NAMED ENTITY RECOGNITION IN TURKISH USING DEEP LEARNING MODELS AND JOINT LEARNING

by

Arda Akdemir

B.S., Computer Engineering, Boğaziçi University, 2017

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering,
Boğaziçi University
2018

NAMED ENTITY RECOGNITION IN TURKISH USING DEEP LEARNING MODELS AND JOINT LEARNING

APPROVED BY:

Prof. Tunga Güngör (Thesis Supervisor)	.....
Assoc. Prof. Gülşen Cebiroğlu Eryiğit	.....
Assoc. Prof. Arzucan Özgür	.....

DATE OF APPROVAL: 14.12.2018

ACKNOWLEDGEMENTS

First of all, I would like to express my deepest gratitude to my supervisor Prof. Tunga Güngör for his incredible support and understanding during the course of my Master's Thesis. He has always been very supportive and considerate during this process, and without his extra efforts this Thesis would not be complete.

I am grateful to my Thesis committee members; Assoc. Prof. Gülşen Cebiroğlu Eryiğit and Assoc. Prof. Arzucan Özgür for sparing their time and for their invaluable comments regarding my Thesis. I believe that their suggestions will help me to improve a lot on this study.

I also would like to take this opportunity to thank many people who have helped and inspired me during my Master's studies. I am very lucky to have met Asst. Prof. Ümit Işlak during my senior year, who has been a great teacher and an even greater inspiration for the past three years. I am privileged to have experienced his way of guidance and teaching which will a big source of inspiration for me in the years to come. I am very thankful to my supervisor at Koc University, Ali Hürriyetoğlu for his great support, and for sharing his insights and experiences with us.

I owe a special thanks to Semiha Kevser Bali for her guidance and patience during the writing of this Thesis.

Finally I would like to thank my family; my parents, Canan and Zafer, my brother, Mert and my aunts for their love and support. I thank my first and best teacher in life, my mother for her diligence to prepare us for life. I thank my father for being the best role model and for guiding us through his actions.

ABSTRACT

NAMED ENTITY RECOGNITION IN TURKISH USING DEEP LEARNING MODELS AND JOINT LEARNING

Named Entity Recognition (NER) is the task of detecting and categorizing the entities in a given text. It is an important task in Natural Language Processing (NLP) and forms the basis of many NLP systems. Previous work on NER that make use of statistical models can be categorized into two main categories: feature-based and embedding-based. Earlier work on NER made frequent use of manually crafted features. In order to use manually crafted features we either automatically annotate the dataset for the given features using third party software or manually annotate the dataset, both of which require additional work. Recent work make use of BiLSTM based neural networks and represent words with embeddings. This relieves systems from relying on manually created feature sets. In this work, we started with analyzing the performance of the feature based systems. In this phase, we reimplemented a previous work and improved the performance by making use of the dependency parsing features. Following these results, we implemented a novel method that makes use of both dependency parsing features and embeddings. We propose a novel BiLSTM CRF based neural model that makes use of the dependency parsing feature to learn both tasks jointly in a unique way. Our model jointly learns both dependency parsing and named entity recognition using separate datasets for each task. The model does not require the named entity recognition dataset to be annotated for the dependency parsing task. Our results show that performance increases when we use a joint learning model instead of annotating the named entity recognition dataset automatically.

ÖZET

TÜRKÇE VARLIK İSİMLERİNİN TANINMASI İÇİN DERİN ÖĞRENME VE BİRLİKTE ÖĞRENME

Varlık İsimlerinin Tanınması, verilen bir döküman içindeki özel isimlerin belirlenip, doğru kategorilere ayrılmasını amaçlayan bir Doğal Dil İşleme görevidir. Bu alandaki erken çalışmaların büyük bölümü el ile seçilmiş özelliklerin istatistik bazlı sistemler tarafından analizine dayanmaktadır. Yakın zamandaki bu alandaki en başarılı sonuçlar ise kelimelerin vektörel olarak temsil edilmesinden faydalanan yapay sinir ağı bazlı sistemler tarafından elde edilmiştir. Birden fazla görevi birlikte öğrenen, birlikte öğrenme sistemleri, genelde verisetlerinin iki görev için de etiketlenmiş olmasını gerektirir. Bu çalışmada, öncelikle Türkçe için yapılmış daha önceki özellik bazlı çalışmaları detaylı bir şekilde inceleyip, farklı özellikler kullanarak sonuçları iyileştiriyoruz. Bu aşamada bağlılık ayrıştırma ile ilgili özelliklerin varlık isimlerinin tanınmasında performansı iyileştirdiğini gösteriyoruz. Sonraki bölümde daha önce denenmemiş bir model kullanarak bağlılık ayrıştırma ve varlık isimlerinin tanınması görevlerinin beraber öğrenilmesini gösteriyoruz. Modelimiz benzer birlikte öğrenme sistemlerinden farklı olarak iki görev için de ayrı verisetlerinden faydalanıyor. Elde ettiğimiz sonuçlar birlikte öğrenmenin ayrı verisetleri kullanarak yapılmasının, aynı verisetinin otomatik olarak etiketlenmesine kıyasla daha iyi performans verdiğini gösteriyor.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . iii
ABSTRACT . . . iv
ÖZET . . . v
LIST OF FIGURES . . . viii
LIST OF TABLES . . . x
LIST OF SYMBOLS . . . xii
LIST OF ACRONYMS/ABBREVIATIONS . . . xiii
1. INTRODUCTION . . . 1
1.1. Motivation and Objectives . . . 6
1.2. Contributions of this Thesis . . . 7
2. BACKGROUND THEORY . . . 9
2.1. Early work . . . 9
2.2. BiLSTM Based Related Work . . . 12
2.2.1. LSTM-CRF . . . 13
2.2.2. Tag Scheme . . . 13
2.2.3. Transition-Based Chunking . . . 13
2.2.4. Word embeddings . . . 14
2.3. Related Work on Turkish . . . 14
2.4. Related Work on Joint Learning . . . 18
2.4.1. jPTDP Model . . . 20
2.4.2. jPTDP Dependency Parsing Architecture . . . 22
3. DATASETS . . . 25
3.1. NER Dataset for Joint Learning . . . 26
3.2. NER Dataset for Initial Models . . . 29
3.3. Dependency Dataset . . . 30
3.3.1. CoNLL-U Format . . . 31
3.3.2. IMST-UD Dataset . . . 32
4. METHODOLOGY . . . 34
4.1. CRF Model . . . 34

4.2. Models using Wapiti toolkit . . . 37
4.2.1. Training models . . . 37
4.2.2. Data Format . . . 38
4.2.3. Modes . . . 39
4.3. Joint Learning Model . . . 41
4.3.1. General Information . . . 42
4.3.2. DyNet Introduction . . . 42
4.3.3. Input Data . . . 45
4.3.4. Output Data . . . 46
4.3.5. Work flow . . . 48
4.3.6. Architecture . . . 50
4.3.7. Named entity recognition module . . . 57
4.3.8. Decoding Algorithms . . . 63
5. EXPERIMENTS AND RESULTS . . . 65
5.1. Re-implementation Phase . . . 65
5.1.1. Features Used . . . 66
5.1.2. Results . . . 69
5.2. Exploration Phase . . . 70
5.2.1. Results . . . 72
5.2.2. Results for Extensive Experiments . . . 74
5.3. Joint Learner Phase . . . 76
5.3.1. Results for NER only model . . . 77
5.3.2. Results for joint learner on a single dataset . . . 78
5.3.3. Results for joint learner on separate datasets . . . 80
5.3.4. Experiments with different parameter configurations . . . 81
5.3.5. Final experiments on the test set . . . 83
6. CONCLUSION AND FUTURE WORK . . . 87
REFERENCES . . . 88

LIST OF FIGURES

Figure 2.1. Architecture of the jPTDP model for jointly learning POS tagging and Dependency Parsing 24

Figure 3.1. An example sentence from the Turkish NER dataset in CoNLL-U format. Misc field is used for Morphological analyses and NER label. 29

Figure 3.2. NER dataset used in initial phases. 30

Figure 3.3. Example sentence from IMST-UD Treebank. 33

Figure 4.1. Example for the template file for the features of the CRF++ tool. 36

Figure 4.2. Illustration of the relationship between various machine learning models. 38

Figure 4.3. Dynamics of the VanillaLSTM Builder 44

Figure 4.4. LSTM cell illustration. 45

Figure 4.5. Vocabulary Generation Algorithm 47

Figure 4.6. Entry Object in the joint learner model. 48

Figure 4.7. Work flow of the joint learner model. 50

Figure 4.8. MLP for dependency arcs. 54

Figure 4.9. Dependency parser of the joint learning model. 55

Figure 4.10. MLP for producing the label probabilities for the dependency arcs. 56

Figure 4.11. Named entity recognition component of the joint learning model. . 58

Figure 4.12. Score matrix for NER. 61

Figure 4.13. Complete architecture for the joint learner model. 62

Figure 4.14. Viterbi Algorithm 64

Figure 5.1. Results for the NER only model in each epoch. 78

Figure 5.2. Results for the dependency parsing task of the joint model on a single dataset annotated jointly. 79

Figure 5.3. Results for the NER task of the joint model on a single dataset annotated jointly. 80

Figure 5.4. Dependency parsing scores for joint learner on separate datasets on the development set. 82

Figure 5.5. NER scores for joint learner on separate datasets on the development set. 82

LIST OF TABLES

Table 3.1. Number of sentences in the Turkish NER dataset. 28

Table 3.2. Number of annotated tokens in the Turkish NER dataset. 28

Table 3.3. Number of annotated entities in the Turkish NER dataset. 28

Table 5.1. CRF++ Results 70

Table 5.2. Early Results with Wapiti 74

Table 5.3. Exploration of combinations of all training models and optimization algorithms. 75

Table 5.4. Final result with 'Wapiti' toolkit. 76

Table 5.5. Best results for the joint learning model on a single dataset. 79

Table 5.6. Parameters of the joint learning model together with the default values. 81

Table 5.7. Results for the joint learning model with different parameter combinations. 83

Table 5.8. Results for the joint learning model using separate datasets on the test set. 84

Table 5.9. Results for the joint learning model using a single dataset on the test set. 84

Table 5.10. Results for the NER only model on the test set for each NE type. . 85

LIST OF SYMBOLS

a_{ij}	Matrix element on i^{th} row and j^{th} column
$\mathbf{e}_w$	Vector representation of a word w
σ	Sigmoid activation function

LIST OF ACRONYMS/ABBREVIATIONS

BiLSTM	Bidirectional Long Short Term Memory
CRF	Conditional Random Fields
LAS	Labeled Attachment Score
LSTM	Long Short Term Memory
MAXENT	Maximum Entropy
MEMM	Maximum Entropy Markov Models
MUC	Message Understanding Conference
NER	Named Entity Recognition
NLP	Natural Language Processing
POS	Part-of-Speech
UAS	Unlabeled Attachment Score
VAT	Varlık İsmi Tanıma

1. INTRODUCTION

Named Entity Recognition (NER) is a sub-task of Natural Language Processing. Natural Language Processing (NLP) refers to the research area in Computer Engineering that aims to solve problems related to natural languages and Artificial Intelligence (AI). It is a general name and contains many sub-fields such as Machine Translation, Language Generation and Text Summarization.

One of the major motivations of NLP research is to create computer programs that can understand, generate and represent a natural language text without losing any information. As the task is so big and complex, researchers pick and tackle sub-tasks or simplified versions of the tasks e.g. restricted domain translation. NLP research has seen tremendous improvements especially in the last decades. Yet researchers still have to settle for simpler representations that extract important information rather than representing the whole meaning and content in a designated data structure [1]. These simpler representations are often categorized into syntactic and semantic levels. Syntactic representation of a given text gives us important structural information which is useful for understanding meaning and content.

Syntactic level information is also important as it is often used as supplementary material for creating higher level semantic representations. Using low level information as additional input to systems tackling high-level tasks is a frequently used method in NLP research. POS tagging is used [2–6] for different high level syntactic tasks. Chunking is used [7] to help improve dependency parser accuracy. Syntactic parsers are used to leverage performance of natural language inference [3] and machine translation systems [8], which are both semantic level tasks.

Semantic level information often contains higher level information about the meaning and the discourse of the text. As representing a given text semantically is a highly demanding task researchers settle for simpler semantic representations and gradually increase the complexity of the programs. These semantic level tasks aim to

describe a limited part of the whole meaning and content of a given text.

Named Entity Recognition is one of these semantic level tasks which focuses on detecting and properly categorizing the named entities. Named entities refer to words or words phrases that consistently designates a specific referent. Named entities are closely related to the 'rigid designators' [9]. Named Entity Recognition is important, because named entities often contain valuable information about a given text. Properly recognizing the named entities in a given document is an important step towards understanding and processing texts especially in the domain of formal documents such as News reports. Named Entity Recognition form the basis of many Information Retrieval systems, and many IR systems include components that detect and categorize named entities.

Named Entity Recognition is first defined as a formal task in the sixth Message Understanding Conference (MUC) in 1995. Chincor *et al.* [10] give the definition of the named entity recognition task. The Named Entity task, defined by MUC, consists of three sub-tasks for extracting the following items:

- entity names: ENAMEX
- temporal expression: TIMEX
- number expression: NUMEX

These three sub-tasks are selected because these are considered to be the most critical expressions for information retrieval. According to their definition, the annotation should be done on the unique identifiers of named entities, times and quantities. Each of these sub-tasks are further divided into sub-categories. Named entities consist of organization, location and person types. Time expressions consist of dates and times. Finally quantity expressions consist of monetary values and percentages. Even though these definitions were subject to change over the last decades by other researchers, these categories and entity sub-tasks still remain to be the most frequently used ones. Recent studies also focus on fine-grained NER or further categorization of the above mentioned categories. 'Location' type is often divided into several categories such as

country and city names. The primary aim of this approach is to increase the level of detail of the information extracted from a given text. Yet these categorizations often further complicate the NER task which has its own challenges even with the first definition given in 1995. Also, these categorization attempts have never reached to a consensus among researchers world-wide so there are many varying annotated datasets with different subcategories. On the other hand, the annotation guideline defined by MUC has reached to an almost universal standard among researchers, and many datasets are available for many languages which are annotated in almost the same way or in a very similar way defined by MUC. The datasets used in our work are annotated only for the entity names. The annotated entity types are exactly the same with the MUC definition: Person, Location and Organization.

The task is defined in two steps: Identifying all the instances of the three types of expressions and properly categorizing them into the subcategories. In the first step the aim is to identify each type of expression regardless of the subcategory information. Identification is an important step which enables information extraction over large texts. The second step can be considered as a more refined task where the aim is to use any kind of contextual or word-level, sub-word level information to distinguish between subcategories of entities. Various studies [11–13] show that this step is a major challenge especially for the 'entity names' sub-task. Entity names contain many ambiguities, and therefore are difficult to distinguish both from common nouns and from other entity names. For example, many languages have city names as person names which could cause NER systems to fail to detect the proper category of named entities. The other sub-tasks are often more straightforward as both temporal expressions and number expressions often follow a rather strict guideline and therefore are easier to categorize. The results attained also support this observation as for many languages the performances of the entity recognition systems on these sub-tasks reached to the human level [14]. Therefore many recent studies focus more on the detection and proper categorization of the entity names, as in the case for the work done in this study.

Another important initiative was done for the NER task in 2003 with the CoNLL-2003 Shared Task [15]. The task focuses on creating language-independent named entity recognition systems. The task only focuses on the ENAMEX type and divide the entity names into four categories. Three of the categories are same with the MUC entity categories. In addition to them a fourth entity type called Miscellaneous as 'MISC' is introduced. This entity type contains very diverse type of named entities. Any word or word sequence that does not belong to the traditional MUC definitions of the subcategories which refer to unique entities are annotated as MISC. Examples of this category include adjectives, like Japanese, and event names. Apart from this difference, the entity definitions and the category types determined by MUC-6 are preserved. The Shared Task is organized for two languages: German and English. The datasets are annotated in the IOB scheme which prevents entity boundary ambiguities which arise when the different entities of the same category follow one another.

In the IOB tagging scheme, all tokens of an entity is tagged with 'I' prefix unless the first word is not an immediate predecessor of another entity token of the same type. In that case the first token is annotated with the 'B' prefix which denotes that a new entity sequence has started. The following entity tokens, if any, are also tagged with the 'I' prefix as usual. The 'O' character in the IOB scheme is used to for tagging the non-entity tokens. This shared task received a great deal of attention from researchers of the NLP community and many systems are created for tackling this task. It became a convention to report the performance of a NER system on the English dataset of the CoNLL-2003 Shared Task. As annotating datasets requires extensive work and consistency is difficult to attain researchers tend to use the same datasets once the datasets reach a certain popularity and confirmed to be valid and consistent by researchers.

The detection of entity names has its own unique ambiguities and challenges. Even in its simplest version, the NER task only for the three entity name categories includes many ambiguities. These ambiguities challenge both the annotators of the datasets and the researchers who create systems to detect these entities. Thus forming detailed annotation guidelines is another important task which sets the basis for

creating high quality datasets.

Named Entity Recognition as mentioned above is a research area with various applications. For this reason, many researchers worked on creating systems with high performances. Various shared tasks [10, 15] and available datasets especially for the frequently used languages such as English helped researchers to test and compare their systems in a consistent and fast way. Thus the results of these systems are impressive especially for the frequently used datasets and languages. Even before the emergence of the deep learning based systems in the field, the results of the systems using various machine learning methods were impressive [16]. Systems that make use of deep learning methods together with recent hardware technologies have even more impressive results on the well-known and well-structured formal datasets.

Generalization of these systems became important especially with the rise in the value of NER in informal texts such as text from social media [17–22]. Even though the generalization of coarse-grained NER systems is an important issue and the impressive results are shown to drop significantly when the same systems are tested on novel and different datasets [23, 24], many researchers tend to focus on different and more challenging tasks related to named entity recognition as well. One of the trending topics about NER is 'fine-grained NER' where each entity type is further divided into various categories and detecting them is a more challenging task [25] both because of the definition of the task itself and the lack of available datasets specific for fine-grained NER. Lack of available datasets forces researchers to use methods that automatically annotate datasets [26] which limit the performance of systems making use of such datasets. Some of the earlier works focus on the sub-categories of a single entity type [27], whereas recent studies mainly focus on the sub-categories of several entity types at once [28, 29].

Another important research area is the joint learning of different NLP tasks including the NER task. Joint learning or multi-task learning is a type of learning approach where a single system is used for tackling multiple tasks. Caruna *et al.* [30] attempt to learn multiple hard tasks at once rather than learning each of them sep-

arately. The main idea is that the information obtained during the training for each task is a useful inductive bias for the other tasks. In the recent joint learning systems, the prediction or output produced for one task is generally fed to the system and used as an additional information to make a prediction for the other task. Such systems have the advantage of not relying on to a separate tool to obtain the information for the other task. Another advantage of joint learning based systems is that both parts of the system are trained simultaneously so that the loss obtained for one task is used for training the parts of the system related to the other task. Hence joint learning usually performs better when used for related or similar tasks. However, previous work on joint learning of different tasks requires datasets to be annotated for both tasks [31]. To overcome this joint annotation problem, Collobert *et al.* [32] implemented a neural network based system for various NLP tasks by annotating a dataset using third-party tools. The dataset is annotated using the Stanford Named Entity Recognizer [33] tool for the NER labels. However, automatically annotating a dataset is prone to errors and may cause the errors made during annotation to propagate to the training phase.

The availability of datasets annotated for both tasks is a major problem even for frequently used languages such as English. For less frequently used languages, the joint labeling problem is even more evident and limits the research in this area. Turkish is one of those languages where NLP related research is limited due to the availability of annotated large high-quality datasets even for single tasks. Thus joint labeling problem puts a limit to the research in this area for the Turkish language and many other less frequently used languages.

1.1. Motivation and Objectives

Through the observations mentioned in the previous section, our main motivation in this Thesis is to create a NER system for Turkish language that also attempts joint learning of another NLP task. In order to find a suitable task for the joint learning along with NER, we start by analyzing the performance of available sequence labeling tools on Turkish language and make improvements over the previous work by exploiting new features and feature combinations. During this phase, we focus on re-implementing a

previous work on Turkish and then improving the performance by using dependency parsing information together with new feature combinations.

Dependency parsing is an important research topic in NLP. It is demonstrated to be highly useful for various NLP tasks. Dependency parsing is shown to be useful for relation extraction [34,35], semantic parsing [36,37], machine translation [38], question answering [39, 40] and named entity recognition [41, 42]. Our main motivation is to combine these early findings and our results from the previous phase with the recent related work on joint learning to create a named entity recognition system which makes use of and jointly trains a dependency parser. We propose a novel system for the joint learning of dependency parsing and named entity recognition that has not been implemented before for the Turkish language using a deep learning based model. We focus on obtaining relative performance improvements over the version of the named entity recognizer which does not make use of a dependency parser output. Through this method we also propose a novel way to solve the joint labeling problem mentioned in the previous section. Our final method works on different datasets for different tasks and does not require the dataset to be jointly annotated for both tasks. Our results show that the system trained using separate datasets outperforms the version of the model trained using a jointly annotated dataset using a third party software for the NER task.

1.2. Contributions of this Thesis

In this section we explain the main contributions of this Thesis. Since a Master's Thesis is a document containing all the research done in a two-year span it may be difficult to track down the main contributions of a Master's Thesis. For this reason, we list the main contributions of this Thesis below. Our main contributions are related with the final proposed joint learning model for dependency parsing and named entity recognition. The main contributions are as follows:

- The joint model proposed in this Thesis is the first deep learning based model that jointly learns dependency parsing and named entity recognition for Turkish.

We also have not seen a similar joint learning scheme applied for these two tasks for other languages as well.

- We have used different datasets for each task. Related works on joint learning, make use of a single dataset annotated for both tasks [1]. Yet only for frequently used languages and frequently tackled tasks such datasets are available. Our model attempts to solve this joint annotation problem by using different datasets for each task. Our main hypothesis is that learnt features about a language for a task can be useful for a different task even when the datasets differ.

These contributions are explained in detail in the subsequent sections.

