in Section 3.4.2, and the training procedure is described in Section 3.4.3 are used.

4.4.2 Datasets and Performance Metrics

The data used in those experiments is presented in Section 2.7. The performance metrics used are presented in Section 2.6.

4.4.3 Results

Table 4.3 shows the averages and standard deviations of 10 models trained with the same parameters and different random initializations, for the three tasks. The results include low and full-rank NeuroCRF baselines. Large margin training is compared to those baselines. With the exception of low-rank NeuroCRF used for NER, large margin training improved performance. Shared parameters where added to full-rank NeuroCRFs, resulting in improvements for the three tasks. The combination of large margin training and shared parameters, once more applied to full-rank NeuroCRFs, further improved performance for the Chunking and WikiNER. Performance on NER (CoNLL-2003), while improved over the baseline, was lower than the performance observed using either large margin training or shared parameters.

Table 4.4 shows the results when those groups of 10 models are combined into a single ensemble models.

The boxes in Figure 4.1 shows the two central quartiles of those 10 models' F_1. Each box correspond to a group of 10 models, used for the indicated task. The vertical lines above and below the boxes indicate the range of the first and last quartile. The central horizontal lines, in the boxes, indicate the median. Finally, the line going from box to box indicates the performance of ensemble models. Figure 4.1a clearly shows a series of successive improvement on the chunking task. Figure 4.1b confirms that large margin training and shared parameter both improved performance, on the NER task, when used individually, but that their combination was less helpful. Figure 4.1c, like Figure 4.1a, shows a succession of improvement on the WikiNER task. In this particular case, full-rank NeuroCRF are equivalent or slightly worst than large margin low-rank NeuroCRF.

Chunking (CoNLL-2000)

Table 4.5 shows detailed results for the chunking task. The performance metric used are described in Section 2.6 The table is divided into three set of experiments: low-rank, full-rank, full-rank with shared parameters. Those base configurations are compared to the same configuration trained with large margin.

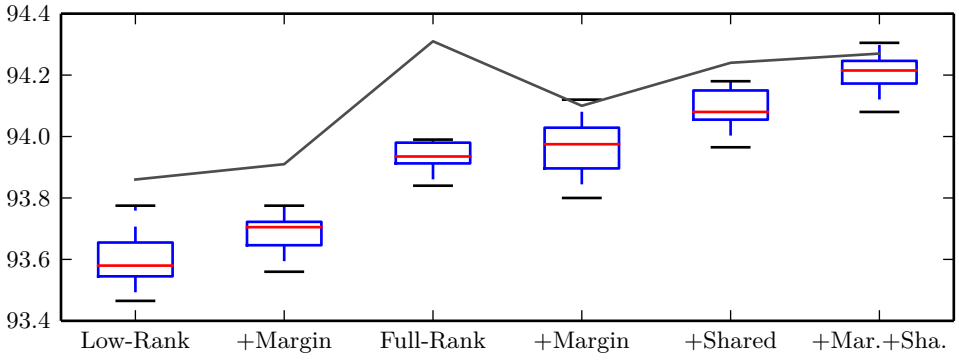
Table 4.5 shows generalized improvements when large margin training is added to low-rank NeuroCRFs. A two-tailed Student's T-tests shows that the change in F_1 is statistically significant ($p \leq 1.74\%$). Those results show that classification accuracy was not significantly affected, but that the segmental F_1 was increased. This indicates that the models trained

Configuration	Chunking		NER		WikiNER	
	μ	σ	μ	σ	μ	σ
Low-Rank	93.58	0.0964	**88.63**	0.2145	87.49	0.1107
+Margin	**93.69**	0.0675	88.49	0.2540	**87.60**	0.1034
Full-Rank	93.94	0.0507	88.75	0.2305	87.58	0.0739
+Margin	93.97	0.0972	89.03	0.1505	87.90	0.1122
+Shared	94.08	0.0760	**89.08**	0.1818	87.95	0.1569
+Margin+Shared	**94.20**	0.0650	88.82	0.1385	**88.10**	0.1082

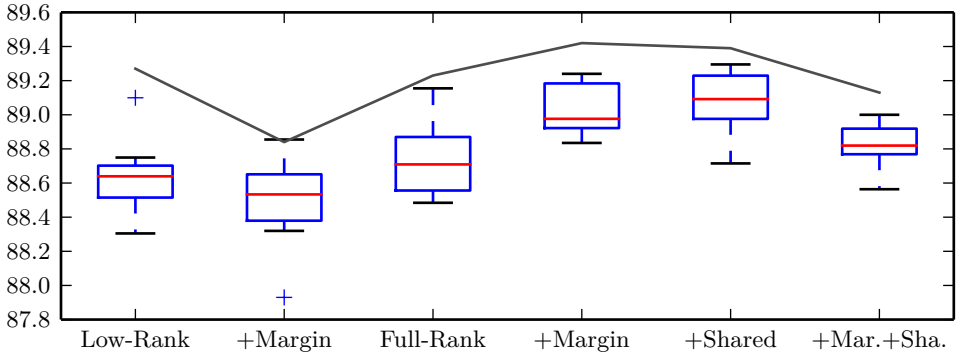
Table 4.3 Comparison of improvements to NeuroCRFs for the Chunking (CoNLL-2000), NER (CoNLL-2003), and WikiNER (Wikipedia) task. Model obtained using large margin training, shared parameter and the combination of both are compared to full and low-rank NeuroCRFs. Results are the average μ and standard deviation σ of 10 models.

Configuration	Chunking		NER		WikiNER	
	μ	ens.	μ	ens.	μ	ens.
Low-Rank	93.58	93.86	88.63	**89.27**	87.49	**88.02**
+Margin	93.69	**93.91**	88.49	88.84	87.60	87.79
Full-Rank	93.94	**94.31**	88.75	89.23	87.58	88.03
+Margin	93.97	94.10	89.03	**89.42**	87.90	88.29
+Shared	94.08	94.24	89.08	89.37	87.95	88.40
+Margin+Shared	94.20	94.27	88.82	89.13	88.10	**88.50**

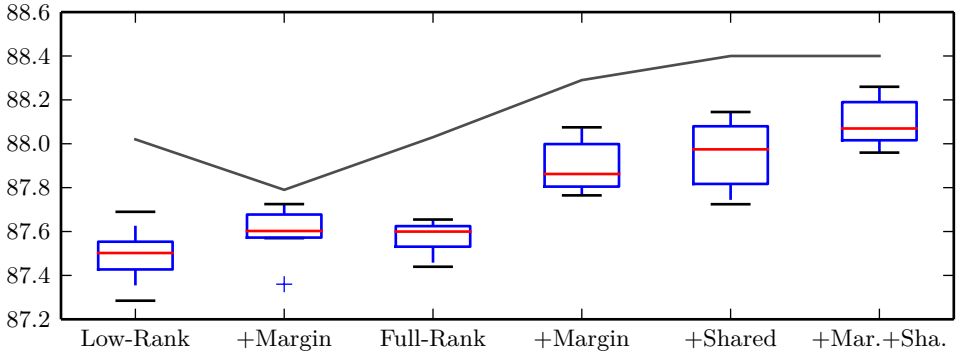
Table 4.4 Comparison of ensemble NeuroCRFs for the Chunking (CoNLL-2000), NER (CoNLL-2003), and WikiNER (Wikipedia) task to the models used to create the ensembles. Compare the average μ of 10 models to an ensemble of those 10 models.



(a) Chunking



(b) NER



(c) WikiNER

Fig. 4.1 Boxplot comparing the performance of improved NeuroCRFs for the Chunking, NER and WikiNER task. Model obtained using large margin training, shared parameter and the combination of both are compared to full and low-rank NeuroCRFs. The line indicates the performance obtained with the corresponding ensemble models.

with large margin are better at identifying the segment boundaries, but are not more capable of identifying classes. Precision and recall were similarly improved.

Table 4.5 also shows some improvements when large margin training is added to full-rank NeuroCRFs. In this case, the effect is much lower, and is not statistically significant ($p \leq 36.00\%$). Segmental F_1 is slightly increased, while classification accuracy is not significantly affected. The effect of shared parameters is statistically significant ($p \leq 0.02\%$). In this case, segmental F_1 is improved, while classification accuracy is not affected.

Finally, Table 4.5 also shows generalized improvements when large margin training is added to full-rank NeuroCRFs with shared parameters ($p \leq 0.2\%$). Once more, classification accuracy is not affected, but segmental F_1 is improved.

Table 4.6 shows improved performance using ensemble models, when compared to Table 4.5. Ensemble learning improved on the average F_1 obtained for all configurations. The ensemble of full-rank NeuroCRF was especially better than the baseline models. Classification accuracy is not significantly improved by ensemble learning. Segmental F_1 is improved. The large improvement obtained using an ensemble of full-rank NeuroCRF, which is the new best configuration, is the results of a combination of better segmentation and better classification. Other configurations have similar performance on either of those measure, but none have the same high performance for both.

In general, those experiments show that large margin training and shared parameters improve the segmentation performance of NeuroCRF, without improving their classification performance. The same is true of ensemble of those models.

Figure 4.2 shows the precision-recall plot of those experiments. It clearly shows the improved performance when large margin training is applied to low-rank NeuroCRFs. Similarly, full-rank models with shared parameters are clearly separated from full-rank models. Full-rank models with large margin training, while showing some improvement, are clearly interleaved with the full-rank baseline. Finally, the combination of shared parameters, large margin training and full-rank NeuroCRFs outperformed the other configuration, and can be separated from full-rank models with shared parameters.

Named Entity Recognition (CoNLL-2003)

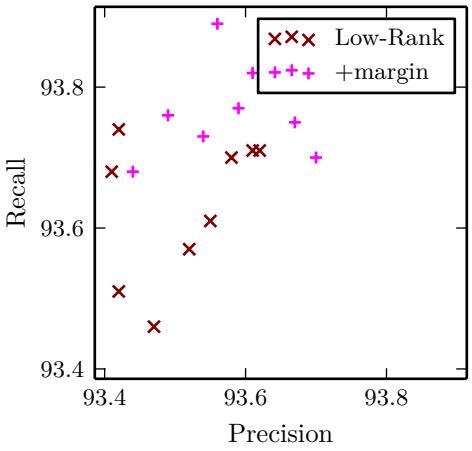
Table 4.7 shows detailed results for the NER (CoNLL-2003) task. The performance metric used are described in Section 2.6 The table is divided into three set of experiments: low-rank,

Measure	Low-Rank μ	Low-Rank σ	+Margin μ	+Margin σ
F_1	93.59	0.0964	*93.69*	0.0675
Precision	93.53%	0.1050	*93.59%*	0.0880
Recall	93.65%	0.1129	*93.79%*	0.0754
Class. Accuracy	*98.63%*	0.0339	98.62%	0.0210
$F_1^{(s)}$	94.89	0.0840	*95.00*	0.0870
	Full-Rank		+Margin	
F_1	93.94	0.0507	*93.97*	0.0972
Precision	93.94%	0.0815	93.94%	0.1071
Recall	93.93%	0.0542	*94.00%*	0.0896
Class. Acc.	**98.74%**	0.0278	98.73%	0.0349
$F_1^{(s)}$	95.14	0.0545	*95.18*	0.0940
	Shared		+Margin	
F_1	94.08	0.0760	**94.20**	0.0650
Precision	94.10%	0.0774	**94.17%**	0.0662
Recall	94.07%	0.0882	**94.24%**	0.0824
Class. Acc.	**98.74%**	0.0212	98.73%	0.0340
$F_1^{(s)}$	95.28	0.0760	**95.42**	0.0721

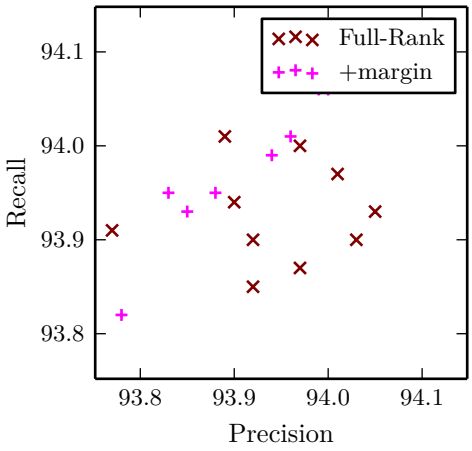
Table 4.5 Detailed results comparing improved NeuroCRFs for the Chunking (CoNLL-2000) task, showing the F_1, precision, recall, classification accuracy and segmental F_1. "Shared" refers to full-rank NeuroCRFs with shared parameters. Results are the average μ and standard deviation σ of 10 models. **Bold** font indicates absolute best result, while *italic* font indicate per-row best result.

