DEEP NEURAL NETWORKS FOR NAMED ENTITY RECOGNITION ON SOCIAL MEDIA

SOSYAL MEDYA ÜZERİNDE VARLIK İSMİ TANIMA İÇİN DERİN SİNİR AĞLARI

EMRE KAĞAN AKKAYA

ASST. PROF. DR. BURCU CAN BUĞLALILAR
Supervisor

Submitted to Graduate School of Science and Engineering of Hacettepe University
as a Partial Fulfillment to the Requirements
for the Award of the Degree of Master of Science
in Computer Engineering

November 2018

This work named **"Deep Neural Networks for Named Entity Recognition on Social Media"** by **EMRE KAĞAN AKKAYA** has been approved as a thesis for the Degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING** by the below mentioned Examining Committee Members.

Prof. Dr. Cem BOZŞAHİN
Head ....

Asst. Prof. Dr. Burcu Can BUĞLALILAR
Supervisor ....

Asst. Prof. Dr. Mehmet KÖSEOĞLU
Member ....

Asst. Prof. Dr. Ayça TARHAN
Member ....

Asst. Prof. Dr. Umut ÖZGE
Member ....

This thesis has been approved as a thesis for the Degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING** by Board of Directors of the Institute for Graduate Studies in Science and Engineering.

Prof. Dr. Menemşe GÜMÜŞDERELİOĞLU
Director of the Institute of
Graduate School of Science and Engineering

To my family…

ETHICS

In this thesis study, prepared in accordance with the spelling rules of Institute of Graduate Studies in Science of Hacettepe University,

I declare that

- all the information and documents have been obtained in the base of the academic rules.
- all audio-visual and written information and results have been presented according to the rules of scientific ethics
- in case of using others works, related studies have been cited in accordance with the scientific standards
- all cited studies have been fully referenced
- I did not do any distortion in the data set
- and any part of this thesis has not been presented as another thesis study at this or any other university.

26/11/2018

EMRE KAĞAN AKKAYA

YAYINLAMA VE FİKRİ MÜLKİYET HAKLARI BEYANI

Enstitü tarafından onaylanan lisansüstü tezimin/raporumun tamamını veya herhangi bir kısmını, basılı (kağıt) ve elektronik formatta arşivleme ve aşağıda verilen koşullarla kullanıma açma iznini Hacettepe Üniversitesi'ne verdiğimi bildiririm. Bu izinle Üniversite'ye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak, tezimin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanım hakları bana ait olacaktır.

Tezin kendi orijinal çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Tezimde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanması zorunlu metinlerin yazılı izin alarak kullandığımı ve istenildiğinde suretlerini Üniversite'ye teslim etmeyi taahhüt ederim.

Yükseköğretim Kurulu tarafından yayınlanan "**Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge**" kapsamında tezim aşağıda belirtilen koşullar haricince YÖK Ulusal Tez Merkezi / H.Ü. Kütüphaneleri Açık Erişim Sisteminde erişime açılır.

- □ Enstitü / Fakulte yönetim kurulu kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren 2 yıl ertelenmiştir.
- □ Enstitü / Fakulte yönetim kurulu gerekçeli kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren ... ay ertelenmiştir.
- □ Tezim ile ilgili gizlilik kararı verilmiştir.

26/11/2018

EMRE KAĞAN AKKAYA

ABSTRACT

DEEP NEURAL NETWORKS FOR NAMED ENTITY RECOGNITION ON SOCIAL MEDIA

Emre Kağan AKKAYA

Master of Science, Computer Engineering Department

Supervisor: Asst. Prof. Dr. Burcu CAN BUĞLALILAR

November 2018, 126 pages

Named entity recognition (NER) on noisy data, specifically user-generated content (e.g. online reviews, tweets) is a challenging task because of the presence of ill-formed text. In this regard, while studies on morphologically-poor languages such as English has been rapidly advancing in recent years, studies on morphologically-rich languages such as Turkish has fallen behind for noisy data. This is mostly due to Turkish being an agglutinative language, having a rich morphology and also having scarce annotated data. Existing studies on Turkish both for noisy and formal (e.g. news text) data still make use of hand-crafted features and/or external domain-specific resources (e.g. gazetteers). In this thesis, we investigate the effects of neural architectures without the help of any external domain-specific resources and/or manually-constructed features. So that the proposed model can also be used for different morphologically-rich languages and for different domains. Moreover, we also experimented with different word and sub-word level (e.g. morpheme, character or character n-gram level) embedding techniques and we argue that sub-word level embeddings provide better word representations for morphologically-rich languages syntactically and semantically. For this purpose, we propose a transfer learning model that is an extension of a baseline, bidirectional LSTM-CRF architecture. The model is trained on two different datasets simultaneously for the purpose of transfer learning from formal to noisy data and it exploits morpheme-level, character n-gram level and orthographic character-level embeddings as its feature set. Consequently, we have obtained an F1 score of 65.72% on Turkish tweet dataset and 41.97% on English WNUT'17 dataset.

Keywords: named entity recognition, recurrent neural networks, long-short term memory, conditional random fields, transfer learning, Turkish, natural language processing, deep learning

ÖZET

SOSYAL MEDYA ÜZERİNDE VARLIK İSMİ TANIMA İÇİN DERİN SİNİR AĞLARI

Emre Kağan AKKAYA
Yüksek Lisans, Bilgisayar Mühendisliği
Danışman: Yrd. Doç. Dr. Burcu CAN BUĞLALILAR
Kasım 2018, 126 sayfa

Gürültülü veri, özellikle kullanıcı tarafından oluşturulan içerik (örn. çevrimiçi yorum, tweet), üzerinde varlık ismi tanıma metnin bozuk yapısından dolayı zorlu bir görevdir. Bu kapsamda, İngilizce gibi göreli morfolojik olarak fakir diller üzerindeki araştırmalar son yıllarda hızla ilerlerken, Türkçe gibi morfolojik olarak zengin dillerde gürültülü veri üzerindeki araştırmalar geri kalmıştır. Bu çoğunlukla Türkçe dilinin morfolojik olarak zengin ve sondan eklemeli olması ile az miktarda etiketli veriye sahip olması nedeniyledir. Türkçe'de varolan araştırmalar hem gürültü hem de resmi (örn. haber metni) veri üzerinde çoğunlukla hala el yapımı öznitelikler ve/veya alana-özgü harici kaynaklardan (isim listeleri) faydalanmaktadır. Bu tezdeyse, el yapımı öznitelikler ve/veya alana-özgü harici kaynaklar kullanmayan yapay sinir ağlarının etkileri incelenmektedir. Öyle ki, önerilen model farklı morfolojik olarak dillerde ve farklı alanlarda da kullanılabilsin. Bununla birlikte, farklı kelime ve kelime-altı (örn. morfem ve karakter n-gram seviyesinde) embedding teknikleriyle de deneyler gerçekleştirdik ve morfolojik olarak zengin diller için kelime-altı embedding'lerin sözdizimi ve anlamsal açıdan daha iyi kelime temsili sunduğunu savunuyoruz. Bu amaçla, temel aldığımız LSTM-CRF mimarisinin uzantısı olan bir *transfer öğrenme* modeli önermekteyiz. Söz konusu model, resmi veriden gürültülü veriye bilgi aktarımı amacıyla aynı anda iki farklı veri kümesi üzerinde eğitilmekte olup; morfem, karakter n-gram ve ortografik embedding'lerden faydalanır. Sonuç olarak, Türkçe gürültülü veri kümesi üzerinde %65.72 ve İngilizce WNUT'17 veri kümesinde %41.97 F1 puanı elde ettik.

Anahtar Kelimeler: varlık ismi tanıma, özyineli sinir ağları, uzun-kısa süreli hafıza, koşullu rastgele alanlar, öğrenme aktarımı, Türkçe, doğal dil işleme, derin öğrenme

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my respectable supervisor Asst. Prof. Dr. Burcu Can Buğlalılar for her persistent encouragement, endless patience, and invaluable guidance in this thesis. It would not have been possible without her efforts for which I am, and always will be, gratefully indebted.

I would like to also show my sincere appreciation to my thesis committee members; Prof. Dr. Cem Bozşahin, Asst. Prof. Dr. Mehmet Köseoğlu, Asst. Prof. Dr. Ayça Tarhan and Asst. Prof. Dr. Umut Özge for taking their time to review and providing insightful comments.

I would like to also thank my dearest friends Burak Gülsaçan, Tutku Demiröz, Serhan Aktar, İrem Aktar, Merve Akşit and Tuğba Kaya for their unfailing support.

Finally, and most importantly, I would like to thank my family, my dear mother Jale Akkaya, my dear father Zafer Akkaya and my brother Samet Parsak for their unconditional love and support.

CONTENTS

Page

FIGURES

Page

TABLES

ABBREVIATIONS

NN	Neural Network
FFNN	Feed-forward Neural Network
RNN	Recurrent Neural Network
BPTT	Backpropagation Through Time
SGD	Stochastic Gradient Descent
CBOW	Continuous Bag-of-words
LSTM	Long-short Term Memory
Bi-LSTM	Bidirectional Long-short Term Memory
CRF	Conditional Random Fields
CNN	Convolutional Neural Network

1. INTRODUCTION

1.1. Overview

Named Entity Recognition (NER) is a problem of information extraction in natural language processing (NLP) that aims to identify and categorize each word into pre-defined categories. The term, *named entity*, first coined by Ralph Grishman and Beth Sundheim [2] during the sixth of Message Understanding Conferences, MUC-6, and defined as ”identifying the names of all the people, organization, and geographic locations in a text” [3]. The term as expected has evolved over time and nowadays it may also include recognizing various different categories such as time and date, money and percentage, corporation, creative-work, group, product names etc.

Example The sentence

Cumhuriyetin ilk kurumu olan Cumhurbaşkanlığı Senfoni Orkestrası Büyük Atatürk'ün yüce makamının adını vermesiyle onurlandırılmıştır. (Presidential Symphony Orchestra as the first institution of the Republic of Turkey was honoured by Great Turk Atatürk naming it with his supreme authority.

is tagged as:

*Türkiye/**ORG** Cumhuriyetinin/**ORG** ilk kurumu olan Cumhurbaşkanlığı/**ORG** Senfoni/**ORG** Orkestrası/**ORG** Büyük/**PER** Atatürk'ün/**PER** yüce makamının adını vermesiyle onurlandırılmıştır.*

Named entity, thus, can also be seen as sequence labeling problem, similar to POS tagging and usually used as a preprocessing step for various NLP tasks such as machine translation, automatic question answering, opinion mining, relation extraction and so on.

1.2. Motivation

As we can see in previous named entity recognition results in the literature, the performance of named entity recognition on formal (e.g. newspaper, academic papers) data is very high and accurate, particularly for languages like English that has abundant annotated data. Recent research such as [4] achieved over 91% F1 score on English for formal data, almost comparable to human annotation performance. So one can unfairly conclude that the NER task has nearly reached its peak performance.

However with the ever-changing nature of Internet, especially after the emergence of social media, we have been introduced to informal/noisy data (user-generated data) such as user comments and tweets. This new type of data is highly valuable for information extraction tasks such as opinion mining due to being widespread and having almost up-to-date nature. This type of noisy and informal text includes missing characters in words (either deliberately or by forgetfulness), missing punctuation marks, emojis, slang words and abbreviations. Examples of this nature can be seen in the following tweets:

”Why doesn’t George R.R. Martin use twitter? Because he killed all 140 characters. #got”
”This is Gunnersss! PL not finished yet! Keep going lads! #CHEARS”

For example, Stanford Named Entity Tagger [5] that is based on Conditional Random Fields (CRF) trained on news data, successfully tags *George R. R. Martins* as *PERSON* but fails to tag *Twitter* due to missing capitalization and also fails to tag *Gunners* (alias of Arsenal FC) and *PL* (Premier League) because of character repetition, alias and abbreviation.

All of these cause new problems for existing NER systems, considering most of them depend on manually-crafted features (e.g. capitalization, numerical/date/time patterns or other rule-based features) and/or external domain-specific resources (e.g. gazetteers, lexicons), therefore ill-suited to noisy data. Some of the existing systems try to solve these new challenging problems either by extending their existing feature set to better suit this new domain or by adding new domain-specific resources. Recent successful researches, however, address these by utilizing neural architectures with the help of auto-generated features. As one of the goals of this thesis, we have researched the effects of different neural architectures and experimented with different word-level and sub-word level representation techniques.

Despite NER is a well-studied topic for English and Turkish formal data, Turkish NER performance is far behind on noisy/informal data. In [6] which is the current state-of-the-art work for Turkish NER, 91.94% F1 score (ENAMEX) for news data and only 67.96% F1 score for noisy data was achieved. This is mostly due to Turkish being an agglutinative language and having rich morphology and scarce annotated data. As we can see in the following chapters, most of the existing Turkish NER solutions still use statistical and/or rule-based approaches that rely mostly on hand-crafted features.

This motivates us to research and adopt proven neural network models for Turkish and in the process obtain valuable features without using any external domain-specific resources or any hand-crafted features. In addition, we have researched the means to transfer learning from formal data to noisy/informal data due to aforementioned scarce annotated data problem. So that, the resulting Turkish NER model can be practically used in different applications on different domains without the need of domain-specific knowledge. We also believe that the proposed model in this thesis can also be trained for different morphologically-rich languages as well.

As a result, we aim to research and experiment with different neural network models and features for named entity recognition and provide a Turkish NER model which is capable of transfer learning and based on a proposed neural architecture without using any domain or language-specific knowledge.

1.3. Research Questions

The aim of this thesis can be summarized as finding answers to these questions:

- Instead of rule-based approaches, can neural networks be effectively used on noisy data?

- Can we achieve successful results using neural networks for morphologically-rich languages such as Turkish?
- Can we learn valuable features without using any hand-crafted features for the NER task?

1.4. Organization of the Thesis

Structure of the thesis can be outlined as follows:

In **Chapter 2**, different approaches used for sequence labeling are thoroughly described. This includes various neural networks such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). We also describe techniques for learning word-level and sub-word level representations such as word2vec [7], FastText [8] and morph2vec [9].

In **Chapter 3**, we present a literature review on Named Entity Recognition, especially focusing on Turkish and English both for formal and noisy/informal data. Various studies on learning different word and sub-word level embeddings are also presented in this chapter.

In **Chapter 4**, we first briefly refer to approaches that we used throughout the experiments which includes neural networks, sequence labeling technique and word representation algorithms. Then we present the proposed models for English and Turkish. Experimental setting and training details are also presented in this chapter.