2. BACKGROUND THEORY

2.1. Early work

Bikel *et al.* [43] published the paper in 1997 which describes the system called 'Nymble'. This is one of the earliest papers that uses statistical tools and supervised learning in NER. They use very standard set of features for each word. By using an if else structure they check for the existence of a feature in a given word such as 'isCapitalized' which checks whether a given token is capitalized or not. Each word has a single feature which can be considered as its category. This feature together with the word itself is used to find transition probabilities of a Hidden Markov Model. Since the data is sparse and their corpus is limited, they used a back-off scheme whenever the transition probability is 0 because there is no previous data to handle the unknown words. HMM has 8 nodes representing 8 NE types and they calculate the transition probabilities between these nodes.

McCallum *et al.* [44] make use of a CRF based model with induced features. CRF's are an extension to the HMM model and use transition probabilities with features and the words for sequential labeling. In CRF models, the task is to learn the weights of the feature functions. Feature functions are often created automatically using the combinations of the given features or just make use of atomic feature functions. Giving too many features with a large window size causes generating too many feature functions which slows down the learning process and may result in over-fitting. To overcome this issue they start with 0 features and add the most useful feature at each step and repeat this process until the net gain is insignificant. When the net gain is insignificant over a development set, this signals a point at which increasing the number of features will likely result in over-fitting the training dataset.

They also suggest finding patterns in the HTML format using web documents to detect NE's. Their assumption is that HTML format around an NE type will be similar in most cases because HTML is structured and follows a specific format.

Chieu *et al.* [45] use Maximum Entropy Models with global features for NER task. The use of global features such as previous occurrence in the document is shown to improve the performance of the system. Many features are used in this system and all of them are explained in detail which makes this paper a valuable contribution for feature related research on NER. Maximum entropy estimates probabilities by using features. The number of features are unlimited and any feature function can be selected freely. Compared to CRF's ME does not depend on the sequential nature of the input. But they also take into account the transition probabilities between classes which prevent inadmissible transitions such as going from $person_{begin}$ followed by $location_{unique}$ since it is not a possible transition. Yet, these transition probabilities are 1 if the transition is admissible and 0 otherwise, so learning is not done on these parameters. Probability of a NE tag conditional to the current word and the history is shown below:

$$p(o|h) = \frac{1}{Z(h)} \prod_{j=1}^{k} \alpha_j^{f_j(h,o)}$$

where f_j represents the feature functions and α_j are their associated weights. The weights are calculated by maximum likelihood estimation which tries to minimize the difference between the empirical and expected number of occurrences of each feature.

Overall process looks as follows:

$$P(c_1, c_2, ...c_n|s, D) = \prod_{i=1}^{n} P(c_i|s, D) * P(c_i|c_{i-1})$$

where the transition probabilities are two valued functions explained as above. In the above equation, c_i, s and D refer to named entity class prediction for i^{th} word, sentence and document respectively. Dynamic programming is used to find the most likely tag sequence.

Altun *et al.* [46] combine HMM's with SVM's for NER. This paper uses a different version of SVM that makes use of interdependency of tag predictions which are typical of HMM based models. In their model, they take into account the tag information of

the previous tokens in the sequence. In a way they have included in the features of the SVM a new type of feature family which checks the previous token's tag information. However, the goal is not to create a generative model which counts the previous occurrence but to find discriminant functions to separate data. Label-label interaction only considers the nearest neighbors as in the case with the HMM based model.

The features are appended for a given token and output label pair. In a hidden markov perceptron model, learning the values of the dual function are incremented or decremented based on the results. This is to train the model to achieve the optimal results. The aim is to find the $a_i(\mathbf{y})$ values given the training data where $\mathbf{y}$ corresponds to the sentence vector. In the end, the system returns the y value which maximizes the F function which gets a higher score if the dot product of the two feature vectors of two tokens are similar.

Zhou *et al.* [47] also use HMM's for tagging chunks. Their approach is similar to the previous work described on HMM in which the system learns the transition probabilities between states, representing NE tag types.

The major difference of this paper is that they use a different version of conditional probabilities. Rather than using:

$$P(T^n|G^n) = \frac{P(T^n)P(G^n|T^n)}{P(G^n)}$$

formalism, they use the following:

$$logP(T^n|G^n) = logP(T^n) - \sum_{i=1}^{n} logP(t_i) + \sum_{i=1}^{n} logP(t_i|G^n)$$

This equation helps using not only the word itself but the neighboring words and their features for making a tag prediction. Yet this approach suffers from sparsity for which they apply a back-off algorithm. The back-off scheme is very straightforward.

The works described in this section was the early research done on this area which used supervised learning and statistical sequence learners. The main idea behind all of these systems is that, sequence transitions are Markovian processes. Both HMM, CRF and ME use the same idea. Next section describes in detail one of the top performing systems on NER for the CoNLL 2003 English dataset which is one of the most frequently used datasets. The NER model implemented in the final phase of this Thesis is similar to the neural network model that uses BiLSTM with a CRF layer. That is why a separate section is devoted for explaining this work in detail.

2.2. BiLSTM Based Related Work

Lample *et al.* [48] created one of the top performing systems for NER on CoNLL 2003 English dataset.

This work describes two methods for the NER task. Both make use of Neural Architectures. They do not use explicit name lists and gazetteers, and they have achieved state-of-the-art results. They also refrain from using language specific hand crafted rules which is considered to be crucial for portability of such systems.

They introduce two models :

- Bidirectional LSTM with an additional CRF layer for label sequence optimization.
- Transition based parsing with stack LSTM.

The intuition behind these models is that NEs are in general multiple token sequences. Second idea is about the orthographic and positional distributions of NEs. For this, they use character based word embeddings. For the orthographic distributions, the two directional LSTM takes into account the context around the word which represents the orthographic information.

2.2.1. LSTM-CRF

LSTM-CRF model makes use of $A_{y_i,y_{i+1}}$ where A is the probability of a transition from tag y_i to y_{i+1} in addition to the probability of a word having a certain NE tag. The probability P_{i,y_i} of i^{th} word having a tag y is learned in the LSTM layers. The transition probabilities make the model somewhat CRF-like. The formulations of the neural architecture are beyond the scope of this study. Adding the transition probabilities and the tag probabilities gives a score of the likelihood of the tag sequence for a given word sequence. During training, the neural net is trained to minimize the loss for this score for each sequence.

BiLSTM takes as input d-dimensional word embeddings, and returns two representations for each word: forward and backward. These are then concatenated, and together they represent the word with which the tag likelihood is calculated.

2.2.2. Tag Scheme

IOBES tagging scheme is used in this work. IOBES is used to differentiate between single token NEs, and differentiate internal and final words of a multiword NE sequence. Single token NEs are tagged with the 'S' prefix. Final words of a multiword sequence is tagged with the 'E' prefix. They did not observed significant improvement over using the IOB representation [48].

2.2.3. Transition-Based Chunking

Rather than treating the input sequentially, stack based approach is used for analyzing stacks of input simultaneously. The program has 3 parts : Stack, Out and Buffer. Initially all words are in a buffer and one-by-one they are either sent to the stack or directly sent to the Out part. If there are words in the Stack, they can be sent to Out or additional words can be stacked from Buffer. All these actions are learned during training. The LSTM takes into account the current context in all 3 parts of the model, the words themselves and previous decisions for making a decision at each step. These

decisions are learned during training with annotated corpus. Details and figures in the paper nicely explain the procedure. Obviously since the NEs are detected in chunks, this approach does not use any tagging scheme. Again the words are represented as word embeddings which are character based models explained in the following section.

2.2.4. Word embeddings

Lample *et al.* [48] make use of character level embeddings obtained by using BiLSTMs together with a word2vec variation of word embeddings. For a given word, characters are given in forward and backward manner, and two representations are learned by LSTMs. Then these are concatenated with the pretrained word representations.

These word embeddings are vectors that represent the words and fed into the LSTMs for predicting tags. A similar embedding mechanism is used in the joint learner model implemented in this Thesis. Yet pretrained embeddings are not used and each embedding is initialized randomly.

2.3. Related Work on Turkish

We continued the literature survey with focusing on the previous work done for the Turkish language. We grasped a general idea about the research done for other languages. Our aim in this phase of the literture survey was to survey through the work done for Turkish language and to find the points we can improve by using the methods that are used for other languages and not applied to Turkish.

Demir *et al.* [49] use word embeddings together with supervised learning. They used the work of Mikolov *et al.* [50] for word embeddings which does not use non-linear hidden layers. This enables having very short training times. Skip-gram model is used which captures the information neighboring the word as well as the word itself.

In an unsupervised manner, the word representations are learned by using a large unannotated data. Then an additional clustering is applied to the words and 200 clusters are formed. During the supervised stage, these cluster membership features are also used as additional features.

In the supervised stage, the NER model is trained using annotated data. The model is a regularized averaged perceptron. Mainly the system calculates the probabilities of having a certain NE tag given the previous tags and neighboring word information. One of the advantages of their approach is that the model only makes use of language independent features. This enables to apply this model easily to other languages.

The only drawback of this approach is that they make use of hand crafted rules which are given to the system manually.

Celikkaya *et al.* [51] explains an application of NER systems on real data in Turkish. Rather than using formal texts which are easy to parse, the study attempts to show the difficulties that arise when NE systems are applied on real data. The F1-measures drop significantly which shows that by using the conventional methods we can not achieve meaningful results on real data. However most of the data that we would like to apply our NE systems on are in fact in the form of real data, meaning that they contain a lot of non regular word forms, typos and grammatical errors.

They proposed several solutions to the most common problems occurring in real data but the main aim of the paper is to show the difficulties rather than proposing clear solutions. This paper can be viewed as a reference when the aim is to work on real data because it shows the possible difficulties that frequently arise. Some of the difficulties mentioned is; NE's in their lower cased form which is ungrammatical for Turkish and not using the apostrophe when it is needed.

Seker *et al.* [52] make use of a morphological preprocessing unit before applying a CRF model on the same dataset we are planning to use. The morphological processing

consists of an analyzer step which gives the information required for morphological features. The disambiguator chooses the most probable way of analyzing a word based on its context. This preprocessing unit is prone to errors.

The study makes use of the frequently used features for Conditional Random Fields (CRF) based systems. Features extracted automatically by applying a morphological analyzer can be considered as a novelty yet these features are not gold labels so they may contain inconsistencies. Inflectional information is also used as features. A binary feature for proper noun indication in the morphological analysis is used to detect whether the analyzer tagged the word as a proper noun or not. The study uses manually created feature templates which can be either based on a single feature or a conjunction of various features. They have created 92 different templates and generated around 2M different functions to be used by the CRF based model. The main difficulty when using CRFs is the determination of these feature functions. In this study the feature functions are determined manually using heuristics. This part can be done automatically by using machine learning algorithms but it is a complicated process on its own and such automatically created features are often outperformed by manually picked feature sets since the human expertise in this area is quite high. Large gazetteers are used for learning entities and these gazetteers are publicly available for other researchers to use.

Eryiğit *et al.* [53] give an in-depth analysis of the effect of learning named entities under the class of multi-word expression on dependency parser performance. In this sense, their model is the first model that learns both named entities and the dependency parsing for Turkish.

Kucuk *et al.* [54] used a hybrid system using manually written rules together with statistical tools for this task. They have used a rule-based system which incorporates extensive look up tables for tagging the words matching with the gazetteers. Then the pattern bases for each NE type are applied on the previous tagged data to tag the words that match with the patterns in the bases. Then this rule-based system is extended into a hybrid design that uses rote learning.

In the hybrid approach, lexical resources and patterns are learned by using rote learning for each specific genre. These genres include news domain, financial domain, historical texts etc. Rote learning counts the number of occurrences, and for instance, for a word that occurred 10 times appears with a person tag in the news domain it is added to the person type entity list learned by rote learning for the news genre. Then a hybrid tagging scheme is applied to texts for which the genre is either known or not known. If the genre is known, the rote learned genre specific resources are applied to tag the entities in the learned list with a higher confidence score. Finally, the rule-based system tags the remaining entities based on the lexical resources or pattern bases.

Yeniterzi [55] used morphological features of words in a CRF based model. The novelty of this paper is that it uses a morphological model which treats every morphological sub-token of a given word as separate tokens. Therefore, after expanding the lexical forms of the words, the morphemes are treated separately in a sequential manner. This way the paper claims to take into account the morphological information in a more robust way.

Neural networks are also used for NER for Turkish [56, 57]. Gungor *et al.* [58] produced a neural network based model for Turkish language. They make use of morphological embeddings in addition to word and character embeddings for NER task. BiLSTM with CRF like tag transition structure is used for predicting the NE tags. BiLSTM uses forward and backward LSTM outputs for word representations. Then these representations are concatenated and used to calculate the score for each tag for this representation. By taking into account the transition probabilities as well the system calculates the tag sequence with the highest score.

Three types of embeddings are used: word, character and morphological. Word embeddings are loaded from pretrained vectors and never learned. Character embeddings are learned by incorporating BiLSTM's. Additional morphological embeddings are learned by morphologically analyzing the words. The analyzed result is turned into an embedding in a couple of different ways and the results are shown in the paper for each method. Best results are obtained when the morphological embedding is used

without the root word. Biggest advantages of this paper over previous approaches can be listed as follows:

- Not using any manually entered features. BiLSTM is trained only by using embeddings.
- Using forward and backward systems which captures information both from previous words and words that follow the current word.
- Using morphological embeddings that capture valuable information for morphologically rich languages. Representing morphological analysis of a word with embeddings is a more robust way than using the analysis directly.

2.4. Related Work on Joint Learning

Joint learning is an emerging research area in machine learning especially for NLP related tasks. Most NLP tasks contain valuable information which can speed up or improve the learning process of other NLP tasks. In many NLP tasks low-level information is used to improve performance of higher-level tasks. The low-level information gives additional information besides the surface form of the word and it is often useful. Systems making use of such low-level information mostly make use of third party software for annotation or implement a separate system and use both systems in a pipeline manner [3, 8]. A CRF-based system [59] is used for analyzing entities in three levels: coreference, named entity recognition and linking. The analysis for each level is used as additional information for other tasks.

Low-level tasks such as POS tagging require less abstraction than high-level tasks such as semantic tasks. Previous work on multi purpose NLP systems does not generally take this phenomenon into account and use the same architecture for each task as in [1]. Using the outermost layer output of a neural network for different tasks is a frequently used method in neural network models especially in the area of computer vision [60, 61]. Yet in most NLP related tasks the level of abstraction required for each task differs.

Hashimoto *et al.* [2] discuss the importance of building a single system that tackles multiple NLP tasks at once. Their Joint Many-Task (JMT) model performs similar to the state-of-the-art systems that specialized on each separate task. The main motivation of the model is that many of the state-of-the-art tools for different NLP tasks make use of similar neural architectures. Especially BiLSTM based models outperformed other models in almost all NLP tasks. Thus most state-of-the-art systems for different NLP tasks have a similar neural architecture and it is straightforward to combine them for jointly solving different tasks.

JMT uses outputs of different layers to be used for different tasks which is shown to perform well [62]. As the complexity of the task increases more neural layers are added to have a higher level of abstraction. This way first layers of the network is affected by the losses back propagated for more tasks but higher layer weights are not updated directly during the training of low level tasks. These layers are more specialized and can be considered as capturing the subtleties of more complex tasks whereas the low level layers give us the fundamental and low-level information.

Their model tackles five different NLP tasks which are put into the following three categories: word-level, syntactic-level and semantic-level. Word level tasks are POS tagging and Chunking which are predicted using the first and second layer outputs of the network respectively. These tasks require relatively less abstraction and therefore first two layer outputs are used. Dependency parsing is the syntactic level task which uses the output of third layer. The third layer receives as input the outputs of the previous tasks in addition to the word representations and the hidden states. The outputs of the previous tasks are the predicted labels which are converted into vector representations before being fed to the third layer.

The final two tasks, Semantic Relatedness and Textual Entailment are the semantic tasks which require considering the whole sentence at once. So a max pooling is applied over all word-level representation which is proven to be an effective method in text classification [63].

Detection and categorization of named entities are separated into different tasks and learnt jointly in [64] where extensive external features are used for improving the performance. Using the outputs produced for each task for the other task is shown to improve performance of both systems [65, 66].

Nguyen *et al.* [4] attempts jointly learning of Part-of-Speech (POS) tagging and Dependency Parsing using the CoNLL 2018 Shared Task dataset. Their model which is called joint POS Tagging and Dependency Parsing (jPTDP) will be explained in detail in the following section. As our final proposed model is similar to the model described in this work we devote a separate section explaining their model in detail. The details of the jPTDP model will be preceded by a short introduction to the CoNLL 2018 Shared Task, as the model is submitted to the shared task for dependency parsing.

CoNLL 2018 Shared Task [67], "Multilingual Parsing from Raw Text to Universal Dependencies" focuses on the Dependency Parsing for many typologically different languages and has three evaluation metrics. The participant systems are expected to predict the labeled dependencies for each word. The datasets used for all languages are publicly available and made possible by the Universal Dependencies initiative (UD). The datasets are in CoNLL-U format. The UD dataset for Turkish is explained in detail in the Datasets chapter which includes the details regarding the CoNLL-U format as well. The task includes datasets for more than 60 languages.

2.4.1. jPTDP Model

jPTDP makes use of a BiLSTM based neural network to jointly learn POS tagging and dependency parsing. Their model, which is an extension of the BIST graph-based dependency parser [5], also includes a BiLSTM based POS tagging component. The output of the POS tagging component is fed into the subsequent BiLSTM layers responsible for dependency parsing. POS tag of a word contains valuable low-level information which can be useful for high-level tasks such as dependency parsing. Thus many dependency parsers make use of POS tag feature of the words. Yet most of these parsers rely on third party POS taggers. So these parsers either require the dataset to

have gold label POS tags which is not available for many languages and require manual labor or they make use of POS taggers which make mistakes. These mistakes cause error propagation and remain unchanged throughout the training of the dependency parser. Since the POS tagger is only used in the beginning to automatically annotate the dependency parsing dataset, the errors made during annotation remain the same throughout the training phase of the dependency parser. Previous work on avoiding the use of a separate POS tagger tool [68–70] can not successfully outperform the version of the models that make use of a separate POS tagger tool. Yet using a separate tool puts an upper limit to the success of the dependency parser. Jointly learning both tasks is an attempt to solve both of these issues at once. Joint learner does not require gold labels for POS tagging and does not require a separate tool for automatically tagging the dataset for POS tags. Additionally joint learning enables improving the accuracy of the POS tagger during the training phase so that the error, which remains unchanged when the dataset is automatically annotated using a separate tool, gets smaller. Another important benefit of using joint learning is that the losses obtained for one task is used for improving the accuracy of the part of the architecture responsible for the lower level task which in this particular model is the POS tagging part. So the losses obtained for dependency parsing is used for updating the POS tagger which is shown to improve the accuracy of the final tagger.

The architecture and the details of the jPTDP model is given in the following section. The final model used for this thesis makes use of the same well-known BIST dependency parsing architecture. The main difference from the dependency parser of jPTDP is that our architecture does not make use of POS tag prediction. So the input to the parser does not contain the additional vector representation of the POS tag, but the architecture and the prediction mechanism is similar for the dependency parser. The main motivation is to show that the dependency parsing prediction improves the accuracy of the NER system rather than improving the state-of-the-art dependency parser systems. So implementing a better dependency parser by changing the architecture or the prediction mechanism is not the primary aim of this study.

2.4.2. jPTDP Dependency Parsing Architecture

This section is devoted for explaining the parsing component of the joint learner. The parsing component implemented during the work done in this Thesis is similar to this component except for the input vector fed into the parser. The overall structure of the model is given in Figure 2.1 which is taken from the work in [4]. In addition to the concatenation of the word and character level vector representations for each word which is called the word vector representation they also concatenate the vector representation of the predicted POS tag for each word. Let $\mathbf{e}_{w_i}^{(W)}$ and $\mathbf{e}_{w_i}^{(C)}$ represent the word and character embeddings for each w_i in a sentence respectively. So the i^{th} word of a sentence has the following word vector representation

$$\mathbf{e}_i = \mathbf{e}_{w_i}^{(W)} \circ \mathbf{e}_{w_i}^{(C)}$$

where $\mathbf{e}_{w_i}^{(W)}$ is initialized randomly and updated during training but can also be initialized with pre-trained word embeddings. Character embedding, $\mathbf{e}_{w_i}^{(C)}$, is obtained by using the character embedding of each character making up the word. If we let $w_i = c_1 c_2 ... c_k$ represent the i^{th} word with k characters where each character c_i is represented with a character embedding $\mathbf{c}_i$ then the character-level vector representation is learned by inputting the k character embeddings $\mathbf{c}_{1:k}$ to a BiLSTM called sequence BiLSTM (BiLSTM$_{seq}$) and concatenating the forward and backward outputs.

$$\mathbf{e}_{w_i}^{(C)} = \text{BiLSTM}_{seq}(\mathbf{c}_{1:k}) = \text{LSTM}(\mathbf{c}_{1:k}) \circ \text{LSTM}(\mathbf{c}_{k:1})$$

For each word w_i the predicted POS tag is represented with a vector embedding $\mathbf{e}_{p_i}^{(P)}$. The word vector representation is concatenated to this vector embedding so that for each word w_i we have a corresponding vector representation $\mathbf{x}_i$ where

$$\mathbf{x}_i = \mathbf{e}_{p_i}^{(P)} \circ \mathbf{e}_i$$

So a sentence with n words is represented with $\mathbf{x}_{1:n}$. These vectors which represent each word of a given sentence is fed into a BiLSTM layer which is named BiLSTM$_{dep}$, and the corresponding vector representations are called the latent feature vectors. Let $\mathbf{v}_i$ represent the latent feature vector of the i^{th} word where

$$\mathbf{v}_i = \text{BiLSTM}_{dep}(\mathbf{x}_{1:n}, i)$$

The latent feature vector for each word is constructed by taking into account the context of the word. Latent feature vectors are used to predict the head and the relation labels of dependency parsing for each word. An arc-factored parsing approach is used to decode the dependency arcs [71]. A multilayer perceptron (MLP) is used to calculate a score for each pair of words in a given sentence by concatenating several vector representations. One can think of the dependency tree as a directed graph where each arc (h, m) is a head-modifier arc. So the scores for (h, m) are not equivalent to (m, h) as the order denotes which word is the head and which word is the modifier. The score for the arc from the i^{th} to the j^{th} word is calculated using the latent feature vectors $\mathbf{v}_i$ and $\mathbf{v}_j$ as follows:

$$\text{score}_{\text{arc}}(i, j) = \mathbf{MLP}_{\text{arc}}(\mathbf{v}_i \circ \mathbf{v}_j \circ (\mathbf{v}_i * \mathbf{v}_j) \circ |\mathbf{v}_i - \mathbf{v}_j|)$$

where $(\mathbf{v}_i * \mathbf{v}_j)$ and $|\mathbf{v}_i - \mathbf{v}_j|$ are element-wise multiplication and absolute element-wise difference respectively. By using these scores for each pair of words, Eisner's decoding algorithm is used to find the highest scoring dependency parse tree. Eisner's decoding algorithm gives full coverage of the sentence so that no word is left without a head index except for the root word of the sentence which has the head index -1. By using these indices for each word another MLP called relation MLP is used to return scores for each predicted arc in the decoded parse tree. The number of output nodes of the relation MLP is the number of relation types. Highest scoring relation type for each arc is used during the prediction phase. Figure 2.1 shows the overall architecture of the above explained jPTDP model.