Measure	Low-Rank Base	Low-Rank +Mar.	Full-Rank Base	Full-Rank +Mar.	Full-Rank +Sha.	Full-Rank +Both
F_1	93.86	93.91	94.31	94.10	94.24	94.27
Precision	93.76%	93.79%	94.30%	94.07%	94.24%	94.22%
Recall	93.95%	94.03%	94.31%	94.13%	94.25%	94.33%
Class. Accuracy	98.67%	98.65%	98.79%	98.78%	98.78%	98.74%
$F_1^{(s)}$	95.12	95.20	95.46	95.26	95.41	95.47

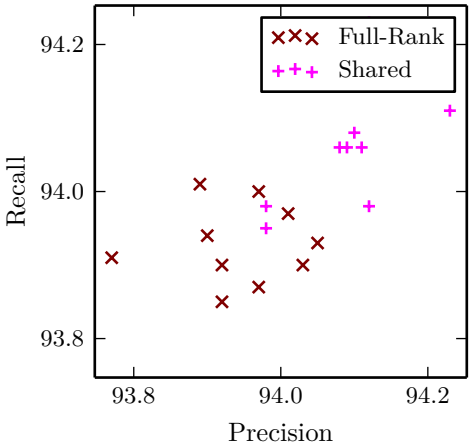
Table 4.6 Detailed results comparing ensemble NeuroCRFs for the Chunking (CoNLL-2000) task, showing the F_1, precision, recall, classification accuracy and segmental F_1. "+Both" refers to the combination of large margin and shared parameters.



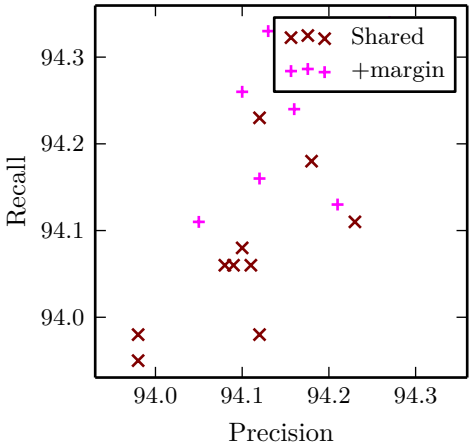
(a) Low-Rank vs Margin



(b) Full-Rank vs Margin



(c) Full-Rank vs Shared



(d) Shared vs Margin

Fig. 4.2 Precision-Recall plot comparing the performance of improved NeuroCRF on the chunking task.

full-rank, full-rank with shared parameters. Those base configurations are compared to the same configuration trained with large margin.

Large margin training improves performance for the full-rank models, but reduces performance for the low-rank models. A two-tailed Student's T-tests shows that the change in F_1 is statistically significant for the full-rank models ($p \leq 0.50\%$), but not for the low-rank models ($p \geq 19.00\%$). Full-rank models with shared parameters are also negatively affected by large margin training ($p \leq 0.30\%$). Table 4.7 shows that large margin training improved classification at the expense of segmentation for low-rank NeuroCRFs, whose NN model label emissions. The table also shows that large margin training improved segmentation for full-rank NeuroCRFs, whose NN model label-to-label transitions, while also improving classification accuracy. Finally, the table shows that the addition of large margin training to models with shared parameters, whose NN models both label emissions and label-to-label transitions, significantly reduced classification accuracy.

Table 4.7 shows significant improvement when shared parameters are added to full-rank NeuroCRFs. A two-tailed Student's T-tests shows that the change in F_1 is statistically significant ($p \leq 0.30\%$). Both classification and segmentation performance are improved. There are also significant improvement to precision and recall.

Table 4.8 shows improved performance using ensemble models, when compared to Table 4.7. Ensemble learning improved the average F_1 for all configuration. Both segmentation and classification are improved by the ensemble models. The significant improvement observed are caused by this join improvement. In general, the performance of the ensemble models depend on the performance of the models used to create them. This is visible in Figure 4.1b, where the line indicating the performance of ensemble models clearly follow the boxes indicating the range of performance obtained with the original models.

WikiNER

Table 4.9 shows detailed results for the WikiNER task. The performance metric used are described in Section 2.6 The table is divided into three set of experiments: low-rank, full-rank, full-rank with shared parameters. Those base configurations are compared to the same configuration trained with large margin.

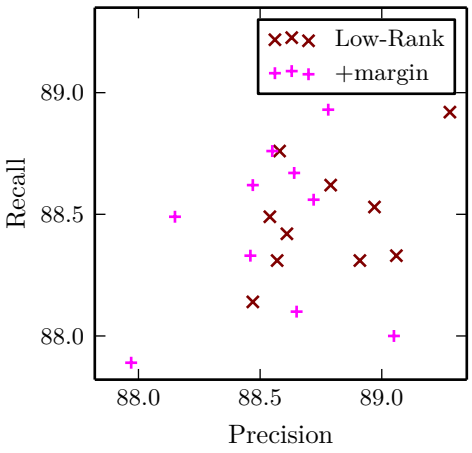
Both large margin training and shared parameters results in improvements, on all five measures. In particular, low-rank NeuroCRFs trained with large margin achieve performance

Measure	Low-Rank μ	Low-Rank σ	+Margin μ	+Margin σ
F_1	*88.63*	0.2145	88.49	0.2540
Precision	*88.78%*	0.2688	88.54%	0.3100
Recall	*88.48%*	0.2322	88.43%	0.3450
Class. Acc.	94.51%	0.2012	*94.66%*	0.1257
$F_1^{(s)}$	*93.78*	0.1402	93.48	0.2477
	Full-Rank		+Margin	
F_1	88.75	0.2305	*89.03*	0.1505
Precision	88.93%	0.2588	*89.14%*	0.3266
Recall	88.57%	0.3145	*88.93%*	0.2403
Class. Acc.	94.66%	0.2095	*94.73%*	0.2348
$F_1^{(s)}$	93.75	0.1648	**93.99**	0.1691
	Shared		+Margin	
F_1	**89.08**	0.1818	88.82	0.1385
Precision	**89.15%**	0.2139	88.94%	0.1393
Recall	**89.00%**	0.2559	88.70%	0.2734
Class. Acc.	**94.80%**	0.1410	94.54%	0.1394
$F_1^{(s)}$	*93.96*	0.1482	93.95	0.1076

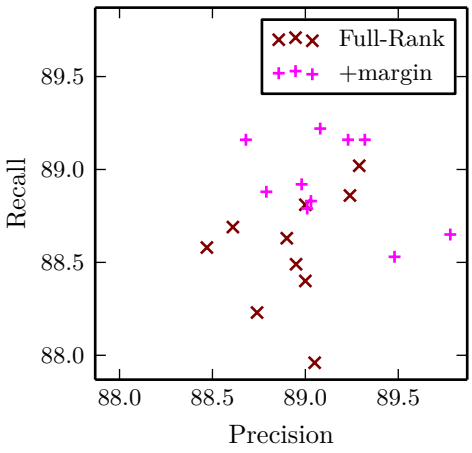
Table 4.7 Detailed results comparing improved NeuroCRFs for the NER (CoNLL-2003) task, showing the F_1, precision, recall, classification accuracy and segmental F_1. "Shared" refers to full-rank NeuroCRFs with shared parameters. Results are the average μ and standard deviation σ of 10 models. **Bold** font indicates absolute best result, while *italic* font indicate per-row best result.

Measure	Low-Rank Base	Low-Rank +Mar.	Full-Rank Base	Full-Rank +Mar.	Full-Rank +Sha.	Full-Rank +Both
F_1	89.27	88.84	89.23	89.42	89.37	89.13
Precision	89.44%	88.84%	89.42%	89.46%	89.39%	89.24%
Recall	89.09%	88.83%	89.04%	89.38%	89.34%	89.02%
Class. Accuracy	94.80%	94.89%	94.94%	94.87%	94.96%	94.67%
$F_1^{(s)}$	94.16	93.62	93.99	94.25	94.11	94.15

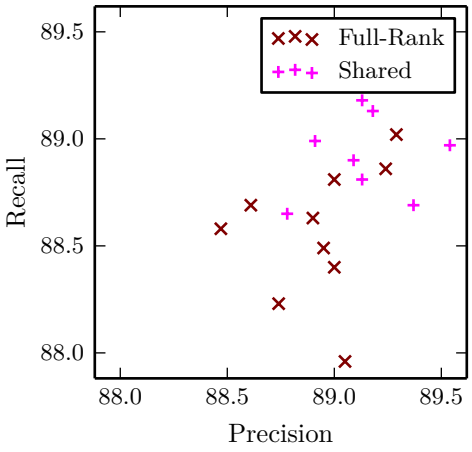
Table 4.8 Detailed results comparing ensemble NeuroCRFs for the NER (CoNLL-2003) task, showing the F_1, precision, recall, classification accuracy and segmental F_1. "+Both" refers to the combination of large margin and shared parameters.



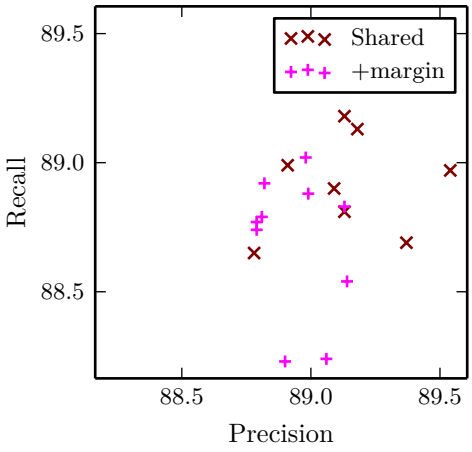
(a) Low-Rank vs Margin



(b) Full-Rank vs Margin



(c) Full-Rank vs Shared



(d) Shared vs Margin

Fig. 4.3 Precision-Recall plot comparing the performance of improved NeuroCRF on the NER task.

similar to the performance of the baseline full-rank NeuroCRFs. A two-tailed Student's T-tests shows that the change in F_1 is statistically significant ($p \leq 3.60\%$ with low-rank, $p \leq 0.01\%$ with full-rank). In general, large margin training improves segmentation and classification performance. The inclusion of shared parameters also improves segmentation and classification performance ($p \leq 0.01\%$). Finally, the combination of shared parameters and large margin training also improve both measures ($p \leq 0.01\%$).

Figure 4.4 confirms the results in Table 4.9. There is a clear separation between full-rank model trained with and without large margin. The inclusion of shared parameters also results in a clear separation. while there is some overlap, low-rank models trained with large margin dominate the upper right corner, corresponding to higher F_1, while low-rank models trained without occupy the lower right corner. Full-rank NeuroCRFs with shared parameters trained with large margin also dominate the upper right corner, while the full-rank NeuroCRFs with shared parameters trained without large margin occupy the lower right corner. In this particular case, there is a significant overlap between the two groups.

4.5 Summary and Discussion

In this chapter, we presented improvements to the NeuroCRF models presented in Chapter 3. One of those improvement is an extension of full-rank NeuroCRF where NN outputs, and therefore parameters, are shared by similar transitions. Those shared parameters are added to the transition specific parameters. The addition of those shared parameters improve performance on the three task of interest.

Large margin training, the second improvement, was also found to improve performance. The combination of shared parameters and large margin training improved performance for two of the three tasks.

Finally, we exploited the complementarity of the 10 models used to evaluate performance by combining them into a single ensemble model. Those ensemble models improved performance over the 10 original models. It was found that their performance is affected by the other improvements similarly to the average performance of the 10 original models.

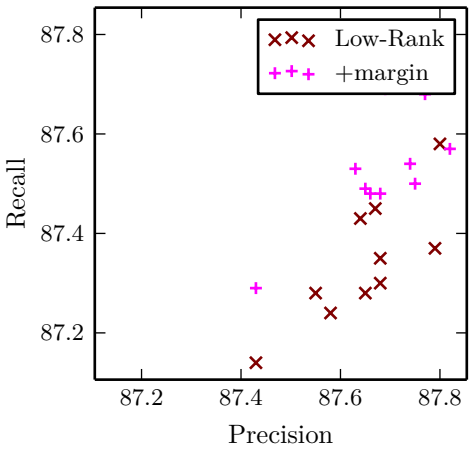
We report results on WikiNER to compensate for the small size of CoNLL-2003. WikiNER is significantly larger, which help training and reduce the variation caused by the random initialization, and by the test set. This is reflected in the experimental results, where the experiments on WikiNER reported in Table 4.9 show consistent improvements, unlike the

Measure	Low-Rank μ	Low-Rank σ	+Margin μ	+Margin σ
F_1	87.49	0.1107	*87.60*	0.1034
Precision	87.65%	0.1092	*87.68%*	0.1079
Recall	87.34%	0.1233	*87.53%*	0.1142
Class. Acc.	93.46%	0.0718	*93.55%*	0.0573
$F_1^{(s)}$	93.62	0.0658	*93.65*	0.0747
	Full-Rank		+Margin	
F_1	87.58	0.0739	*87.90*	0.1122
Precision	87.75%	0.0738	*88.04%*	0.0971
Recall	87.41%	0.0797	*87.76%*	0.1328
Class. Acc.	93.61%	0.0722	*93.76%*	0.0887
$F_1^{(s)}$	93.56	0.0532	*93.74*	0.0489
	Shared		+Margin	
F_1	87.95	0.1569	**88.10**	0.1082
Precision	88.14%	0.1435	**88.27%**	0.0983
Recall	87.75%	0.1764	**87.94%**	0.1211
Class. Acc.	93.81%	0.0810	**93.89%**	0.0658
$F_1^{(s)}$	93.75	0.0993	**93.83**	0.0767

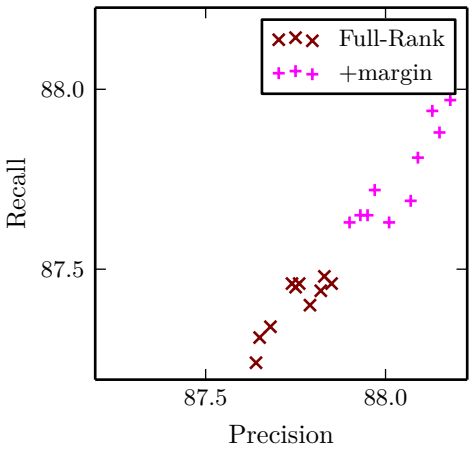
Table 4.9 Detailed results comparing improved NeuroCRFs for the WikiNER task, showing the F_1, precision, recall, classification accuracy and segmental F_1. "Shared" refers to full-rank NeuroCRFs with shared parameters. Results are the average μ and standard deviation σ of 10 models. **Bold** font indicates absolute best result, while *italic* font indicate per-row best result.