In **Chapter 5**, results are presented which comprise different experiments conducted on Turkish and English with formal and informal/noisy data. The chapter also presents comparison to related research in the literature and provides discussion on the results.

In **Chapter 6**, we conclude the thesis by briefly reviewing the contributions made and approaches undertaken and provide possible future improvements based on the results we obtained.

2. BACKGROUND

In this chapter, the methods and models used for NER as a sequence labeling task will be described. First, we will explain *Conditional Random Fields* as one of the most commonly used models in sequence labeling tasks because of its ability to compute a conditional probability for the most probable label sequence. Then we will explore neural networks in general and give recurrent neural networks (RNN), especially *Long-Short Term Memory (LSTM)*, as an example to learn valuable features in a sequence. We also explain *Convolutional Neural Network (CNN)* as an alternative to LSTM in this regard. Subsequently we describe different *word-level and subword-level representation* techniques that can be used as features in order to train the neural networks. After improving on these building blocks, we finally explain *transfer learning* model in order to learn better features by incorporating an additional CRF which is trained on different dataset.

2.1. Conditional Random Fields

As we have discussed before, named entity recognition is the task of assigning each word of an input sequence into a pre-determined categories. Each sentence in this input is a training example for our model and there is only a limited number of named entity categories (i.e. labels). Solving this labeling task is different from other classification models in these contexts:

- Valuable information will surely be lost if the classifier works only on word basis. It must also take neighboring tags into account.
- Vector representation of sentences with fixed size is not feasible. There must be a way to represent different sentences with different lengths.
- The set of all possible label sequences is too large to find the most probable one.

Conditional Random Fields (CRF) [10] is one of the statistical models based on graphical models that are often used for sequence learning.

2.1.1. Mathematical Definition

Our goal is to find the most probable label sequence given an input sequence. Thus, given that $\underline{x}$ is an input sequence of $x_1 \cdots\ x_m$ and $\underline{s}$ is a sequence of labels $s_1 \cdots\ s_m$, we denote the set of all possible labels as S and set of all possible label sequences that are valid or not as S^m, then we build a probabilistic model to define the conditional probability:

$$p(s_1 \cdots s_m | x_1 \cdots x_m) = p(\underline{s} \mid \underline{x}) \tag{1}$$

where $\underline{s}$ refer to a sequence of states and $\underline{x}$ refer to a sequence of input words.

Moreover, we define a feature vector to map a pair of input sequence and state (label) sequence $\underline{s}$ to d-dimensional feature vector.

$$\underline{\Phi}(\underline{x}, \underline{s}) \epsilon \mathbb{R}^d \tag{2}$$

where $\underline{\Phi}$ denotes the global feature function. Building log-linear model gives us the form:

$$p(\underline{s}|\underline{x}; \underline{w}) = \frac{exp(\underline{w}.\underline{\Phi}(\underline{x}, \underline{s}))}{\sum_{\underline{s'} \epsilon S^m} exp(\underline{w}.\underline{\Phi}(\underline{x}, \underline{s'}))} \tag{3}$$

where $\underline{w}$ is a parameter vector and the inner product $\underline{w}.\underline{\Phi}(\underline{x}, \underline{s})$ refers to the likelihood of state s given input x.

As we can see, solving this log-linear model is quite challenging due to (1) high number of possible values for $\underline{s}$ and (2) having a denominator (normalization constant) that involves a sum over the set of S^m which is a complex computation.

The global feature vector we have mentioned earlier can be defined as:

$$\underline{\Phi}(\underline{x}, \underline{s}) = \sum_{j=1}^{m} \underline{\phi}(\underline{x}, j, s_{j-1}, s_j) \tag{4}$$

As we can see the local feature vector takes input sequence, current label and previous label into account and that means for every $k = 1 \cdots d$, $\underline{\Phi_k}(\underline{x}, \underline{s})$ is computed as the sum of the local feature vectors over the label transitions.

In order to better understand CRF, we must again stress that it represents input sequences by using these feature functions. Various feature functions for a certain type of sequence labeling task could be used. For example, some features for the NER task can be defined as follows:

- Whether a word is capitalized or not
- Looking into neighboring words that may indicate certain type of entities (e.g. ”Co.”, ”Inc.” for indication of corporation or ”TL”, ”dollars” for indication of money)
- Whether a word is the beginning (or end) of a sentence or not

Next, we describe how to learn the weights w for the defined features, which is called parameter estimation.

2.1.1.1. Parameter Estimation

Given a set of n labeled examples $\{(\underline{x}^i, \underline{s}^i)\}_{i=1}^{n}$, where each $\underline{x}^i$ is an input sequence $x_1^i \cdots x_m^i$, and each $\underline{s}^i$ is a label sequence $s_1^i \cdots s_m^i$, the regularized log-likelihood function is defined as:

$$L(\underline{w}) = \sum_{i=1}^{n} logp(\underline{s}^i \mid \underline{x}^i; \underline{w}) - \frac{\lambda}{2}||\underline{w}||^2 \tag{5}$$

We want to find good parameter estimates for the weights that maximizes the given log-likelihood:

$$\underline{w}^* = \arg\max_{\underline{w}\epsilon\mathbb{R}^d} \sum_{i=1}^{n} logp(\underline{s}^i \mid \underline{x}^i; \underline{w}) - \frac{\lambda}{2}||\underline{w}||^2 \tag{6}$$

By using gradient-based optimization, the partial derivatives take the form:

$$\frac{\partial}{\partial w_k} L(\underline{w}) = \sum_i \Phi_k(\underline{x}^i, \underline{s}^i) - \sum_i \sum_{\underline{s}\epsilon S^m} p(\underline{s} \mid \underline{x}^i; \underline{w})\Phi_k(\underline{x}^i, \underline{s}) - \lambda w_k \tag{7}$$

For the first term, all we have to do is to sum over all examples of $i = 1 \cdots n$ and for each one of them sum over all positions $j = 1 \cdots m$ since it is equal to:

$$\sum_i \Phi_k(\underline{x}^i, \underline{s}^i) = \sum_i \sum_{j=1}^{m} \phi_k(\underline{x}^i, j, s^i_{j-1}, s^i_j) \tag{8}$$

Then we have to compute the second term using dynamic programming where it involves computation over S^m. The derivation is given below:

$$\sum_{\underline{s}\epsilon S^m} p(\underline{s} \mid \underline{x}^i; \underline{w})\Phi_k(\underline{x}^i, \underline{s}) \tag{9}$$

$$= \sum_{\underline{s}\epsilon S^m} p(\underline{s} \mid \underline{x}^i; \underline{w}) \sum_{j=1}^{m} \phi_k(\underline{x}^i, j, s^i_{j-1}, s^i_j) \tag{10}$$

$$= \sum_{j=1}^{m} \sum_{\underline{s}\epsilon S^m} p(\underline{s} \mid \underline{x}^i; \underline{w})\phi_k(\underline{x}^i, j, s^i_{j-1}, s^i_j) \tag{11}$$

$$= \sum_{j=1}^{m} \sum_{a\epsilon S, b\epsilon S} \sum_{\substack{\underline{s}\epsilon S^m: \\ s_{j-1}=a, s_j=b}} p(\underline{s} \mid \underline{x}^i; \underline{w})\phi_k(\underline{x}^i, j, s^i_{j-1}, s^i_j) \tag{12}$$

$$= \sum_{j=1}^{m} \sum_{a\epsilon S, b\epsilon S} \phi_k(\underline{x}^i, j, s^i_{j-1}, s^i_j) \sum_{\substack{\underline{s}\epsilon S^m: \\ s_{j-1}=a, s_j=b}} p(\underline{s} \mid \underline{x}^i; \underline{w}) \tag{13}$$

$$= \sum_{j=1}^{m} \sum_{a\epsilon S, b\epsilon S} q^i_j(a, b)\phi_k(\underline{x}^i, j, a, b) \tag{14}$$

where

$$q^i_j(a,b) = \sum_{\substack{\underline{s} \epsilon S^m: \\ s_{j-1}=a, s_j=b}} p(\underline{s} \mid \underline{x}^i; \underline{w}) \tag{15}$$

Hence we have reduced the derivatives into the terms of $q^i_j(a,b)$ which allows us to compute the whole derivatives efficiently. Another interpretation is that the term $q^i_j(a,b)$ is the probability of the ith example $\underline{x}^i$ with state a at position $j-1$ and state b at position j, under the distribution of $p(\underline{s} \mid \underline{x}; \underline{w})$. Using forward-backward algorithm which is similar to Viterbi algorithm, this term can be solved in $O(mk^2)$ time efficiently.

2.1.1.2. Decoding

Given the input sequence $\underline{x} = x_1 \cdots x_m$, the aim of decoding is to find the most likely label sequence that can be denoted as:

$$\arg\max_{\underline{s} \epsilon S^m} p(\underline{s} \mid \underline{x}; \underline{w}) \tag{16}$$

This can be simplified as below:

$$\arg\max_{\underline{s} \epsilon S^m} p(\underline{s} \mid \underline{x}; \underline{w}) = \frac{exp(\underline{w}.\underline{\Phi}(\underline{x}, \underline{s}))}{\sum_{s' \epsilon S^m} exp(\underline{w}.\underline{\Phi}(\underline{x}, \underline{s'}))} \tag{17}$$

$$\arg\max_{\underline{s} \epsilon S^m} p(\underline{s} \mid \underline{x}; \underline{w}) = exp(\underline{w}.\underline{\Phi}(\underline{x}, \underline{s})) \tag{18}$$

$$\arg\max_{\underline{s} \epsilon S^m} p(\underline{s} \mid \underline{x}; \underline{w}) = \underline{w}.\underline{\Phi}(\underline{x}, \underline{s}) \tag{19}$$

$$\arg\max_{\underline{s} \epsilon S^m} p(\underline{s} \mid \underline{x}; \underline{w}) = \underline{w}. \sum_{j=1}^{m} \underline{\phi}(\underline{x}, j, s_{j-1}, s_j) \tag{20}$$

$$\arg\max_{\underline{s} \epsilon S^m} p(\underline{s} \mid \underline{x}; \underline{w}) = \sum_{j=1}^{m} \underline{w}.\underline{\phi}(\underline{x}, j, s_{j-1}, s_j) \tag{21}$$

This indicates that finding the most likely label sequence is equivalent to finding the sequence that maximizes the conditional probability:

$$\arg\max_{\underline{s}\epsilon S^m} p(\underline{s} \mid \underline{x}; \underline{w}) = \sum_{j=1}^{m} \underline{w}.\underline{\phi}(\underline{x}, d, s_{j-1}, s_j) \tag{22}$$

Given that each transition from state s_{j-1} to s_j has an associated score shown below, it can be intuitively assumed that for a likely label sequence these transition scores will also be relatively high compared to an implausible label sequence.

$$\sum_{j=1}^{m} \underline{w}.\underline{\phi}(\underline{x}, d, s_{j-1}, s_j) \tag{23}$$

Thus only thing we need to do is to find a label sequence with the maximum transition scores. Again, this can also be solved by dynamic programming.

For the initial step, given that $s\epsilon S$, we have;

$$\pi[1, s] = \underline{w}.\underline{\phi}(\underline{x}, 1, s_o, s) \tag{24}$$

where s_0 is the initial label, and $\pi[j, s]$ is a table of entries that store the maximum probability for any label sequence ending with label s at position j.

For others, the equation takes the form:

$$\pi[j, s] = max_{s'\epsilon S}[\pi[j-1, s'] + \underline{w}.\underline{\phi}(\underline{x}, j, s', s)] \tag{25}$$

where $j = 2 \cdots m$ and $s = 1 \cdots k$.

Finally, we can use backpointers to calculate the label sequence with the highest transition score in $O(mk^2)$ time:

$$max\sum_{j=1}^{m}\underline{w}.\underline{\phi}(\underline{x},j,s_{j-1},s_j) = max_s\pi[j,s] \tag{26}$$

2.2. Artificial Neural Networks

2.2.1. Definition

As we have seen CRF allows tagging input sequences but in order to do so, it needs meaningful features that represent the data and in this section we explain different artificial neural networks which normally learn these features automatically.

A neural network is defined as ”a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs” [11]. In terms of computing, an artificial neural network (ANN) is a group of interconnected nodes called neurons and each one of them is responsible for computation of a specific objective function with the help of some weight and bias values.

In order to understand neural networks, we first need to refer perceptron which is a simple linear classifier that can often also be called as a single-layer neural network. As seen in Figure 2.1., given an input vector $\underline{x} = x_1 \cdots x_i$, the first half of the perceptron computes the weighted sum of the input as $\sum_i x_i$ and the second half, output, of the perceptron applies a sign function on this value. As a result, a perceptron outputs $+/-1$ and this can be used to solve linearly-separable problems.

2.2.1.1. Gradient Descent

During training, we have to find good weights in a correct way to solve the objective problem. So we need a **cost function** that we want to minimize to find good weights. For a

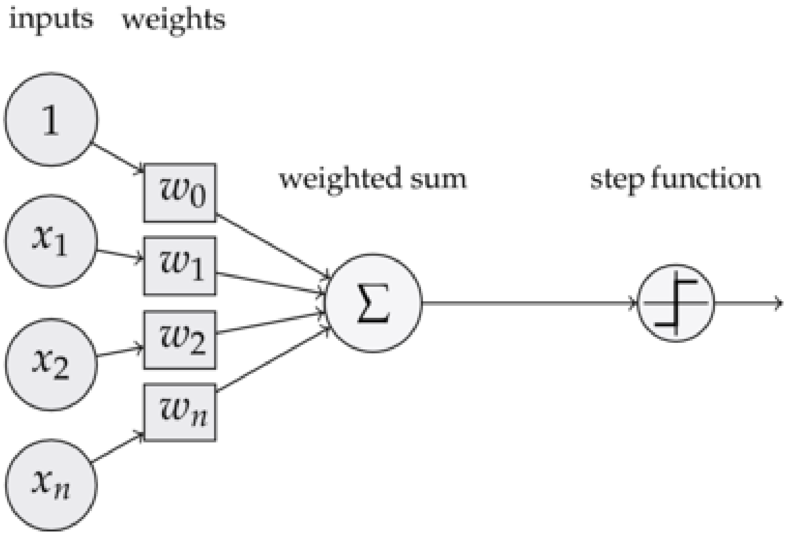


FIGURE 2.1. Perceptron

d-dimensional space, this is a hyperplane which classifies the labels correctly (Similarly it is a line or a decision boundary for a 2-dimensional space such as the case for a perceptron). We can define such hyperplane as:

$$\underline{w}^T.\underline{x} + w_0 \tag{27}$$

Given a set of $\{(x_i, y_i)\}_{i=1}^{n}$ such that $\underline{y_i} \epsilon \{-1, 1\}$, for any $\underline{x_1}$ and $\underline{x_2}$ points on this hyperplane, we can infer that;

$$\underline{w}^T.\underline{x_1} + w_0 = \underline{w}^T.\underline{x_2} + w_0 \tag{28}$$

$$\underline{w}^T.(\underline{x_1} - \underline{x_2}) = 0 \tag{29}$$

which makes $\underline{w}^T$ and $(\underline{x_1} - \underline{x_2})$ orthogonal to each other.