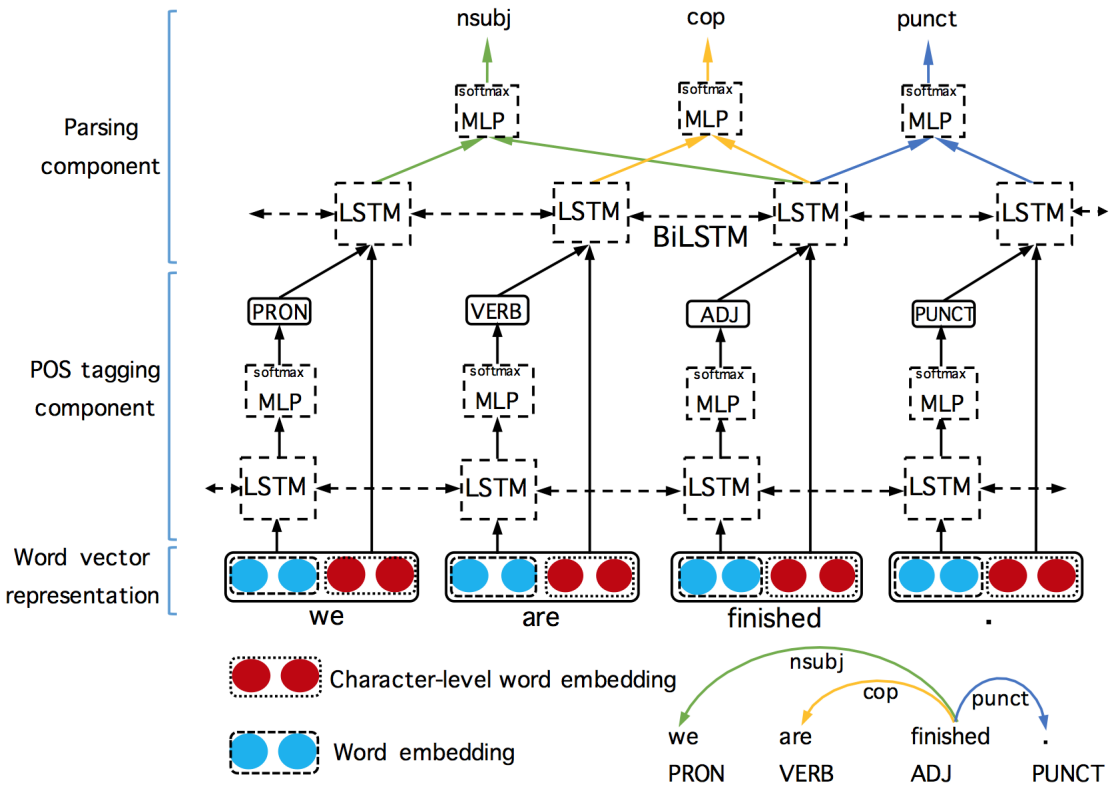

Figure 2.1. Architecture of the jPTDP model for jointly learning POS tagging and Dependency Parsing

3. DATASETS

The datasets used during the experiments are explained in detail in this chapter. Machine Learning based systems rely heavily on both annotated and unannotated datasets. These systems are mostly data driven so the quality and the size of the dataset affects the final performance of such systems especially in the case of NLP related tasks. Thus creating high-quality and large size corpora is an important task. Yet creating high-quality annotated corpora requires extensive manual labor and can be really costly. Many efforts have been put in for the last decades to create such corpora. The increase in the computing power and the emergence of deep learning methods with easy-to-use libraries and tools enabled fast implementation of systems with impressive results. Yet such systems are useful only if the datasets that they use for learning is available and has high-quality. In other words, the performance of such systems is bounded by the limitations enforced by the dataset itself. Limitations of datasets can be grouped in several categories and may differ in importance for different research fields.

Frequently occurring limitations for both annotated and unannotated corpora in NLP is their generalizability. Language itself is an ever growing phenomenon and it is simply infeasible and unsustainable to try to include all possible characteristics of even a single language in a corpus of finite size. Recent research shows that the frequently used datasets are not able to generalize to other domains and genres especially in the case of NER task. Augenstein *et al.* [23] give a detailed analysis of the generalizability of datasets for English language for the NER task. They report that the unseen entities and unseen features of those entities that occur in a test set cause significant performance drops. Even the majority of the performance scores of the state-of-the-art systems on frequently used test sets are made up of the previously seen entities. By analyzing the seen/unseen entity ratios in several datasets they show that the performance of systems using those datasets are highly correlated with this ratio. Thus memorization of entities and entity features is still a big issue for machine learning based systems in NER task.

3.1. NER Dataset for Joint Learning

For training and testing the final joint learning systems described in this work for the NER task for Turkish, a commonly used dataset is used. This dataset is created as part of a work [72] which tackles four information extraction task including NER. It is made up of articles from a national newspaper. This section will give a detailed explanation of the dataset along with some observations and statistics.

The dataset is created in 2003 and many researcher working on NLP tasks for Turkish made use of this dataset in the past two decades. As researchers most naturally make changes on the dataset or use only some parts of it according to their needs, there are many versions of this dataset available. For example, at the beginning of the experiments which are done using publicly available machine learning tools, a version of this dataset is used where the training set is not splitted into training and development sets. Also different researchers may use different sentence splitting mechanisms which can have different outputs. The final proposed joint learner model makes use of a development set. Using a development set with machine learning based systems is very common for various purposes such as preventing over fitting. In order to have more consistent results and be able to compare the results, the exact same version which was used in [57] is used for all the experiments done with the joint model. In this version of the dataset some parts of the dataset is taken out as they contained inconsistency as described in the same work. These inconsistencies are related to the morphological golden tags of the words. Even though those inconsistencies are not related to the NER task for which the dataset is used for, same version is kept for the reasons mentioned above. So this particular version of the dataset will be explained here. More information about the specific version of this dataset and more information about the way some sentences has been deleted can be obtained from the referenced work [57].

The version of the dataset used in this work has the following fields of the CoNLL-U format annotated:

- ID : Word index, starting from 1. Multiword tokens are ignored and every token is counted separately.
- FORM : Surface form of the token.
- FEATS : List of morphological features. The gold label morphological analysis is used for this field. Yet during the training of the joint learner this field is not used.
- MISC : This field is reserved for language specific annotations that does not belong to other fields. In this version of this dataset MISC field is used for candidate morphological analyses, NER tag and correct morphological analysis which is the same analysis with the list of morphological features given in FEATS field.

Remaining fields of the CoNLL-U format is not used and therefore marked with an underscore (_).

Number of sentences in this version is given in Table 3.1. Number of tagged entity tokens and number of tagged entities are given in Table 3.2 and in Table 3.3 respectively. Some evaluation metrics does not count the partial matches for a multiword named entity as matches. So partial match and complete miss of the entity is counted as the same. For such metrics the difference between the number of entities and number of entity tokens becomes an important parameter about the dataset.

The most important part of this version of the dataset is the content of the MISC field. The field is designed in a dictionary format where each entry is a key-value pair. The first key is 'ALL_ANALYSES' and the value is a list of all possible morphological analyses for the given token. The second key is 'NER_TAG' and the value is the NER tag of the token. This is the only key-value pair used in this Thesis but the data format is kept to same in order to have consistency. The final key in the version which is used during the experiments is 'CORRECT_ANALYSIS' and the value is the gold label morphological analysis of the token. As the number of key-value pairs of this

field may increase this description should not be considered as a final description. It is rather given to describe the dataset used during this work. An annotated example sentence from this dataset is given in Figure 3.1. The annotated example sentence is: 'Ayvalık Türkiye'nin büyük patronlarının yöreye duydğu ilgiden memnun.' which corresponds to the following sentence in English: 'Ayvalık is happy with the interest shown by the heads of the big companies of Turkey to the area.'

Table 3.1. Number of sentences in the Turkish NER dataset.

Subset	**Number of Sentences**
Training	25,514
Development	2,954
Test	2,915

Table 3.2. Number of annotated tokens in the Turkish NER dataset.

Subset	**LOC**	**ORG**	**PER**
Training	7,865	13,561	10,292
Development	904	2,026	1,347
Test	1,103	1,711	1,260

Table 3.3. Number of annotated entities in the Turkish NER dataset.

Subset	**LOC**	**ORG**	**PER**
Training	6,720	9,260	6,249
Development	769	1,412	824
Test	907	1,174	670

```
1	Ayvalık _	_	_	ayvalık+Noun+Prop+A3sg+Pnon+Nom _	_	_	{"ALL_ANALYSES":
["ayva+Noun+A3sg+Pnon+Nom^DB+Noun+Ness+A3sg+Pnon+Nom", "ayva+Noun+A3sg+Pnon+Nom^DB+Adj+FitFor", "ayval\u0131k+Noun
+Prop+A3sg+Pnon+Nom"], "NER_TAG": "I-LOC", "CORRECT_ANALYSIS": "ayval\u0131k+Noun+Prop+A3sg+Pnon+Nom"}
2	,	_	_	,+Punc	_	_	_	{"ALL_ANALYSES": [",+Punc"], "NER_TAG": "O",
"CORRECT_ANALYSIS": ",+Punc"}
3	Türkiye'nin	_	_	türkiye+Noun+Prop+A3sg+Pnon+Gen _	_	_
{"ALL_ANALYSES": ["t\u00fcrkiye+Noun+Prop+A3sg+P2sg+Gen", "t\u00fcrkiye+Noun+Prop+A3sg+Pnon+Gen"], "NER_TAG": "I-
LOC", "CORRECT_ANALYSIS": "t\u00fcrkiye+Noun+Prop+A3sg+Pnon+Gen"}
4	büyük	_	_	büyük+Adj	_	_	{"ALL_ANALYSES": ["b\u00fcy\u00fck
+Adj"], "NER_TAG": "O", "CORRECT_ANALYSIS": "b\u00fcy\u00fck+Adj"}
5	patronlarının	_	_	patron+Noun+A3pl+P3sg+Gen	_	_	_
{"ALL_ANALYSES": ["patron+Noun+A3pl+P3pl+Gen", "patron+Noun+A3sg+P3pl+Gen", "patron+Noun+A3pl+P3sg+Gen", "patron
+Noun+A3pl+P2sg+Gen"], "NER_TAG": "O", "CORRECT_ANALYSIS": "patron+Noun+A3pl+P3sg+Gen"}
6	yöreye	_	_	yöre+Noun+A3sg+Pnon+Dat _	_	_	{"ALL_ANALYSES": ["y\u00f6re
+Noun+A3sg+Pnon+Dat"], "NER_TAG": "O", "CORRECT_ANALYSIS": "y\u00f6re+Noun+A3sg+Pnon+Dat"}
7	duyduğu _	_	_	duy+Verb+Pos^DB+Noun+PastPart+A3sg+P3sg+Nom	_	_
{"ALL_ANALYSES": ["duy+Verb+Pos^DB+Adj+PastPart+P3sg", "duy+Verb+Pos^DB+Noun+PastPart+A3sg+P3sg+Nom"], "NER_TAG":
"O", "CORRECT_ANALYSIS": "duy+Verb+Pos^DB+Noun+PastPart+A3sg+P3sg+Nom"}
8	ilgiden _	_	_	ilgi+Noun+A3sg+Pnon+Abl _	_	_	{"ALL_ANALYSES": ["ilgi+Noun
+A3sg+Pnon+Abl"], "NER_TAG": "O", "CORRECT_ANALYSIS": "ilgi+Noun+A3sg+Pnon+Abl"}
9	memnun	_	_	memnun+Adj	_	_	_	{"ALL_ANALYSES": ["memnu+Adj^DB+Noun
+Zero+A3sg+P2sg+Nom", "memnun+Adj"], "NER_TAG": "O", "CORRECT_ANALYSIS": "memnun+Adj"}
```


Figure 3.1. An example sentence from the Turkish NER dataset in CoNLL-U format. Misc field is used for Morphological analyses and NER label.

3.2. NER Dataset for Initial Models

As mentioned above, the NER dataset for Turkish contains many versions. During the initial experiments the version of the dataset was different than the dataset explained above. The dataset is made up of same sentences but this version is splitted to training and test sets. So the development set of the above mentioned version is included in the training set. Also as will be explained in detail in following sections this version of the dataset includes several features which are used by the initially implemented models. The dataset is not in CoNLL-U format. This version is also in token-per-line format where each line contains a token and sentences are separated with a blank line. Each word line contains several hand crafted features which are fed as input to feature based machine learning tools. Initially the dataset contained only the surface forms of each word together with the gold NER label. Then by using heuristics and reading the related literature the number of features are increased gradually. Figure 3.2 gives an example annotated sentence from the final version of this dataset. Each field in the dataset will be explained in Chapter 4 so here we only give the title for each field in the order they appear in Figure 3.2. They are as follows: surface form, final POS tag, capitalization, stem of the word, start of sentence, proper noun feature, acronym feature, nominal feature, final suffix, index of the dependency head

and dependency relation. Previous versions of the dataset can be inferred easily as the features always appended to the second to last field since the last field is always the gold label that is being predicted. Dependency related features are added at the end so they appear at 10^{th} and 11^{th} positions. The example sentence given in Figure 3.2 is: 'POZİTİF ve Açık Radyo işbirliğiyle düzenlenecek olan İstanbul Müzik Şenliği 2, müzikseverlere Aralık ayında merhaba demeye hazırlanıyor', which can be translated into English as: 'Istanbul Music Festival 2 which will be organized in collaboration with POZITIF and Open Radio will welcome the music lovers in December.'

```
POZİTİF Adj 2 pozitif 0 0 Notacro Notnom None 9 nmod:poss BORG
ve Unkn 0 ve 0 0 Notacro Notnom None 8 cc IORG
Açık Adj 1 açık 0 0 Notacro Notnom None 8 amod IORG
Radyo Noun 1 radyo 0 0 Notacro Nom None 9 nmod:poss IORG
işbirliğiyle Noun 0 İşbir 0 1 Notacro Notnom YlA 10 obl O
düzenlenecek Adj 0 düzenle 0 0 Notacro Notnom None 11 obj O
olan Adj 0 ol 0 0 Notacro Notnom None 12 acl O
İstanbul Prop 1 İstanbul 0 1 Notacro Notnom None 22 nsubj BLOC
Müzik Noun 1 müzik 0 0 Notacro Nom None 12 flat O
Şenliği Noun 1 şen 0 0 Notacro Notnom SH 12 flat O
2 Unkn 0 2 0 0 Notacro Notnom None 12 flat O
, Punc 0 , 0 0 Notacro Notnom None 22 punct O
müzikseverlere Noun 0 müziksever 0 0 Notacro Notnom YA 22 obl O
Aralık Noun 1 aralık 0 0 Notacro Nom None 19 nmod:poss O
ayında Noun 0 Ay 0 0 Notacro Notnom NDA 22 obl O
merhaba Noun 0 merhaba 0 0 Notacro Notnom None 22 obl O
demeye Noun 0 de 0 0 Notacro Notnom YA 22 nmod O
hazırlanıyor Verb 0 hazırla 0 0 Notacro Notnom Hyor 0 root O
```

Figure 3.2. Final version of the NER dataset used in the initial phases of this Thesis. Each line includes a token, 10 features about the token and the gold NER label.

3.3. Dependency Dataset

Final proposed joint learning model make use of the publicly available Universal Dependencies framework for the labeled dependency parsing dataset. The dataset is provided for the CoNLL 2018 Shared Task [67] for multilingual parsing. Universal Dependencies is a framework for cross-linguistically consistent grammatical annotation. The primary aim of this framework is to create datasets for multiple languages in a consistent format. This will allow the development and testing of multilingual parsers.

Another benefit of cross-linguistically consistent grammatical annotations is that it allows cross-lingual learning. It is harder to obtain a high-quality corpora for all languages. On the other hand, several frequently used languages have vast grammatically annotated sources. By creating a framework which contains all datasets in a consistent format, knowledge gained on a certain language can be used in the other language settings with limited annotated resources. Universal Dependencies is an important step towards creating generalizable systems and enabling fast research on grammatical tasks. Before going on explaining the IMST dependency dataset for Turkish, which is the dataset used for the dependency task in this Thesis, Universal Dependencies framework and the specific dataset format which is called CoNLL-U or CoNLLU will be explained.

3.3.1. CoNLL-U Format

CoNLL-U format is an extended and revised version of CoNLL-X. Detailed information about these dataset formats can be accessed from the relevant website of the framework. Below a brief summary of the CoNLL-U format is given which is sufficient if the main purpose is to make use of a dataset in CoNLL-U format. The basic structure of the CoNLL-U format is as follows:

- A single token occurs in a line with a total of 10 fields each separated by a tab character. Fields with no entries are marked with an underscore ('_'), so that none of the fields are left blank.
- Each sentence is separated with blank lines.
- '#' Character precedes comment lines and those lines are not taken into account.

The definitions of each of the 10 fields of a word line given in the official website of Universal Dependencies are as follows :

(i) ID: Word index, integer starting at 1 for each new sentence.
(ii) FORM: Word form or punctuation symbol.

(iii) LEMMA: Lemma or stem of word form.

(iv) UPOS: Universal part-of-speech tag.

(v) XPOS: Language-specific part-of-speech tag; underscore if not available.

(vi) FEATS: List of morphological features from the universal feature inventory or from a defined language-specific extension; underscore if not available.

(vii) HEAD: Head of the current word, which is either a value of ID or zero (0).

(viii) DEPREL: Universal dependency relation to the HEAD (root iff HEAD = 0) or a defined language-specific subtype of one.

(ix) DEPS: Enhanced dependency graph in the form of a list of head-deprel pairs.

(x) MISC: Any other annotation.

Above given information is necessary and sufficient for researchers who would just like to use a CoNLL-U format dataset for their research. An example annotated sentence from this dataset is given in Figure 3.3. The example sentence in Turkish is: 'Karşısında, pantolonu dizlerine dek ıslak, önlük torbası ham eriklerle dolu İbrahim dikiliyordu', which corresponds to the following English sentence: 'Against him Ibrahim was standing with his pants wet up to the knees and with his bag filled with plums.' The final field is made available so that annotation teams and researchers who would like to use additional annotations can append them to the final field. This way the CoNLL-U format allows flexibility while keeping the format consistent. Researchers who would like to ignore the final entry or any other entry can just skip the certain entry. This way CoNLL-U format allows tackling different NLP tasks without making any changes on the dataset.

3.3.2. IMST-UD Dataset

UD Turkish Treebank or IMST-UD Treebank is used for the dependency parsing task for Turkish in this study. This treebank is a part of the Universal Dependencies framework and annotated according to the CoNLL-U format. It is a semi-automatic conversion of the IMST Treebank [73], which is itself a reannotated version of the METU-Sabanci Turkish Treebank [74]. The dataset is publicly available and can be

obtained easily online. Detailed statistics about the dataset is given in the relevant GitHub repository of the work as well. The dataset is made up of 5,635 sentences from daily news reports and novels. The details of the annotations and other characteristics of the dataset is given in the official website of the Universal Dependencies.

This dataset is used at the later stages of this Thesis with the joint model. Below is the list of the cases this dataset is used in:

(i) Training a dependency parser on the training set and using the trained model to annotate the NER dataset for Turkish for dependency relations. This annotation are used as additional features for the feature based systems used in the exploration phase of the experiments.

(ii) Training and testing a new dependency parser without joint learning to see whether performance increases or decreases when a joint learning is incorporated.

(iii) Training and testing the dependency parser part of the joint learner model.

This dataset has all of the fields of the CoNLL-U format annotated except for the 9^{th} field which is given the name 'DEPS' and is reserved for enhanced dependency graph annotation. MISC field is used for denoting the existence and nonexistence of a space right after the token. Figure 3.3 shows an example sentence from this dataset. The joint learner only makes use of the surface form, dependency relation type and dependency head index fields.

```
# sent_id = mst-0006
# text = Karşısında, pantolonu dizlerine dek ıslak, önlük torbası ham eriklerle dolu İbrahim dikiliyordu.
1   Karşısında  karşı   ADJ NAdj    Case=Loc|Number=Sing|Number[psor]=Sing|Person=3|Person[psor]=3  14  amod    _   SpaceAfter=No
2   ,   ,   PUNCT   Punc    _   14  punct   _   _
3   pantolonu   pantolon    NOUN    Noun    Case=Nom|Number=Sing|Number[psor]=Sing|Person=3|Person[psor]=3  4   nmod:poss   _   _
4   dizlerine   diz NOUN    Noun    Case=Dat|Number=Plur|Number[psor]=Sing|Person=3|Person[psor]=3  6   obl _   _
5   dek dek ADP PCDat    _   4   case    _   _
6   ıslak   ıslak   ADJ Adj _   13  amod    _   SpaceAfter=No
7   ,   ,   PUNCT   Punc    _   14  punct   _   _
8   önlük   önlük   NOUN    Noun    Case=Nom|Number=Sing|Person=3   9   nmod:poss   _   _
9   torbası torba   NOUN    Noun    Case=Nom|Number=Sing|Number[psor]=Sing|Person=3|Person[psor]=3  14  obl _   _
10  ham ham NOUN    Noun    Case=Nom|Number=Sing|Person=3   11  nmod    _   _
11  eriklerle   erik    NOUN    Noun    Case=Ins|Number=Plur|Person=3   12  obl _   _
12  dolu    dolu    ADJ Adj _   13  amod    _   _
13  İbrahim İbrahim PROPN   Prop    Case=Nom|Number=Sing|Person=3   14  nsubj   _   _
14  dikiliyordu dikil   VERB    Verb    Aspect=Prog|Mood=Ind|Number=Sing|Person=3|Polarity=Pos|Polite=Infm|Tense=Past   0   root    _
SpaceAfter=No
15  .   .   PUNCT   Punc    _   14  punct   _   _
```

Figure 3.3. An example sentence from the IMST-UD Treebank dataset used by the dependency parser of the joint learner.