Measure	Low-Rank Base	Low-Rank +Mar.	Full-Rank Base	Full-Rank +Mar.	Full-Rank +Sha.	Full-Rank +Both
F_1	88.02	87.79	88.03	88.29	88.40	88.50
Precision	88.16%	87.88%	88.18%	88.43%	88.57%	88.66%
Recall	87.89%	87.70%	87.88%	88.14%	88.23%	88.34%
Class. Accuracy	93.71%	93.58%	93.81%	93.92%	93.98%	94.05%
$F_1^{(s)}$	93.94	93.81	93.84	94.00	94.06	94.10

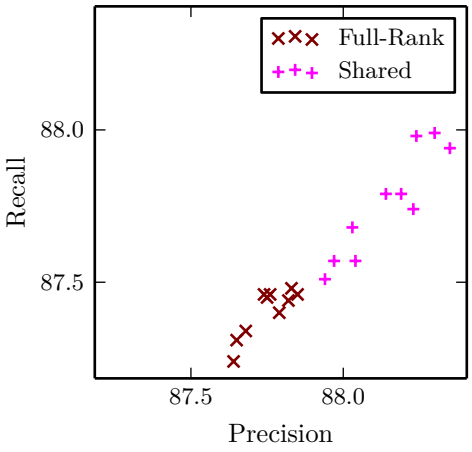
Table 4.10 Detailed results comparing ensemble NeuroCRFs for the WikiNER task, showing the F_1, precision, recall, classification accuracy and segmental F_1. "+Both" refers to the combination of large margin and shared parameters.



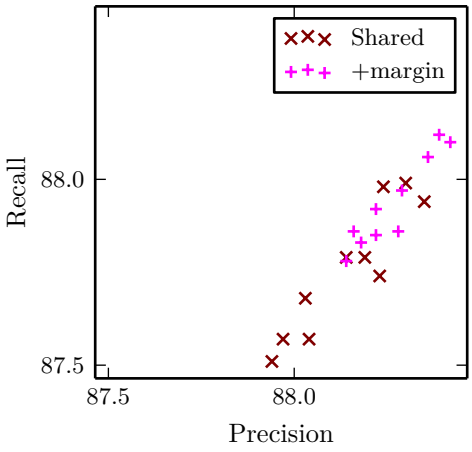
(a) Low-Rank vs Margin



(b) Full-Rank vs Margin



(c) Full-Rank vs Shared



(d) Shared vs Margin

Fig. 4.4 Precision-Recall plot comparing the performance of improved NeuroCRF on the WikiNER task.

experiments on CoNLL-2003 reported in Table 4.7, where large margin training degraded performance for two of the three base configurations. Those consistent improvements match the results of the experiment on the chunking task, reported in Table 4.5.

More feature engineering is required to take advantage of the added shared parameters. In particular, a natural extension of this method would be to apply clustering to tie transitions, rather than adding generalized events. This would reduce the number of parameters required, and could improve performance when there is not enough training data.

Chapter 5

Recurrent NeuroCRFs

NeuroCRFs using recurrent layers are presented in this chapter.[1] The output of a recurrent layer is a function of its input and of its previous output. This can support long term dependencies between input and output without extending the size of the input window or extending the Markov order of the model.

Our experiment used two forms of recurrent layer. They will be presented in the following sections. The second form, long short-term memory (LSTM) layers, was found to significantly improve performance. This will be the focus of this chapter.

5.1 Motivation

In a NeuroCRF, the NN is used to learn and extract useful features from the input. With a feed forward NN (FF NN), the features must be extracted from a sliding window. This limits the capacity of the NeuroCRF to learn and model long term dependencies between input and output. For example, no features related to the topic of a sentence can be extracted by a FF NN. While the topic might be identifiable for a given window, the FF NN cannot provide the corresponding features to the previous and following windows, severely limiting their utility. Segments can also be significantly longer than the sliding window, which will complicate their classification.

In general, the CRF component of the NeuroCRFs can assemble the local potentials provided by the FFNN into longer term potentials. For example, $\alpha_{t,\mathrm{I-LOC}}$ accumulates

[1] The content of this chapter is based on work published in "LSTM-Based NeuroCRFs for Named Entity Recognition" [13].

potentials for a LOC segment, even when the beginning of the segment is outside the sliding window. The CRF Markov order limits the CRF capacity to address long term dependencies. In the previous example, $\alpha_{t,\text{I}-\text{LOC}}$ combines all possible LOC segments starting before t. This can prevent accurate detection of the segment's end, or prevent the model from realizing that the segment is, in fact, the name of a municipal organization, whose correct class is ORG.[2] The performance of a CRF trained on a given corpus depends almost totally on the quality of its features. Improved feature analysis should improve performance.

Finally, the recurrent layer can also increase the effective Markov order. This is because recurrent layers are continuous state machines. Those continuous state machines are not constrained to a finite set of states, determined by the Markov order. They can instead learn a function that creates states as needed to analyze the input.

5.2 Related Works

Recurrent neural networks (RNNs) have been combined with CRFs in [60, 82], forming a low-rank NeuroCRFs based on RNN. This has also been done in the domain of image processing [83].

LSTMs were used for the CoNLL-2003 NER challenge [84], with a very low performance ($F_1 = 60.15$). Those models were used to generate a vector representation of the label, resulting in an architecture completely different from the CRF based models described in this chapter. Hammerton hypothesized that the low performance was, in part, caused by the low informativeness of their word representation.

More recently, bidirectionnal LSTMs with rich features have [85] obtained excellent performance on the CoNLL-2003 task. In [85], a FFNN low-rank NeuroCRF with those rich features obtained $F_1 = 89.67$[3]. Using a bidirectional LSTM model with those rich features augmented with a complex word representation improved performance to 91.62[4]. A similar model was presented in [86], using a conventional feature set but without the complex word representation, reaching $F_1 = 90.10$ on the CoNLL-2003 task. When gazetteer features, which are highly informative for NER, are removed, this model reached 88.83, which

[2]This is a common problem in NER, complicated by a combination of inconsistent naming conventions and the placement of adjectives in English.

[3]Our equivalent result was 88.63, improved to 88.75 with a full-rank model.

[4]Our equivalent result was 89.30

is inferior to our full-rank model presented below[5].

Those results show the importance of feature engineering. In particular, "bad" features did decrease the performance of the models in [86], while "good" features improved performance significantly in [85].

The models described in [87] are very similar to the models described in this chapter. The main difference is the word representation used, in particular the addition of a character based representation. On CoNLL-2003, a basic bidirectional LSTM models, trained using cross-entropy, obtained $F_1 = 87.00$, the character representation improved performances to 89.36, and using a full-rank NeuroCRF improved performance further to 91.21. Those results illustrate the importance of the word representation and of the CRF component.

LSTMs and the similar gated recurrent units have also been used in for joint intent classification and slots filling in [88, 89, 90, 91]. The labeling component of those models is very similar to the models presented in this section, but they are trained using cross-entropy rather than CRF log-likelihood.

5.2.1 Sequence-to-Sequence Models

Sequence-to-sequence (seq2seq) models [70, 92, 93] were originally developed for machine translation. As their name indicates, they are used to generate an output sequence given an input sequence. This task is a more general form of the sequence labeling task addressed in this thesis.

Sequence-to-sequence models were developed for a different problem, and are missing some fundamental constraints required for the tasks we approached with sequence labeling, while also having extra requirements that are not meet by sequence labeling. We can expect well trained seq2seq models to achieve reasonable performances on sequence labeling tasks, but we do not expect that they would exceed the performances of the models presented in this thesis, or of similar models.

Seq2seq models are composed of two major components: an encoder and a decoder. Both are recurrent NNs. The encoder is used to create a summary of the input sequence, as well as to extract features. The decoder is used to generate the output sequence, based on the summary encoded by the encoder. The features extracted by the encoder are used by the decoder through an attention mechanism [92] which uses an internal state to weight the

[5]Most of this difference is attributed to our word representation.

features extracted at various point in the input sequence.

The decoder is required in order to support output sequences whose length is not fixed to the input sequence's length. In sequence labeling, those two lengths are always equal.

The attention mechanism is required for machine translation, as there is not always a clear one-to-one mapping between input and output. For our sequence labeling tasks, there is a known one-to-one mapping between input and output. Fundamentally, this is the key difference: seq2seq models have to learn a complex alignment between input and output, but this alignment is not part of their output, while our models assume a one-to-one sequential mapping between input and output. A seq2seq model used for sequence labeling would have to learn this *exact* mapping but in our model this constraint is built-in.

If a length constraint is added to the decoder and the attention mechanism is replaced by the one-to-one mapping, seq2seq models are simply NN with two recurrent layers.

5.3 Recurrent Layer

Recurrent layers were described in Section 2.4. We use RNN (recurrent NN) to refer to NNs built from those layers, in order to distinguish them from NN built from the LSTM layers presented in the next section.

RNNs add a set of recurrent weights, R_l, to the parameters of the standard feed forward NN. Those recurrent weights combine the layer's previous output $G_{l,t-1}$ to its current input $G_{l-1,t}$ and biases B_l to create a new output $G_{l,t}$.

RNNs' ability to manipulate their memory is limited. An input $G_{l-1,t}$ is only remembered by the recurrent connection, and its contribution to $G_{l,t+d}$ will decay at a fixed rate. Similarly, information can only be forgotten through decays, caused by successive multiplication with R_l, or through an immediate opposed input $G_{l-1,t}W_l$. This opposed input must be a close approximation of the units in $G_{l,t-1}R_l$ that must be forgotten.

Another limitation is that the memory is also the layer's output. The needs of the recurrent connection, used to keep track of state, and the needs of feature extraction can conflict. Ideally, the two would be separate, and recurrent layers would have a set of units used as memory, and another set that would be used as output. This is possible if the following layer has zeroed weights for the output units. While it is possible for a RNN to learn such a topology, it is unlikely to occur without explicit constraints.

Despite those limitation, RNN have been recently combined with CRF [60, 82]. They

also have been used for sequence labelling without using CRFs [94] and for joint training with a topic and/or intent classifier [95, 88].

5.4 Long Short-Term Memory Layer

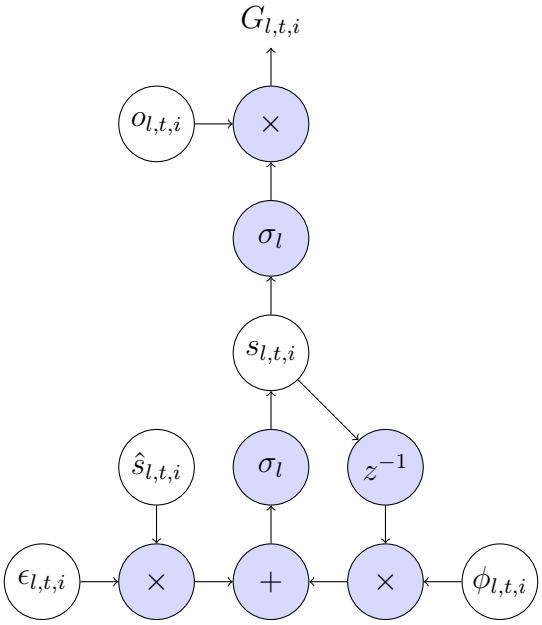


Fig. 5.1 Computation graph of long short term memory cell.

Long short term memory (LSTM) [81] layers address some of the issues with RNNs. In particular, the recursion is now controlled by the input. There is also an added internal memory, with an associated recurrent connection, that can preserve observations independently of the output. Access to this internal memory is controlled by gates. The gates protect the internal memory, which reduces the impact of the vanishing gradient problem described in Section 2.4 [47]. LSTM have been used for sequence labelling, in particular as part of the CoNLL-2003 challenge [84]. More recently they were combined with MEMMs [96].