Similarly, for any $\underline{x_0}$, we have $\underline{w}^T.\underline{x_0} + w_0 = 0$ which also means $w_0 = -\underline{w}^T\underline{x_0}$.

Given a point $\underline{w}$ in d-dimensional space, we can compute its distance to the hyperplane by:

$$\underline{w}^T.(\underline{x} - \underline{x_0}) \tag{30}$$

$$\underline{w}^T.\underline{x} - \underline{w}^T.\underline{x_0} \tag{31}$$

$$\underline{w}^T.\underline{x} + w_0 \tag{32}$$

Thus, we can conclude the distance function of the point $\underline{w_i}$ as $d_i = y_i(\underline{w}^T.\underline{x_i} + w_0)$. Then, we can define our cost function as the summation of distance functions such as

$$Err(\underline{w}, w_0) = \sum_M -y_i.(\underline{w}^T\underline{x_i} + w_0) \tag{33}$$

where M is the set of all misclassified points. As we can see for a bad hyperplane the equation gives higher error value, however, for a better hyperplane the error value gets smaller. To achieve our goal of minimizing the error value, so that we can find good weights, we apply gradient descent.

Gradient descent is the method of taking derivative of the cost function with respect to the parameters (i.e. w) and then updating the related parameters accordingly. It nudges the parameter in a way that the value of the cost function gets smaller until it finds the global minimum or stuck in a local minima. Taking derivatives, we have

$$\frac{\partial Err}{\partial \underline{w}} = \sum -y_i.\underline{x_i} \quad w.r.t.\ \underline{w}\ and \tag{34}$$

$$\frac{\partial Err}{\partial \underline{w_0}} = \sum -y_i \quad w.r.t.\ \underline{w_0} \tag{35}$$

Finally, we can update parameters such as:

$$w_new \leftarrow w_old - \rho \frac{\partial Err}{\partial w} \tag{36}$$

$$w_new \leftarrow w_old + \rho \sum y_i . \underline{x_i} \tag{37}$$

$$w_new \leftarrow w_old + \rho y_i . \underline{x_i} \tag{38}$$

where ρ is the constant, also called **learning rate**, which determines how big a step it should take to find the minima. Often summation over M also ignored and only the multiplication is taken into account as we can see in Equation 38. This approximation is preferable because of being computationally faster than the more accurate one and it is called **Stochastic Gradient Descent (SGD)**.

2.2.1.2. Backpropagation

While perceptron can be seen as the model of a neuron, a group of neurons interconnected with each other is an **artificial neural network**, as we have already mentioned. An example of this can be seen in Figure 2.2. which is called a **feed-forward neural network** (FFNN). An FFNN is a neural network without any feedback mechanisms and it consists of *(1)* one input layer where the input is fed, *(2)* one output layer where the result (prediction) is obtained and *(3)* one or more hidden layers.

As we have already explained, during training of perceptron we can apply gradient descent which is a derivation of cost function with respect to weight, so that we can update the weight. But for a neural network consisting of multiple neurons in multiple layers, we have to define a way to apply gradient descent for all weights. This algorithm is called **backpropagation** which we will explain in the example below.

The input part of a neuron is the weighted summation of its inputs and the output part is the computation of a non-linearity such as *sigmoid* or *tanh*.

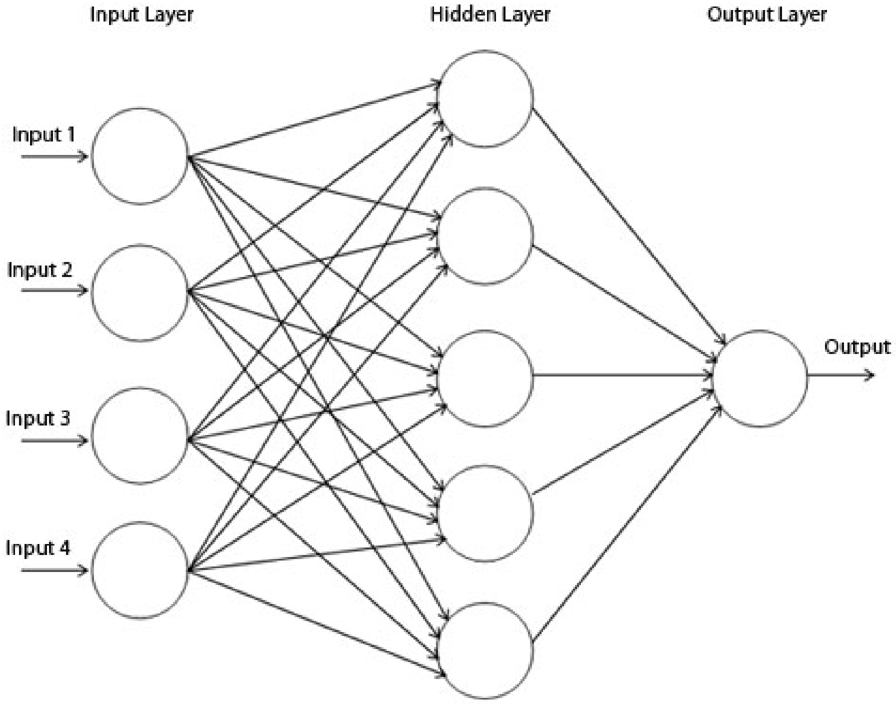


FIGURE 2.2. FFNN with one hidden layer

Given that training examples $\{(x_i, y_i)\}_{i=1}^{n}$ and prediction $\hat{y}$, we can denote, for example, the input part of the neuron k as $a_k = \sigma_j w_{kj}.z_j$ and output part as $z_k = \sigma(a_k)$ and the non-linearity function (sigmoid) of it can be defined as:

$$\sigma(a_k) = \frac{1}{1 + \epsilon^{-a_k}} \tag{39}$$

An example of this scenario can be seen in Figure 2.3.. We can also define our cost function (error) as the absolute distance between our predictions $\hat{y}$ and the true labels y as:

$$Err = |y - \hat{y}|^2 \tag{40}$$

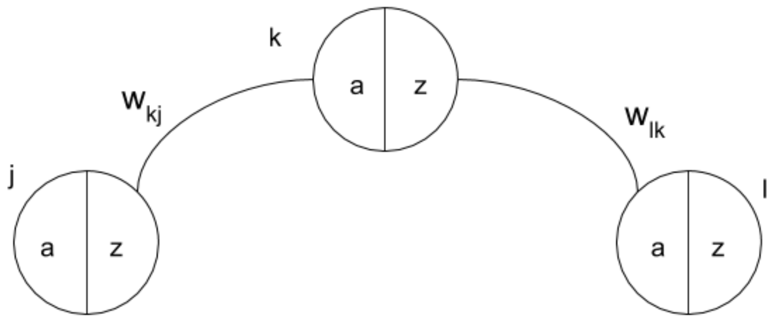


FIGURE 2.3. Simplified feed-forward neural network with one neuron on each layer

In order to find the derivation of cost function with respect to weight w_{kj}, we apply chain rule:

$$\frac{\partial Err}{\partial w_{kj}} = \frac{\partial Err}{\partial a_k}.\frac{\partial a_k}{\partial w_{kj}} \tag{41}$$

$$\frac{\partial Err}{\partial w_{kj}} = \frac{\partial Err}{\partial a_k}.z_j \tag{42}$$

$$\frac{\partial Err}{\partial w_{kj}} = \delta_k.z_j \tag{43}$$

In order to calculate the first derivative δ_k, we can again apply chain rule such as:

$$\frac{\partial Err}{\partial a_k} = \sum_l \frac{\partial Err}{\partial a_l}.\frac{\partial a_l}{\partial a_k} \tag{44}$$

$$\frac{\partial Err}{\partial a_k} = \sum_l \delta_l.\frac{\partial a_l}{\partial a_k} \tag{45}$$

and the second derivative can be computed as

$$\frac{\partial a_l}{\partial a_k} = \frac{\partial a_l}{\partial z_k}.\frac{\partial z_k}{\partial a_k} \tag{46}$$

$$\frac{\partial a_l}{\partial a_k} = w_{lk}.\sigma'(a_k) \tag{47}$$

Putting all together, δ_k can be computed as

$$\delta_k = \frac{\delta Err}{\delta a_k} \tag{48}$$

$$\delta_k = \sum_l \delta_l w_{lk}.\sigma'(a_k) \tag{49}$$

$$\delta_k = \sigma'(a_k) \sum_l \delta_l w_{lk} \tag{50}$$

This formula has the meaning that in order to compute δ_k, the computation of δ_l must be done. This proves that the derivation of the cost function with respect to the weights in that layer depends on the derivation of the cost function with respect to the weights in the next layer. Thus, if we can compute δ_k for the output layer we can compute the derivation for all of the layers that comes before.

Assume that the output layer o consists of a single neuron without any activation function, then the delta δ_o for the last layer takes the form:

$$\delta_o = \frac{\partial Err}{\partial \hat{y}} \tag{51}$$

$$\delta_o = \frac{\partial |y - \hat{y}|^2}{\partial \hat{y}} \tag{52}$$

$$\delta_o = -2(y - \hat{y}) \tag{53}$$

Therefore knowing δ_k, we can effectively compute the derivation of the cost function with respect to the weights as the equation we have found earlier dictates:

$$\frac{\partial Err}{\partial w_{kj}} = \delta_k.z_j \tag{54}$$

So, in summary, the backpropagation is the algorithm utilizing chain rule to find the derivation of the cost function with respect to each weight so that this derivation, gradient, can be used to update the weights as we have already seen in gradient descent.

2.2.2. Recurrent Neural Networks

2.2.2.1. Definition

Unlike the traditional neural networks, such as feed-forward neural networks as we have seen earlier, **recurrent neural networks** (RNNs) have a feedback mechanism and they are commonly used when the input data is sequential and of variable-length. To give an example, if we would like to build a speech-to-text application, we would use previous words or utterances to narrow down the possibilities for the word spoken now. Similarly, if we would like to build an automatic question answering system, we would want to keep relations between the current word and the words in a few sentences earlier if they are related in context. It's unclear how a feed-forward neural network could use its reasoning to achieve this.

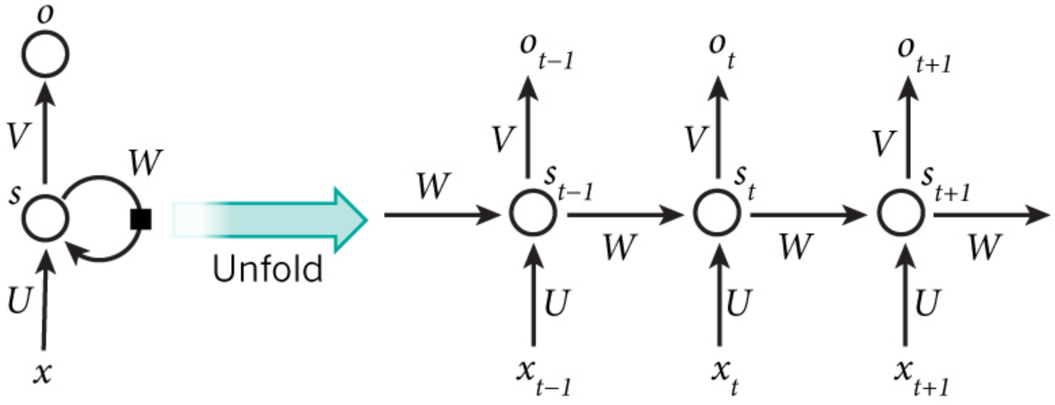


FIGURE 2.4. A recurrent neural network with its unfolded time steps

As we can see in Figure 2.4., an unfolded (unrolled) RNN depicts its different states in different time steps with each time step processing of another data point (different states). The key part here is sharing the parameters U, V, W that allow hidden state s_t to depend on the previous one, so that, it can capture information about all previous time steps. Hidden state at time step t can also be denoted as below:

$$s_t = g_t(x_t, x_{t-1}, x_{t-2}, \cdots, x_{t1}) \tag{55}$$

Figure 2.4. shows an RNN being unrolled into a full network which is a simplified depiction of the network for the complete sequence where x_t, s_t, o_t are the input data (word), hidden state and output at time step t respectively. Hidden state s_t (memory of the network) is computed based on the previous hidden state and the current input at this step. The equation of the computation can be found below:

$$s_t = f(U.x_t + W.s_{t-1} + b) \tag{56}$$

where f is a non-linearity function which is usually a *tanh* or *sigmoid.*

o_t is the output at time step t which gives us a vector of probabilities across the vocabulary and the vector can later be used for prediction purposes.

$$o_t = V.s_t + c \tag{57}$$

$$p_t = softmax(o_t) \tag{58}$$

Backpropagation Through Time (BPTT) Similar to traditional neural networks such as feed-forward neural network we define a loss function (cost function) to find good parameters U, V, W that minimize the error. A proven choice for this is to use *cross-entropy loss* which is fairly common because of its simple ability to compute how far the predictions are away from the labels.

$$L(y, o) = -\frac{1}{N} \sum_{n \epsilon N} y_n . \log o_n \tag{59}$$

For the simplicity of the examples, we take L as the summation of loss at every time step which is $L = \sum_t L_t$.