4. METHODOLOGY

4.1. CRF Model

After finalizing the initial literature survey phase, implementation of a NER system is started. A simple model is implemented and the complexity is gradually increased by observing the weaknesses and by using the knowledge obtained during the literature survey. We do not view this as the final architecture. Rather we have found it more useful to start with a simple model and then by observing the weaknesses and the limitations move onto a new system as necessary. For this purpose the initial model is developed using a CRF-based third party software. The main reference article of this initial model, the work of Seker *et al.* [52], also applies CRF's on the same Turkish dataset we have used.

The CRF-based tool uses an annotated training data and feature functions and tries to find the optimal linear combination of weights by observing the annotated dataset. We try to maximize the conditional expectation of the transitions that occur frequently in the training data. CRF's are suitable for NER task because they can capture the sequential nature of a natural language and make use of features.

The CRF-based tool used is named 'CRF++' and the version of the tool used is 0.58. We have started with a baseline model and gradually improved the model by using different features and feature functions by observing the weaknesses of the system and using the methodology described in the reference article. We have obtained a Conll score of 89.79% without using gazetteers. This is the best obtained result for Turkish using CRF as the learning algorithm without using gazetteers.

This section describes the 'CRF++' toolkit. The results are given in the Experiments and Results chapter. This toolkit is an open source implementation of Conditional Random Fields and can be used for various sequential labeling tasks including NER. Below is a list of some of the features the toolkit offer:

- Feature sets can be redefined. This allows testing the system with different feature sets.
- Fast training based on L-BFGS. Compared to the 'Wapiti' toolkit which will be explained later this tool does not offer options for the optimization algorithm being used.
- The output probabilities can be retrieved which enables observing the confidence of the model about a specific prediction. This feature is especially important when making use of several tools and combining their predictions to boost the performance.

The tool requires inputting a feature template which is used for automatically generating all possible features that will be used by the CRF model. The feature functions are generated by considering all possible cases in the training set which fit the definition of a feature given in the feature template. Thus even if the dataset with the features stay the same the tool enables training the model with different feature combinations. This allows faster research as the same dataset can be preserved throughout the experiments. The only requirements for the datasets given as input to the tool is as follows:

- The datasets must be in token-per-line format as in the case of CoNLL-2003 datasets.
- All the datasets (training and testing) must have equal number of features with the same ordering. The number of features also must be kept the same for all tokens in both datasets.
- The features must be separated by a blank space and the final field must contain the label that will be learned and predicted.

The template file for the patterns also has a specific definition. An example segment from the template file is given in Figure 4.1. The lines starting with 'U' refer to the unigram features. The feature functions are only generated for the current token when U prefix is used. 'B' prefix is used for bigram features which are generated

```
# Unigram
U00:%x[-2,0]
U01:%x[-1,0]
U02:%x[0,0]
U03:%x[1,0]
U04:%x[2,0]
U05:%x[-1,0]/%x[0,0]
U06:%x[0,0]/%x[1,0]

U10:%x[-2,1]
U11:%x[-1,1]
U12:%x[0,1]
U13:%x[1,1]
U14:%x[2,1]
U15:%x[-2,1]/%x[-1,1]
U16:%x[-1,1]/%x[0,1]
U17:%x[0,1]/%x[1,1]
U18:%x[1,1]/%x[2,1]

U20:%x[-2,1]/%x[-1,1]/%x[0,1]
U21:%x[-1,1]/%x[0,1]/%x[1,1]
U22:%x[0,1]/%x[1,1]/%x[2,1]

# Bigram
B
```

Figure 4.1. Example for the template file for the features of the CRF++ tool.

considering the label of both the previous and the current token. In Figure 4.1, B is used without any other specification. This means that only the bigram tag transition probabilities will be generated using this pattern template. Below is an example line:

$$U15 : \%x[-2, 1]/\%x[-1, 1]$$

The following number of the first character, which is '15' in this case, denote the 'ID' of a feature. This enables looking at the weights of each layer in the model file generated at the end of the learning mode. '%x' denote a feature. '[-2,1]' means that the second previous token's (-2), second feature (1) must be used. '/' sign denotes that the first feature must be combined with the succeeding feature which is the second feature (1) of the previous word (-1). So this way any number of features can be combined but this will increase the memory used and the time required to train the model. So the feature template should only include relatively important features and only important combinations of those features so that the model can be trained efficiently without suffering from low performance.

4.2. Models using Wapiti toolkit

This section explains in detail the 'Wapiti' toolkit used for training several feature based models. This toolkit explained in [75], is a sequence classifier toolkit which allows training models using various model types and optimization algorithms. The results achieved by this toolkit on the CoNLL-2003 English dataset is comparable to the state-of-the-art neural network based systems even though the training time is shorter and the memory requirement is significantly lower. After observing that this toolkit is more suitable for conducting extensive experiments with different configurations, we have switched to this toolkit. The toolkit is chosen primarily because it enables fast configuration of various training models as well as fast configuration of the features that are being used by the model. Following subsections will describe the specific aspects of this toolkit.

4.2.1. Training models

The wapiti toolkit allows using various machine learning models for training as mentioned previously. The models and their brief description are as follows:

- Maximum Entropy (MAXENT): Maximum Entropy models are very general probabilistic methods that pick the output with the highest entropy by considering the observations and the prior knowledge. These models are frequently used in NLP related tasks. A statistical machine translation system making use of maximum entropy models for reordering of the phrases is explained in [76]. A Maximum Entropy based model is used in [77] for the POS tagging task.
- Maximum Entropy Markov Models (MEMM): It is an extension of the Maximum Entropy models which consider the hidden features of Hiddden Markov Models. It is also frequently used in NLP, especially for the sequence labeling tasks such as POS tagging [78] and NER [79]. Main difference between MEMM and MAXENT is that the former calculates the conditional probability of each prediction by taking into account the previous information. MAXENT assumes independence between individual predictions.

- Conditional Random Fields (CRFs): This model calculates the transition probabilities from one prediction to another in addition to the Markovian assumption of the MEMM. Thus the output prediction of CRF's is a sequence of outputs for a given sequence of inputs such as words in a given sentence. Transition probabilities between tags are learned from the training dataset.

A very nice and detailed explanation of CRF model is given in [80]. The work also contains a very nice illustration of the relationship between the models explained above. This illustration is given in Figure 4.2.

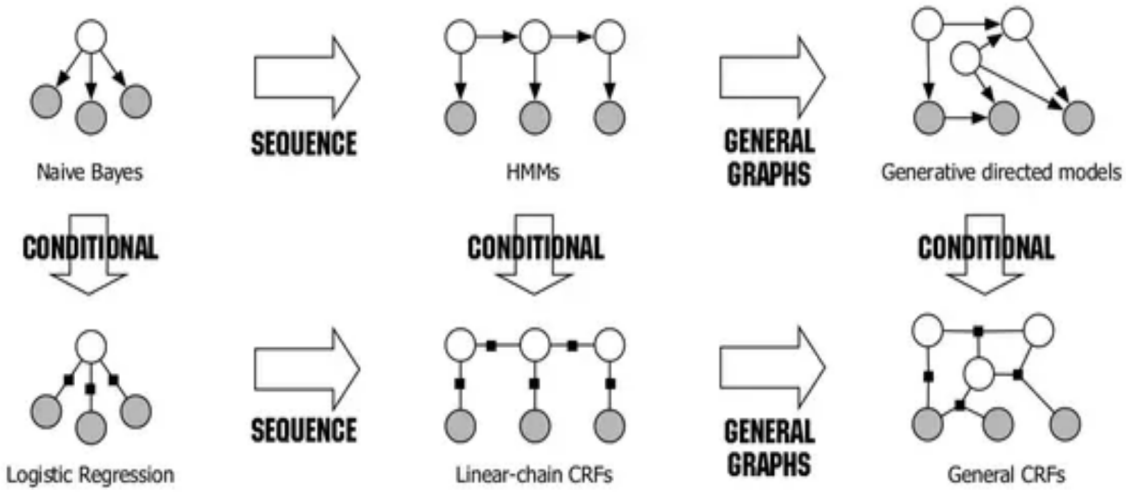

Figure 4.2. Illustration of the relationship between various machine learning models taken from [80].

4.2.2. Data Format

Wapiti can use only a specific data format. The data for training and testing must be in the same token-per-line format with same number of features. The last token of each line must be the label of the token. The sequence that corresponds to sentences must be separated by a blank line. If not the program considers the document as a

single sequence and learns transition probabilities from previous sentences meanwhile decreasing the training speed.

The number of features must be the same for each word of each dataset in order the toolkit to function. If no annotation is available for a given word for a specific feature or field then a special character must be replaced for each such case. Thus none of the fields must be empty.

The token-per-line format is a highly frequently used data format in NLP related tasks. This format allows word level annotation which is enough in most cases. Sub-word level annotation can be required for some tasks for which this format must be revised. In the case of NER task word level annotation is the most frequent methodology.

4.2.3. Modes

Wapiti has 2 main modes: training and labeling. During training the user can choose among different training models and different learning algorithms which are explained previously. The training mode requires a pattern file which contains the information about how to generate feature functions in CRF which will be used in transition likelihood calculations. Given a pattern file and a training corpus the tool creates a model which contains the transition probabilities. This model is used for prediction.

Labeling mode is the prediction mode of the wapiti program and requires the model file and test file as inputs and outputs the prediction file containing the label prediction at the end of each line for each token. The tool has the option to calculate and output the recall, precision and F1-score at the end of the prediction. The scores will not be correct if the tagging format is different from the format of the corpus(BIO, BIOES etc.).

As the toolkit offers a wide range of configuration options for both training and testing modes the following section is devoted for explaining them.

Wapiti toolkit allows full flexibility in configuration. The user can change every hyper parameter of the model by a simple command. Below is the list of some parameters that can be configured for the training mode along with their definitions:

- –train: type of model to train.
- –algo: type of optimization algorithm used for training the model.
- –nthread: number of threads to be used which can increase the model performance.
- –sparse: used for enabling sparse computation during forward and backward iterations.
- –rho1: defines the L1 component of the elastic-net penalty.
- –rho2: defines the L2 component of the elastic-net penalty. Increasing these penalties result in smaller models.
- –clip: Enables the gradient clipping for L-BFGS. Generally useful for sparse models regularized with L2-only penalty.

The above list is not an exhaustive list of all the configurable parameters of the training mode of the toolkit. The above given ones are picked by their effect on the resulting performance of the final trained model.

Labeling mode also contain various configurable parameters two of which are as follows:

- –me: used for activating the pure maxent mode. In default the program switches between linear-chain and maxent mode depending on the data observed. Specifying the pure maxent mode forces the program to use this for all observations.
- –post: used for allowing the posterior decoding algorithm instead of the default Viterbi algorithm. This often results in an increase in performance with slower decoding times.

The patterns given as input determine the feature functions that will be generated. Feature functions are binary-valued functions such as: 1 if previous word is 'in' else 0. They can combined to produce more complicated patterns. The patterns can be the features given in the corpus as shown in the example above or Regex patterns. Wapiti allows simple Regex patterns which enables adding features like capitalization without making changes in the dataset itself through annotations. The program requires a specific pattern file format to be inputted. The format is very similar to the pattern format used by the CRF++ tool. In addition there is a specific format for Regex patterns as CRF++ does not have that functionality. The details of the pattern file format can be accessed from the relevant website of the toolkit.

4.3. Joint Learning Model

This section is devoted to explain the final proposed model of this Thesis. It contains the details of the most important contribution of the work and explains the model implemented for the main hypothesis of the work. The model is BiLSTM based and attempts joint learning of two tasks: Dependency Parsing and Named Entity Recognition. The architecture and the details of the models will be explained in detail. The final proposed model has different variations which tackle each task separately or tackle both tasks in different ways. Each variation will also be described in detail in this section. Yet these variations are highly similar and explaining each variation separately will cause redundancy. Thus, initially the main version of the model is explained in detail. Only the differences with the main version is given for the remaining variations.

This model is used to test our main hypothesis in this Thesis. The main hypothesis is that we can do joint learning by training different parts of a neural network using different datasets. This approach will increase the flexibility of data driven systems as it does not require the same dataset to be annotated for each task. Rather the model attempts learning whatever information is available in a given dataset to improve the performance for all tasks. For example, a trained model can be retrained on a dataset which only has the NER tags and this will help tuning the parameters of the part of the neural network which is responsible for the dependency parsing task as well.

Related work is explained in detail in the Chapter 2. The extensive literature survey reveals that the joint learning of Dependency Parsing and Named Entity Recognition is not attempted previously. Also joint learning by using different datasets for each task is another novelty of this model as other work on joint learning use a single dataset annotated for each task.

4.3.1. General Information

The model is implemented using the version 2.7 of Python programming language. DyNet neural network library [81] is used for the implementation of the neural network architecture. Below, a short introduction of this library is given.

4.3.2. DyNet Introduction

DyNet is a neural network library which can be used in Python programming language. The library is especially designed for neural networks with dynamical structures. Dynamical networks are useful for natural language related tasks as the part of the neural network used for each training instance is different. Conventional libraries do not offer convenient ways to handle this difference in the input data. DyNet offers a convenient parameter based approach to define the neural network. Only the relevant parameters for a given instance is taken into account and updated. In DyNet terminology a neural network is a 'Parameter Collection' where the parameters correspond to the weights in the network. Each layer is defined by the number of input and output dimensions which correspond to the number of connections to the previous and next layers.

Consider the task of named entity recognition. The typical input received by a machine learning system is a text of varying size such as a document or a sentence. A general approach is to represent all possible words in the vocabulary and the additional special word 'unkw' for unknown words with a vector embedding. Let *wemb* and *slen* represent the size of a word embedding and the number of words/tokens in some

sentence S. Then the size $|I_S|$ of the input I_S to the neural network is,

$$wemb \times slen$$

This means that the size of the input varies for each input instance as well as the words in each input sentence. Thus only a small portion of the weights of the neural network is used. A rough estimate of the ratio of the first layer connections used for each sentence can be calculated as follows. Let $\hat{s}$, $\hat{v}$ represent the average number of words in a sentence and the number of distinct words in a vocabulary for a particular language respectively. By making use of the huge data available, we can find really close estimates of these values. Then the ratio r of the first layer connections used will be,

$$r = \frac{\hat{s}}{\hat{v}}.$$

DyNet offers denoting each connection in a key-based manner with a lookup table structure. Thus for each training instance the small portion of the relevant connections is considered and used. Two possible ways to interpret this mechanism is as follows:

(i) For each training instance a new neural network is constructed which has the input layer node size equivalent to the size of the input sentence, which varies for each sentence. The weights are the corresponding weights of each word in the instance sentence in the order the words appear.

(ii) There is a huge neural network for all possible words in a dictionary. For each instance sentence only the relevant nodes are switched on and used, and the rest of the network is inactive.

DyNet library offers builders for the initialization of recurrent neural network (RNN) layers. RNN layers are frequently used in applications which make use of dynamic network structures. As the input is fed into the network sequentially RNN's are suitable for sequential applications such as the NER task. Unlike classical multi

layer perceptrons the value of the previous input to the network affects the final output for the current input. This feature has particular importance for sequential inputs. In the joint learning model a specific type of RNN Builder is used: Vanilla LSTM Builder. Vanilla LSTM Builder creates a standard LSTM with decoupled input and forget gates and no peehole connections. Below the details of this builder is given. The dynamics of the Vanilla LSTM Builder are given in Figure 4.3.

$$
\begin{aligned}
i_t &= \sigma\left(W_{ix}x_t + W_{ih}h_{t-1} + b_i\right) \\
f_t &= \sigma\left(W_{fx}x_t + W_{fh}h_{t-1} + b_f + 1\right) \\
o_t &= \sigma\left(W_{ox}x_t + W_{oh}h_{t-1} + b_o\right) \\
\tilde{c_t} &= tanh\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right) \\
c_t &= c_{t-1} \circ f_t + \tilde{c_t} \circ i_t \\
h_t &= tanh\left(c_t\right) \circ o_t
\end{aligned}
$$

Figure 4.3. Dynamics of the VanillaLSTM Builder

In the Figure 4.3, σ denotes the sigmoid activation function and the input connections and recurrent connections are initialized by sampling from specific distributions. i_t and f_t are calculated using the current token x_t and the hidden output h_t of the previous LSTM cell. They correspond to the input and forget gates respectively. These are used as gates that allow or block the memory information and the current cell information respectively. Output vector o_t is also calculated by taking into account the current token and the hidden output of the previous LSTM cell, and it is used to calculate the hidden output of the current cell. $\tilde{c_t}$ is the candidate cell vector. The cell vector c_t of the current cell is calculated by using the cell vector of the previous cell c_{t-1}, candidate cell vector $\tilde{c_t}$ and the above mentioned memory gates f_t and i_t. Final line shows the hidden output vector h_t of the current LSTM cell which uses the context vector c_t and o_t. The outputs of the LSTM cells are the h_t vectors calculated at

each step using these dynamics. Figure 4.4 gives an illustration of the above explained calculations made in each cell.

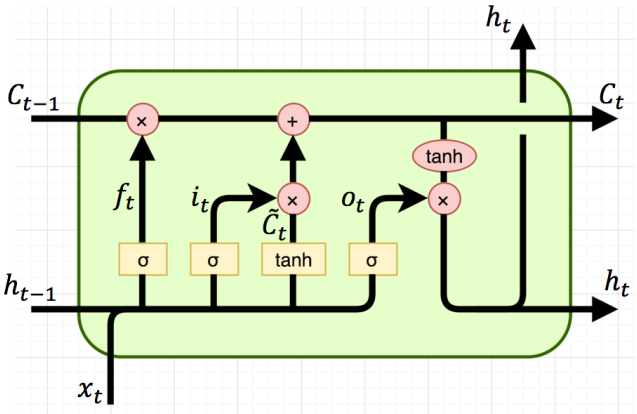

Figure 4.4. LSTM cell illustration.

The following are the parameters of the Vanilla LSTM Builder:

- layers: Number of layers.
- input_dim: Dimension of the input x_t.
- hidden_dim: Dimension of the hidden states h_t and c_t.
- model: ParameterCollection which holds the parameters.
- ln_lstm: binary feature for using layer normalization.
- forget_bias: value to use as bias for the forget gate. The default value is 1.0.

4.3.3. Input Data

As explained previously in the relevant section of the datasets chapter, the input to the system must be in the CoNLL-U format. The details of this format is given in the relevant section. After receiving the data in the CoNLL-U format, preprocessing is done on the input data. Preprocessing is done to store the given CoNLL-U format data in a convenient data structure. Vocabulary of the dataset is generated in the beginning

and then various features are generated by observing the vocabulary. These features are used during the training and prediction phases of the system. As the neural model takes as input vector representations which are numerical, every feature is converted into a vector representation by random initialization.

The vocabulary generation algorithm shown in Figure 4.5 is used to store all entries in the dataset into dictionaries which maps entries and characters to indexes. This algorithm is followed by another algorithm which converts the dataset into sentences which are made up of Entry objects. The initialization function of the Entry objects is given in Figure 4.6. Each token in a line in the dataset is an Entry object with variables representing the values of each field in CoNLL-U format. During training and prediction phases each sentence is considered separately. Thus the main input to the model can be considered as an array which is made up of Entry objects.

4.3.4. Output Data

The program has two main modes: Training and Prediction. The output of the model changes depending on the current mode. Besides, the output of the neural network and the overall program can be considered as different outputs. The output of the neural network in both modes is the scores for the Dependency and Named Entity tags. These scores are used to calculate losses and to find the best prediction of the model. The output of the overall program during training is a model file which stores all the information necessary for rebuilding the trained neural network and a parameters file which stores the information related to the training process including the hyper parameter information. The output of the overall program during the prediction mode is an annotated dataset with predictions for each token appended at the end of each corresponding line. The joint learner makes predictions about both the dependency tags and the named entity tags, so two output files are produced.

Input CoNLL-U format dataset
for each annotated field that will be used **do**
  Initialize a Counter to store relevant data
end for
Initialize char_dict
Set char_dict['unkw'] = 0
Set char_dict['< w >'] = 1
Set char_dict['< /w >'] = 2
for each **line** in dataset **do**
  if line ≠ empty_line OR comment **then**
    Update each Counter
    for each **c** in line[1] **do**
      if c not in char_dict.keys() **then**
        char_dict[c] = char_dict[len (char_dict.keys())]
      end if
    end for
  end if
end for
for all Counter$_i$ in initialized Counters **do**
  j = 0
  for all key in Counter.keys() **do**
    Dict$_i$ [key] = j ++
  end for
end for
Return all Dict$_i$ and char_dict

Figure 4.5. Vocabulary Generation Algorithm. This algorithm converts the vocabulary and the entries in each field into indexes. These indexes is used to initialize the vector representations.

```
class Entry:

    def init(id1,word,nertag,reltag,capinfo,parent_id):
        self.id1 = id1
        self.form = word
        self.capinfo = capinfo
        self.nertag = nertag
        self.relation = reltag
        self.parent_id = parent_id
        self.pred_parent_id = None
        self.pred_nertag = None
        self.pred_relation = None
```

Figure 4.6. Entry object is initialized by reading each line of the CoNLL-U format dataset

4.3.5. Work flow

There are various steps in a neural network based machine learning model. In the case of joint learning models, the number of steps increase even further. Thus, it can be difficult to track down the overall work flow. This section explains what the joint learning model does step by step. Figure 4.7 shows the overall workflow of the model in a diagram.