LSTM layers are function of the form $G_{l,t,i} = f(G_{l-1,t}, G_{l,t-1}, s_{l,t-1,i})$, where $s_{l,t-1,i}$ is the internal memory, forming a state vector $s_{l,t}$. For unit i, the internal memory is updated by

$$s_{l,t,i}(G_{l-1,t}, G_{l,t-1}, s_{l,t-1,i}) = \sigma_l \left(\begin{array}{l} \phi_{l,t,i}(G_{l-1,t}, G_{l,t-1}) s_{l,t-1,i} + \\ \epsilon_{l,t,i}(G_{l-1,t}, G_{l,t-1}) \hat{s}_{l,t,i}(G_{l-1,t}, G_{l,t-1}) \end{array} \right), \tag{5.1}$$

where $\hat{s}_{l,t,i}(G_{l-1,t}, G_{l,t-1})$ is a candidate value, $\phi_{l,t,i}(\cdot)$ is the forget gate, and $\epsilon_{l,t,i}(\cdot)$ is the input gate. The new value is a function of a linear combination of the previous value and a candidate value, where the contribution of each is controlled by the forget and input gates. Finally, the internal memory is exposed to the following layers, and to the next time step, through the output gate

$$G_{l,t,i}(G_{l-1,t}, G_{l,t-1}, s_{l,t-1,i}) = o_{l,t,i}(G_{l-1,t}, G_{l,t-1}) s_{l,t,i}(G_{l-1,t}, G_{l,t-1}, s_{l,t-1,i}) \tag{5.2}$$

Figure 5.1 shows the resulting computation graph.

The forget gate is

$$\phi_{l,t,i}(G_{l-1,t}, G_{l,t-1}) = \sigma_\phi(\sum_{j=1}^{N_{l-1}} G_{l-1,t,j} W_{l,\phi,j,i} + \sum_{j=1}^{N_l} G_{l,t-1,j} R_{l,\phi,j,i} + B_{l,\phi,i}), \tag{5.3}$$

the input gate is

$$\epsilon_{l,t,i}(G_{l-1,t}, G_{l,t-1}) = \sigma_\epsilon(\sum_{j=1}^{N_{l-1}} G_{l-1,t,j} W_{l,\epsilon,j,i} + \sum_{j=1}^{N_l} G_{l,t-1,j} R_{l,\epsilon,j,i} + B_{l,\epsilon,i}), \tag{5.4}$$

the output gate is

$$o_{l,t,i}(G_{l-1,t}, G_{l,t-1}) = \sigma_o(\sum_{j=1}^{N_{l-1}} G_{l-1,t,j} W_{l,o,j,i} + \sum_{j=1}^{N_l} G_{l,t-1,j} R_{l,o,j,i} + B_{l,o,i}), \tag{5.5}$$

and the candidate value is

$$\hat{s}_{l,t,i}(G_{l-1,t}, G_{l,t-1}) = \sigma_l(\sum_{j=1}^{N_{l-1}} G_{l-1,t,j} W_{l,s,j,i} + \sum_{j=1}^{N_l} G_{l,t-1,j} R_{l,s,j,i} + B_{l,s,i}), \tag{5.6}$$

where σ_l is the layer's activation function, and σ_ϕ, σ_ϵ, and σ_o are the gates' activation function, usually the logistic function.

The forget gate can erase the content of the memory cell, the input gate control the update of the memory cell and the output gate can temporarily suppress outputs. While the layer's output is part of the recursion, the memory cell also contributes.

Overall, the recursion is more complex that the RNN's recursion, and is controlled by the input, rather than fixed.

5.4.1 Back Propagation

Given $\frac{\partial L(\mathbf{x},\mathbf{y})}{\partial a_{l+1,t,j}(\mathbf{x})}$, the loss back propagated to the linear activation of layer $l+1$ unit j at time step t, and $\frac{\partial L(\mathbf{x},\mathbf{y})}{\partial a_{l,t+1,g,j}(\mathbf{x})}$, the loss back propagated to the linear activation of gate g of layer l unit j at time step $t+1$, the loss is back propagated into the units at time t, with

$$\text{Rec}_\phi = \sum_j R_{l,\phi,i,j} \frac{\partial L(\mathbf{x},\mathbf{y})}{\partial a_{l,t+1,\phi,j}}, \tag{5.7}$$

$$\text{Rec}_\epsilon = \sum_j R_{l,\epsilon,i,j} \frac{\partial L(\mathbf{x},\mathbf{y})}{\partial a_{l,t+1,\epsilon,j}}, \tag{5.8}$$

$$\text{Rec}_o = \sum_j R_{l,o,i,j} \frac{\partial L(\mathbf{x},\mathbf{y})}{\partial a_{l,t+1,o,j}}, \tag{5.9}$$

$$\text{Rec}_s = \sum_j R_{l,s,i,j} \frac{\partial L(\mathbf{x},\mathbf{y})}{\partial a_{l,t+1,s,j}}, \tag{5.10}$$

$$\frac{\partial L(\mathbf{x},\mathbf{y})}{\partial G_{l,t,i}} = \sum_j W_{l+1,i,j} \frac{\partial L(\mathbf{x},\mathbf{y})}{\partial a_{l+1,t,j}} + \text{Rec}_\phi + \text{Rec}_\epsilon + \text{Rec}_o + \text{Rec}_s. \tag{5.11}$$

It it then back propagated to the output gate by

$$\frac{\partial L(\mathbf{x},\mathbf{y})}{\partial o_{l,t,i}} = \frac{\partial L(\mathbf{x},\mathbf{y})}{\partial G_{l,t,i}} s_{l,t,i}. \tag{5.12}$$

The gradient is propagated to the forget gate by

$$\frac{\partial L(\mathbf{x},\mathbf{y})}{\partial \phi_{l,t,i}} = \frac{\partial L(\mathbf{x},\mathbf{y})}{\partial G_{l,t,i}} o_{l,t,i} \sigma_l' s_{l,t-1,i}, \tag{5.13}$$

to the input gate by

$$\frac{\partial L(\mathbf{x},\mathbf{y})}{\partial \epsilon_{l,t,i}} = \frac{\partial L(\mathbf{x},\mathbf{y})}{\partial G_{l,t,i}} o_{l,t,i} \sigma_l' \hat{s}_{l,t,i}, \tag{5.14}$$

and finally to the candidate value by

$$\frac{\partial L(\mathbf{x},\mathbf{y})}{\partial \hat{s}_{l,t,i}} = \frac{\partial L(\mathbf{x},\mathbf{y})}{\partial G_{l,t,i}} o_{l,t,i} \sigma_l' \epsilon_{l,t,i}. \tag{5.15}$$

5.4.2 Bi-directional LSTM Layer

Bi-directional LSTM (BLSTM) layers are the concatenation of two sub-layers. The first sub-layer remember previous inputs. The second sub-layer reverses the recursion and remembers future inputs. This anti-causal sub layer can be obtained by reversing the recursion used for the causal LSTM layer. The sub-layers cannot be connected to each other, as this would create cycles in the computation graphs.

The main advantage of this approach is that the entire layer has seen the entire input at all t. This can be used by the following layers to create features based on the entire input, rather than being limited to a sliding window, or to the current and previous inputs.

5.5 Experimental Study

Our experimental study is divided into three parts. First, we compared the performance on WikiNER of a RNN-based NeuroCRF to the performance of a LSTM-based NeuroCRF. The LSTM-based NeuroCRF was found to be significantly better; in order to limit the number of variables, the following experiments used LSTM-based NeuroCRF.

The second set of experiments compares the performance of LSTM-based NeuroCRFs combined with the improvements presented in the previous chapters. Those experiments used full-rank NeuroCRFs.

Finally, the third set of experiments investigates the performance of bi-directional LSTM layer, forming BLSTM-based NeuroCRF. BLSTM-based NeuroCRF are combined to the improvement presented in the previous chapters. This set of experiment also include an investigation of the importance, to BLSTM-based NeuroCRF, of the context included in the sliding window. This last investigation was only applied to WikiNER, since its large size reduces the variance of the results.

5.5.1 Model Configuration and Training Procedure

The improvements presented in this chapter do not significantly affect the model configuration and training procedure. As such, the model configuration procedure described in Section 3.4.2, and the training procedure is described in Section 3.4.3 are used.

The recurrent layers are unrolled completely during training; it is common to establish a cut-off point, where the recurrence is only allow edto go backward for a fixed number of time

steps, in order to reduce the computational requirement of model training. This practice is useful when operating over very long sequences, such as speech frames in an acoustic model This is not required for the three tasks used in our experimental studies, as the sentences have, at most, a few dozen words.

Bi-directional LSTM layers are split evenly, so that half the units are causal and half are anti-causal. While this might not be the optimal split, this arbitrary choice avoided the addition of another hyper-parameter, which would have increased the search space.

Finally, we should note that the dropout mask is not involved in the recursion. Dropout is only applied to the non-recurrent connections between layers.

5.5.2 Datasets and Performance Metrics

The data used in those experiments is presented in Section 2.7. The performance metrics used are presented in Section 2.6.

5.5.3 RNN-based NeuroCRF

Table 5.1 shows the performance of three configurations: a baseline full-rank FF-based NeuroCRF, a LSTM-based NeuroCRF and a RNN-based NeuroCRF. While the RNN-based model outperforms the FF-based model, the improvement is limited. The performance of the LSTM-based model is a significant increase over both. Given those results, the following experiments do not use RNN-based NeuroCRFs.

Table 5.1 shows improved classification performance for the RNN-based NeuroCRFs, compared to the baseline. This can be explained by the longer memory, which ensures that the entire segment can be used during classification. The improved classification performance of the RNN-based models is partially cancelled by their lower segmentation performance.

Table 5.1 also shows significant improvement for the LSTM-based NeuroCRFs, over both the baseline and RNN-based NeuroCRFs. Both classification and segmentation performance are significantly improved.

Figure 5.2 is a boxplot of the results shown in Table 5.1. It shows that the performance of RNN-based NeuroCRFs is indeed improved compared to the FF-based NeuroCRFs, there is a significant overlap. Figure 5.2 also shows the significant improvement obtained with LSTM-based NeuroCRFs, compared to the FF-based models and to the RNN-based models. Those experimental results justify the use of LSTM layers in the following experiments.

While improvements can be obtained with RNN-based NeuroCRFs, larger gain are obtained with LSTM-based NeuroCRFs.

Measure	NeuroCRF (FF)		NeuroCRF (RNN)		NeuroCRF (LSTM)	
	μ	σ	μ	σ	μ	σ
F_1	87.58	0.0739	87.66	0.1386	**89.11**	0.0795
Precision	87.75%	0.0738	87.88%	0.1242	**89.26%**	0.0999
Recall	87.41%	0.0797	87.44%	0.1584	**88.96%**	0.0621
Class. Acc.	93.61%	0.0722	93.76%	0.1564	**94.57%**	0.0709
$F_1^{(s)}$	93.56	0.0532	93.50	0.0691	**94.23**	0.0557

Table 5.1 Comparison of the performance of FF-based, LSTM-based and RNN-based NeuroCRFs on WikiNER

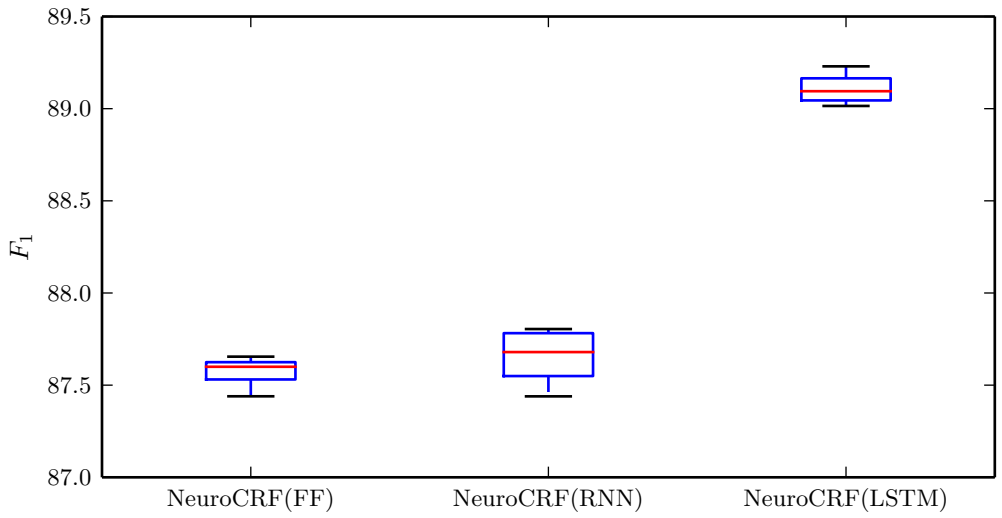


Fig. 5.2 Boxplot comparing the performance of FF-based, LSTM-based and RNN-based NeuroCRFs on WikiNER

5.5.4 LSTM-based NeuroCRF

Table 5.2 shows the results obtained when the FF NN used in the previous chapters is replaced by a LSTM layer. Results from Table 4.3 are included for comparison. In general, the LSTM improved performance for all task and all configurations. For the Chunking and WikiNER task, the worst LSTM configuration is better than the best FFNN configuration.