Given that time step $t = 1 \cdots T$, taking the derivative of the loss L with respect to output o_t takes the form:

$$\frac{\partial L}{\partial o_t} = \frac{\partial L}{\partial L_t} . \frac{\partial L_t}{\partial o_t} \tag{60}$$

$$\frac{\partial L}{\partial o_t} = 1.\delta_{o_t} \tag{61}$$

Since the loss function L is known δ_{o_t} can be computed. Similarly, the derivative of the loss L with respect to hidden state at the last time step s_T can be computed as below:

$$\frac{\partial L}{\partial s_T} = \frac{\partial L}{\partial o_T} . \frac{\partial o_T}{\partial s_T} \tag{62}$$

$$\frac{\partial L}{\partial s_T} = \delta_{o_T}.V \tag{63}$$

However the derivative of the loss L with respect to any hidden state s_t is not so trivial to compute as we can see in the equation below. This is mainly because the derivative has effects in two different direction: (1) changes in current output o_t and (2) changes in the next hidden state s_{t+1}.

$$\frac{\partial L}{\partial s_t} = \frac{\partial L}{\partial o_t} . \frac{\partial o_t}{\partial s_t} + \frac{\partial L}{\partial s_{t+1}} . \frac{\partial s_{t+1}}{\partial s_t} \tag{64}$$

$$\delta_t = \delta_{o_T}.V + \delta_{t+1}.W(1 - tanh^2(U.x_t + W.s_{t-1} + b)) \tag{65}$$

$$\delta_t = \delta_{o_T}.V + \delta_{t+1}.W(1 - s_t^2) \tag{66}$$

This equation clearly implies that each δ_t depends on the next δ_{t+1} and just like the backpropagation we see in traditional neural networks, we go until the last time step T and after computing its value we compute backpropagate to compute the previous ones, hence the name **backpropagation through-time (BPTT)**.

Assuming BPTT is applied to compute the derivatives of loss L with respect to hidden states, we can now compute the derivatives of loss L with respect to the parameters U, V, W effectively such as these:

$$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L}{\partial o_t} . \frac{\partial o_t}{V} \tag{67}$$

$$\frac{\partial L}{\partial V} = \sum_t \delta_{o_t} . s_t \quad and \tag{68}$$

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L}{\partial s_t} . \frac{\partial s_t}{W} \tag{69}$$

$$\frac{\partial L}{\partial W} = \sum_t \delta_{s_t} . \frac{\partial s_t}{W} \tag{70}$$

2.2.2.2. Long-Short Term Memory Networks

Although using BPTT we can compute the derivatives which can later be used to update the parameters in theory, however in practice the multiplication of multiple derivatives has tendency to cause one of the two problems called **exploding or vanishing gradient problem**. That means multiplication of the derivatives through the time steps will tend to be either very large or very small. Unlike traditional neural networks with only a few layers subject to backpropagation, RNNs (or any other deep neural networks) have many time steps (or layers) that cause this problem and it is a major obstacle that prevented RNNs to be used in any real world applications. More detail can be found on vanishing gradient in [12] which is a German article describing the problem and in another survey [13].

To give a simple example, assume that all values are scalar θ and given that number of time steps T goes to ∞, then the result θ^T goes to ∞ if $\theta > 1$ or goes to 0 if $\theta < 1$. To interpret this situation, a *vanishing gradient* leads to misleading results which has the wrong impression that the network is learning but in fact strays around and does not guide to minima. Gradients from a few time steps back does not contribute to the learning process. *Exploding gradient*, contrarily, results in taking big learning steps and consequently passes by minima and never reaches it. But clipping the gradients at a pre-defined threshold is an effective solution to this issue. That's why vanishing gradient is more problematic since it is not obvious when to occur or how to handle it.

In order to prevent the vanishing gradient problem, **Long Short-Term Memory (LSTM)** [14] and **Gated Recurrent Unit (GRU)** [15] architectures are proposed and widely adopted in the literature in recent years.

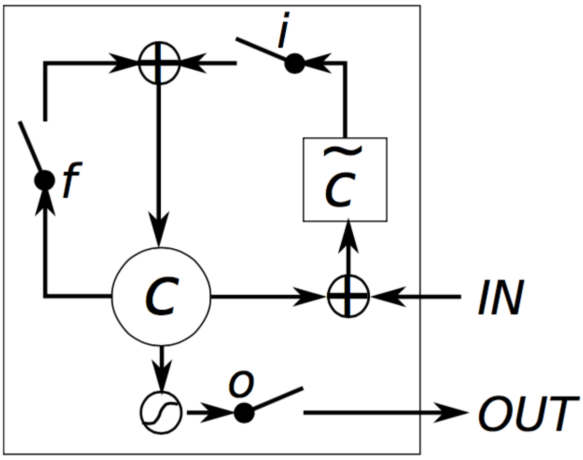


FIGURE 2.5. Input, forget and output gating mechanisms of an LSTM network. [15]

LSTM offers a number of gating mechanism to be able to decide how much of the old information it wants to keep. It is called gate because of the sigmoid function which outputs a vector with values between 0 and 1 from its input vectors. The mathematical definition of LSTM is as follows (See Figure 2.5.):

$$i = \sigma(x_t U^i + s_{t-1} W^i) \tag{71}$$

$$f = \sigma(x_t U^f + s_{t-1} W^f) \tag{72}$$

$$o = \sigma(x_t U^o + s_{t-1} W^o) \tag{73}$$

$$g = tanh(x_t U^g + s_{t-1} W^g) \tag{74}$$

$$c_t = c_{t-1} \circ f + g \circ i \tag{75}$$

$$s_t = tanh(c_t) \circ o \tag{76}$$

As we can see in the equations above; input i, forget f and output o gates are almost the same sigmoid functions except for their different parameter vectors and they decide how much of the previous hidden state for the current input the network wants to keep. Hidden state g is the same function as the hidden state computation of RNNs, but this hidden state is used as an internal (or candidate) state to compute the real hidden state. Another mechanism of LSTM is the memory of the network, c_t, that is used to decide how much of the old memory to keep. Finally, using c_t and the output gate, the new hidden state is computed. The important thing to notice here is the gating mechanisms that allow to forget/keep some of the old hidden states in order to prevent vanishing gradients during matrix multiplication.

2.2.3. Convolutional Neural Networks

Convolutional neural network (CNN) is a special kind of artificial neural network which makes use of convolution instead of standard matrix multiplication such a way that it is specifically designed to exploit certain image-specific properties. CNNs are usually used in image processing (i.e. object detection, pattern recognition). They are also used for learning character-level features in natural language processing tasks.

Typically, compared to a feed-forward neural network, a CNN consists of a number of convolutional and pooling layers along with fully-connected (affine) layers. *Convolutional layer* is the one that computes the dot product of neuron weights and a region of the previous layer (input volume). *Fully-connected layer* is similar to a layer of a regular fully-connected neural network. *Pooling layer*, on the other hand, performs a subsampling operation which results in dimensionality reduction with the aim of keeping most of the significant information. Finally, the network can be trained using gradient-based optimization like a traditional neural network to update parameters.

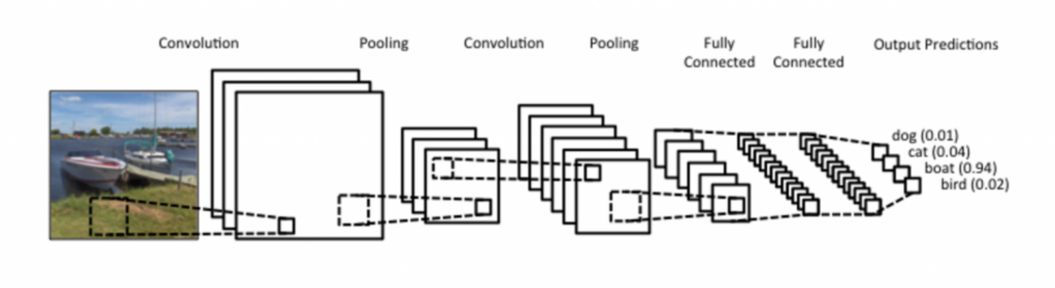


FIGURE 2.6. An example of CNN architecture for image recognition. Each convolutional layer captures another set of features from the previous layer. Source: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

2.2.3.1. Convolution

A convolution in mathematics is an operation of shifting/sliding a function w over another function x and computing the overlap between them while doing so. This can be expressed by this equation:

$$s[t] = (x \star w)(t) = \sum_{a=-\infty}^{\infty} x[a]w[t-a] \tag{77}$$

where $s[t]$ is the convolution function over t, t and a are the positional arguments that move function w over function x. This operation is adapted to process multi-dimensional input and a *kernel* (window) function is used as the shifting function with the aim of capturing valuable features of the image which is being processed. Hence the equation becomes:

$$s[i,j] = (I \star K)[i,j] = \sum_{m}\sum_{n} I[m,n]K[i-m,j-n] \tag{78}$$

where I and K are the multidimensional input and kernel functions respectively, i and j are the positional arguments. As we have seen in feed-forward neural networks, the network consists of multiple layers that each one of them contains a set of neurons. Each one of these neurons is connected to all neurons in the previous layer and naturally these connections have different weights to learn. Due to the fact that size of images can be varied and comparably high, a neuron in the first layer of a network should compute a huge number of weights. For example, to encode an image with the size of $1920 \times 1080 \times 3$ (width x height x depth, depth is the color channel: e.g. R, G, B), a neuron should have $1920 \times 1080 \times 3 = 6220800$ weights to train. Assuming a number of layers and a much more neurons, this naturally leads to a high number of neurons which is an unmanageable network and prone to over-fitting. CNNs overcome this problem such that neurons of a layer does not connect to all neurons of the previous layer which is succeeded by using a smaller kernel than the input and that allows to capture only the necessary features. Another outcome of using a smaller kernel is that the weights of the kernel are shared during shifting which naturally leads to a reduced number of trainable parameters.

Parameters of a convolutional layer are actually learnable filters that each of them is small along width and height (spatially) but is full-length in depth (e.g. 6x6x3). During forward pass, each filter is slided (convolved) over a region of the input volume (along width and height). As they slide, each one produces a 2D activation map which is a set of values computed by dot products of the filter entries and the input at a related position. So it can be concluded that each filter ”learn to activate” for a particular set of features (e.g. edges of images, faces). While filters of previous convolutional layers learn to activate for, say, edges of a plane wing, filter of next convolutional layers may learn to active for whole image of the plane.

Sparse Connections As we have briefly explained, each neuron of a convolutional layer connect to only a sub-region of the previous layer (i.e. input volume/matrix). The hyperparameter of this connectivity that determines the spatial extent is called the **filter size**. However, the size of the depth is a constant which is always the full-length of the depth. For example, assume that we have an image of size 48x48x32. With a filter size of 3x3, each layer in the convolutional layer would have (3x3x32) 288 connections to the input volume.

Spatial Arrangement The neuron and their arrangement in the output volume is another different approach CNNs take. There are three different hyperparameters that determine this:

- **Depth**: of the output volume is closely related to the number of filters. Meaning, a depth column (or fibre) is a set of neurons which are connected to the same region.
- **Stride**: determines how many steps (pixels) we move our filters. For example if the stride is 2, then filters would have move two pixels at a time. As a result, we would have a relatively smaller output volume along width and height axes.
- **Padding**: is the operation of padding zero the input volume which allows us to preserve the spatial size of the input volume.

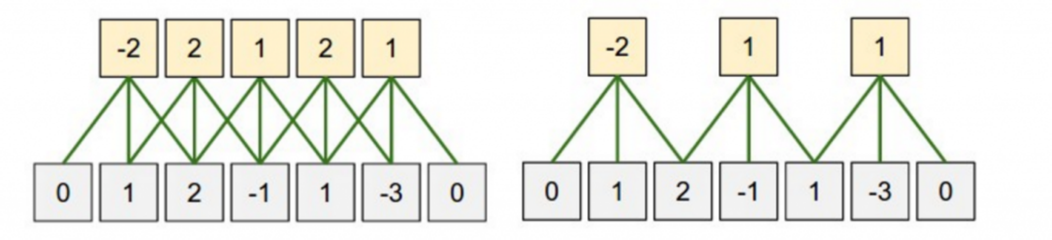


FIGURE 2.7. Examples of stride=1 on the left and stride=2 on the right. We can see smaller output when a bigger stride value is used. Source: http://cs231n.github.io/convolutional-networks/

After convolution, an activation (non-linear) function is applied to the output. This stage is called **detector** and the output is usually followed by a pooling layer.

2.2.3.2. Pooling

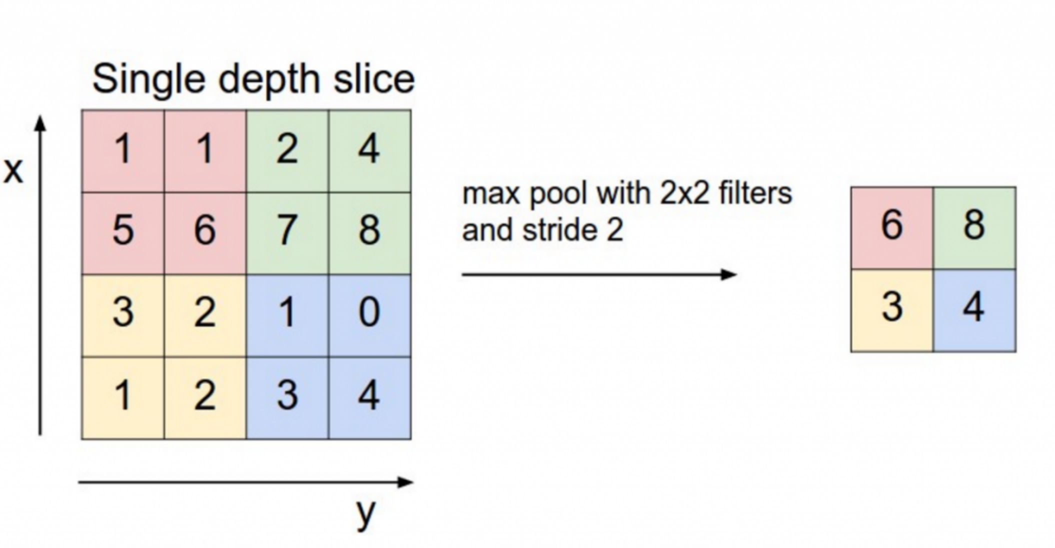
In order to reduce the amount of parameters and try to prevent overfitting, pooling is commonly used between successive convolutions. Pooling can be seen as summary of the features we need to learn most which reduces the computational cost.