Below is the step by step explanation of the overall work flow of the joint learning model in the training mode:

(i) Reading the dataset and creating the vocabulary. Initializing dictionaries for each field of the CoNLL-U format that is not empty. The vocabulary generation algorithm is shown in Figure 4.5. Each token in the dataset is initialized as an 'Entry' type instance. The initialization of an instance is shown in Figure 4.6.

(ii) Initializing the parameter collection weights using the vocabulary data and the initialized dictionaries.

(iii) Initializing the neural network architecture using the parameters given to the system (embedding dimensions, lstm output dimension) and using the vocabulary.

(iv) Training is started and the dependency parser is trained only for 1 epoch on the dependency parsing dataset. After calculating the scores for the dependency arcs for each pair of words Eisner's decoding algorithm [82] is used to find the highest scoring spanning tree. Then another neural network layer with softmax activation is used for calculating the likelihood of each possible label for all the arcs in the previously found spanning tree. This approach is similar to the method employed in [83]. During this step the neural network part that is responsible for the NER task is not used and the weights are not updated. Loss function only makes use of the loss from the dependency parser output. Yet, since the dependency parser output is used in the NER training, that part can also said to be trained in an indirect way.

(v) Training continues with 1 epoch of NER task on the NER dataset. This time the model makes use of both parts of the network. First, the dependency parser is run and the parser generates scores for arcs and labels for a given sentence. These information are fed into the NER part of the neural network. Yet the loss function does not take into account the dependency parser output at this step. Only the scores generated by the neural network for the NER tags is used to calculate the loss value by making use of the gold NER labels.

(vi) After the NER epoch the system continues with making predictions on the development sets of each task separately. First dependency parser accuracy is measured over the relevant dataset using the dependency dataset. Then NER accuracy is calculated again by making use of the dependency parser predictions of the network for the NER dataset sentences. During the NER part of this prediction step the system has an additional CRF layer for finding the highest scoring NER label sequence for each sentence. The highest scoring sequence, found using the Viterbi decoding algorithm, is used as the final prediction for the NER task.

(vii) Single epoch for the overall system ends. The process is repeated starting from step 4 for the designed number of epochs in the same manner.

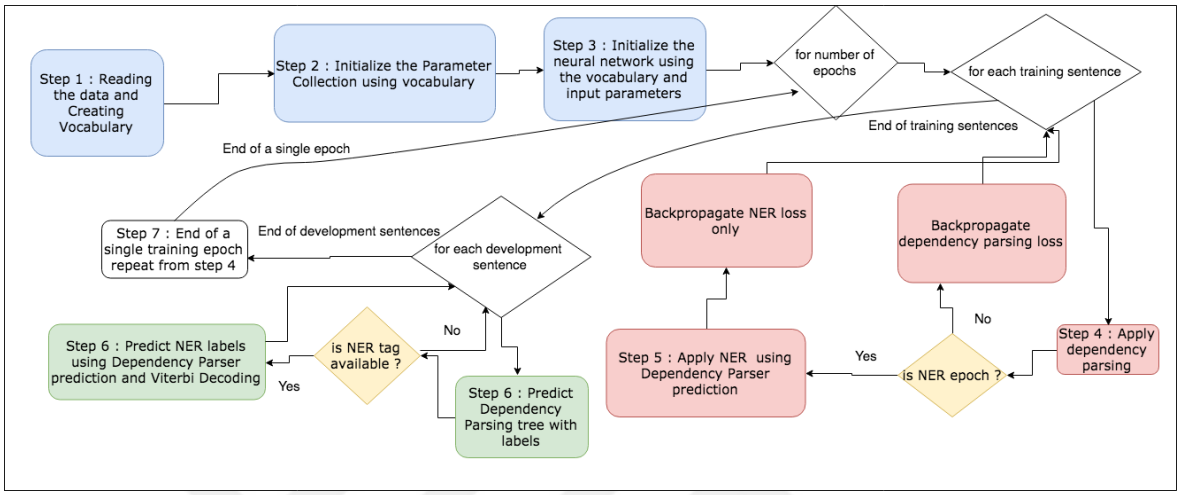

Figure 4.7. Work flow of the joint learner model. The dependency parser is used for all training sentences. During the training of the NER model it is used as additional information.

4.3.6. Architecture

This section explains the details of the neural network. A neural network is created following neural architectures of the previous state-of-the-art systems for NER task in Turkish and English. This BiLSTM based architecture is commonly used not just for the NER task but also for many NLP related tasks. Even though each system have their own unique part and each have specialized parts for improving the performance for the relevant task the fundamental idea is similar. The idea is to represent each token with the concatenation of forward and backward vector outputs of LSTM layers. The input to the LSTM layer in each system is subject to changes but in general word and character embeddings are given as inputs. Character embeddings are shown to boost performance of Named Entity Recognition systems [84] as well as many other systems tackling NLP-related tasks .In addition, task specific features can be converted into vector representations and can be appended to the word and

character embeddings. Different number of layers of BiLSTM can be used where output of each layer can be considered as a different level of abstraction for a given token. These representations are then given to a neural network which performs task specific calculations.

Following the related state-of-the-art work in NLP related tasks a BiLSTM is used to create vector representations of length $ldims$ for both tasks. The value of $ldims$ is a hyper parameter and experiments with various values can be done to find its optimal value. In the main joint learning model, the output of the dependency parsing network is given as input to the neural network responsible for the named entity recognition task. The input to the first BiLSTM layer is the concatenation of word embedding w_{emb}, forward and backward character embeddings c^f_{emb} and c^b_{emb} which are generated using a single LSTM layer and the vector representation cap_{emb} of the capitalization information of the token. cap_{emb} vector is the vector representation of the capitalization information and initialized by randomly mapping all possible capitalization types into vectors of fixed length, since LSTMs take as input only vectoral representations. Using vector representations also has the benefit of enabling the learning of the relations between different types of capitalizations of words.

The output of the first BiLSTM layer is the concatenation of the vectors created by going over a given sentence in forward and backward directions. Since the output of the LSTM layer has length $ldims$ for each direction, the output is of length $2 \times ldims$. The second BiLSTM layer takes as input a vector of size $2 \times ldims$ for each token in a sentence. The program again goes over these vector representations to create forward and vector outputs of size $ldims$ each.

The outputs of the second BiLSTM layer are used as the input for the multi layer perceptrons (MLP) responsible for calculating the scores for the dependency parsing task. Dependency parsing task contains two sub-tasks: creating a parse tree for a given sentence and labeling the arcs of the parse tree.

For the first sub-task the system finds the highest scoring parse tree which is a directed graph that covers all edges where each vertex corresponds to a token in a sentence. The head of the sentence is called 'root' and has the 'parent_id' as '-1'. All other tokens have 'parent_id' set equal to the index of the arc that points to that token. The algorithm for finding the highest scoring parse tree will be explained in the following sections. This algorithm takes as input arc scores for all possible arcs in a given sentence. Since the parse tree is a directed graph, for a sentence with n tokens $n^2 - n$ scores must be calculated. The multi layer perceptron responsible for calculating these arc scores takes as input a vector of length $8 \times ldims$. For tokens w_i and w_j in a sentence, the input to the perceptron is the concatenation of the following vectors:

(i) Concatenated forward and backward vectors of w_i and w_j. Since each vector has dimension $2 \times ldims$, this vector is of size $4 \times ldims$.

(ii) Absolute value of the point wise difference of the concatenated forward and backward vectors of each token. This vector is of size $2 \times ldims$.

(iii) Point wise multiplication of the concatenated forward and backward vectors of each token. This vector is also of size $2 \times ldims$.

In total the input to the multilayer perceptron is of size $8 \times ldims$. The number of layers and the number of hidden units of this perceptron are also hyper parameters which can be tuned by making experiments with different configurations. In this work number of hidden layers is 1 and the number of hidden units is 100. State-of-the-art work on dependency parsers which make use of a similar neural network [4] is taken as reference for obtaining these hyper parameter values. The output of this perceptron is the score of the directed arc from w_i to w_j. It is important to remember that since the dependency is a directed relation this score is different for the arc from w_j and w_i. Eisner's decoding algorithm [82] is used to find the highest scoring dependency tree which covers all tokens by using these scores calculated for each pair (w_i, w_j). Figure 4.8 shows the multi-layer perceptron for producing the arc scores for each pair. To avoid confusion each vector is represented separately but the overall input is a single vector which is the concatenation of these four vectors.

Let $\mathbf{e}_w$ denote the vector representation of a given token w. Also let, $\mathbf{w}_{emb}$, $\mathbf{c}_{emb}$ and $\mathbf{cap}_{emb}$ represent the corresponding word, character and capitalization embeddings of the word respectively . The vector representation is calculated by concatenating the above mentioned three vectors:

$$\mathbf{e}_w = \mathbf{w}_{emb} \circ \mathbf{c}_{emb} \circ \mathbf{cap}_{emb}$$

where $\mathbf{c}_{emb}$ is calculated by using a BiLSTM layer. For a character x, we randomly initialize a character embedding $\mathbf{c}_x$. To calculate the character embedding for a given word each character embedding $\mathbf{c}_i$ is fed into a BiLSTM which produces forward and backward character representations, $\mathbf{c}^f_{emb}$ and $\mathbf{c}^b_{emb}$. These are concatenated to produce the character embedding:

$$\mathbf{c}_{emb} = \mathbf{c}^f_{emb} \circ \mathbf{c}^b_{emb}$$

So the overall vector representation of a given word is:

$$\mathbf{e}_w = \mathbf{w}_{emb} \circ \mathbf{c}^f_{emb} \circ \mathbf{c}^b_{emb} \circ \mathbf{cap}_{emb}$$

These vector representations for the words are given as input to the first BiLSTM layer. The output of the first BiLSTM layer is the concatenations of the vectors created by going over a given sentence in forward and backward directions. The output of the LSTM layer has length $ldims$ for each direction, thus the output is of length $2 \times ldims$. The second BiLSTM layer thus takes as input a vector of size $2 \times ldims$ for each token in a sentence. The program again goes over these vector representations to create forward and vector outputs of size $ldims$. Let $\mathbf{e}^{lstm}_{w_i}$ represent the final lstm output for word w_i. Following the related work [4], four vectors are concatenated and given as input to the MLP called $\mathbf{MLP}_{arc}$, which outputs the score for a directed arc from w_i

and w_j:

$$\text{score}_{\text{arc}}(i,j) = \mathbf{MLP}_{\text{arc}}(\mathbf{e}^{lstm}_{w_i} \circ \mathbf{e}^{lstm}_{w_j} \circ (\mathbf{e}^{lstm}_{w_i} * \mathbf{e}^{lstm}_{w_j}) \circ (|\mathbf{e}^{lstm}_{w_i} - \mathbf{e}^{lstm}_{w_j}|)$$

where $(\mathbf{e}^{lstm}_{w_i} * \mathbf{e}^{lstm}_{w_j})$ and $|\mathbf{e}^{lstm}_{w_i} - \mathbf{e}^{lstm}_{w_j}|$ are element-wise multiplication and absolute element-wise difference, respectively. Loss $Loss_{arc}$ is calculated by maximizing the difference between the gold parse tree and the highest scoring incorrect parse tree following the related work [5].

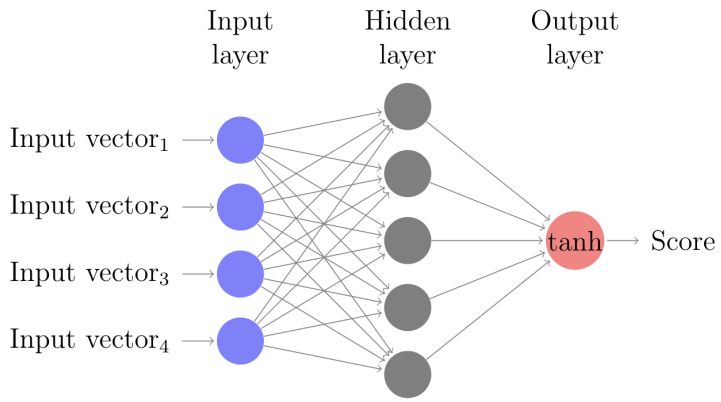

Figure 4.8. MLP for producing the arc scores for each pair of tokens. Each input vector$_i$ is explained above and each have size $2 \times ldims$.

For the second sub-task the system uses a separate multi layer perceptron to determine the relation type for each arc in the highest scoring dependency parse tree. The input to this multilayer is the same with the perceptron for the first task. Thus the input is of size $8 \times ldims$. This relation MLP, called $\mathbf{MLP}_{rel}$, takes the same input with the previous MLP and outputs a vector containing a score for each relation type:

$$\mathbf{scores}_{rel}(i,j) = \mathbf{MLP}_{rel}(\mathbf{e}^{lstm}_{w_i} \circ \mathbf{e}^{lstm}_{w_j} \circ (\mathbf{e}^{lstm}_{w_i} * \mathbf{e}^{lstm}_{w_j}) \circ |\mathbf{e}^{lstm}_{w_i} - (\mathbf{e}^{lstm}_{w_j}|)$$

Cross entropy loss $Loss_{rel}$ is computed over this score vector for gold label.

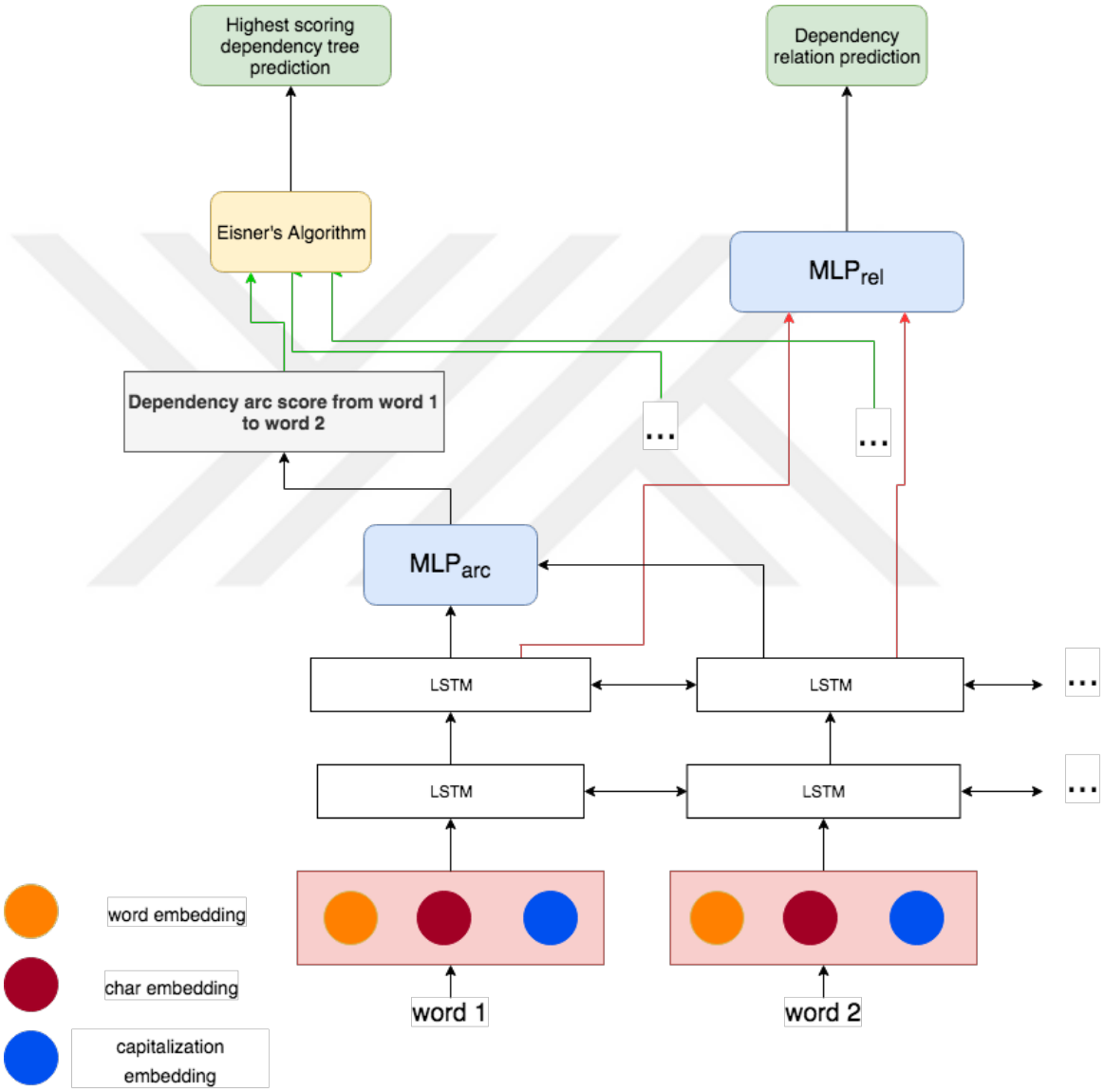

Figure 4.9. Dependency parser of the joint learning model.

The number of output nodes is equal to the number of possible relation types.This number is obtained either by observing the training dataset or by loading a pretrained model which also contain all such information extracted using the vocabulary generation algorithm explained previously. A hidden layer of 100 hidden units is used with 'tanh' activation function and the hidden layer is connected to the output layer. Finally a softmax layer is used to convert the scores of the output layer into probabilities which represent the likelihood of each relation type for the given arc from w_i to w_j. It is important to notice that, since this MLP is used only for predicting the labels of the arcs in the dependency parse tree these scores are not calculated for each word pair. Figure 4.10 shows the multi-layer perceptron for producing the likelihood score for each relation type for a given dependency arc. Each input vector$_i$ is explained above and each have size $2 \times ldims$. The number of the output nodes is equal to the number of possible dependency relations. The final softmax layer converts the scores into probabilities which add up to 1. The relation type with the highest probability is picked during the prediction mode.

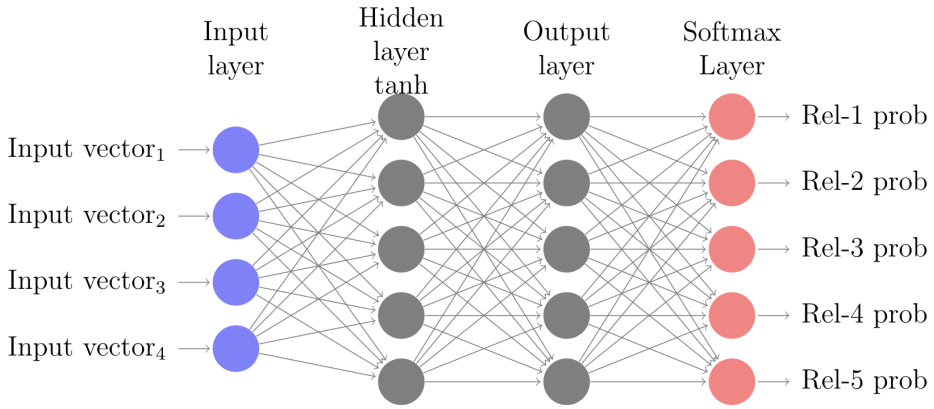

Figure 4.10. MLP for producing the likelihood score for each relation type for a given dependency arc from w_i to w_j.

The part of the overall architecture which is responsible for the dependency parsing task is shown in Figure 4.9. Until this point, the part of the system which is used of dependency parsing is described. This part of the system is made up of two BiLSTM layers connected to two multi layer perceptrons. The way the joint learning model

makes use of the dependency parser output is explained in the following subsection which describes the part of the system responsible for the named entity recognition task.

4.3.7. Named entity recognition module

This subsection explains the neural network for the named entity recognition task. This network is connected to the neural network responsible for the dependency parsing. This enables joint training of both networks simultaneously. Even though the named entity recognition dataset is not annotated for the dependency parsing task, the back propagated error for the named entity recognition task is used to tune the weights of the neural network responsible for the dependency parsing task.

The architecture of this part of the system is similar to the tagging component of the previously described dependency parsing model jPTDP [4], and is shown in Figure 4.11. In addition to the word embeddings, forward and backward character embeddings and the capitalization embedding, the first BiLSTM layer also takes as input the embedding of the relation type prediction of the previous network rel_{emb} for each token which is of size $rdims$ and set equal to 100 in our model. Thus the total size of the input to the first BiLSTM layer is $wdims + (3 \times cdims) + reldims$ where $wdims$ and $cdims$ correspond to the sizes of the word and character embeddings. The size of the capitalization embedding cap_{emb} is set equal to $cdims$ which is the reason $cdims$ is multiplied by three.