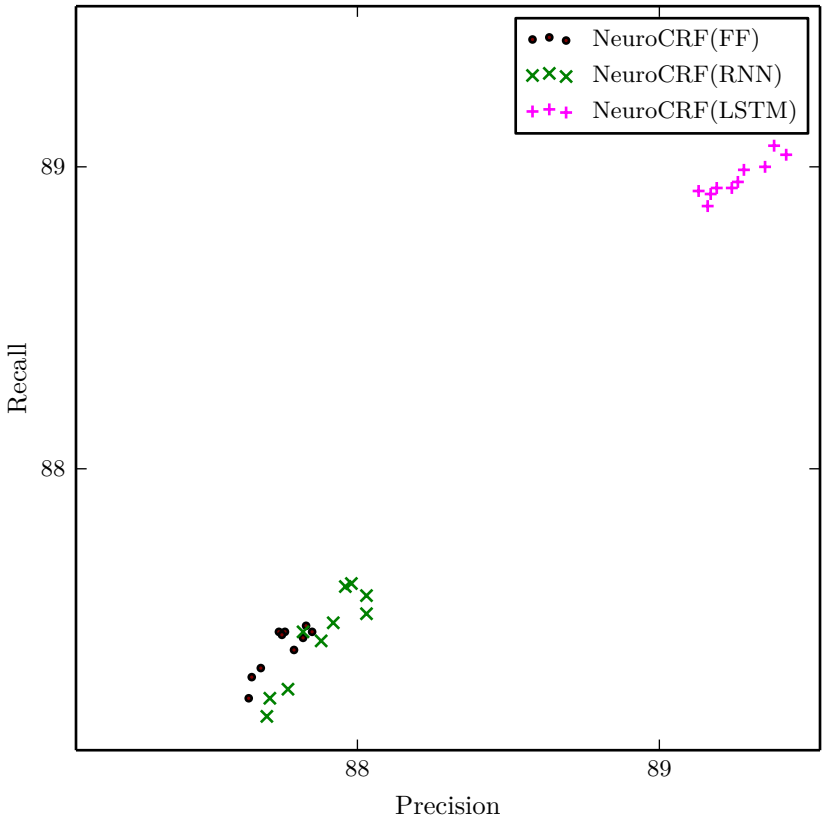


Fig. 5.3 Precision-recall graph comparing the performance of FF-based, LSTM-based and RNN-based NeuroCRFs on WikiNER

On the NER (CoNLL-2003) task, the results are less clear. The best LSTM configurations outperform the best FFNN configurations, but the worst LSTM configuration is significantly worst than the worst FF NN configuration. Nonetheless, the results in Table 5.2 show significant improvement when using LSTM layers.

Configuration	Chunking		NER		WikiNER	
	μ	σ	μ	σ	μ	σ
NeuroCRF(LSTM)	94.46	0.0924	**89.30**	0.2432	89.11	0.0795
+Margin	94.46	0.0948	**89.30**	0.1526	88.82	0.0835
+Shared	94.31	0.0849	88.25	0.1833	89.04	0.1155
+Margin+Shared	**94.56**	0.0550	89.00	0.3119	**89.13**	0.1098
NeuroCRF(FF)	93.94	0.0507	88.75	0.2305	87.58	0.0739
+Margin	93.97	0.0972	89.03	0.1505	87.90	0.1122
+Shared	94.08	0.0760	**89.08**	0.1818	87.95	0.1569
+Margin+Shared	**94.20**	0.0650	88.82	0.1385	**88.10**	0.1082

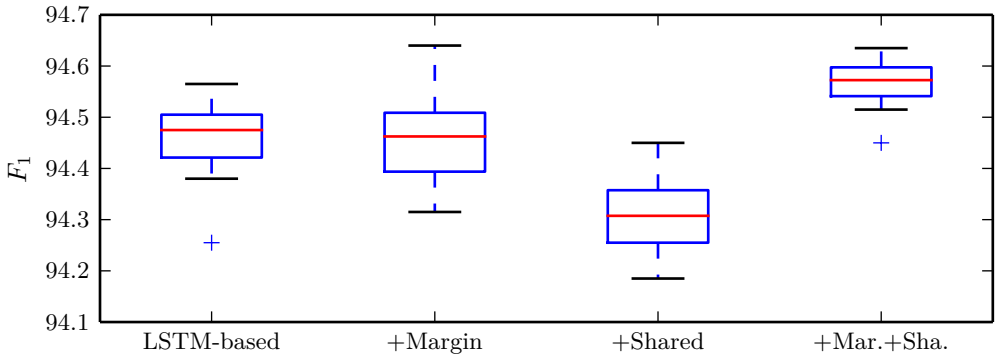
Table 5.2 Comparison of LSTM-based NeuroCRFs and full-rank FF-based NeuroCRFs for the Chunking (CoNLL-2000), NER (CoNLL-2003), and WikiNER (Wikipedia) task. Large margin training, shared parameter and the combination of both are also compared. Results are the average μ and standard deviation σ of 10 models.

Chunking

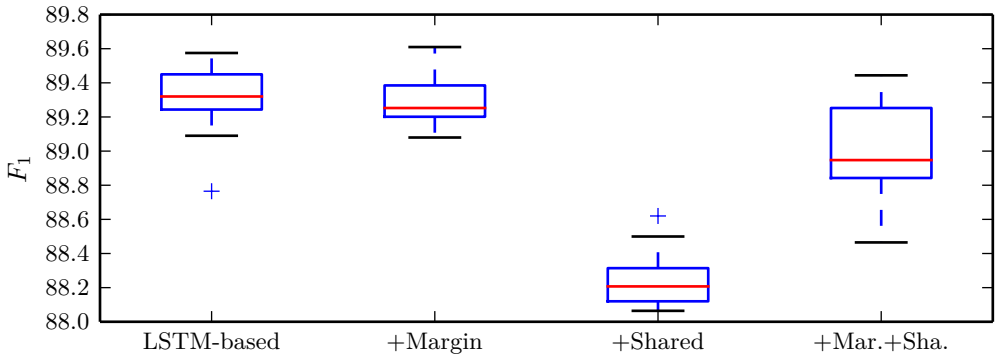
Table 5.3 shows detailed results obtained when LSTM-based NeuroCRFs are used for the chunking task. The performance metric used are described in Section 2.6 The table is divided into two set of experiments: full-rank LSTM-based NeuroCRFs and LSTM-based NeuroCRF with shared parameters. Those base configurations are compared to the same configuration trained with large margin.

The results in Table 5.3 show that the best configuration is the combination of large margin training and shared parameters. This configuration has the best $F_1^{(s)}$, although it also has the worst classification accuracy. This illustrates the importance of good segmentation for this task.

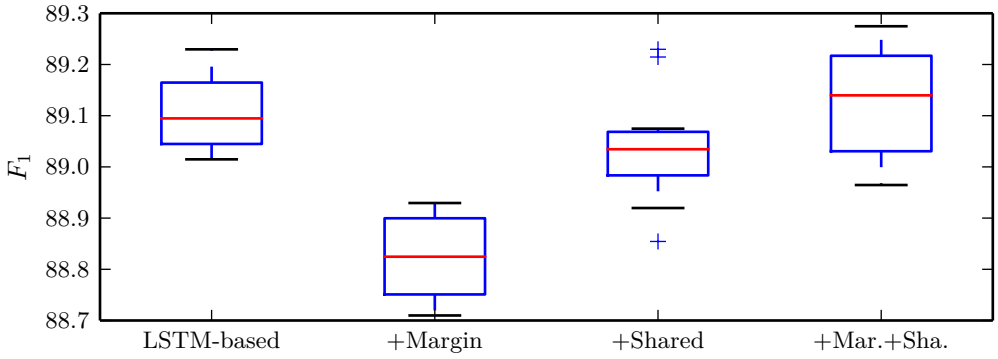
While the combination of shared parameter and large margin training resulted in the best configuration, Table 5.3 shows that LSTM-based NeuroCRFs without shared parameters outperform LSTM-based NeuroCRFs with shared parameters. In this particular case, both $F_1^{(s)}$ and classification accuracy were degraded, reducing overall performance.



(a) Chunking



(b) NER



(c) WikiNER

Fig. 5.4 Boxplots comparing the performance of LSTM-based NeuroCRFs for the Chunking, NER and WikiNER task.

Table 5.3 show that large margin training improved segmentation. It also reduced classification accuracy. Without shared parameters, the two changes cancel each other, resulting in similar overall performance. With shared parameters, the improved $F_1^{(s)}$ more than compensate for the reduction in classification accuracy.

The impact of the LSTM layer is observable by comparing the results in Table 5.3 and the results shown in Table 4.5. The performance of the NeuroCRFs using a LSTM layer shows significant improvement over the baseline NeuroCRFs. Most of the improvement are caused by better segmentation. There are no significant change to the already high classification accuracy, but they are significant improvement to the segmental F_1, indicating better segmentation.

Figure 5.4a is a boxplot of the results in Table 5.3. It shows significant improvements for the combination of large margin training and shared parameters. Figure 5.4a shows the similar performance of LSTM-based NeuroCRF with and without large margin training. It also show that performance is lower when shared parameters are added without large margin training.

Measure	LSTM-based NeuroCRF μ	σ	+Margin μ	σ
F_1	*94.46*	0.0924	*94.46*	0.0948
Precision	*94.48%*	0.0890	94.47%	0.1068
Recall	94.43%	0.1016	*94.45%*	0.0959
Class. Acc.	**98.81%**	0.0223	98.76%	0.0215
$F_1^{(s)}$	95.60	0.0958	*95.64*	0.0891
	Shared		+Margin	
F_1	94.31	0.0849	**94.56**	0.0550
Precision	94.31%	0.1107	**94.57%**	0.0618
Recall	94.30%	0.0773	**94.56%**	0.0823
Class. Acc.	*98.75%*	0.0371	98.72%	0.0162
$F_1^{(s)}$	95.50	0.0784	**95.79**	0.0565

Table 5.3 Detailed results comparing LSTM-based NeuroCRFs for the Chunking (CoNLL-2000) task, showing the F_1, precision, recall, classification accuracy and segmental F_1. "Shared" refers to full-rank LSTM-based NeuroCRFs with shared parameters. Results are the average μ and standard deviation σ of 10 models. **Bold** font indicates absolute best result, while *italic* font indicate per-row best result.

Named Entity Recognition

Table 5.4 shows detailed results obtained when LSTM-based NeuroCRFs are used for the NER task. The performance metric used are described in Section 2.6 The table is divided into two set of experiments: full-rank LSTM-based NeuroCRFs and LSTM-based NeuroCRF with shared parameters. Those base configurations are compared to the same configuration trained with large margin.

Table 5.4 shows that the best configurations are the full-rank LSTM-based NeuroCRFs and the full-rank LSTM-based NeuroCRFs trained with large margin. Those two models are not equivalent, even if their overall performance are. The segmentation performance of the large margin model is higher, but its classification performance is lower. We used a two-tailed Student's T-test to verify the statistical significance of those results. As expected, the differences in precision and recall are not statistically significant ($p \geq 94\%$), but the differences in $F_1^{(s)}$ and A_c are significant ($p \leq 2.85\%$ and $p \leq 3.12\%$, respectively).

Table 5.4 and Table 5.2 show degraded performance when using shared parameters on the NER (CoNLL-2003) task. This degradation affects all performance metrics, and is statistically significant ($p \leq 0.01\%$.) This combination of LSTM layer and added shared parameter has significantly more parameters than the baseline NeuroCRF. CoNLL-2003 is the smallest training corpus used in our experiment, and we do not observe a similar degradation with the other tasks.

Figure 5.4b is a boxplot of the results in Table 5.4. It shows the similar performance of LSTM-based NeuroCRF with and without large margin training. It also show that performance is significantly lower when shared parameters are added without large margin training. With the chunking task, large margin training improved performance when combined with shared parameters. Figure 5.4b shows that large margin training only compensate for part of the degradation caused by the shared parameters. This indicates that the combination of added shared parameters and the parameters required by the LSTM layer gates increases the overfitting, and that the training corpus is too small to properly train those models.

WikiNER

Table 5.5 shows detailed results obtained when LSTM-based NeuroCRFs are used for the WikiNER task. The performance metric used are described in Section 2.6 The table is divided into two set of experiments: full-rank LSTM-based NeuroCRFs and LSTM-based

Measure	LSTM-based NeuroCRF μ	σ	+Margin μ	σ
F_1	**89.30**	0.2432	**89.30%**	0.1526
Precision	**89.42%**	0.3862	89.41%	0.1986
Recall	**89.19%**	0.2061	89.18%	0.2613
Class. Acc.	**94.83%**	0.1849	94.64%	0.1735
$F_1^{(s)}$	94.17	0.1576	**94.36**	0.1878
	Shared		+Margin	
F_1	88.25	0.1833	*89.00*	0.3119
Precision	88.50%	0.1471	*89.03%*	0.4034
Recall	88.00%	0.2967	*88.98%*	0.2751
Class. Acc.	94.23%	0.1563	*94.56%*	0.3009
$F_1^{(s)}$	93.66	0.2088	*94.12*	0.1298

Table 5.4 Detailed results comparing LSTM-based NeuroCRFs for the NER (CoNLL-2003) task, showing the F_1, precision, recall, classification accuracy and segmental F_1. "Shared" refers to full-rank LSTM-based NeuroCRFs with shared parameters. Results are the average μ and standard deviation σ of 10 models. **Bold** font indicates absolute best result, while *italic* font indicate per-row best result.

NeuroCRF with shared parameters. Those base configurations are compared to the same configuration trained with large margin.