Using a specific operation (e.g. max pooling, average pooling, L2 norm), the pooling computes a fixed function without parameters on every depth slice of the input and resizes it along width and height. In order to better understand this operation, assuming the size of input as W_1xH_1xD1, it requires only two fixed hyperparameters which one of them is spatial extent F (filter size), and the other is the stride S. As output, it computes a volume of size $W_2 x H_2 x D_2$ where

$$W_2 = (W_1 - F)/S + 1 \tag{79}$$

$$H_2 = (H_1 - F)/S + 1 \tag{80}$$

$$D_2 = D_1 \tag{81}$$

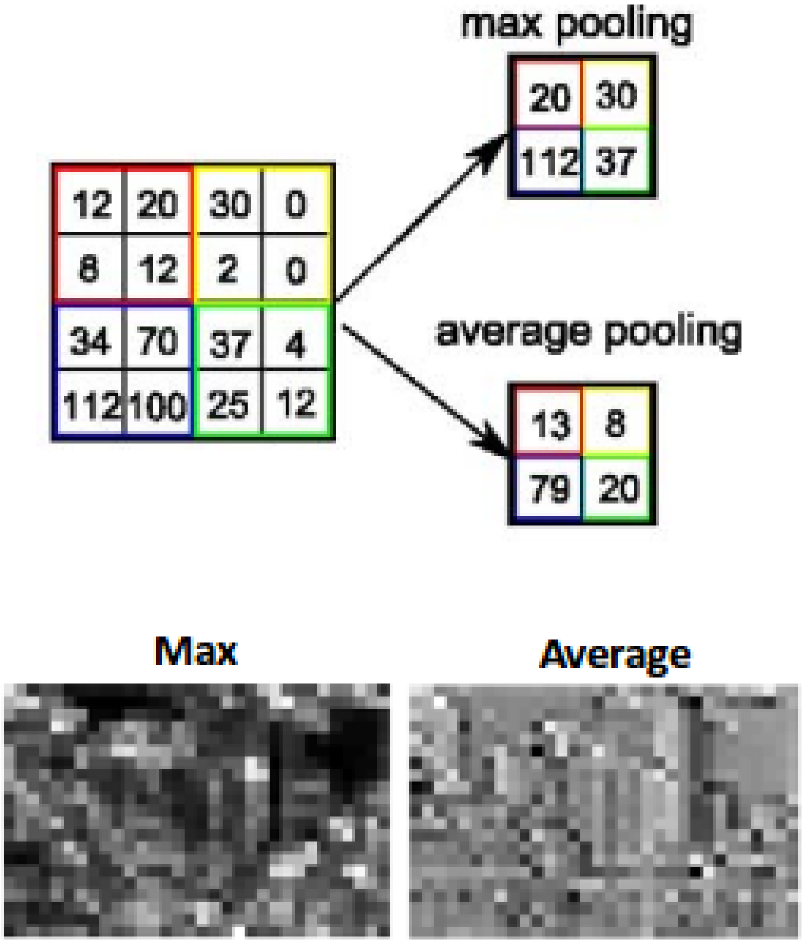


1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

FIGURE 2.8. An example of max pooling operation. Source: http://cs231n.github.io/convolutional-networks/

As we have earlier mentioned, there are different pooling operations; **max pooling** and **average pooling** to be the most common ones. Max pooling is usually preferred to make use of ”extreme” features. Assume that there are mostly smaller values in a filter and only a few of bigger ones, by using max pooling, we will lose information about smaller ones. While max pooling only takes the maximum-valued feature (”activation”) into account, average pooling uses the average of all features in a filter. When bigger values are balanced by smaller values in a filter, then average values seem like less significant for an average pooling however this way it retains more information about smaller ones. To conclude, there is not any best pooling operation to fit all cases but the choice usually depends on the dataset and the task.



12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

20	30
112	37

13	8
79	20

FIGURE 2.9. An example of difference between max pooling and average pooling

NLP Adaptation Although we have seen how CNNs can be used effectively in image processing, we have not yet discussed how to use CNNs for NLP tasks or build effective models with it.

When we use CNN for an NLP task, such as NER, the input to our network is naturally matrices of sentences instead of images. As we have previously seen for LSTM, each row of the matrix is a word representation. This word representation is usually word embeddings (e.g. word2vec [7], FastText [8]) or a *one-hot vector*. In order to ”convolve” filter over the input, filters with full-width of the rows are used. So that a filter should take full row (length of word) into account while sliding over. That’s why as a preprocessing step all words are usually zero-padded to preserve the same width. There is no explicit rule for height of the

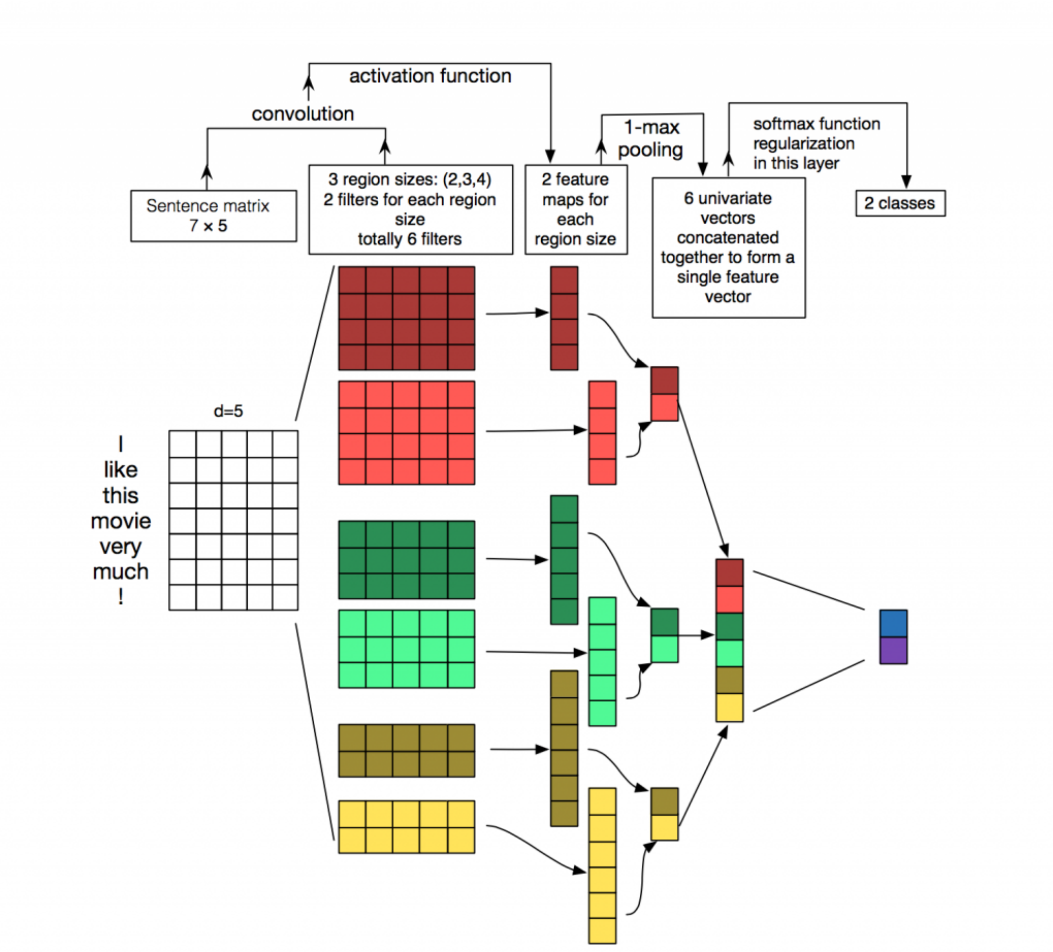

activation function
convolution
Sentence matrix
7 × 5
3 region sizes: (2,3,4)
2 filters for each region size
totally 6 filters
2 feature maps for each region size
1-max pooling
6 univariate vectors concatenated together to form a single feature vector
softmax function regularization in this layer
2 classes
d=5
I
like
this
movie
very
much
!

FIGURE 2.10. An example of CNN architecture for sentence classification. [16]

filter but usually 2-5 is used to slide over a few words at a time.

In most of the NLP tasks, the position of where a word appears in a sentence is very significant. In addition, words that are far apart to each other can be semantically related and therefore also important. These important aspects cannot be captured by CNN but in researches conducted in the recent years, we can clearly see that CNN performs very well on many NLP tasks including NER. This is mostly contributed to how fast they are trained and how efficient they are in terms of word (or character) representations. As a result we have also experimented with character-level CNN to learn good features of words before feeding them into a word-level LSTM that we have discussed earlier.

2.3. Word Representations

Representing the words by a fixed dimensional vector with the help of neural networks have shown superior performance in the recent years.

Words (or tokens) are usually mapped to discrete numeric values so that we can mathematically represent them. But using unique vectors for each word such as one-hot vector where a word is represented by 1 in one bit and 0s everywhere else, leads to sparsity problem since the cardinality of vectors must match the vocabulary size. This method also does not encode any information about words or relationships between them. However, by using word embeddings, we can represent words in a dense vector space while keeping information and relationships between them as much as possible.

2.3.1. Word2vec

Word2vec [7] is a computationally-efficient word representation algorithm that is designed to represent words without losing their semantic knowledge in a dense vector space [7]. There are two different models of word2vec, namely *Skipgram* and *continuous bag-of-words (CBOW)*. Skipgram is trained to predict the surrounding words based on the current word. CBOW, on the other hand, is trained to predict the current word based on the context words.

In this regard, the aim is to find good word representation for the predicting the surrounding words. Given a corpus of T, the objective is to maximize the probability that:

$$\arg\max_{\theta} \prod_{w \in T} [\prod_{c \in C(w)} p(c|w;\theta)] \tag{82}$$

$$\arg\max_{\theta} \prod_{(w,c) \in D} p(c|w;\theta) \tag{83}$$

where $p(c|w;\theta)$ is the probability of context word c given that word w and $C(w)$ is the context words of word w. Finally D denotes the set of all word-context word pairs in the T.

Probability of context word can be represented as follows:

$$p(c|w;\theta) = \frac{\epsilon^{v_c \cdot v_w}}{\sum_{c' \in C} \epsilon^{v'_c \cdot v_w}} \tag{84}$$

where $v_c, v_w \in \mathbb{R}^d$ are the vector representations of context word c and word w respectively and C represents the set all of context words. After applying log function to switch to summation, the aim is to find good parameters that give rise to the following equation:

$$\arg\max_{\theta} \sum_{(w,c) \in D} logp(c|w) = \sum_{(w,c) \in D} (log\epsilon^{v_c \cdot v_w} - log \sum_{c' \in C} \epsilon^{v'_c \cdot v_w}) \tag{85}$$

Although this can be computed, it is noted in the paper that this is computationally very expensive due to the summation over all the context words c' and they argue that the full probabilistic model is not needed and instead a *binary classification objective* can be used with the aim of differentiating the target words from the noise words in the same context. In order to find parameters that maximize the probability of all the observations come from the data (all observations are target words not noise words), they change the objective function such that:

$$\arg\max_{\theta} \prod_{(w,c) \in D} p(D = 1|w, c; \theta) \tag{86}$$

$$= \arg\max_{\theta} log \prod_{(w,c) \in D} p(D = 1|w, c; \theta) \tag{87}$$

$$= \arg\max_{\theta} \sum_{(w,c) \in D} logp(D = 1|w, c; \theta) \tag{88}$$

where $p(D = 1|w, c; \theta)$ denotes the probability that the pair of (w, c) comes the corpus data, and similarly $p(D = 0|w, c; \theta) = 1 - p(D = 1|w, c; \theta)$ denotes the probability that the pair of (w, c) does not come the corpus data. Using softmax on the probability, the objective function becomes:

$$\arg\max_{\theta} \sum_{(w,c)\in D} log\frac{1}{1+\epsilon^{-v_c . v_w}} \tag{89}$$

This can be solved if the set of parameters θ is set such that the probability $p(D=1|w,c;\theta)$ becomes 1 for every pair of (w,c). In this regard, the parameters $v_c = v_w$ must be equals such that the product of them K is large enough. As a consequence another mechanism must also be used to prevent that all vectors from having the same value. This is done by presenting some (w,c) pairs that are not in the data. To do so, a randomly-generated pairs are produced as "negative samples" so that the objective function becomes:

$$\begin{aligned}
&\arg\max_{\theta} \prod_{(w,c)\in D} p(D=1|w,c;\theta) \prod_{(w,c)\in D'} p(D=0|w,c;\theta) && (90)\\
&= \arg\max_{\theta} \prod_{(w,c)\in D} p(D=1|w,c;\theta) \prod_{(w,c)\in D'} 1-p(D=1|w,c;\theta) && (91)\\
&= \arg\max_{\theta} \prod_{(w,c)\in D} logp(D=1|w,c;\theta) \prod_{(w,c)\in D'} log(1-p(D=1|w,c;\theta)) && (92)\\
&= \arg\max_{\theta} \sum_{(w,c)\in D} log\frac{1}{1+\epsilon^{-v_c . v_w}} + \sum_{(w,c)\in D'} log(1-\frac{1}{1+\epsilon^{-v_c . v_w}}) && (93)\\
&= \arg\max_{\theta} \sum_{(w,c)\in D} log\frac{1}{1+\epsilon^{-v_c . v_w}} + \sum_{(w,c)\in D'} log(\frac{1}{1+\epsilon^{v_c . v_w}}) && (94)
\end{aligned}$$

This is a binary logistic regression probability where high probabilities are assigned to the real words w_i in the corpus D and low probabilities for the noise words in D'. It is again noted that optimization of loss function corresponds to the number of noise words instead of the whole vocabulary. Another important peculiarity of this approach is that the model learns both the word representation and context representation simultaneously which makes it non-convex.

2.3.2. FastText

FastText [8] is another word representation technique based on *Skipgram* model of word2vec and it represents each word with a collection of *character n-grams*. Given a word w, a set of character n-grams G_w is generated. For example, for a word *heceleme* (meaning *spelling*) and $n = 3$, these are the n-grams;

$$< he, hec, ece, cel, ele, lem, eme, me > \tag{95}$$

Additionally, the word itself w (e.g "heceleme") is added to the G_w to allow it to learn the vectorial representation of the whole word. By using character n-grams, FastText can even represent out of vocabulary (OOV) words by generating their vectorial representation from their vectors of n-grams. This concept makes it suitable for especially morphologically-rich languages such as Turkish because of the extensive number of rare words, unlike other word representation algorithms where they represent each word in a vocabulary with a single unique vector and fail to handle out-of-vocabulary words by doing so.