The first BiLSTM layer takes the above mentioned input vectors and again produces two output vectors by going over the input sentence in forward and backward directions. These two vectors are fed into a second BiLSTM layer. Thus the second BiLSTM layer takes as input the concatenation of forward and backward vectors for each , and outputs two vectors of size $ldims$. These vector representations are used for calculating the scores for each named entity tag in the named entity recognition task. The main advantage of these vectors over other vector representations is that they take into account the context information. Following the work in [57] these vector

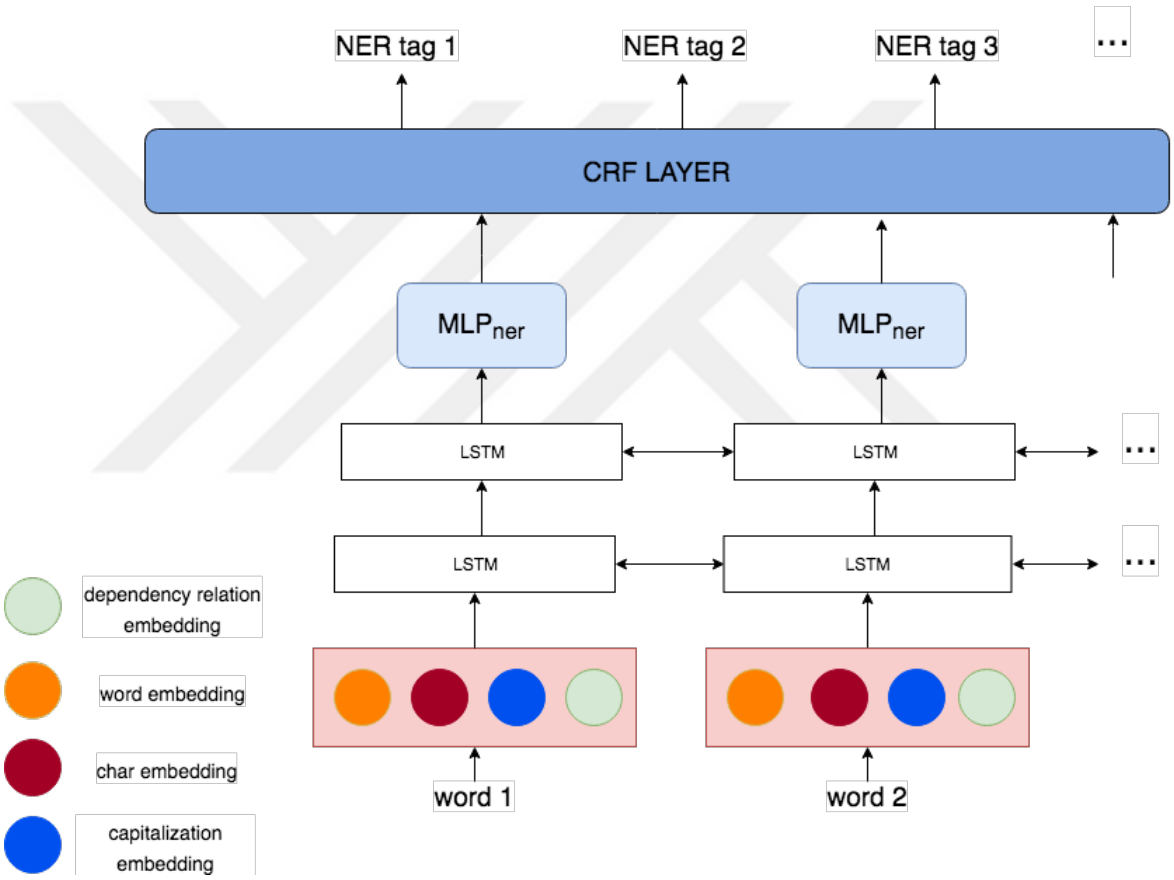

Figure 4.11. Named entity recognition component of the joint learning model.

representations are given to a neural network which outputs scores for each named entity tag. For a given sentence, output scores for each token is kept in a matrix. If we have a sentence with n tokens and if the named entity task contains t different named entity tags ,which in the case of this work is equal to 7 (B-PER, I-PER, B-LOC, I-LOC, B-ORG, I-ORG, O), then the matrix is of size (n, t). Each row of the matrix contains the scores for each possible named entity tag type for each word. Figure 4.12 shows the score matrix used in the NER task. Each row represent the scores for each possible named entity tag for a given token. These scores are decoded using the Viterbi algorithm during the prediction to find the optimal tag sequence. $score_{i,j}$ refers to the score for the i^{th} token having the j^{th} entity tag.

Each dependency relation type rel is represented with an embedding $\mathbf{e}_{rel}$. Given the relation type prediction $\mathbf{e}_{rel_i}$, of the dependency parsing component for a given word w_i, the word is represented by concatenating the embedding of this relation to the vector representation of the word:

$$\mathbf{e}^{ner}_{w_i} = \mathbf{e}_{w_i} \circ \mathbf{e}_{rel_i}$$

For a given input sentence with n words, we represent each sentence with the sequence of vector representations for each word $\mathbf{e}^{ner}_{w_{1:n}}$ and feed this sequence of vectors into LSTMs in forward and backward directions. The LSTM outputs vectors for each word w_k in a given sentence by taking into account the context in both directions:

$$\mathbf{v}_k = \text{LSTM}_f(\mathbf{e}^{ner}_{w_{1:k}}) \circ \text{LSTM}_b(\mathbf{e}^{ner}_{w_{n:k}})$$

These vectors of size $2 \times ldims$ are fed into a second LSTM layer which outputs the final vector representation of each token:

$$\mathbf{v}^{fin}_k = \text{LSTM}_f(v_{1:k}) \circ \text{LSTM}_b(v_{n:k})$$

Each vector $\mathbf{v}_k^{fin}$ are given as input to an MLP called $\mathbf{MLP}_{ner}$. The score matrix of size (n, t) is created where $score_{i,j}$ refers to the score for the i^{th} token having the j^{th} tag:

$$\mathbf{Scores}_{ner}(i) = \mathbf{MLP}_{ner}(\mathbf{v}_i^{fin})$$

During the prediction mode, these scores are normalized into probabilities to be used for finding the optimal tag sequence. For each tag for a given word probability is calculated by normalizing the scores produced for each entity tag type tag_j:

$$P(i, j) = \frac{exp(score(i, j))}{\sum_{(j' \in tags)} exp(score(i, j'))}$$

where $score(i, j)$ represents the score produced for tag_j for a given word w_i in a sentence.

Finally, we model the tag sequence jointly rather than predicting each label independently. For this a CRF layer [85] is used to find the highest scoring named entity tag sequence. Transitions between named entity tags are important because of the sequential nature of the task and CRFs are used frequently for the NER task [57,86,87]. Using the score vectors produced for each word and a randomly initialized score matrix $\mathbf{score}_{trans}(tag_i, tag_j)$ for transitions between each named entity tag, we find the optimal tag sequence. Viterbi decoding algorithm [88] is used to find the optimal tag sequence during prediction. Negative log likelihood loss $Loss_{ner}$ is used to calculate the loss of the gold label ner tag for each word in the sentence. The transition probabilities between entity tag types are included implicitly in $Loss_{ner}$ as the final prediction of the model is calculated using the Viterbi algorithm.

This (n, t) matrix is used to find the named entity tags for a given sentence. Named entity recognition is a sequence labeling task. In sequence labeling tasks the labels of each token is related with the labels of the neighboring words. In such tasks using a greedy algorithm and taking the highest scoring tag type for a given token without considering the score of a transition from one tag type to another performs

$$\begin{bmatrix}
score_{1,1} & score_{1,2} & \dots & \dots & \dots & \dots & \dots & score_{1,t} \\
\vdots & score_{2,2} & score_{2,3} & \dots & \dots & \dots & \dots & \vdots \\
\vdots & score_{3,2} & \dots & \dots & \dots & \dots & \dots & \vdots \\
\vdots & \dots & score_{4,3} & \dots & \ddots & \dots & \dots & \vdots \\
\vdots & \dots & \dots & \ddots & \ddots & \ddots & \dots & \vdots \\
\vdots & \dots & \dots & \dots & \ddots & \ddots & \dots & \vdots \\
\vdots & \dots & \dots & \dots & \dots & \ddots & \ddots & \vdots \\
score_{n,1} & \dots & \dots & \dots & \dots & \dots & \dots & score_{n,t}
\end{bmatrix}$$

Figure 4.12. $n \times t$ score matrix for a given sentence for the named entity recognition task.

poorly. For this reason, by following the related work an additional CRF-layer is used to find the optimal named entity tag sequence for a given sentence. The optimal tag sequence is found by using the Viterbi algorithm. The input to the Viterbi algorithm is the above described matrix in addition to a randomly initialized $(t+2, t+2)$ matrix which correspond to the tag transition scores. 2 additional entries is used for transitions from 'BOS' and 'EOS' which correspond to beginning and end of sentence tags in order to be able to take into account the transition probabilities properly. It must be noted that some of the entries in the matrix correspond to invalid tag transitions such as a transition from 'BOS' to 'EOS'. These transition scores does not cause a problem as the system quickly learns that such transitions never occur by observing the training dataset. The tag transitions are initialized randomly at the beginning of the training when the model is first initialized together with all other weights of the neural networks.

The complete architecture for the joint learner model is shown in Figure 4.13. This architecture is the union of the two parts of the architecture shown in Figure 4.9 and Figure 4.11. First the dependency parsing part of the network is fed with inputs and then the part of the model responsible from named entity recognition is fed with the output of the dependency parser together with other input vectors.

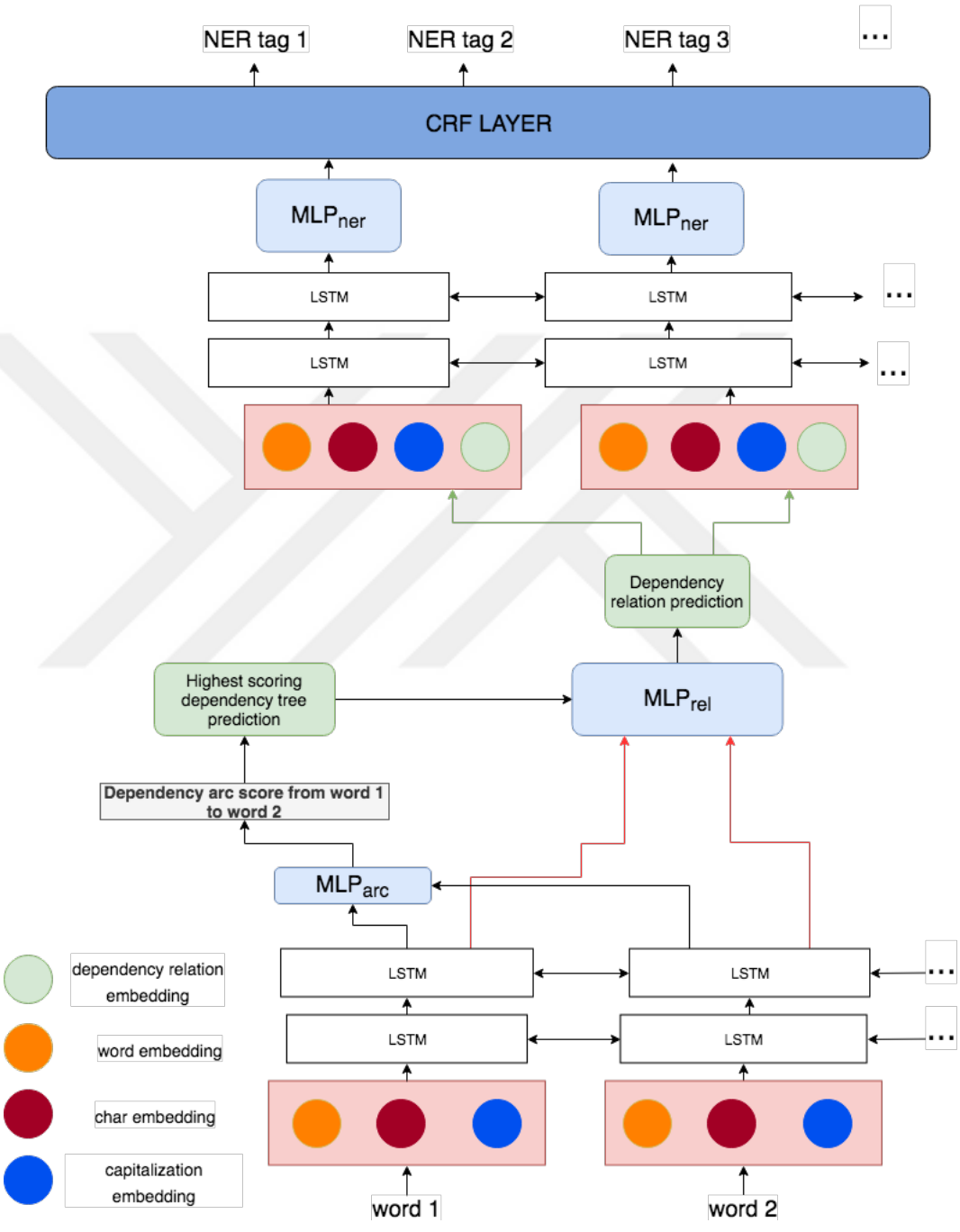

Figure 4.13. Complete architecture for the joint learner model.

4.3.8. Decoding Algorithms

This section explains several algorithms that have critical importance for the system in detail.

Eisner's decoding algorithm [82] is used during the dependency parsing to find the highest scoring dependency tree that spans the whole sentence. Eisner's decoding algorithm is a dynamic programming algorithm that always finds the best tree for a given scoring matrix which contains scores for arcs between each pair of words in a given sentence.

The Viterbi decoding algorithm [88] is used in the prediction mode of the joint learning model. Viterbi algorithm is a popular algorithm for finding the best scoring sequence of states where there is state transition scores in addition to the static state scores. Figure 4.3.8 shows a pseudocode for the algorithm. The Viterbi-path is equivalent to the path obtained by choosing the highest scoring state for each element whenever there is no state transition available or all transition scores are equal. Viterbi algorithm is a dynamic programming algorithm which finds the optimal solution faster than a trivial algorithm which compares all possible state sequences by using more memory.

The main idea behind the algorithm is to start from the initial state and for each state finding the optimal state sequence until that state by considering the state transition scores. If we let N number of possible states that an element in the sequence can be in then for each new element at position i, the algorithm only calculates N scores because the best scoring state sequences are already calculated before arriving to this new element, rather than making $N \times (i-1)$ as in the case of a trivial solution.

Input. $N \times T$ matrix 'S' and $T \times T$ matrix 'A' for state-tag scores and tag-tag transition scores respectively where N is the number of elements in a sequence and T is the number of possible tag types.

Output. An N element long array 'V' which represents the 'Viterbi-path'.

```
best_parents = list([-1 for x in range(T)])
best_scores = S[0]
for i = 1 to N do
   Initialize best_score , best_parent_inds same way as above
   for k = 0 to (T − 1) do
      best_score[k] = max(S[i][k]+A[j][k]) where j is from 0 to T-1
      best_parent_inds[k] = argmax(S[i][k]+A[j][k]) among j from 0 to T-1
   end for
   best_scores.append(best_score)
   best_parents.append(best_parent_inds)
end for
final = argmax(best_scores[-1])
V = [final]
parent = best_parents(-1)[final]
for i = N − 2 to 1 do
   V.append(best_parents[i][parent])
   parent = best_parents[i][parent]
end for
Return reversed(V) {Return the best parents in reverse direction.}
```

Figure 4.14. Viterbi Algorithm. The algorithm is written in pseudo-code format similar to the python programming language syntax.

5. EXPERIMENTS AND RESULTS

This chapter explains the experiments done and the results obtained in detail. The experiments done can be grouped into three phases: Re-implementation, exploration and testing. Accordingly, this chapter is divided into three sections and the experiments done for each category will be explained. Re-implementation and exploration phases contain experiments only for the NER task. The testing phase include experiments for the final proposed joint learning model. Thus this phase includes experiments for both NER and Dependency Parsing tasks. First two phases aim at getting familiar with the previous models and finding ways to improve on the previous implemented NER systems. At the end of these two phases several hypotheses are put forward mainly regarding the improvement that can be obtained by using the dependency parsing information for each word for the task of NER. The experiments done in the final testing phase is for testing these hypotheses by using various models as explained in detail, in the methodology chapter.

5.1. Re-implementation Phase

The initial experiments are conducted for obtaining the previously achieved results for the NER task for Turkish. A CRF-based model is used in this stage, and the features used as inputs to the CRF-based model are obtained from the previous works [49, 52, 55]. The main motivations of this phase are to put in practice the extensive analysis done during the literature survey and to get familiar with the NER task and commonly used techniques. We have re-implemented the model described in [52] and showed that by using new features we can obtain improvements over the version of their model which does not make use of gazetteers. This section explains in detail the features used and the results obtained.

During this stage 'CRF++' toolkit is used for conducting experiments. It is a CRF-based machine learning toolkit for sequence labeling. The toolkit is convenient and allows conducting experiments using different configurations with ease. This al-

lowed starting with simple models, which only make use of the surface form of the words to make predictions about the NER label, and gradually increasing the complexity of the model by taking into account more features and larger window sizes. This is a frequently used approach in feature-based models as considering all candidate features at once makes it difficult to track the useful features. It is a common practice to start with the most fundamental feature such as the surface form of a token and gradually take into account other less frequently used features. Some of them turn out to increase the complexity of the model without bringing a significant improvement, some of the features degrade the performance as they mislead the model and finally some turn out to be useful and bring significant improvements. The latter features are then considered as core features for the task and new features are tested. This is a commonly used technique which works well in most cases. On the other hand, certain feature combinations are shown to degrade the performance even though the addition of each feature separately improves the performance. In order to detect such combinations and all possible feature combinations must be tested extensively which can not be feasible in models with a large number of candidate features.

5.1.1. Features Used

This part explains the features used in the re-implementation phase with the CRF++ toolkit. As mentioned these features are added one by one and the performance change obtained is observed. These features require feature engineering and each of them are included using heuristics rather than using automatic feature extraction techniques. Yet such features are also shown to be important as they contain highly specialized information about the task. Below is a list of all the features used at this phase. As mentioned above, these features are not considered altogether but rather included to the model one by one. A third party morphological analyzer [89] is used to annotate the dataset for several features.

(i) **Surface form** : The surface form of each word is the first feature. This is the most fundamental especially for CRF-based models which use complete matching of feature values. Yet, compared to word embeddings this method performs worse

as it can not detect the similarity between tokens that are not the same.

(ii) **Initial POS tag** : The POS tag prediction for the lemma form of the word by a third party morphological analyzer. As Turkish is a morphologically rich language, words may undergo several POS changes as different suffices are appended. Lemmatization is done to extract the stem of the word and then the POS tag prediction of the analyzer is used as a feature. This feature turned out to degrade the performance of the system.

(iii) **Final POS tag** : The POS tag prediction for the complete surface form of the word by the third party morphological analyzer. Experiments showed that Final POS tag increases the performance.

(iv) **Capitalization Feature** : A four valued feature giving information about the lowercase and uppercase letters. Capitalization feature is a fundamental feature for the NER task for Turkish as almost all named entities are capitalized or at least supposed to be capitalized. This feature significantly increases the performance in languages like Turkish.

- 0 : All lowercase letters e.g. araba
- 1 : Only the first letter is uppercase, also called proper name case e.g. Istanbul
- 2 : All letters are uppercase , e.g. ABD
- 3 : Mixed capitalization : e.g. LaPalombara

(v) **Stem of the word** : A morphological analyzer is used to predict the morphological analysis of the word and the root word of the prediction is used. The main motivation of this feature is to overcome the data sparsity which occur frequently for morphologically rich languages. As the CRF++ use complete matching it can not detect that two words are different versions of the same stem word with different suffixes. By using the stem form the model more complete matches are obtained. The stem form is also obtained from the morphological analysis of the analyzer.

(vi) **Start of sentence** : This is a binary feature which denotes whether the token is the first word of a sentence. This is used to tackle the ambiguity caused by the case feature, because in Turkish, as in the case for many other languages, the

first word in each sentence has the first letter uppercased by default.

(vii) **Proper noun** : The third party morphological analyzer also differentiates between regular nouns and proper nouns during prediction. This binary feature takes the value 1 if the morphological analyzer predicts the word to be a proper noun and 0 otherwise. This prediction is not necessarily through but used as a supporting evidence.

(viii) **Acronym feature** : Binary feature denoting whether the morphological analyzer predicts the word to be an acronym or not, e.g. ABD - Acro and Istanbul - Notacro.

(ix) **Nominal feature** : This is a complex feature, which is a combination of three atomic features. Observing the morphological analyses of the labeled entities in the training set showed that, most of them share the following three features: They are capitalized, they are in their stem form so there are no inflections appended to the token and finally the morphological analyzer predicts them to be a nominal denoted as 'Nom'. This nominal features is a binary feature which takes the value 'Nom' only if the above mentioned three conditions are satisfied simultaneously for a given token and 'Notnom' otherwise.

(x) **Final suffix** : The final suffix of the word is given in the morphological analysis format. If the word does not have any suffix 'None' value is given. In order to overcome the data sparsity of complete matching the surface form of the suffix is not used. For example the final suffix of the word 'kalitesinin' is 'nin' but the feature value is 'NHn' where the uppercased letters denote the letters are subject to change in other words but the suffix itself is the same. By using this feature CRF++ can detect the words that have the same suffix even though the surface form of them differ as in the case of 'kalitesinin' - 'NHn' and 'ormanın' - 'NHn'.

In order to increase readability the names of the features explained previously are shortened as follows :

- Surface form: Surf
- POS tag: POS

- Capitalization feature: Cap
- Stem form: Stem
- Start of sentence feature: SS
- Proper noun: Prop
- Acronym feature: Acro
- Nominal feature: Nom
- Final suffix: Suf

5.1.2. Results

As explained above, simple models are trained and tested initially and then gradually the model complexity is increased. Four features given below are selected for the base model trained using the 'CRF++' tool and new features are tested one by one. The four features are as follows:

- surface form
- final POS tag
- Capitalization
- Stem of the word

The model with the above given features are called 'CRF++BM' (BM as an intuitive abbreviation for 'Baseline Model'). All the models make use of a window size of 2 unless specified. Only one the models is tested with a window size of 3 and it did not a significant improvement, even though the training time increased. So the rest of the models are trained with a window size of 2. The results of this phase is given in Table 5.1. '+' sign refers to the addition of a feature abbreviated in the above explained way.

Table 5.1. Initial results obtained using the 'CRF++' toolkit together with the training times.

	MUC	**F1-Measure**	**Training Time**
CRF++BM	0.919	0.889	3,000s
+SS	0.921	0.889	3,300s
+Prop	0.924	0.896	3,400s
+Acro	0.924	0.897	3,900s
+Nom	0.925	0.896	4,800s

5.2. Exploration Phase

Previous phase aimed at starting with a simple model and build up gradually by reimplementing the previous work done for the NER task for Turkish. In the previous phase, similar features and the same machine learning toolkit is used with a previous work [52]. In this phase 'Wapiti' toolkit [75] is used instead of 'CRF++' as it includes more training options.

The dataset used in this phase is explained in the dataset chapter and an example annotated sentence is given as well.