Table 5.5 shows that the best configuration is a full-rank LSTM-based NeuroCRFs with shared parameters trained with large margin training. The second best configuration is a full-rank LSTM-based NeuroCRFs. The F_1, precisions, recalls, classification accuracies and $F_1^{(s)}$ of those two configurations are similar. Two-tailed Student's T-tests show that the differences are not statistically significant (all $p \geq 17\%$). The third best configuration is a full-rank LSTM-based NeuroCRFs with shared parameters. Its F_1, precisions, recalls, and $F_1^{(s)}$ is not statistically difference from the ones of the best configuration ($p \geq 7\%$), but its classification accuracy is ($p \leq 4\%$). Finally, the worst configuration is a full-rank LSTM-based NeuroCRFs trained with large margin training. Its performance is statistically significantly different from the performance of the other configurations ($p \leq 0.1\%$).

Table 5.5 and Table 5.2 shows that the worst LSTM-based NeuroCRF configuration outperforms the best FFNN-based NeuroCRF configuration. The difference is statistically significant ($p \leq 0.01\%$). Comparing the detailed full-rank results found in Table 4.9 with Table 5.5 shows that both segmental and classification accuracy are improved by the LSTM layer.

Figure 5.4c is a boxplot of the results in Table 5.5. It shows that the performance of the best and second best configuration are similar, and confirm the degradation observed with large margin training without added shared parameters.

5.5.5 BLSTM-based NeuroCRF

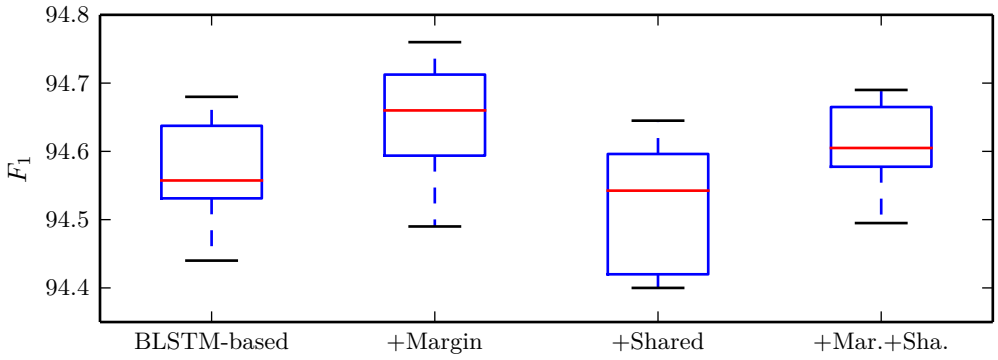
Table 5.6 shows the results obtained when the FF NN used in the previous chapters is replaced by a bi-directional LSTM layer. Results from Table 5.2 are included for comparison. For the WikiNER task, the worst BLSTM configuration is better than the best LSTM configuration. For the Chunking task, performance is improved by the BLSTM layer, but the best LSTM configuration is better than the worst BLSTM configuration. Results are more complex for the NER (CoNLL-2003) task. For this task, there is no clear ranking between LSTM and BLSTM-based NeuroCRF.

Measure	LSTM-based NeuroCRF μ	σ	+Margin μ	σ
F_1	*89.11*	0.0795	88.82	0.0835
Precision	*89.26%*	0.0999	89.00%	0.0949
Recall	**88.96%**	0.0621	88.65%	0.0865
Class. Acc.	**94.57%**	0.0709	94.40%	0.0681
$F_1^{(s)}$	*94.23*	0.0557	94.09	0.0684
	Shared		+Margin	
F_1	89.04	0.1155	**89.13**	0.1098
Precision	89.22%	0.1114	**89.31%**	0.1042
Recall	88.86%	0.1280	*88.95%*	0.1256
Class. Acc.	94.45%	0.0963	*94.54%*	0.0791
$F_1^{(s)}$	94.27	0.0653	**94.28**	0.0812

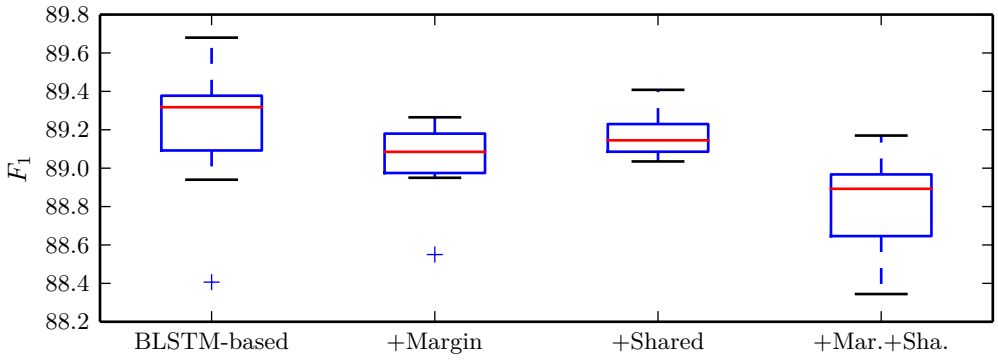
Table 5.5 Detailed results comparing LSTM-based NeuroCRFs for the WikiNER task, showing the F_1, precision, recall, classification accuracy and segmental F_1. "Shared" refers to full-rank LSTM-based NeuroCRFs with shared parameters. Results are the average μ and standard deviation σ of 10 models. **Bold** font indicates absolute best result, while *italic* font indicate per-row best result.

Configuration	Chunking μ	σ	NER μ	σ	WikiNER μ	σ
BLSTM-based NeuroCRF	94.57	0.0743	**89.23**	0.3703	89.28	0.1070
+Margin	**94.65**	0.0871	89.05	0.2064	**89.45**	0.1337
+Shared	94.52	0.0944	89.17	0.1155	89.23	0.0712
+Margin+Shared	94.61	0.0613	88.83	0.2496	89.32	0.1406
LSTM-based NeuroCRF	94.46	0.0924	**89.30**	0.2432	89.11	0.0795
+Margin	94.46	0.0948	**89.30**	0.1526	88.82	0.0835
+Shared	94.31	0.0849	88.25	0.1833	89.04	0.1155
+Margin+Shared	**94.56**	0.0550	89.00	0.3119	**89.13**	0.1098

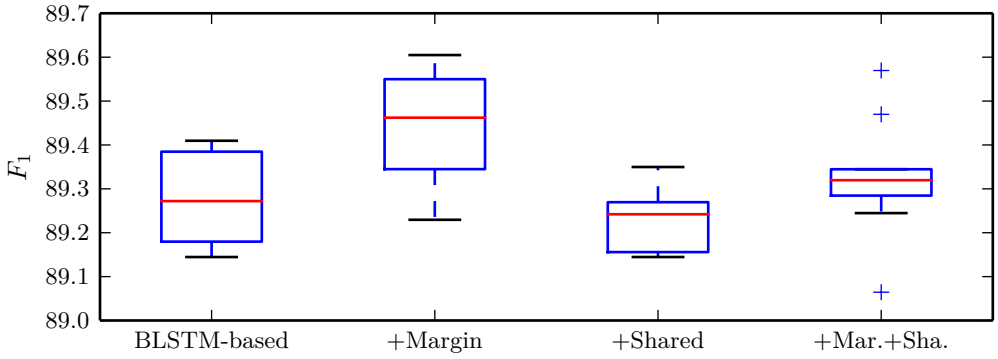
Table 5.6 Comparison of BLSTM-CRFs and LSTM-CRFs for the Chunking (CoNLL-2000), NER (CoNLL-2003), and WikiNER (Wikipedia) task. NeuroCRFs using bi-directional LSTM layers are compared to NeuroCRFs using LSTM layers. Large margin training, shared parameter and the combination of both are also compared. Results are the average μ and standard deviation σ of 10 models.



(a) Chunking



(b) NER



(c) WikiNER

Fig. 5.5 Boxplots comparing the performance of bi-directional LSTM-based NeuroCRFs for the Chunking, NER and WikiNER task.

Chunking

Table 5.7 shows detailed results obtained when BLSTM-based full-rank NeuroCRFs are used for the chunking task. The performance metric used are described in Section 2.6 The table is divided into two set of experiments: full-rank BLSTM-based NeuroCRFs and BLSTM-based NeuroCRF with shared parameters. Those base configurations are compared to the same configuration trained with large margin.

The best configuration is a BLSTM-based full-rank NeuroCRF trained with large margin training. Two-tailed Student's T-tests show that the differences between this configuration and BLSTM-based full-rank NeuroCRF and BLSTM-based full-rank NeuroCRF with shared parameter are statistically significant ($p \leq 4.4\%$ and $p \leq 0.4\%$, respectively). The difference between this best configuration and the second best, BLSTM-based full-rank NeuroCRF with shared parameter trained with large margin training, is not statistically significant ($p \geq 25\%$). This is also visible in Figure 5.5a.

In general, we see that with BLSTM-layer, performance is improved by large margin training and not significantly affected by shared parameters.

Named Entity Recognition

Table 5.8 shows detailed results obtained when BLSTM-based full-rank NeuroCRFs are used for the NER (CoNLL-2003) task. The performance metric used are described in Section 2.6 The table is divided into two set of experiments: full-rank BLSTM-based NeuroCRFs and BLSTM-based NeuroCRF with shared parameters. Those base configurations are compared to the same configuration trained with large margin.

The best configuration is a BLSTM-based full-rank NeuroCRF. The second best is a BLSTM-based full-rank NeuroCRF with shared parameter. A two-tailed Student's T-tests shows that the differences between those two configurations is not statistically significant ($p \geq 61\%$). The difference between the best configuration and the third best, a BLSTM-based full-rank NeuroCRF trained with large margin training, is also not statistically significant ($p \geq 19\%$). Finally, the difference between the best and worst configuration is statistically significant ($p \leq 2\%$). This is also visible in Figure 5.5b.

Given that the results in Table 5.8 and Table 5.4 cannot be clearly separated, further analysis is required. There is not a statistically significant difference between the best configuration in Table 5.8 and the best configurations in Table 5.4 ($p \geq 61\%$). This indicates that

Measure	BLSTM-based NeuroCRF μ	σ	+Margin μ	σ
F_1	94.57	0.0743	**94.65**	0.0871
Precision	94.64%	0.0983	*94.67%*	0.0967
Recall	94.51%	0.0704	**94.63%**	0.0861
Class. Acc.	98.88%	0.0341	**98.89%**	0.0345
$F_1^{(s)}$	95.65	0.0467	**95.71**	0.0854
	Shared		+Margin	
F_1	94.52	0.0944	*94.61*	0.0613
Precision	94.55%	0.1050	*94.60%*	0.0911
Recall	94.48%	0.0975	**94.63%**	0.0703
Class. Acc.	98.85%	0.0512	*98.85%*	0.0214
$F_1^{(s)}$	95.62	0.0758	**95.71**	0.0652

Table 5.7 Detailed results comparing BLSTM-based NeuroCRFs for the Chunking (CoNLL-2000) task, showing the F_1, precision, recall, classification accuracy and segmental F_1. "Shared" refers to full-rank BLSTM-based NeuroCRFs with shared parameters. Results are the average μ and standard deviation σ of 10 models. **Bold** font indicates absolute best result, while *italic* font indicate per-row best result.

the lower performance, from $F_1 = 89.30$ to $F_1 = 89.23$, is caused by the random initialization.

In general, for this task, the BLSTM layer did not improve performance significantly. Any difference is masked by the significant variability caused by the random initialization.

Measure	BLSTM-based NeuroCRF μ	σ	+Margin μ	σ
F_1	**89.23**	0.3703	89.05	0.2064
Precision	**89.38%**	0.2925	89.15%	0.0991
Recall	**89.09%**	0.4888	88.94%	0.3958
Class. Acc.	**94.83%**	0.2693	94.68%	0.1525
$F_1^{(s)}$	*94.10*	0.1766	94.05	0.1678
	Shared		+Margin	
F_1	*89.17*	0.1155	88.83	0.2496
Precision	*89.27%*	0.2740	88.91%	0.3352
Recall	*89.07%*	0.2215	88.75%	0.2927
Class. Acc.	*94.73%*	0.1327	94.54%	0.1882
$F_1^{(s)}$	**94.13**	0.1245	93.96	0.1214

Table 5.8 Detailed results comparing BLSTM-based NeuroCRFs for the NER (CoNLL-2003) task, showing the F_1, precision, recall, classification accuracy and segmental F_1. "Shared" refers to full-rank BLSTM-based NeuroCRFs with shared parameters. Results are the average μ and standard deviation σ of 10 models. **Bold** font indicates absolute best result, while *italic* font indicate per-row best result.

WikiNER

Table 5.9 shows detailed results obtained when BLSTM-based full-rank NeuroCRFs are used for the WikiNER task. The performance metric used are described in Section 2.6 The table is divided into two set of experiments: full-rank BLSTM-based NeuroCRFs and BLSTM-based NeuroCRF with shared parameters. Those base configurations are compared to the same configuration trained with large margin.

The best configuration is a BLSTM-based full-rank NeuroCRF trained with large margin training. The second best configuration is the equivalent with shared parameters. A two-tailed Student's T-tests shows that the differences between those two configurations is not statistically significant ($p \geq 7\%$). The best configuration is statistically different from the

two remaining configurations ($p \leq 1\%$), while the second best configuration is not ($p \geq 9\%$). This is also visible in Figure 5.5c.