Due to the fact that FastText is an extension of Skipgram model of Word2vec, we will first discuss the Skipgram model and then provide the extension to explain FastText. Given that we have a set of words with the size T and each word is indexed by $t \in \{1, ..., T\}$, objective function is

$$\sum_{t=1}^{T} \sum_{c \in C_t} logp(w_c|w_t) \tag{96}$$

where C_t is the set of context words and a word is a context word if it surrounds the word w_t. Assume that we have a predefined score function s that outputs scores given a word w_t and its context C_t. Given w_t the probability of w_c can be defined as

$$p(w_c|w_t) = \frac{\epsilon^{s(w_t,w_c)}}{\sum_{j=1}^{W} \epsilon^{s(w_t,j)}} \tag{97}$$

Despite this probability can be interpreted as the prediction of a single context word w_c surrounding w_t, we would like to compute the probability of the presence of multiple context words. Given a word w_t, its surrounding (context) words can be used as positive example and some other (non-context) words can be used as negative examples in this regard:

$$log(1 + e^{-s(w_t,w_c)}) + \sum_{n \in N_{t,c}} log(1 + e^{s(w_t,n)}) \tag{98}$$

Here $N_{t,c}$ is the randomly-sampled negative examples. Using a vectorial representation for word w_t and one of its context words w_c, the aforementioned score function s can be defined as

$$s(w_t, w_c) = u_{w_t}^{\intercal}.v_{w_c} \tag{99}$$

where u_{w_t} is the vector of w_t and v_{w_c} is vector of w_c. This log-likelihood and scoring functions are the proposed solution by Skipgram which does not take morphological structure into account.

In FastText, given that a word w is decomposed to its n-grams $G_w \subset \{1, ..., G\}$, it has been proposed a different scoring function as follows:

$$s(w, c) = \sum_{g \in G_w} z_g^{\intercal}.v_c \tag{100}$$

where z_g is the vectorial representation of an n-gram g of word w. Using vectorial representation of n-grams allow it to share some of the parameters between words and, thus, learn meaningful representations for OOV words.

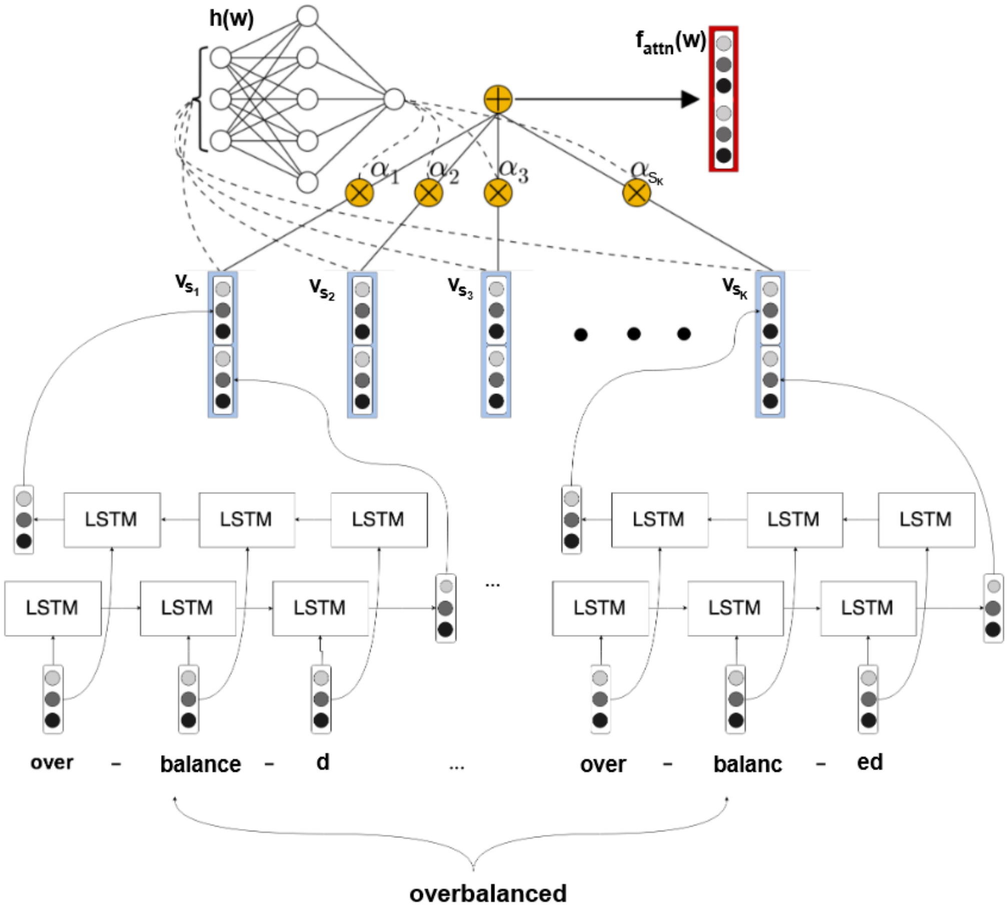


FIGURE 2.11. Architecture of morph2vec with an attention mechanism on top of bi-LSTMs. [9]

2.3.3. Morph2vec

Morph2vec [9] is another method learning to generate word representations from its subword information. It generates a group of candidate word segmentations and learns the vectorial representation of word from these candidate segmentations using an attention model. Without using any external morphological segmentation tool that produces one segmentation for a word, this unsupervised approach generates a number of candidate segmentations in which all of them contribute to the final word representation.

Each morphological segmentation (*morpheme*) of a word has its own representation and word representation of a group of segmentation is computed as follows

$$v_{s_i} = f(w_{s_i}) = f(v_{m_0}, ..., v_{m_n}) \tag{101}$$

where v_{m_i} is a vector of morpheme m_i with a dimension of d_{word}. With the aim of obtaining a function f, bidirectional LSTM is trained on the morphemes of a word. Then an attention mechanism is used on top of bidirectional LSTMs to optimize segmentation weights α_i. Summation of these segmentation weights is 1.

$$\sum_{i}^{S_w} \alpha = 1 \tag{102}$$

where S_w is the set of all segmentations of word w.

Finally using these weights α_i and vectorial representation v_i of segmentation s_i that we obtain from bidirectional LSTM, the word representation can be computed as follows:

$$f_{\text{attn}}(w) = \sum_{i} \alpha_i . v_{s_i} \tag{103}$$

In order to train the model, an objective (loss) function is defined which aims to minimize the cosine distance between the word vectors of this model and pretrained word vectors obtained from a word2vec model. The objective function is as follows:

$$J(\theta) = \sum_{i=k}^{N} h(w_k) + \frac{\lambda}{2} . \, ||\theta||_2^2 \tag{104}$$

where N is the size of the training set and $h(w_k)$ is the cost function of word w_k.

2.4. Transfer Learning

As a machine learning technique to improve performance of the model, **transfer learning** is the process of using knowledge learned from a source task in another (target) task, so that the model improves its results on the target task. Transfer learning approaches can be grouped into three categories as suggested by [17]:

- **Cross-domain transfer**: aims to transfer knowledge from a domain to another domain with preferably fewer labeled data in the same language. These two domains may or may not have mappable label sets.
- **Cross-application transfer**: aims to transfer knowledge from an application to another application in the same language such as POS tagging, chunking and named entity recognition where all of them are different applications of sequence labeling.
- **Cross-lingual transfer**: aims to transfer knowledge from a language to another one, as the name suggests. This may be achieved only with languages that share the same alphabetical characters and have similar morphology, so that character-level information can be exploited.

Transfer learning is accomplished by either (1) training a model on a source task and then using this model as an additional source of knowledge to train another model on a target task or (2) training a model on both source task and target task simultaneously while sharing some of the parameters. We have taken the latter approach in this thesis. Here, the source task is the named entity recognition on news data and the target task is the named entity recognition on tweet (noisy) data which is smaller dataset and has comparably less number of labeled entities.

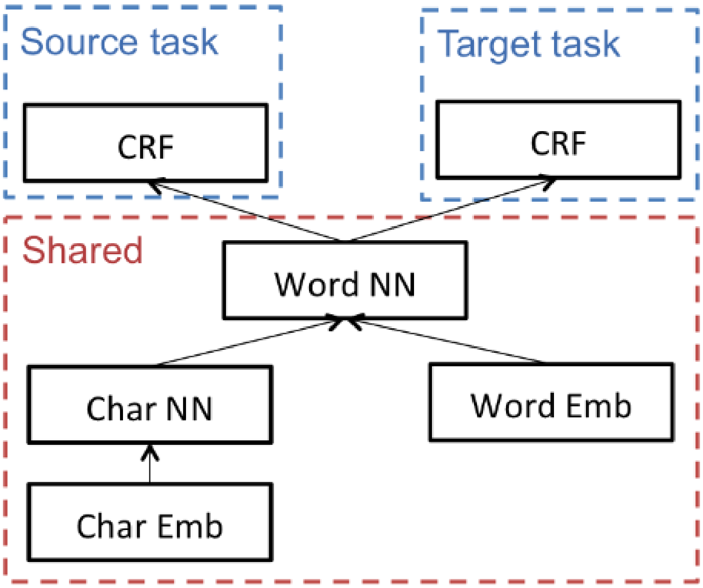


FIGURE 2.12. Architecture of the proposed transfer learning model with separate CRF layers for different domains. [17]

3. LITERATURE REVIEW

In this chapter, we thoroughly review the studies on Turkish named entity recognition on both formal data and noisy/informal data. Similarly, we review the studies on English named entity recognition with the primary focus of noisy data. We also explicate their related features and word (and sub-word) level embedding techniques. Finally, the contributions made for transfer learning are reviewed.

3.1. Named Entity Recognition on Turkish

We can group NER in Turkish into two categories, namely formal data and noisy/informal data.

3.1.1. Studies on Formal Data

Gungor et al. (2017) [18] propose a similar model to that of Lample [4]; they use bidirectional-LSTM along with CRF on top of it. Input to this model is a combination of word, character and morphological embeddings. Firstly, character embeddings are learned using a character-level bidirectional LSTM. Secondly, morphological embeddings are also learned similarly using another bidirectional LSTM. Input to this morphological-level LSTM is a sequence of tags retrieved from the morphological analysis. Thirdly, word embeddings are learned by Skipgram model of Word2vec are concatenated with character embeddings and morphological embeddings. This final embeddings is then fed into another bidirectional-LSTM which is responsible for capturing the representations of words. The importance of using bidirectional LSTM is to capture both forward and backward representations. Finally output of the LSTM is used to decode the possible tagging sequence using CRF. With this model, they have reported 93.59% F1 score for Turkish and 79.59% F1 score for Czech.

Şeker and Eryiğit (2017) [6] is the state-of-the-art on Turkish NER which proposes another CRF-based model using an extensive set of features. They also provide the re-annotated

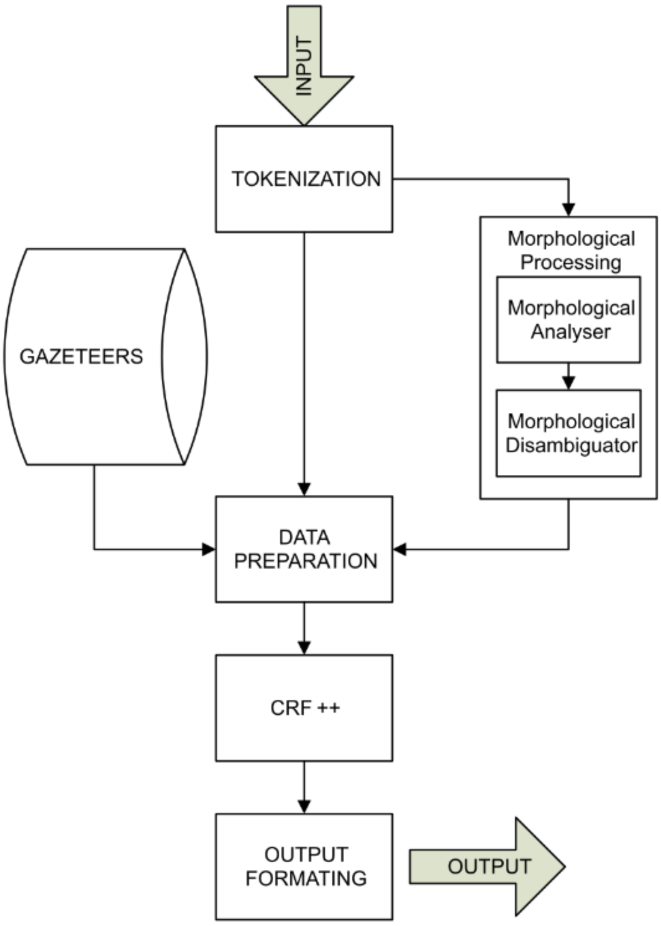


FIGURE 3.1. A CRF-based state-of-the-art Turkish NER model incorporating hand-crafted features and gazetteers. [6]

versions of the two commonly used Turkish datasets: news dataset [19] and Twitter dataset [20]. Re-annotated versions also include TIMEX and NUMEX types along with ENAMEX types. In addition, they also annotated a new Turkish treebank from social media: ITU Web Treebank (IWT) [21]. Proposed model uses respectively: tokenization, morphological processing (morphological analyzer and morphological disambiguator) [22], data preparation (feature selection using morphological tags and gazetteers among others), and finally CRF. During tokenization, punctuation characters including apostrophe are also considered as a token. Morphological analyzer produces possible word analysis while morphological disambiguator aims to select the most probable one, hence morphological processing outputs morphological tags for a given word. They also make use of two kind of gazetteers: base gazetteer which is a list of person and location names (261K tokens) and generator gazetteer which is a list of PLOs and currency units (156 tokens). During data preparation, feature vectors are formed using the raw data, the gazetteers and the morphological tags. They also noted that using IOB or IOBES (IOB2) decreased their model performance so, instead, they used raw tags (e.g. ”PERSON” instead of ”B-PERSON”, ”E-PERSON”) during training. Features used for CRF labeling include: morphological features (such as stem, POS tags, noun case, proper noun, inflectional features), lexical features (case feature, is-start-of-the-sentence), gazetteers lookup features, TIMEX and NUMEX related features (is-numeric-value, has-percentage-sign, is-oclock-term, has-column-indicator, month gazetteer, currency gazetteer). In order to adapt the model for user-generated content (noisy data), they also used: auto capitalization gazetteer (a list of names with little chance of being used as common nouns) as feature. Hence, they aim to differentiate proper names from common nouns. Additionally, they used mentions as feature. Finally, they reported 67.96% F1 score on TDS1 v4 (ENAMEX, TIMEX, NUMEX) [20], 56.02% (ENAMEX, TIMEX, NUMEX) and %49.02 (ENAMEX) on TDS2 [23] [24], 51.61% (ENAMEX, TIMEX, NUMEX) on TDS3 [25].