Dependency information is used frequently to improve the performance of NER systems [28, 42]. Sasano *et al.* [42] uses the output of a parser such as the head of a word and the relation as features given to an SVM based model. As they do not use gold standard labels for the dependency features the performance of the parser puts an upper limit to the performance. The parser used in their work is a Japanese parser called 'KNP' (Kurohashi Nagao Parser) [90]. An efficient method for using dependency information for NER systems is explained in [41] which attempts using the global structure of the dependency tree in addition to the dependency relation information.

Through these observations, dependency parsing information is explored in this stage by using the 'Wapiti' toolkit. Using the dependency parsing information for the NER task for Turkish is not a common methodology. Many of the feature based systems for Turkish make use of the similar feature sets. As Turkish is a morphologically rich language, most of the feature sets include morphological features in order to reflect the underlying syntactic information for a given word. Yet dependency parsing information which also contains important clues about the semantic-level NER task is not used frequently.

The dependency parser named 'jPTDP' [4] is used to tag the NER dataset explained above. The final two fields before the last gold NER label seen in the Figure 3.2 is the predictions made by this system. The system is submitted to the CoNLL 2018 Shared Task which focuses on creating multilingual dependency parsers. This system is trained and tested for many languages including Turkish. We used the pretrained model for Turkish to annotate the datasets and trained various models which take into account these features as well as the previous features used in the re-implementation phase. In this phase, we have also used several regular expression features which capture the character level information for a given word. Below is a list of newly added features that are frequently used in the feature based systems:

- 1,2,3,4 character long suffixes
- 1,2,3,4 character long prefixes
- punctuation feature: binary feature which takes the value of 1 if the first character of the token is a punctuation mark and 0 otherwise.
- all-punctuation feature: binary feature which takes the value of 1 if the all characters of the token a punctuation mark and 0 otherwise.
- inside-punctuation feature: binary feature which takes the value of 1 if a punctuation character occurs inside of the token. So it must be preceded and succeeded with non-punctuation characters.
- number feature: binary feature with value 1 if the token includes a digit.
- all-number feature: binary feature with value 1 if the token only includes digits.

The addition of these sub-word level features are straight forward with the 'Wapiti toolkit', as it also supports a simple kind of regular expression matching. Addition of the regular expressions to the pattern list is enough to automatically generate all features values for each token.

Finally, the two predictions made by the 'jPTDP' model for dependency parsing is used as features in this phase which are as follows:

- index of the head: This is the index of the word which is predicted to be the head of the word in question in the dependency parse tree. The root word of a sentence does not have a head word, so by default it takes the value '-1'. The indexes start from '1' so the index of the first word is '1' rather than '0'.
- dependency relation: The predicted relation between the word in question and its predicted head word. The parser first predicts a parse tree and then for each predicted arc in the parse tree, makes a second prediction about the relation type. The relation types are determined by the Universal Dependencies.

5.2.1. Results

First, the experiments done with 'CRF++' is redone with the 'Wapiti' toolkit. During these experiments, the tool does not make use of the dependency features but only make use of the features used previously and the sub-word level regex features. Each experiment will be explained in detail.

Also the models in this section make use of two new features called: 'index of the head' and 'dependency relation' which are referred to as 'Depind' and 'Deprel' in the following sections.

Again in this section, a similar approach is used and features are included one-by-one. First model trained using 'Wapiti' only takes into account the surface form and the POS tag feature with a window size of 2. In addition the above mentioned sub-word level regex features are used in all of the experiments done using 'Wapiti'.

This model is called the base model (WapitiBM) and each feature is added to this model one-by-one.

Second trained model also takes into account capitalization feature, stem form of the word feature and start of sentence feature. In total, this model takes into account the first 5 fields of the dataset explained previously. As can be seen on Table 5.2 that gives the results obtained for all 'Wapiti' models there is a slight decrease in performance when more features are included. Yet, this is not a significant performance difference and the features are preserved.

Third trained model takes into account three additional features previously used by the 'CRF++' model. This model outperforms the previously trained two models slightly and shows that taking into account all of the features gives the best performing model. Yet, the exhaustive search of previously used feature combinations is not done as the primary aim of this section is not to obtain a state-of-the-art performing model. Rather the main aim is to show that adding certain features like regex features and most importantly the dependency related features give relative performance improvements.

Fourth model takes into account all features used previously in the reimplementation phase. Taking into account the final suffix increased the performance slightly.

Finally the dependency relation features are taken into account to train the fifth 'Wapiti' model. All of the results explained above are given in Table 5.2. A performance drop is observed when all the dependency related features are included. Yet this drop is not significant enough to suggest that such information degrades the performance.

In order to better explore the change of performance when these new features are included, we have performed extensive experiments testing all combinations of all model types and training algorithms provided by the 'Wapiti' toolkit. Following section is devoted for explaining the results obtained from these extensive experiments.

Table 5.2. Early results using Wapiti toolkit.

	Precision	**Recall**	**F1-Measure**
WapitiBM	0.914	0.879	0.896
+Cap+Stem+SS	0.916	0.873	0.894
+Prop+Acro+Nom	0.920	0.879	0.899
+Suf	0.920	0.880	0.900
+Depind+Deprel	0.917	0.881	0.899

5.2.2. Results for Extensive Experiments

This section explains the extensive search done over the following parameters: features used, training model type and training algorithm. Previous experiments using the 'Wapiti' toolkit made use of the default setting of the tool. In the default setting the training model is Conditional Random Fields (CRF) and the optimization algorithm is Limited-memory BFGS which approximates the Broyden-Fletcher-Goldfarb-Shannon (BFGS) algorithm using a limited computer memory [91]. The algorithm is the most popular algorithm from a family of methods called Quasi-Newton methods [92].

In this part all possible training model and optimization algorithm combinations are tried, and the results are given in a Table 5.3. The models make use of all the 11 feature fields available in the final version of the dataset. F1-Measure scores are given for each entity type and for overall together with the MUC score. Some variations are made on the feature list being used as well. Initial experiments showed that using 'blockwise coordinate descent' (bcd), 'stochastic gradient descent with L1 regularization' (sgd-l1), 'rprop-' which is a variant of 'rprop+' without backtracking as mentioned in the description of the toolkit [75] is not suitable for our task.

Thus the experiments done here only made use of the following optimization algorithms :

- l-bfgs
- rprop
- rprop+

And all available training models are tested. They are as follows:

- maxent : Maximum Entropy Model
- memm : Maximum Entropy Markov Model
- crf : Conditional Random Fields Modes

Table 5.3. Exploration of combinations of all training models and optimization algorithms.

Model	Optimization Algorithm	**PER**	**LOC**	**ORG**	**Overall F1**	**MUC**
crf	l-bfgs	0.909	0.898	0.883	0.899	0.919
	rprop	0.904	0.892	0.860	0.887	0.904
	rprop+	0.903	0.892	0.860	0.887	0.905
maxent	l-bfgs	0.910	0.890	0.865	0.892	0.915
	rprop	0.913	0.887	0.847	0.887	0.908
	rprop+	0.913	0.887	0.847	0.887	0.908
memm	l-bfgs	0.910	0.879	0.845	0.883	0.909
	rprop	0.911	0.883	0.826	0.879	0.901
	rprop+	0.911	0.883	0.826	0.879	0.901

During the early results of this exploration phase, we have observed that the index feature for the head of a dependency arc given directly to the system has several issues and cause the performance to drop. Also we have observed that the best performing 'training model + optimization algorithm' combination is 'crf + l-bfgs'. This is the

default combination of the 'Wapiti' toolkit and we have validated this choice for the default setting. These two observations resulted in updating the feature set and setting the model to perform in the default mode. The resulting setting is the final version of the model we have used in this exploration phase. The result for this version is given in Table 5.4. This version of the model outperforms all the previous models both in this phase and in the re-implementation phase. This version does not make use of the head index feature but makes use of the dependency relation feature. Thus, this result successfully shows that using the dependency relation feature brings a relative performance improvement. This final version is named 'WapitiFIN'.

Table 5.4. This table shows the results for the final version of the model with 'Wapiti' toolkit.

	PER	**LOC**	**ORG**	**Overall F1**	**MUC**
WapitiFIN	0.916	0.896	0.886	0.902	0.923

5.3. Joint Learner Phase

The experiments of this phase are done for testing the final system implemented which attempts joint learning of the dependency parse tree and named entity tags for a given sentence. The architecture and the model is explained previously on Chapter 4. The system has different variations and trained with different configurations in order to be confident with the results obtained. To test the validity of the hypotheses made, different variations of the model are implemented. These variations are as follows:

(i) Named entity recognition only model: This model is the baseline model for the NER task. The architecture of this model is the same with the other variations. The only difference is that the model does not make use of the dependency parser and its output. This model only tackles the NER task. Thus only the NER results are given for this model.

(ii) Joint learning model using single dataset annotated for both tasks: This model includes a dependency parser as well as the above mentioned architecture for the NER task. This model assumes that there is a dataset which gold label tags for both tasks available. So a single dataset is used and joint learning is done on the same dataset. Thus, results of this model include both the dependency parser related results as well as the named entity recognition related results. Since there is no available dataset which includes gold standard labels for both tasks for Turkish, we have used the version of the NER dataset tagged for dependency features using a third party tool. This version of the dataset was previously used during the exploration phase.

(iii) Joint learning model using separate datasets annotated for each task: This model is the same, in terms of the architecture, with the above mentioned model. The main difference is that it can receive different datasets for each task. Thus this model also tackles both tasks and the results of this model include results related for both dependency parsing and named entity recognition. This version of this model is the main contribution of this work.

This section is divided into three parts corresponding to the above mentioned variations of the final model. The main motivation of using these variations is to compare the relative performances of the final proposed joint learning model with other variations on NER performance.

5.3.1. Results for NER only model

This section contains the results obtained for the NER only model. The architecture of this model is the same, but the part of the neural network responsible for the dependency parsing task is not used. One can think of this model with the dependency part of the network 'silenced'. We have used the same training and development sets in all these models to have consistency. The dataset used is explained in the 'Datasets' section. Figure 5.1 shows the results obtained for running this model for 40 epochs. Best F1-Measure of **0.915** is obtained over the development set in the 34^{th} epoch. This

score outperforms the F1-Measure scores obtained in all previous experiments using the CRF-based third-party tools. So the neural network model implemented in this study brings a relative improvement over the CRF-based tools for this dataset.

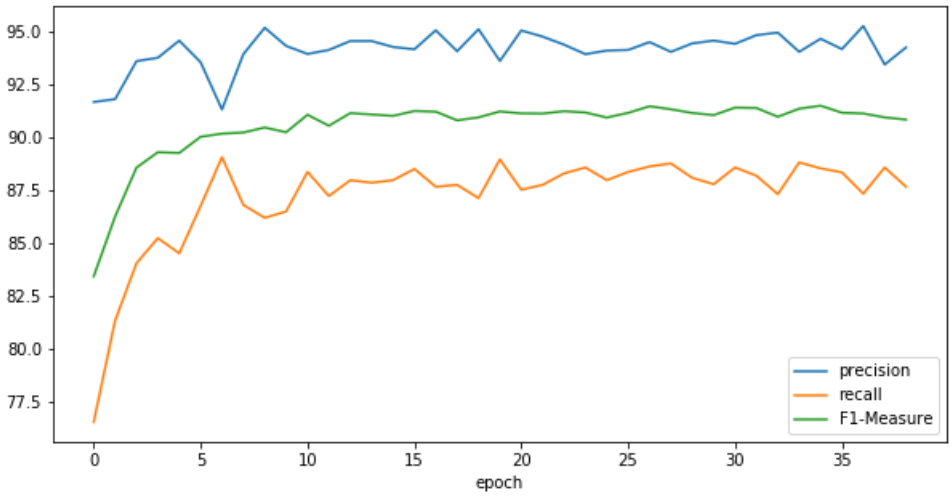

Figure 5.1. Results for the NER only model in each epoch tested on the development set.

5.3.2. Results for joint learner on a single dataset

The results of this section are obtained for running the final version of the joint learner model on the same dataset for both tasks. The NER dataset used by the previous NER only model is annotated using a separate dependency parsing tool [4]. So at each epoch loss is calculated by taking into account the loss obtained from the NER predictions and also the dependency parsing predictions. This is the main difference between this model and the final proposed joint learner model. As the corpus used for these experiments are not annotated with gold standard labels for the dependency parsing, the results should not be viewed as the final performance of this version. Future work includes finding datasets which includes gold standard labels for both tasks and repeating these experiments on these datasets.

Figure 5.2 and Figure 5.3 show the results after each epoch for the joint learning model on jointly annotated dataset for dependency parsing and NER tasks respectively. The best results obtained for each task and the epoch number are given in Table 5.5. Best score for dependency parsing is calculated by taking the average of LAS and UAS accuracy scores for each epoch. For the NER task, the best result refers to the result with the highest F1-Measure score. The high dependency parser results are probably due to the fact that the architecture of the tool used for annotating the dataset is quite similar to the dependency parser implemented in our system. This similarity can make it easier for the system to quickly learn and mimic the parser used.

Table 5.5. Best results for the joint learning model on a single dataset. DEP refers to the dependency parser score.

Task Name	Metric	**Best Results**	**Best Epoch**
DEP	**Average LAS - UAS Accuracy**	0.760	13
NER	**F1-Measure**	0.878	12

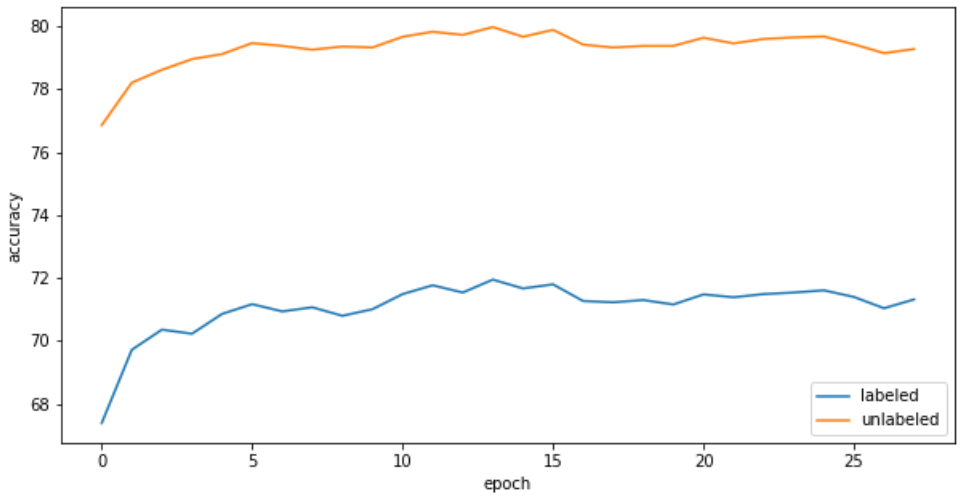

Figure 5.2. Results for the dependency parsing task of the joint model on a single dataset annotated jointly.

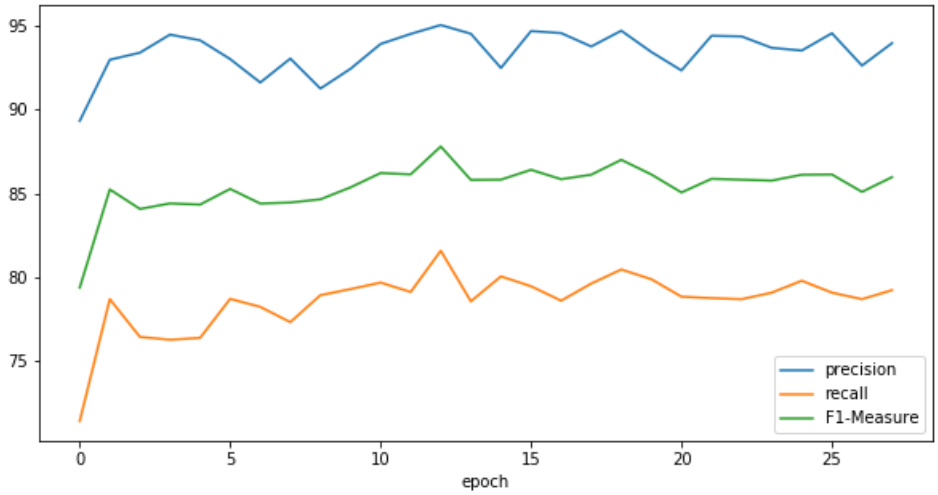

Figure 5.3. Results for the NER task of the joint model on a single dataset annotated jointly.

5.3.3. Results for joint learner on separate datasets

This section explains the results for the final version of the joint learning model. Thus, the experiments done in this section is more extensive compared to the experiments for the previous two versions. The final version is trained using various parameter combinations on the same dataset. All of these parameter combinations is explained in detail together with the results obtained.

First, we give the results for the default configuration. Figure 5.4 and Figure 5.5 show the results obtained at each epoch for dependency parsing and named entity recognition tasks respectively. The parameters of the default configuration are set using the previous work on joint learning [4, 57]. The list of parameters together with their default values is given in Table 5.6. This will be followed by the results of the grid search done over various parameter value combinations. Grid search is done to find a better performing parameter combination which does not increase the complexity of the model substantially. For the evaluation of the performance of the both parts, frequently used evaluation metrics are used. For the dependency task

'Labeled Attachment Score' (LAS) and 'Unlabeled Attachment Score' (UAS) is used. For the named entity recognition task CoNLL evaluation metric is used which is the standard metric for the NER task. Partial matching is ignored in the CoNLL evaluation metric for NER, so performance scores considering the partial matches are only given on the development sets.

Table 5.6. Parameters of the joint learning model together with the default values.

Parameter Name	**Default Value**
Word embedding size	100
Character embedding size	50
Capitalization feature embedding size	50
Relation embedding size	100
Hidden units	100
Activation function	tanh
Lstm layers	2
Lstm dimensions	128
Enable dependency parsing	True
Enable viterbi decoding	True

5.3.4. Experiments with different parameter configurations

This section gives the results obtained for training the system with different parameter combinations and testing on the development sets. Experiments for different values of the 'character embedding size' parameter showed that the default setting gives the best performance. Thus for the following experiments this parameter always has the default value. Table 5.7 shows the results obtained for different configurations for the 'word embedding dimension size' and 'lstm dimensions'. Yet one can observe that best results obtained for the NER task has the smallest lstm dimension size which can be counter-intuitive. F1-Measure and Average of Labeled Attachment Score (LAS) and Unlabeled Attachement Score (UAS) are given for NER and DEP tasks respectively.

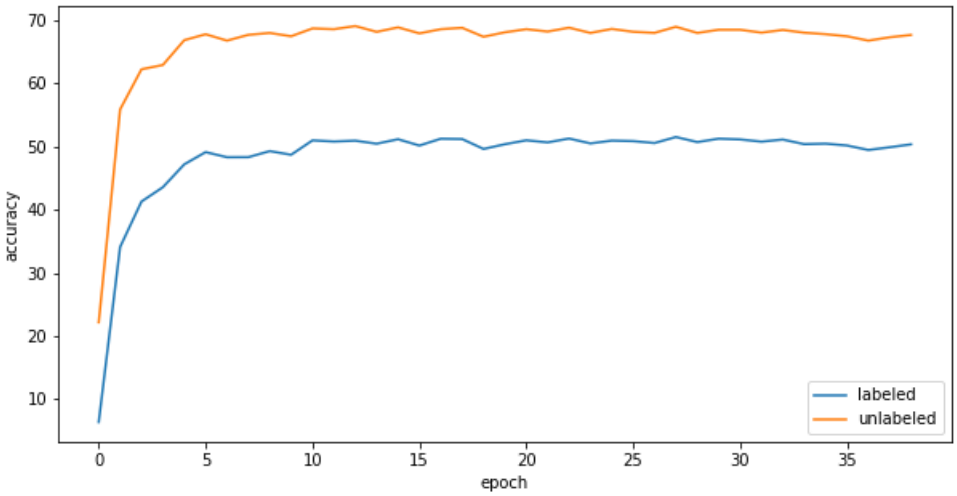

Figure 5.4. Dependency parsing scores for joint learner on separate datasets on the development set.

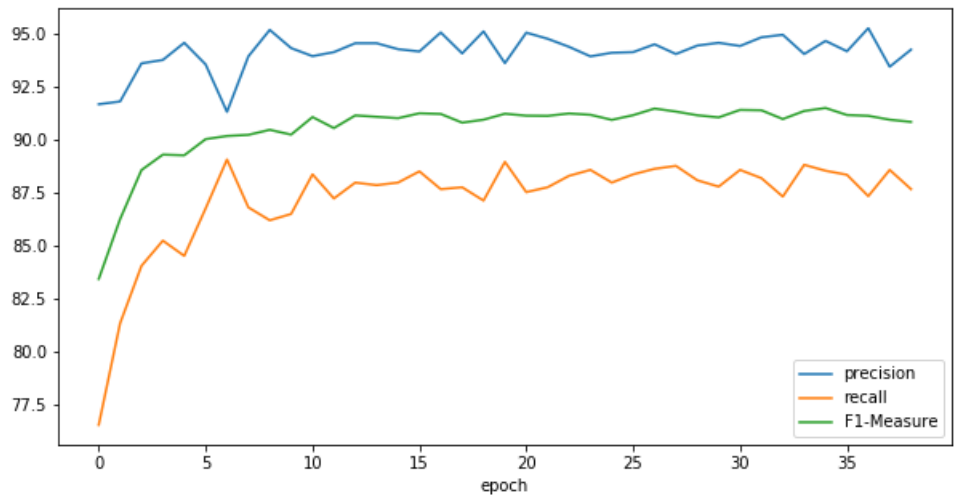

Figure 5.5. NER scores for joint learner on separate datasets on the development set.

All results are quite close which makes it difficult to conclude that one configuration outperforms others significantly. Best F1-Measure scores are obtained for the NER task with the following two configurations: (word embedding dim: 100 , lstm dim: 64) and (word embedding dim: 150 , lstm dim: 64).