Comparing the results in Table 5.9 and Table 5.5, the worst BLSTM-based configuration is better than the best LSTM-based configuration. The difference is statistically significant ($p \leq 3.5\%$).

Measure	BLSTM-based NeuroCRF μ	σ	+Margin μ	σ
F_1	89.28	0.1070	**89.45**	0.1337
Precision	89.46%	0.1125	**89.58%**	0.1305
Recall	89.10%	0.1049	**89.32%**	0.1507
Class. Acc.	94.69%	0.0932	**94.77%**	0.0863
$F_1^{(s)}$	94.29	0.0524	**94.38**	0.0903
	Shared		+Margin	
F_1	89.23	0.0712	*89.32*	0.1406
Precision	89.41%	0.0604	*89.50%*	0.1313
Recall	89.05%	0.0927	*89.15%*	0.1586
Class. Acc.	94.60%	0.0533	*94.64%*	0.0913
$F_1^{(s)}$	94.32	0.0776	**94.38**	0.0797

Table 5.9 Detailed results comparing BLSTM-based NeuroCRFs for the WikiNER task, showing the F_1, precision, recall, classification accuracy and segmental F_1. "Shared" refers to full-rank BLSTM-based NeuroCRFs with shared parameters. Results are the average μ and standard deviation σ of 10 models. **Bold** font indicates absolute best result, while *italic* font indicate per-row best result.

5.5.6 Importance of context size

The computational complexity of LSTM layers is usually dominated by the large matrix multiplication between the input and the corresponding weight matrix. This, in turn, is a function of the size on the input vector, itself a function of the sliding window's size. This window is composed of a central word, the C words preceding the central word, and the C words following the central word. Removing the context from this sliding window would reduce the computational complexity of LSTM layer.

The context provided by the preceding and following words is an important source of information for feature analysis. Without it, FF NNs would have to extract features from isolated words, since they have no memory of past or future. In the case of bi-directional LSTM layers, both the past and future can be available at all time step. The internal memory can be used to replace the important information that would be provided by the context.

A small experiment on WikiNER, where models are trained with no context, confirms this. Table 5.10 shows that the performance of a full-rank NeuroCRF using a FFNN is significantly degraded with the context is removed. The segmentation performance is reduced. More interestingly, so is the classification accuracy, confirming that having access to the entire named entity is required in order to classify it correctly. Those results confirm NeuroCRFs using a FFNN cannot learn essential features, as those features require the context.

Table 5.10 also shows that a bi-directional LSTM layer can reconstruct this context. Those results show that NeuroCRFs using a BLSTM NN can learn to reproduce the essential parts of the context, by using their internal memory as a substitute.

Model	$F_1^{(s)}$	A_c	F_1
NeuroCRF (FF)	93.56	93.61%	87.58
Without Context	91.98	91.99%	84.61
NeuroCRF (BLSTM)	94.10	94.83%	89.23
Without Context	94.23	94.70%	89.23

Table 5.10 Impact of context component of input for WikiNER.

5.6 Summary and discussion

In this chapter, we presented recurrent versions of the NeuroCRF models presented in Chapter 3 and Chapter 4. The resulting recurrent NeuroCRFs are able to use the entire input sequence when learning and extracting features.

Our initial experiment compared two forms of recurrent NeuroCRFs. The first is based on a conventional recurrent layer, where the previous output of the layer is used to compute the current output; this is a full-rank form of R-CRF. The second form is based on a LSTM layer, where an internal memory is also used to carry the summary of previous inputs. Our experimental results shows significant improvements with the LSTM-based NeuroCRF

compared to the RNN-based NeuroCRF and the FFNN baseline.

This initial experiment was followed by the combination of the improved NeuroCRF presented in Chapter 3 with LSTM layers. Performance was significantly improved for the WikiNER and chunking tasks. Results are less clear on the NER (CoNLL-2003) task, due to the high variance caused by the random initialization.

Finally, we investigated bi-directional LSTM layers, which are divided in a causal and anti-causal section. Our experimental results showed significant improvements over purely causal LSTM layers. We also confirmed that bi-directional LSTM layers can replace the context component of the input window without suffering from degraded performance. This property can be used to decrease the computational complexity of LSTM-based system, which is dominated by the matrix multiplication of the input and weight of the LSTM layer.

Chapter 6

Conclusion and Future Work

This chapter will present a summary of the major contributions found in Chapters 3, 4 and 5 of this thesis to the problem of natural language understanding (NLU). This will be followed by an overview of some potential extension of the work presented in this thesis.

6.1 Full-rank NeuroCRFs

Chapter 3 presented full-rank NeuroCRFs, the first contribution of this thesis. Full-rank NeuroCRFs are the combination of a conditional random field (CRF) and a neural network (NN). A CRF's factor functions are used to factorize a complex distribution into a set of simpler factors. A full-rank NeuroCRF's factor functions are the outputs of a NN. Linear-chain CRFs, commonly used in natural language understanding for information extraction through sequence labelling, have factor functions corresponding to label-to label transitions. In a full-rank NeuroCRF, each possible label-to-label transition has a corresponding NN output. The corresponding factor function is a function of the origin label y_{t-1}, the destination label y_t and of the observation $\mathbf{x}_t$.

Previous work [59, 3, 60, 61] in this area focussed on low-rank NeuroCRFs, where the NN is used to model label emission. Full-rank NeuroCRFs were able to obtain significant improvements on a chunking task, where the origin label is a strong predictor of the destination label. Improvements were also obtained on two NER tasks. Those improvements were limited by the low mutual information between successive labels, and by the higher class entropy of the NER tasks. The experimental results showed that full-rank NeuroCRFs were able to improve, compared to low-rank, the modelling of inter-labels dependencies, without

degrading performance in the cases where those dependencies are limited.

6.2 Shared Parameters

Chapter 4 presented full-rank NeuroCRFs with added shared parameters. NN outputs, corresponding to generalized events, were added, and combined linearly to form factor functions corresponding to label-to-label transitions. An initial sharing scheme, suitable for most tasks, was developed.

This sharing scheme creates emission events based on the class corresponding to a label, as well as to the position in a segment (i.e. first or following word) indicated by the same label. Those emission events are then used to create a set of transitions events, and output units corresponding to those events are added to a full-rank NeuroCRF. The full factor functions average the NN outputs corresponding to the generalized events set of a specific transition.

Those shared parameters were found to improve performance on all three tasks. They were also combined with large margin training, intended to improve the model's generalization, and with ensemble learning.

6.3 Recurrent NeuroCRF

Chapter 5 presented the use of recurrent NN, specifically LSTM layers, in NeuroCRFs. LSTM layers use an internal memory cell as part of their recursion, allowing them to model long term dependencies between input and output. The use of bi-directional LSTM layers, which consist of causal and anti-causal halves, was also investigated.

Significant improvements were observed with LSTM and bi-directional LSTM layers. Their performance when combined with the shared parameters presented in Chapter 4, as well as large margin training, was also investigated. Finally, an experimental study showed that bi-directional layers eliminate the need for the sliding window.

6.4 Future Work

This section presents an overview of some possible future work extending the contributions of this thesis.

6.4.1 More Datasets

While full-rank NeuroCRFs improved performance over low-rank NeuroCRF, the improvements were limited when the mutual information between successive labels was low. In the case of the NER task, the mutual information is limited by the sparseness of named entities in the datasets used. An obvious extension of the work presented in this thesis would be to apply full-rank NeuroCRFs to other tasks with high mutual information between labels.

Our experimented studies used labelled data based on high quality written English. While this should not affect the mutual information, the high regularity of this form of English should simplify the feature analysis. Feature analysis of spontaneous speech, with its inherent irregularities and the errors introduced by automatic speech recognition, is significantly more difficult. FF-based and LSTM-based NeuroCRFs should be negatively affected by those irregularities, but we can expect the degradation to be smaller for LSTM-based NeuroCRFs.

Finally, the size and depth of our models were limited by the available data, and the complexity of the tasks used. Larger datasets should support larger and deeper models without overfitting the training data. More complex tasks, requiring more complex feature extraction and with high mutual information between labels, would be ideal candidates for LSTM-based full-rank NeuroCRFs.

6.4.2 Semi-Supervised Learning

Large amount of unlabelled written text is available through the web. Similarly, unlabelled speech can be obtained inexpensively. This data cannot be used for the supervised learning algorithm used in this thesis, but could be used by semi-supervised learning algorithms, which use a mix of labelled and unlabelled training examples to train models.

The process used to pre-train the word representation could be extended to pre-train the hidden layer of a NeuroCRF. This would be done by using the unlabelled data to train a deep NN language model, whose parameters would be used to initialize the hidden layers of a NeuroCRF.

Self-training [97, 98, 99, 100] consists of using the model being trained to label the unlabelled training examples. First, the labelled data is used to learn parameters. The resulting model is used to label the unlabelled training examples. Those examples are used to update the model's parameters. The process can be repeated, depending on the algorithm. This approach requires enough data to train a good initial model, and can be computationally

expansive, due to its iterative nature.

Finally, the algorithm presented in [101] could be adapted for NeuroCRFs. This algorithm minimizes a loss function composed of a discriminative model and a generative model, both based on NNs. In [101], this algorithm is used to train image classifiers, where the class is scalar. Linear chain NeuroCRFs are used with sequences, and the algorithm would have to be modified to take this into account.

6.4.3 Data Driven Parameters Sharing Scheme

Shared parameters were introduced in Chapter 4. An a-priori sharing scheme was used. This scheme makes two major assumptions. First, that all the generalized events associated with a given transition are relevant. Two, that the set of a-priori similarities between labels were the only possible similarities.

This points to the development of a data driven sharing scheme. In particular, an approach based on iteratively clustering transitions, automatically creating groups of similar labels and label-to-label transitions based on their internal representation found in the hidden layer, should result in a more efficient use of the model's parameters. This clustering would remove irrelevant generalized events, and would create missing links between transitions. This could also be used to extend the Markov order of the model without significantly increasing the size of the NN output layer.

In our implementation, the added NN outputs are combined using a fixed linear transform. An obvious approach for data driven parameters sharing would be to learn this transform, which was considered during the work leading to Chapter 4. Further consideration showed that this would, in fact, simply result in a deeper network, as this new transform would simply add a hidden layer. This does underline a key point: the parameter sharing is, fundamentally, a very constrained layer. Future work, in this area, is essentially about determining the required constraints.

6.4.4 System Combination

The ensemble models presented in Chapter 4 were products of experts. This type of ensemble was selected in order to combine NeuroCRFs whose outputs have the same dimensionality and interpretation. We did not investigate ensembles of NeuroCRFs with other model classes. Since the performance of ensemble models rely on complementarity between models, using

heterogenous models should improve performance.

The experiments were also limited to product of experts. Other combinations, such as ROVER [79] or the CRF-based ensemble method presented in [102], could outperform the product of expert, while supporting heterogenous models who cannot be used with a product of expert.

6.4.5 Information Extraction with Attention Mechanisms

An attention mechanism [92] was used in machine translation to indicate the words in the source language utterance relevant for a word in the target language utterance. The attention model can learn complex many-to-many mapping. This could be used to extract information contained in non-consecutive words, as well as to support overlapping information, which cannot be extracted with sequence labelling. While this restriction does not have negative effects for NER, this limitation is the source of the main difference between the structure obtained with chunking and the full parse tree. Attention mechanism have been used for this purpose in[103], without using CRF. Attention models could be used as factor functions in a NeuroCRF. The model output variables would not be a sequence of label, which would required the use of task-specific clique templates. This general approach could also support NLU tasks such as document summarization, which are not well supported by sequence labelling. It could also be used to extract information directly, without labelling. This is especially useful for tasks such as slot or form filling, where field in a fixed structure must be filled by extracting information from one or more utterance, usually as part of a dialog system.

References

[1] K. Gimpel and N. A. Smith, "Softmax-margin crfs: Training log-linear models with cost functions," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 733–736.

[2] E. F. Tjong Kim Sang and S. Buchholz, "Introduction to the conll-2000 shared task: Chunking," in *Proceedings of the 2Nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning - Volume 7*, ser. ConLL '00. Stroudsburg, PA, USA: Association for Computational Linguistics, 2000, pp. 127–132. [Online]. Available: http://dx.doi.org/10.3115/1117601.1117631

[3] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *The Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, 2011.

[4] E. F. Tjong Kim Sang and F. De Meulder, "Introduction to the conll-2003 shared task: Language-independent named entity recognition," in *Proceedings of CoNLL-2003*, W. Daelemans and M. Osborne, Eds. Edmonton, Canada, 2003, pp. 142–147.

[5] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282–289. [Online]. Available: http://dl.acm.org/citation.cfm?id=645530.655813

[6] A. McCallum and W. Li, "Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons," in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, ser. CONLL '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 188–191. [Online]. Available: http://dx.doi.org/10.3115/1119176.1119206

[7] F. Sha and F. Pereira, "Shallow parsing with conditional random fields," in *Proceedings of the 2003 Conference of the North American Chapter of the Association*

for Computational Linguistics on Human Language Technology - Volume 1, ser. NAACL '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 134–141. [Online]. Available: http://dx.doi.org/10.3115/1073445.1073473

[8] 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*, ser. ACL '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 384–394. [Online]. Available: http://dl.acm.org/citation.cfm?id=1858681.1858721

[9] A. Passos, V. Kumar, and A. McCallum, "Lexicon infused phrase embeddings for named entity resolution," *CoNLL-2014*, p. 78, 2014.