3.1.2. Studies on Noisy Data

Celikkaya et al. (2013) [20] is the first work that focus on user-generated/noisy data for Turkish. They used conditional random fields (CRF) as the probabilistic model to label data which has proved to be successful for overlapping features focusing on conditional distribution. Thus making it a better choice to label sequential data such as NER. In order to train CRF some hand-crafted morphological and lexical features such as stem, POS tag, noun case, lower/upper case are used along with gazetteers. Another contribution is the creation of three new datasets (forum, speech and twitter) which the Twitter dataset is also the one we used for our NER research. Finally they reported 19.28% F1 score on Twitter data and 91.64% on news data. This is also another indication that NER on noisy data does not perform as well as NER on news texts.

Küçük and Steinberger (2014) [23] employs normalization scheme and proposes some expansion of existing lexical resources with the aim of adapting an existing rule-based NER system for Turkish Twitter data. During tweet normalization phase; they removed consecutively repeated characters excluding the valid ones existing in a unique words list but this resulted in removing some valid words (e.g. Çanakkale) if it is not in the unique words list. Therefore, they first applied NER phase then used normalization for remaining words then applied NER phase again. During NER phase, an existing lexical resource (list of person, location, organization (PLOs) names and patterns for NE types time, date, money, percent and PLos) are expanded based on the fact that most of the Turkish tweets misses the diacritics (ç, ğ, ı, ö, ş, ü) resulting in ill-formed words (e.g. şiir [poem] → siir or İstanbul → Istanbul). As a result, they achieved 53.45% F1 score on *Tweet Set-1* [24] with normalization and 46.93% F1 score on *Tweet Set-2* [20]

Eken and Tantug (2015) [25] use another CRF-based approach in order to NER in Turkish tweets. Instead of using morphological analyzers which do not give good results on noisy data, they prefer to use first and last four characters of words. Additionally, in order to train CRF on news data, they use these features as well: existing of apostrophe character, case of the word, start of sentence, gazetteers for NE types of PLOs (with optional Levenshtein distance-based matching). As a result they reported 46.97% F1 score on *Tweets Set-1* which is an imbalanced dataset provided by them, and 28.53% on *Tweet Set-2* [20].

Okur et al. (2016) [26] is another important work due to being the first neural-network based approach without language-dependent feature engineering on Turkish noisy data. Trained on a news corpus (BOUN web corpus) of 423M words [27], [28] and 21M Turkish tweets [29], they obtained word embeddings using Skipgram model of word2vec. We are also using these embeddings for our pre-trained word vectors which further details can be seen in the upcoming sections. Expending the work of [30], they use a regularized averaged multiclass perceptron with the following features: context (window of two tokens), capitalization, (BILOU) tags of previous two tokens, word type flags (all-capitalized, is-capitalized, all-digits, contains-apostrophe, is-alphanumeric), first and last three (or four) characters of token, word embeddings obtained from word2vec. They also employ a tweet normalization as a preprocessing step using the work of [31]. The normalization takes two phases: ill-formed word detection and candidate word generation. For the first phase, a morphological analyzer (Şahin et al., 2013) is used with the help of an abbreviation list and list of abbreviated words so that all out-of-vocabulary words can be filtered out and sent directly to the second phase. During the second phase which is basically a pipeline of seven different steps, the output of each step is checked again by the morphological analyzer to detect if it became an in-vocabulary word. If it is, then the word no longer continues the next step. Other steps that is reported by them include: ”letter case transformation, replacement rules and lexicon lookup, proper noun detection, deasciification, vowel restoration, accent normalization, and spelling correction” [26]. They reported 48.96% F1 score on *TwtDS-1* [20] with the model trained on Turkish tweets (*TwtDS-2* [24]) using word embeddings, normalization, and filtering out non-Turkish. Furthermore, they also reported 56.79% F1 score on *TwtDS-2* with the model trained on Turkish news using word embeddings and normalization only.

3.2. Named Entity Recognition on English

We can also categorize NER in English into two categories; studies on formal and informal/noisy data.

3.2.1. Studies on Formal Data

Huang et al. (2015) [32] is the first paper in the literature that uses a combination of bidirectional LSTM and CRF for the purpose of sequence labeling in NLP. They experiment with a variety of these models including; LSTM, BiLSTM-CRF, CRF, LSTM-CRF, BiLSTM-CRF and also their models incorporate a number of hand-crafted features such as spelling features (capitalization, punctuation, various word patterns), context features (uni-gram and bi-gram features) and Senna word embeddings [33]. Consequently, they report an F1 score of 88.83% using the BiLSTM-CRF model and 90.10% using the same model and gazetteers on CoNLL’03 dataset.

Chiu and Nichols (2015) [34] proposes a model of bidirectional LSTM and CNN without heavy feature engineering. They extract character-level features using a CNN and then concatenates these with pre-trained word embeddings (either from Senna, GloVe or word2vec). They also make use of some hand-crafted features such as lexicon features and character-level features (capitalization, existence of uppercase, lowercase or punctuation). As a result, they achieve an F1 score of 91.55% on CoNLL'03 dataset.

Lample (2016) [4] proposes a neural network model that uses no hand-crafted features or external resources with specific domain-knowledge such as gazetteers. Consequently the model achieves the state-of-the-art results in four languages on experiments conducted on formal text. The proposed model is a combination of a bidirectional LSTM and CRF on top of it. While BiLSTM is used to obtain word representations, CRF is responsible for tagging named entities. As we have already seen, this is the base model for most of the newer NER models evaluated on formal and/or noisy data.

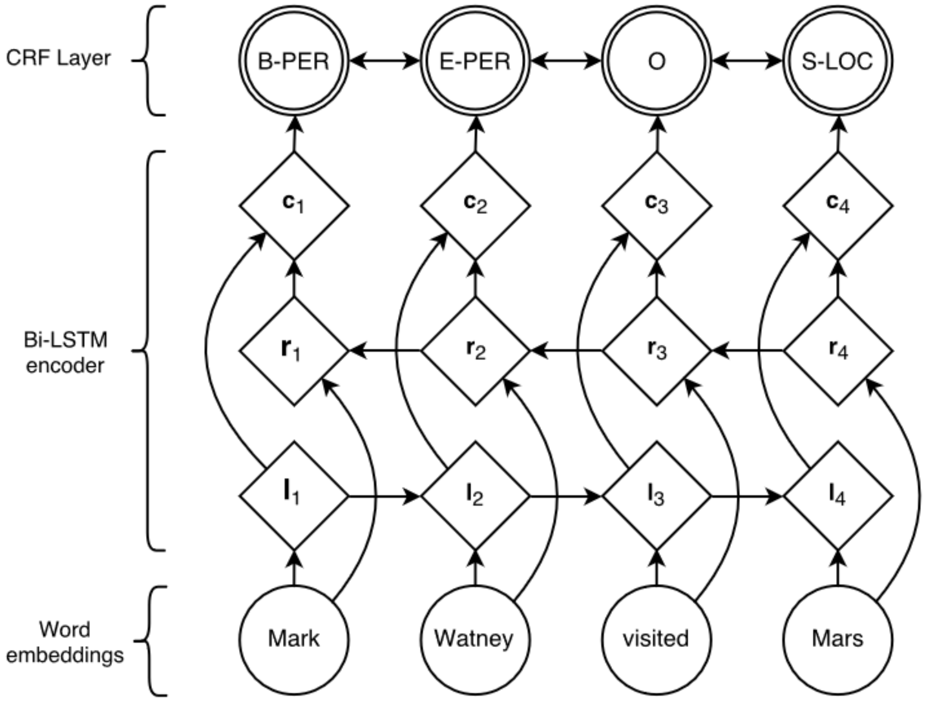


FIGURE 3.2. Overview of the BiLSTM-CRF model [4]

In Figure 3.2., we can see the architecture of this model. Word embeddings are input to a BiLSTM network that is used to capture right and left side context of the sentences. Thus the concatenation of these two representations include the word representation in context. These word representations are then fed into a CRF layer to label the named entities. CRF, as we can see later in 2., is widely used for jointly tagging tasks such as POS tagging and NER. This is because tagging usually comprises dependencies between different output labels and CRF allows us to tag output labels while taking these dependencies into account. As tagging scheme, IOBES is used to impose these dependencies (e.g. B-LOC must be followed by either I-LOC or E-LOC, S-PER cannot be neighbor with any other PERSON tag.).

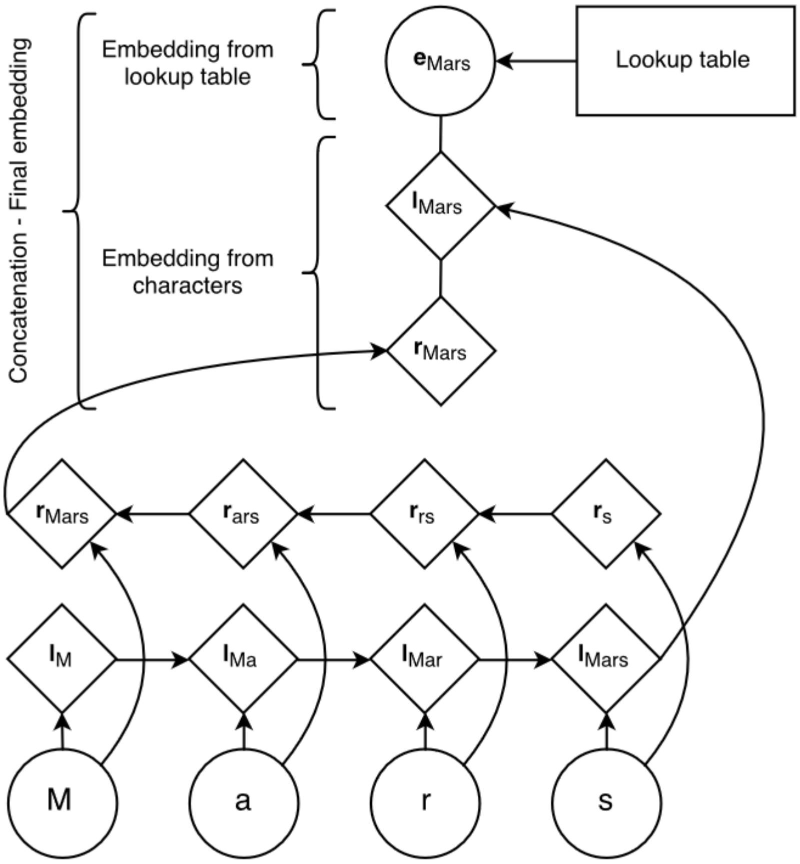

Concatenation - Final embedding
Embedding from lookup table
Embedding from characters
e_Mars
Lookup table
l_Mars
r_Mars
r_Mars
r_ars
r_rs
r_s
l_M
l_Ma
l_Mar
l_Mars
M
a
r
s

FIGURE 3.3. Word embeddings consists of pretrained word embeddings and character embeddings [4]

Word embeddings are obtained from pre-trained word embeddings and character representations which is output of another character-level BiLSTM network. The whole architecture can be seen in Figure 3.3.. Pre-trained word embeddings are embedding trained on Word2vec

(Skipgram) model which is another dense vector representation adopted widely. Using another BiLSTM network for word embeddings is to be able to capture any character-level features. Similar to word-level BiLSTM, output of this network consists of left and right context of words. Concatenation of these LSTM outputs and pre-trained embeddings form the final word representations. They have also used dropout on the final word embeddings just before word-level BiLSTM. This ensures the model does not solely depend on one of the embeddings and therefore improves the performance. As a result, they achieved 90.94 F1 score on CoNLL'03 dataset for English which is a clear indication that using neural architectures may also represent character-level and word-level features instead of using hand-crafted features or depending on domain-specific resources.

Ma and Hovy (2016) [35] proposes a similar bidirectional LSTM-CRF model that incorporates word embeddings obtained from GloVe and character-level embeddings trained on a CNN layer. They also use SGD with gradient clipping of 5.0 and learning decay rate of 0.05, moreover, apply dropout operation before the CNN layer and before and after the bidirectional LSTM layer. As a result they obtain an F1 score of 91.21% on ConLL'03 dataset.

3.2.2. Studies on Noisy Data

Ritter (2011) [36] proposes a pipeline incorporating a tweet POS tagger (T-POS), chunker (T-CHUNK) and NE recognizer (T-NER) with hand-crafted features and dictionaries. T-POS uses CRF with POS dictionaries, Brown clusters and some other unmentioned "spelling and contextual features". T-CHUNK, again, also uses CRF with POS tag outputted by T-POS, Brown clusters and some other features.They also use a capitalization classifier (T-CAP) with the aim of determining whether a tweet has informative capitalization or not. Informative capitalization is when a tweet is properly capitalized. They use SVM for training this classifier. Finally they use CRF with previously mentioned/outputted features for NE segmentation and apply LabeledLDA for NE classification. During this step, they also make use of lists of entity-type pairs gathered from Freebase dictionaries.

In order to mention briefly, LabeledLDA (Labeled Latent-Dirichlet Allocation) [37] is a probabilistic topic model that aims to associate a label with a topic. LabeledLDA is a strong generative model in the sense that it can model ”each document with as a mixture of underlying topics and generates each word from one topic” [37]. LabeledLDA is an extension of traditional LDA with a supervised approach that limits the topic model from a set of label.

In [1], winner of 3rd workshop on WNUT’17, proposes a multi-task learning approach that provides both NE segmentation and NE categorization using a conditional random fields (CRF) fed by features extracted from a character-level convolutional neural network (CNN) and a word-level bidirectional long-short term memory (LSTM). The features for character-level CNN are extracted from an orthographic encoder which is similar to that of [38] and used to represent orthographic characteristics of a word such as capitalization, punctuation. CNN is, therefore, used to learn word shapes and orthographic features. CNN architecture consists of two stacked convolutional layers followed by global average pooling and fully-connected layer with ReLU activation function. On the other hand, the features for word-level bidirectional LSTM are Twitter word embeddings from Skipgram model using word2vec and POS tag embeddings. As a result, bidirectional LSTM with 100 neuron for each direction learns both forward and backward representations of words. Additionally, lexicons for each NE category are fed into a fully-connected layer with 32 neurons and ReLU activation function. Resulting vectors which obtained by concatenating character-level embeddings, word-level embeddings and lexicon vectors, are fed into the multi-task network where a single-neuron layer with sigmoid function is used for NE segmentation task whereas a 13-neuron layer with softmax function is used for NE categorization task. Addition of both losses are used during training and finally output of the network is used to feed CRF-layer in order to jointly predict the sequence of labels. Using this methodology, they achieved 41.86 F1 score on entities and 40.24 F1 score for surface forms.