Table 5.7. Results for joint learning model with different parameter combinations.

wemb dim	**lstm dim**	**NER F1-Measure**	**Average LAS & UAS**
50	64	0.905	0.587
	128	0.908	0.580
	256	0.904	0.588
100	64	**0.909**	0.595
	128	0.904	**0.600**
	256	0.907	0.593
150	64	**0.909**	0.594
	128	0.908	0.595
	256	0.908	0.591

5.3.5. Final experiments on the test set

This section gives the results obtained on the test set of the NER dataset used throughout experiments with the joint learner model. Previous experiments were done on the development set to have an idea about the relative performances of different models with various configurations. In this section we compare performance of each model using the CoNLL evaluation metric. Previous results are calculated taking into account partial matches of entities so every match between a gold label and a prediction is counted without checking whether a multi-word entity is completely predicted by the system or not. Yet, in order to have consistent results we use the CoNLL metric to evaluate the performance of each system. The CoNLL 2003 Shared Task provides a ready-to-use program which evaluates the predictions of a system using the CoNLL evaluation metric. This ensures that each system is evaluated by using the exact same

code so that the results obtained will be consistent. We have used that same CoNLL evaluation program to evaluate the performance of each version of the joint learner in this section. Table 5.8 shows the results for the joint learning model using separate datasets, with the following parameter combination:

- Activation function: tanh
- Word Embedding size: 100
- Lstm output size: 128

Table 5.8. Results for the joint learning model using separate datasets on test set.

	Precision	**Recall**	**F1-Measure**
PER	86.29	86.66	86.48
LOC	86.84	85.89	86.36
ORG	80.97	76.41	78.63
Overall	85.23	83.91	84.56

Table 5.9 shows the results for the joint learning model which uses a single dataset annotated for both tasks.

Next, the NER only model is tested on the test set. This model has a faster training time because the model is trained using only the NER dataset. So in each

Table 5.9. Results for the joint learning model using a single dataset on the test set.

	Precision	**Recall**	**F1-Measure**
PER	92.43	77.82	84.50
LOC	77.07	87.53	81.97
ORG	81.21	75.67	78.34
Overall	83.78	80.51	82.11

epoch the program observes less sentences compared to the joint learner model. This enables training the model for longer epochs. Table 5.10 shows the results for the NER only model on the test set. NER only model outperforms the joint learning model on the test set with the CoNLL evaluation metric.

Table 5.10. Results for the NER only model on the test set for each NE type.

	Precision	**Recall**	**F1-Measure**
PER	89.74	89.89	89.81
LOC	89.95	90.04	89.99
ORG	87.56	86.65	87.10
Overall	89.28	89.15	89.21

Results show that our suggested model brings a relative improvement over the joint learning model which uses a single dataset, as can be seen from Table 5.8 and Table 5.9. The joint learning model using different datasets for each task outperforms the other model on each entity types. The results on the test obtained using the NER only model suggest updating the dependency parsing information used by the NER component of the model. Even though we have shown a relative improvement with the feature based models by using the dependency relation information, same improvements are not observed with our final joint learning model. Yet, the results obtained are comparable to the NER only model and we have shown that both tasks can be learnt jointly using different datasets without sacrificing from performance.

We would like to further emphasize the difference between the two versions of the joint learner model. Apart from the performance improvement, there are several advantages of the novel method that we propose over the conventional joint learning model:

- The proposed model works with gold labeled datasets for each task whenever a dataset is available for each task. On the other hand, conventional systems require a single dataset that is annotated for all task to be available.

- The proposed model relieves systems from relying on third-party tools for annotating a dataset. This is a crucial step towards creating multi-purpose and generalizable NLP systems.
- By making use of different datasets the proposed model is being trained on a larger dataset. Thus, the common parts of the overall architecture generalize the given language better. Also using data from with multiple sources prevents overfitting to a single dataset which is a major problem for most NLP systems.

6. CONCLUSION AND FUTURE WORK

In this study, we started with the analysis of some available tools and improved their performance by using the dependency parsing information. Next, we have implemented a novel neural network based system that jointly learns dependency parsing and named entity recognition tasks using separate datasets. Our results show that the joint learner outperforms several previous work on the Turkish dataset and has comparable results with the version of the system that focus only on the NER task.

Thus far, we have focused on the implementation of the novel joint learner system. Future work includes focusing on improving the performance of this system in several ways. Using embedding representations for outputs instead of the output directly is shown to increase performance of NLP systems for various tasks [29, 93]. Representation learning is applied to the coarse label NER task in various studies [1, 94, 95]. Yogatama *et al.* [29] show that using output representations of NER labels increase the performance for both coarse-grained and fine-grained NER tasks. Future work includes learning the representations of output labels to calculate losses in a more robust way.

We will also try changing the proposed architecture to take into account the dependency prediction of the dependency parsing component in different ways. We will implement a joint learning model that takes into account the head of the dependency arc for a given word as well as the dependency relation.

Future work also includes combining our joint learner model with the model of Gungor *et al.* [57] to create a system that jointly learners more than two tasks. Our final objective is to have a system that jointly learns many NLP tasks by using separate datasets for each task to solve the joint labeling problem completely. The systems will not be limited to the Turkish language but will be applicable to all languages with datasets available.

REFERENCES

1. Collobert, R., J. Weston, L. Bottou *et al.*, "Natural language processing (almost) from scratch", *Journal of Machine Learning Research*, Vol. 12, No. Aug, pp. 2493–2537, 2011.

2. Hashimoto, K., C. Xiong, Y. Tsuruoka *et al.*, "A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks", *EMNLP*, 2017.

3. Chen, Q., X. Zhu, Z. Ling *et al.*, "Enhancing and combining sequential and tree LSTM for natural language inference", *arXiv preprint arXiv:1609.06038*, 2016.

4. Nguyen, D. Q., M. Dras and M. Johnson, "A Novel Neural Network Model for Joint POS Tagging and Graph-based Dependency Parsing", *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, 2017, `http://dx.doi.org/10.18653/v1/K17-3014`.

5. Kiperwasser, E. and Y. Goldberg, "Simple and accurate dependency parsing using bidirectional LSTM feature representations", *arXiv preprint arXiv:1603.04351*, 2016.

6. Tjong Kim Sang, E. F. and S. Buchholz, "Introduction to the CoNLL-2000 shared task: Chunking", *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, pp. 127–132, Association for Computational Linguistics, 2000.

7. Attardi, G. and F. Dell'Orletta, "Chunking and dependency parsing", *LREC Workshop on Partial Parsing: Between Chunking and Deep Parsing*, 2008.

8. Eriguchi, A., K. Hashimoto and Y. Tsuruoka, "Tree-to-sequence attentional neural machine translation", *arXiv preprint arXiv:1603.06075*, 2016.

9. Saul, K., "Identity and Individuation", *New York University Press, New York*, 1971.

10. Chinchor, N. and P. Robinson, "MUC-7 named entity task definition", *Proceedings of the 7th Conference on Message Understanding*, Vol. 29, 1997.

11. Ratinov, L. and D. Roth, "Design challenges and misconceptions in named entity recognition", *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pp. 147–155, Association for Computational Linguistics, 2009.

12. Zhang, Z., *Named entity recognition: challenges in document annotation, gazetteer construction and disambiguation*, Ph.D. Thesis, University of Sheffield, 2013.

13. Hoffart, J., M. A. Yosef, I. Bordino *et al.*, "Robust disambiguation of named entities in text", *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 782–792, Association for Computational Linguistics, 2011.

14. Marsh, E. and D. Perzanowski, "MUC-7 evaluation of IE technology: Overview of results", *Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29-May 1, 1998*, 1998.

15. Tjong Kim Sang, E. F. and F. De Meulder, "Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition", *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pp. 142–147, Association for Computational Linguistics, 2003.

16. Nadeau, D. and S. Sekine, "A survey of named entity recognition and classification", *Lingvisticae Investigationes*, Vol. 30, No. 1, pp. 3–26, 2007.

17. Ritter, A., S. Clark, O. Etzioni *et al.*, "Named entity recognition in tweets: an experimental study", *Proceedings of the conference on empirical methods in natural language processing*, pp. 1524–1534, Association for Computational Linguistics, 2011.

18. Liu, X., S. Zhang, F. Wei *et al.*, "Recognizing named entities in tweets", *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 359–367, Association for Computational Linguistics, 2011.

19. Plank, B., D. Hovy, R. McDonald *et al.*, "Adapting taggers to Twitter with not-so-distant supervision", *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pp. 1783–1792, 2014.

20. Rowe, M., M. Stankovic and A.-S. Dadzie, "Microposts 2015–5th Workshop on'Making Sense of Microposts': Big things come in small packages", *Proceedings of the 24th International Conference on World Wide Web*, pp. 1551–1552, ACM, 2015.

21. Baldwin, T., M.-C. de Marneffe, B. Han *et al.*, "Shared tasks of the 2015 workshop on noisy user-generated text: Twitter lexical normalization and named entity recognition", *Proceedings of the Workshop on Noisy User-generated Text*, pp. 126–135, 2015.

22. Okur, E., H. Demir and A. Özgür, "Named Entity Recognition on Twitter for Turkish using Semi-supervised Learning with Word Embeddings", *arXiv preprint arXiv:1810.08732*, 2018.

23. Augenstein, I., L. Derczynski and K. Bontcheva, "Generalisation in named entity recognition: A quantitative analysis", *Computer Speech & Language*, Vol. 44, pp. 61–83, 2017.

24. Akdemir, A., A. Hürriyetoğlu, E. Yörük *et al.*, "Towards Generalizable Place Name Recognition Systems", *Proceedings of the 12th Workshop on Geographic Information Retrieval*, GIR'18, ACM, New York, NY, USA, 2018.

25. Ekbal, A., E. Sourjikova, A. Frank *et al.*, "Assessing the challenge of fine-grained named entity recognition and classification", *proceedings of the 2010 Named Enti-*

ties Workshop, pp. 93–101, Association for Computational Linguistics, 2010.

26. Gillick, D., N. Lazic, K. Ganchev *et al.*, "Context-dependent fine-grained entity type tagging", *arXiv preprint arXiv:1412.1820*, 2014.

27. Fleischman, M. and E. Hovy, "Fine grained classification of named entities", *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pp. 1–7, Association for Computational Linguistics, 2002.

28. Ling, X. and D. S. Weld, "Fine-Grained Entity Recognition.", *AAAI*, Vol. 12, pp. 94–100, 2012.

29. Yogatama, D., D. Gillick and N. Lazic, "Embedding methods for fine grained entity type classification", *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, Vol. 2, pp. 291–296, 2015.

30. Caruna, R., "Multitask learning: A knowledge-based source of inductive bias", *Machine Learning: Proceedings of the Tenth International Conference*, pp. 41–48, 1993.

31. Sutton, C., A. McCallum and K. Rohanimanesh, "Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data", *Journal of Machine Learning Research*, Vol. 8, No. Mar, pp. 693–723, 2007.

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

33. Finkel, J. R., T. Grenager and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling", *Proceedings of the 43rd annual meeting on association for computational linguistics*, pp. 363–370, Association for Computational Linguistics, 2005.

34. Culotta, A. and J. Sorensen, "Dependency tree kernels for relation extraction", *Proceedings of the 42nd annual meeting on association for computational linguistics*, p. 423, Association for Computational Linguistics, 2004.

35. Bunescu, R. C. and R. J. Mooney, "A shortest path dependency kernel for relation extraction", *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pp. 724–731, Association for Computational Linguistics, 2005.

36. Reddy, S., O. Täckström, M. Collins *et al.*, "Transforming dependency structures to logical forms for semantic parsing", *Transactions of the Association for Computational Linguistics*, Vol. 4, pp. 127–140, 2016.

37. Poon, H. and P. Domingos, "Unsupervised semantic parsing", *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pp. 1–10, Association for Computational Linguistics, 2009.

38. Galley, M. and C. D. Manning, "Quadratic-time dependency parsing for machine translation", *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pp. 773–781, Association for Computational Linguistics, 2009.

39. Cui, H., R. Sun, K. Li *et al.*, "Question answering passage retrieval using dependency relations", *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 400–407, ACM, 2005.

40. Liang, P., M. I. Jordan and D. Klein, "Learning dependency-based compositional semantics", *Computational Linguistics*, Vol. 39, No. 2, pp. 389–446, 2013.

41. Jie, Z., A. O. Muis and W. Lu, "Efficient Dependency-Guided Named Entity Recognition.", *AAAI*, pp. 3457–3465, 2017.

42. Sasano, R. and S. Kurohashi, "Japanese named entity recognition using structural natural language processing", *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II*, 2008.

43. Bikel, D. M., S. Miller, R. Schwartz *et al.*, "Nymble: a high-performance learning name-finder", *Proceedings of the fifth conference on Applied natural language processing*, pp. 194–201, Association for Computational Linguistics, 1997.

44. McCallum, A. and W. Li, "Early Results for Named Entity Recognition with Conditional Random Fields, Feature Induction and Web-enhanced Lexicons", *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pp. 188–191, Association for Computational Linguistics, Stroudsburg, PA, USA, 2003, https://doi.org/10.3115/1119176.1119206.

45. Chieu, H. L. and H. T. Ng, "Named entity recognition: a maximum entropy approach using global information", *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pp. 1–7, Association for Computational Linguistics, 2002.

46. Altun, Y., I. Tsochantaridis and T. Hofmann, "Hidden markov support vector machines", *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 3–10, 2003.

47. Zhou, G. and J. Su, "Named entity recognition using an HMM-based chunk tagger", *proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pp. 473–480, Association for Computational Linguistics, 2002.

48. Lample, G., M. Ballesteros, S. Subramanian *et al.*, "Neural architectures for named entity recognition", *arXiv preprint arXiv:1603.01360*, 2016.

49. Demir, H. and A. Ozgur, "Improving Named Entity Recognition for Morphologically Rich Languages Using Word Embeddings.", *ICMLA*, pp. 117–122, 2014.

50. Mikolov, T., I. Sutskever, K. Chen *et al.*, "Distributed representations of words and phrases and their compositionality", *Advances in neural information processing systems*, pp. 3111–3119, 2013.

51. Celikkaya, G., D. Torunoglu and G. Eryigit, "Named entity recognition on real data: a preliminary investigation for Turkish", *Application of Information and Communication Technologies*, pp. 1–5, IEEE, 2013.

52. Şeker, G. A. and G. Eryiğit, "Initial explorations on using CRFs for Turkish named entity recognition", *Proceedings of COLING 2012*, pp. 2459–2474, 2012.

53. Eryiğit, G., T. Ilbay and O. A. Can, "Multiword expressions in statistical dependency parsing", *Workshop on Statistical Parsing of Morphologically Rich Languages*, pp. 45–55, Association for Computational Linguistics, 2011.

54. Küçük, D. *et al.*, "A hybrid named entity recognizer for Turkish", *Expert Systems with Applications*, Vol. 39, No. 3, pp. 2733–2742, 2012.

55. Yeniterzi, R., "Exploiting morphology in Turkish named entity recognition system", *Proceedings of the ACL 2011 Student Session*, pp. 105–110, Association for Computational Linguistics, 2011.

56. Güneş, A. and A. C. Tantuğ, "Turkish named entity recognition with deep learning", *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, IEEE, 2018.

57. Güngör, O., S. Üsküdarlı and T. Güngör, "Improving Named Entity Recognition by Jointly Learning to Disambiguate Morphological Tags", *arXiv preprint arXiv:1807.06683*, 2018.

58. Gungor, O., E. Yildiz, S. Uskudarli and T. Gungor, "Morphological Embeddings for Named Entity Recognition in Morphologically Rich Languages", *arXiv preprint arXiv:1706.00506*, 2017.

59. Durrett, G. and D. Klein, "A joint model for entity analysis: Coreference, typing, and linking", *Transactions of ACL*, Vol. 2, pp. 477–490, 2014.

60. Zhang, C. and Z. Zhang, "Improving multiview face detection with multi-task deep convolutional neural networks", *Applications of Computer Vision (WACV), 2014 IEEE Winter Conference on*, pp. 1036–1041, IEEE, 2014.

61. Yim, J., H. Jung, B. Yoo, C. Choi, D. Park and J. Kim, "Rotating your face using multi-task deep neural network", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 676–684, 2015.

62. Søgaard, A. and Y. Goldberg, "Deep multi-task learning with low level tasks supervised at lower layers", *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Vol. 2, pp. 231–235, 2016.

63. Lai, A. and J. Hockenmaier, "Illinois-lh: A denotational and distributional approach to semantics", *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pp. 329–334, 2014.

64. Nguyen, D. B., M. Theobald and G. Weikum, "J-NERD: joint named entity recognition and disambiguation with rich linguistic features", *Transactions of the Association for Computational Linguistics*, Vol. 4, pp. 215–229, 2016.

65. Luo, G., X. Huang, C.-Y. Lin *et al.*, "Joint entity recognition and disambiguation", *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 879–888, 2015.

66. Radford, W., X. Carreras and J. Henderson, "Named entity recognition with document-specific KB tag gazetteers", *Proceedings of the 2015 EMNLP*, pp. 512–517, 2015.

67. Zeman, D., J. Hajič, M. Popel *et al.*, "CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies", *Proceedings of the CoNLL*

2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, pp. 1–21, 2018.

68. Dyer, C., M. Ballesteros, W. Ling *et al.*, "Transition-based dependency parsing with stack long short-term memory", *arXiv preprint arXiv:1505.08075*, 2015.

69. Ballesteros, M., C. Dyer and N. A. Smith, "Improved transition-based parsing by modeling characters instead of words with LSTMs", *arXiv preprint arXiv:1508.00657*, 2015.

70. de Lhoneux, M., Y. Shao, A. Basirat *et al.*, "From Raw Text to Universal Dependencies-Look, No Tags!", *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pp. 207–217, 2017.

71. McDonald, R., K. Crammer and F. Pereira, "Online large-margin training of dependency parsers", *Proceedings of the 43rd annual meeting on association for computational linguistics*, pp. 91–98, Association for Computational Linguistics, 2005.

72. Tür, G., D. Hakkani-Tür and K. Oflazer, "A statistical information extraction system for Turkish", *Natural Language Engineering*, Vol. 9, No. 2, pp. 181–210, 2003.

73. Sulubacak, U., G. Eryiğit, T. Pamay *et al.*, "IMST: A revisited Turkish dependency treebank", *Proceedings of TurCLing 2016, the 1st International Conference on Turkic Computational Linguistics*, EGE UNIVERSITY PRESS, 2016.

74. Oflazer, K., B. Say, D. Z. Hakkani-Tür *et al.*, "Building a Turkish Treebank, Invited chapter in Building and Exploiting Syntactically-annotated Corpora, Anne Abeille Editor", *Kluwer Academic Publishers*, 2003.

75. Lavergne, T., O. Cappé and F. Yvon, "Practical very large scale CRFs", *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 504–513, Association for Computational Linguistics, 2010.

76. Xiong, D., Q. Liu and S. Lin, "Maximum entropy based phrase reordering model for statistical machine translation", *Proceedings of the 21st CICLing*, pp. 521–528, Association for Computational Linguistics, 2006.

77. Ratnaparkhi, A., "A maximum entropy model for part-of-speech tagging", *Conference on Empirical Methods in Natural Language Processing*, 1996.

78. Owoputi, O., B. O'Connor, C. Dyer *et al.*, "Improved part-of-speech tagging for online conversational text with word clusters", *Proceedings of the 2013 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 380–390, 2013.

79. Fresko, M., B. Rosenfeld and R. Feldman, "A hybrid approach to NER by MEMM and manual rules", *Proceedings of the 14th ACM international conference on Information and knowledge management*, pp. 361–362, ACM, 2005.

80. Sutton, C., A. McCallum *et al.*, "An introduction to conditional random fields", *Foundations and Trends® in Machine Learning*, Vol. 4, No. 4, pp. 267–373, 2012.

81. Neubig, G., C. Dyer *et al.*, "DyNet: The Dynamic Neural Network Toolkit", *arXiv preprint arXiv:1701.03980*, 2017.

82. Eisner, J. M., "Three new probabilistic models for dependency parsing: An exploration", *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pp. 340–345, Association for Computational Linguistics, 1996.

83. Dozat, T., P. Qi and C. D. Manning, "Stanford's Graph-based Neural Dependency Parser at the CoNLL 2017 Shared Task", *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pp. 20–30, 2017.

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

85. Lafferty, J., A. McCallum and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data", *UPenn*, 2001.

86. Ma, X. and E. Hovy, "End-to-end sequence labeling via bi-directional lstm-cnns-crf", *arXiv preprint arXiv:1603.01354*, 2016.

87. Chen, T., R. Xu, Y. He *et al.*, "Improving sentiment analysis via sentence type classification using BiLSTM-CRF and CNN", *Expert Systems with Applications*, Vol. 72, pp. 221–230, 2017.

88. Forney, G. D., "The viterbi algorithm", *Proceedings of the IEEE*, Vol. 61, No. 3, pp. 268–278, 1973.

89. Sak, H., T. Güngör and M. Saraçlar, "Turkish language resources: Morphological parser, morphological disambiguator and web corpus", *Advances in natural language processing*, pp. 417–427, Springer, 2008.

90. Kurohashi, S. and M. Nagao, "Building a Japanese parsed corpus while improving the parsing system", *Proceedings of The 1st International Conference on Language Resources & Evaluation*, pp. 719–724, 1998.

91. Malouf, R., "A comparison of algorithms for maximum entropy parameter estimation", *proceedings of the 6th conference on Natural language learning-Volume 20*, pp. 1–7, Association for Computational Linguistics, 2002.

92. Broyden, C. G., "The convergence of a class of double-rank minimization algorithms 1. general considerations", *IMA Journal of Applied Mathematics*, Vol. 6, No. 1, pp. 76–90, 1970.

93. Srikumar, V. and C. D. Manning, "Learning distributed representations for structured output prediction", *Advances in Neural Information Processing Systems*, pp. 3266–3274, 2014.

94. Turian, J., L. Ratinov and Y. Bengio, "Word representations: a simple and general method for semi-supervised learning", *Proceedings of the 48th annual meeting of the association for computational linguistics*, pp. 384–394, Association for Computational Linguistics, 2010.

95. Li, Q. and H. Ji, "Incremental joint extraction of entity mentions and relations", *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1, pp. 402–412, 2014.