[10] Y. Bengio, H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain, "Neural probabilistic language models," in *Innovations in Machine Learning*. Springer, 2003, pp. 137–186.

[11] M.-A. Rondeau and Y. Su, "Full-rank linear-chain neurocrf for sequence labeling," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, April 2015.

[12] ——, "Recent improvements to neurocrfs for named entity recognition," in *Automatic Speech Recognition and Understanding Workshop (ASRU), 2015 IEEE*, December 2015.

[13] ——, "Lstm-based neurocrfs for named entity recognition," in *Interspeech 2016*, 2016.

[14] T. Kudo and Y. Matsumoto, "Chunking with support vector machines," in *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*. Association for Computational Linguistics, 2001, pp. 1–8.

[15] H. Isozaki and H. Kazawa, "Efficient support vector classifiers for named entity recognition," in *Proceedings of the 19th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 2002, pp. 1–7.

[16] J. Kazama, T. Makino, Y. Ohta, and J. Tsujii, "Tuning support vector machines for biomedical named entity recognition," in *Proceedings of the ACL-02 workshop on Natural language processing in the biomedical domain-Volume 3*. Association for Computational Linguistics, 2002, pp. 1–8.

[17] S. Sekine, R. Grishman, and H. Shinnou, "A decision tree method for finding and classifying names in japanese texts," in *Proceedings of the Sixth Workshop on Very Large Corpora*, 1998.

[18] G. Paliouras, V. Karkaletsis, G. Petasis, and C. D. Spyropoulos, "Learning decision trees for named-entity recognition and classification," in *ECAI Workshop on Machine Learning for Information Extraction*, 2000.

[19] D. M. Bikel, R. Schwartz, and R. M. Weischedel, "An algorithm that learns what's in a name," *Machine learning*, vol. 34, no. 1-3, pp. 211–231, 1999.

[20] K. Seymore, A. McCallum, and R. Rosenfeld, "Learning hidden markov model structure for information extraction," in *AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999, pp. 37–42.

[21] G. Zhou and J. Su, "Named entity recognition using an hmm-based chunk tagger," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ser. ACL '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 473–480. [Online]. Available: http://dx.doi.org/10.3115/1073083.1073163

[22] D. Shen, J. Zhang, G. Zhou, J. Su, and C.-L. Tan, "Effective adaptation of a hidden markov model-based named entity recognizer for biomedical domain," in *Proceedings of the ACL 2003 workshop on Natural language processing in biomedicine-Volume 13*. Association for Computational Linguistics, 2003, pp. 49–56.

[23] G. Zhou and J. Su, "Error-driven hmm-based chunk tagger with context-dependent lexicon," in *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*. Association for Computational Linguistics, 2000, pp. 71–79.

[24] A. Molina and F. Pla, "Shallow parsing using specialized hmms," *The Journal of Machine Learning Research*, vol. 2, pp. 595–613, 2002.

[25] A. McCallum, D. Freitag, and F. C. Pereira, "Maximum entropy markov models for information extraction and segmentation." in *ICML*, vol. 17, 2000, pp. 591–598.

[26] D. Klein, J. Smarr, H. Nguyen, and C. D. Manning, "Named entity recognition with character-level models," in *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*. Association for Computational Linguistics, 2003, pp. 180–183.

[27] M. Fresko, B. Rosenfeld, and R. Feldman, "A hybrid approach to ner by memm and manual rules," in *Proceedings of the 14th ACM international conference on Information and knowledge management*. ACM, 2005, pp. 361–362.

[28] G.-L. Sun, Y. Guan, X.-L. Wang, and J. Zhao, "A maximum entropy markov model for chunking," in *2005 International Conference on Machine Learning and Cybernetics*, vol. 6, Aug 2005, pp. 3761–3765.

[29] C. Sutton and A. McCallum, *An introduction to conditional random fields for relational learning*. Introduction to statistical relational learning. MIT Press, 2006.

[30] B. Settles, "Biomedical named entity recognition using conditional random fields and rich feature sets," in *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*. Association for Computational Linguistics, 2004, pp. 104–107.

[31] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2005, pp. 363–370.

[32] N. Sobhana, P. Mitra, and S. Ghosh, "Conditional random field based named entity recognition in geological text," *International Journal of Computer Applications*, vol. 1, no. 3, pp. 143–147, 2010.

[33] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains," *The annals of mathematical statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[34] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[35] D. L. Vail, J. D. Lafferty, and M. M. Veloso, "Feature selection in conditional random fields for activity recognition," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*. IEEE, 2007, pp. 3379–3384.

[36] X. Wang, H. T. Ng, and K. C. Sim, "Dynamic conditional random fields for joint sentence boundary and punctuation prediction." in *INTERSPEECH*, 2012, pp. 1384–1387.

[37] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[38] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1, no. 4, pp. 339–356, 1988.

[39] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *INTERSPEECH*, vol. 2, 2010, p. 3.

[40] S. Kombrink, T. Mikolov, M. Karafiát, and L. Burget, "Recurrent neural network based language modeling in meeting recognition." in *INTERSPEECH*, 2011, pp. 2877–2880.

[41] A. Graves, A. r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, May 2013, pp. 6645–6649.

[42] H. Sak, A. Senior, K. Rao, O. İrsoy, A. Graves, F. Beaufays, and J. Schalkwyk, "Learning acoustic frame labeling for speech recognition with recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, April 2015, pp. 4280–4284.

[43] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3104–3112. [Online]. Available: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf

[44] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[45] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 157–166, 1994.

[46] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of The 30th International Conference on Machine Learning*, 2013, pp. 1310–1318.

[47] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 5, pp. 855–868, 2009.

[48] H.-S. Le, I. Oparin, A. Allauzen, J. Gauvain, and F. Yvon, "Structured output layer neural network language model," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, May 2011, pp. 5524–5527.

[49] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[50] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Neural Information Processing Systems conference*, 2013.

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

[52] L. Qu, G. Ferraro, L. Zhou, W. Hou, N. Schneider, and T. Baldwin, "Big data small data, in domain out-of domain, known word unknown word: The impact of word representations on sequence labelling tasks," *CoNLL 2015*, p. 83, 2015.

[53] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, October 2014, pp. 1532–1543.

[54] Z. Wu and C. L. Giles, "Sense-aaware semantic analysis: A multi-prototype word representation model using wikipedia." in *AAAI*. Citeseer, 2015, pp. 2188–2194.

[55] W. Ling, T. Luís, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, and I. Trancoso, "Finding function in form: Compositional character models for open vocabulary word representation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, September 2015, pp. 1520–1530.

[56] H. Schmid, "Part-of-speech tagging with neural networks," in *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*, ser. COLING '94. Stroudsburg, PA, USA: Association for Computational Linguistics, 1994, pp. 172–176. [Online]. Available: http://dx.doi.org/10.3115/991886.991915

[57] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," *COMPUTATIONAL LINGUISTICS*, vol. 19, no. 2, pp. 313–330, 1993.

[58] J. Nothman, N. Ringland, W. Radford, T. Murphy, and J. R. Curran, "Learning multilingual named entity recognition from Wikipedia," *Artificial Intelligence*, vol. 194, pp. 151–175, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.artint.2012.03.006

[59] T. Do, T. Arti *et al.*, "Neural conditional random fields," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 177–184.

[60] K. Yao, B. Peng, G. Zweig, D. Yu, X. Li, and F. Gao, "Recurrent conditional random field for language understanding," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 4077–4081.

[61] L. Deng and J. Chen, "Sequence classification using the high-level features extracted from deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6844–6848.

[62] J. Sánchez and F. Perronnin, "High-dimensional signature compression for large-scale image classification," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 1665–1672.

[63] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[64] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[65] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1701–1708.

[66] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014, pp. 806–813.

[67] P. Xu and R. Sarikaya, "Convolutional neural network based triangular crf for joint intent detection and slot filling," in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, 2013, pp. 78–83.

[68] J. Peng, L. Bo, and J. Xu, "Conditional neural fields," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, Eds. Curran Associates, Inc., 2009, pp. 1419–1427. [Online]. Available: http://papers.nips.cc/paper/3869-conditional-neural-fields.pdf

[69] D. Yu, S. Wang, and L. Deng, "Sequential labeling using deep-structured conditional random fields," *IEEE Journal of Selected Topics in Signal Processing*, vol. 4, no. 6, pp. 965–973, 2010.

[70] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[71] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, "Grammar as a foreign language," in *Advances in Neural Information Processing Systems*, 2015, pp. 2773–2781.

[72] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, April 1967.

[73] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements," Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[74] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[75] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2188385.2188395

[76] N. S. Keskar and G. Saon, "A nonmonotone learning rate strategy for sgd training of deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, April 2015.

[77] B. T. C. G. D. Roller, "Max-margin markov networks," *Advances in neural information processing systems*, vol. 16, p. 25, 2004.

[78] F. Sha and L. K. Saul, "Large margin hidden markov models for automatic speech recognition," in *Advances in neural information processing systems*, 2006, pp. 1249–1256.

[79] J. G. Fiscus, "A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (rover)," in *Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on*, Dec 1997, pp. 347–354.

[80] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140. [Online]. Available: http://dx.doi.org/10.1007/BF00058655

[81] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[82] G. Mesnil, Y. Dauphin, K. Yao, Y. Bengio, L. Deng, D. Hakkani-Tur, X. He, L. Heck, G. Tur, D. Yu *et al.*, "Using recurrent neural networks for slot filling in spoken language understanding," *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, vol. 23, no. 3, pp. 530–539, 2015.

[83] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr, "Conditional random fields as recurrent neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1529–1537.

[84] J. Hammerton, "Named entity recognition with long short-term memory," in *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*. Association for Computational Linguistics, 2003, pp. 172–175.

[85] J. P. Chiu and E. Nichols, "Named entity recognition with bidirectional lstm-cnns," *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 357–370, 2016.

[86] Z. Huang, W. Xu, and K. Yu, "Bidirectional LSTM-CRF Models for Sequence Tagging," *ArXiv e-prints*, Aug. 2015.

[87] X. Ma and E. Hovy, "End-to-end sequence labeling via bi-directional lstm-cnns-crf," *arXiv preprint arXiv:1603.01354*, 2016.

[88] Y. Shi, K. Yao, H. Chen, Y.-C. Pan, M.-Y. Hwang, and B. Peng, "Contextual spoken language understanding using recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5271–5275.

[89] B. Liu and I. Lane, "Attention-based recurrent neural network models for joint intent detection and slot filling," *arXiv preprint arXiv:1609.01454*, 2016.

[90] D. Hakkani-Tür, G. Tur, A. Celikyilmaz, Y.-N. Chen, J. Gao, L. Deng, and Y.-Y. Wang, "Multi-domain joint semantic frame parsing using bi-directional rnn-lstm," in *Proceedings of The 17th Annual Meeting of the International Speech Communication Association*, 2016.

[91] X. Zhang and H. Wang, "A joint model of intent determination and slot filling for spoken language understanding." IJCAI, 2016.

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

[93] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[94] B. Liu and I. Lane, "Recurrent neural network structured output prediction for spoken language understanding," in *Proc. NIPS Workshop on Machine Learning for Spoken Language Understanding and Interactions*, 2015.

[95] D. Guo, G. Tur, W.-t. Yih, and G. Zweig, "Joint semantic utterance classification and slot filling with recursive neural networks," in *Spoken Language Technology Workshop (SLT), 2014 IEEE*. IEEE, 2014, pp. 554–559.

[96] K. Yao, B. Peng, Y. Zhang, D. Yu, G. Zweig, and Y. Shi, "Spoken language understanding using long short-term memory neural networks," in *Spoken Language Technology Workshop (SLT), 2014 IEEE*. IEEE, 2014, pp. 189–194.

[97] M. Steedman, M. Osborne, A. Sarkar, S. Clark, R. Hwa, J. Hockenmaier, P. Ruhlen, S. Baker, and J. Crim, "Bootstrapping statistical parsers from small datasets," in *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1*. Association for Computational Linguistics, 2003, pp. 331–338.

[98] C. Rosenberg, M. Hebert, and H. Schneiderman, "Semi-supervised self-training of object detection models," 2005.

[99] 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*. Association for Computational Linguistics, 2006, pp. 152–159.

[100] R. Reichart and A. Rappoport, "Self-training for enhancement and domain adaptation of statistical parsers trained on small datasets," in *ACL*, vol. 7, 2007, pp. 616–623.

[101] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, "Semi-supervised learning with deep generative models," in *Advances in Neural Information Processing Systems*, 2014, pp. 3581–3589.

[102] A. Celikyilmaz and D. Hakkani-Tur, "Investigation of ensemble models for sequence learning," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5381–5385.

[103] O. Vinyals, L. u. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, "Grammar as a foreign language," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2773–2781. [Online]. Available: http://papers.nips.cc/paper/5635-grammar-as-a-foreign-language.pdf