In another work [39], comprehensive word embeddings with multi-channel information are used in a bidirectional LSTM-CRF network ”without using any hand-crafted features such as gazetteers or lexicons” [39]. Following the example of [4], word embeddings are made of pre-trained word embeddings and character-level embeddings obtained from another bidirectional LSTM. They also incorporate syntactical information by using POS tags, dependency roles and word position in the sentence and head position. Concatenation of these embeddings results in the final word embeddings which is then fed into the word-level bidirectional LSTM and CRF layers. In order to obtain POS tags and dependency roles, TweeBank is used as part of the syntactic information. Moreover, for pretrained word embeddings, GloVe is used with dimension of 100. Finally, they achieved 40.42 F1 score for entity-level and 37.62 F1 score for surface-forms.

Another participant of WNUT'17 [40], uses a CRF as an ensemble-based approach which uses features learned from CRF, support vector machine (SVM) and an LSTM. For CRF-based features, a CRF was trained with L2 regularization and with a group of hand-picked features such as POS tag, local context, chunk, suffix and prefix, word frequence and a collection of flags (is-word-length-less-than-5, is-all-digit etc.). For SVM-based features, an SVM classified is built with polynomial kernel with the same features used for CRF-based model above. Thirdly, for LSTM-based features, two different LSTM networks are used which the first one is used to find out NE boundaries then this is used as input to the second one which aims to identify NEs with the help of a collection of hand-picked features and pre-trained embeddings of 150 length which is learned from word2vec. Finally, the output of these three models are concatenated and fed into another CRF layer to predict the final sequence of NE labels. It is noted that this final CRF layer is used as a selector in the ensemble with the aim of utilizing output of three models. Consequently, they achieved 38.35 F1 score for entity-level and 36.31 F1 score for surface forms.

[41] tries a statistical approach, context-sensitivity, where each word is associated with its contexts and context conditional probabilities are used to figure out NE tag probabilities. For entities with multiple words, a pre-determined threshold value and harmonic mean of word likelihood values are used to determine a potential NE membership. As a consequence of using probabilistic approach, for the conflicting situations where a token is predicted for multiple NE membership; **(1)** predictions appeared first, **(2)** longer predictions and **(3)** predictions with higher likelihood have higher precedence respectively. TO achieve better results, in

addition to WNUT'17 dataset, WNUT'16 dataset and a group of lexicon and external dictionaries are also used. In conclusion, they achieved a result of 26.30 F1 score for entity-level and 25.26 F1 score for surface forms.

In [42], inspired by the work of [38], a bidirectional LSTM-CRF model is used but instead of orthographic features from the original work, LDA topic modeling and POS tags are chosen. Character embeddings are obtained from a character-level bidirectional LSTM which are then fed into word-level bidirectional LSTM along with pretrained word embeddings extracted by GloVE. Finally, another bidirectional LSTM is used to obtain word representations. The input to this LSTM is the concatenation of word embeddings from the previous LSTM, POS tags obtained from GATE and LDA topics. In order to generate topics, a combination of three different datasets is used where each entry (e.g. tweet) is handled as one document. After training of 250 topics, for each word a document-level and word-level topics are assigned as 250-dimensional embeddings. Final output of the last LSTM is then fed into CRF layer. Additionally two fully-connected layers are used between the last LSTM and CRF layers in order to better capture higher order representations. As a result, they achieved a performance of 39.98 F1 score on entity-level and 37.77 F1 score on surface forms.

3.3. Transfer Learning

Yang et al. (2017) [17] propose a comprehensive framework for transfer learning in named entity recognition. Transfer learning is the procedure of improving the performance of a target task by incorporating knowledge from a source task. Three different neural architectures are presented in this framework that all of them are based on a (character-level NN + word-level NN + CRF) model, similar to [4]. It is stated that neural layers in these architectures can be either CNN or LSTM. The first architecture targets cross-domain transfer learning by applying label mapping on top of the CRF layer. The second architecture is applicable when there are two sequence labeling tasks with different label sets for the same language (e.g. POS tagging and NER). Two different CRF layers are used to tackle this problem while sharing all other NN layers and their parameters. The third architecture is presented as solution to multi-lingual transfer learning by sharing the same character-level representations between two different (word-level NN + CRF) models. As a result, they have achieved an F1 score of 91.26 for NER on CoNLL2003 dataset and 95.41 for chunking on CoNLL2000 dataset. Comparison of results on formal data can be further analyzed in Table 3.1..

TABLE 3.1. Comparison of results of different studies on different formal data

Model	CoNLL2000	CoNLL2003	Spanish	Dutch
Ma & Hovy (2016)	-	91.21	-	-
Luo et al. (2015)	-	91.20	-	-
Collobert et al. (2011)	94.32	89.59	-	-
Huang et al. (2015)	94.46	90.10	-	-
Yang et al. (2017)	95.41	91.26	85.77	85.19
Lample et al. (2016)	-	90.94	85.75	81.74

von Däniken (2017) [43] which obtained the second position in WNUT'17, uses sentence-level features and exploits additional annotated data for a bidirectional LSTM-CRF network. To be more precise, the combined model consists of two different CRF layers which one of them is trained on WNUT'17 dataset whereas the other one is trained on WNUT'16. The architecture of this proposed model can be seen in Table 3.4.. The aim is to improve generalization ability by incorporating both datasets. Input to these CRF layers are shared and contains sentence-level features and word-level features obtained from a bidirectional LSTM. For sentence-level features, they use sent2vec which introduced by [44] and for word-level features, they use **(1)** word embeddings obtained by FastText on both of the datasets and a corpus of 200 million tweets, **(2)** word capitalization features **(3)** character convolution features obtained by a character-level CNN and **(4)** character capitalization convolution features. As a result, they obtained entity-level 40.78 F1 score and 39.33 F1 score for surface forms.

Enghoff et al. (2018) [45] propose a cross-lingual transfer learning model with the main focus of *annotation projection* from multiple sources to target dataset of the low-resourced language. The proposed model requires a parallel corpora which has many languages with parallel sentences and that each language has its own NE tagger. Parallel corpora is obtained

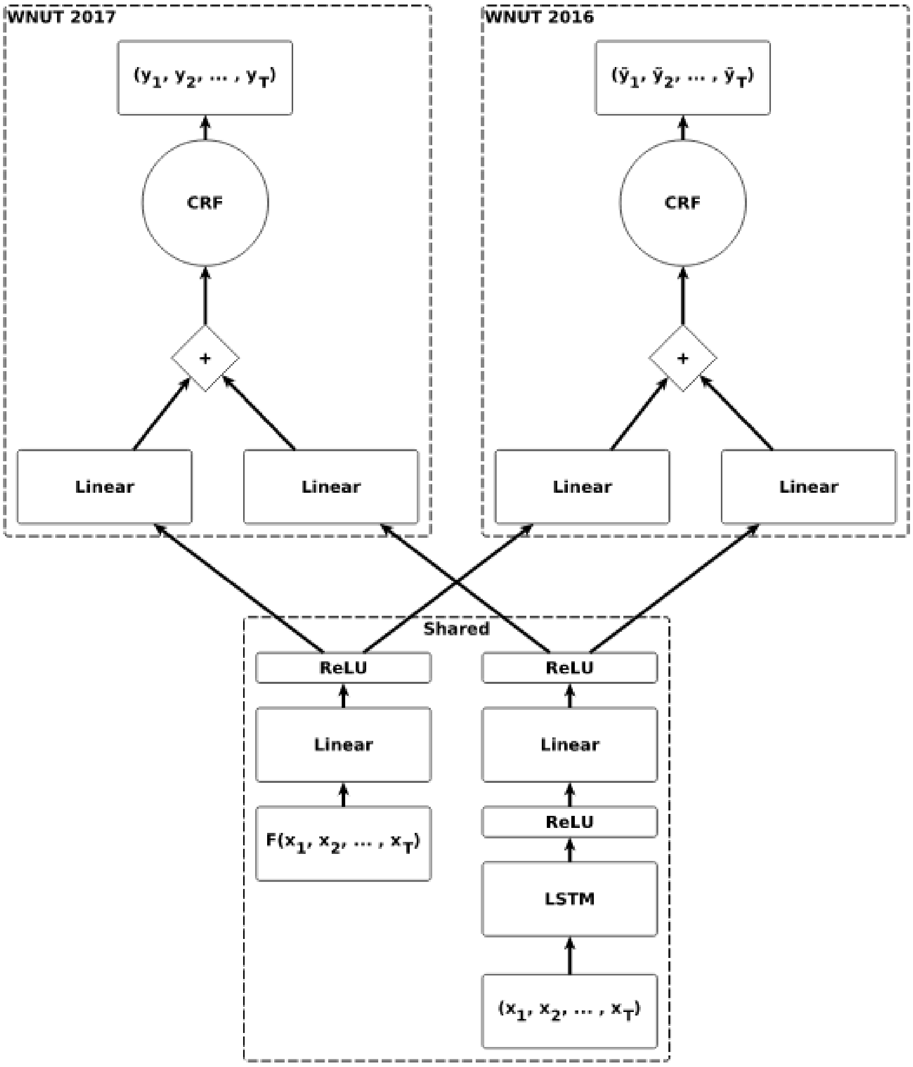


FIGURE 3.4. Architecture of Transfer Learning in a BiLSTM-CRF network [43]

by using unsupervised sentence and word aligners that assign confidences to each tokens as weights. Since each token in these source languages are assigned to NE types with different probabilities, and each aligned token has a weight (aligner confidence), predicting NE label of the target language is the weighted sum of the NE probabilities of its associated (aligned) tokens in different languages, hence the annotation projection. The NE label of the target token is then either selected by *(1)* selecting the NE label with the highest probability or *(2)* by training a NE tagger on the projection with some normalization. The experiments are conducted on two different parallel corpus; Europarl and Watchtower where the first one covers 21 languages with rich resources in quality and quantity and the latter one covers more than 300 languages with low-resource, low-quality data. As a result, they achieved a mean F1 score of 60.7% with $n = 3$ source languages and a mean F1 score of 62.23% with optimal number of source languages n_{max} on Europarl. However, they achieved a mean F1 score of 16.3% with $n = 3$ source languages and a mean F1 score of 21.12% with optimal number of source languages n_{max} on Watchtower.

4. THE PROPOSED MODEL

In this chapter, we first explain our word (and sub-word) embeddings which are used as input to train the neural architectures. We, then, detail our baseline model which is a basic *LSTM-CRF* architecture, then present our proposed, *transfer learning*, model which is an extension of the baseline model by incorporating another CRF layer in order to train the model on two datasets simultaneously with the aim of a better generalization.

4.1. Word Embeddings

We use dense vector representations of words as input to our baseline and transfer learning models. These vector representations, named *embeddings*, are the concatenation of different embedding techniques such as FastText [8], word2vec [7], morph2vec [9] and orthographic character-level embeddings. By using these techniques, we aim to capture orthographic, morphological and contextual information of words ”without using any hand-crafted features such as lexicons or gazetteers” [4]. An overview of our approach can be seen in Figure 4.1..

4.1.1. Orthographic character-level embeddings

Following the example of Aguilar et al. [1], we use an orthographic character encoder that encodes alphabetic characters with their orthographic counterparts such that;

- Every alphabetic character becomes a **”c”** or a **”C”** if the character is capitalized
- Every numeric character becomes an **”n”**
- Every punctuation character becomes a **”p”**

Additionally we have also replaced any non-ascii characters (e.g. some foreign or deteriorated characters) with an **”x”**. We did not replace any white space characters. Thus, an example sentence of;

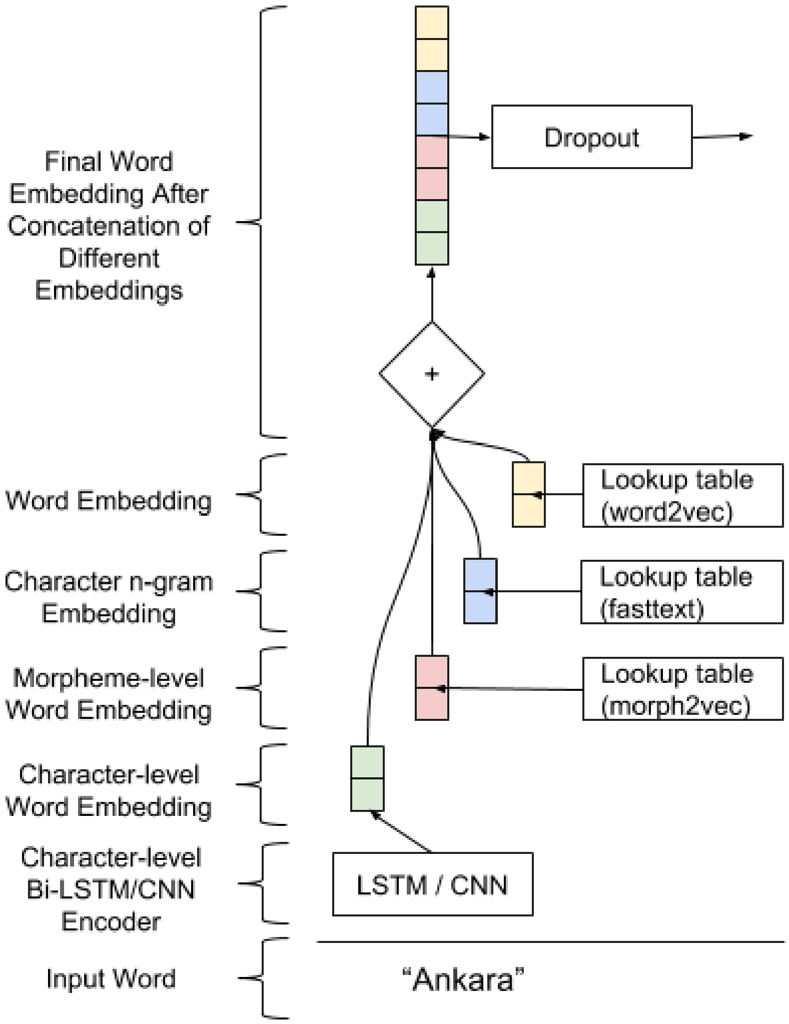

Final Word Embedding After Concatenation of Different Embeddings
Dropout
+
Word Embedding
Lookup table (word2vec)
Character n-gram Embedding
Lookup table (fasttext)
Morpheme-level Word Embedding
Lookup table (morph2vec)
Character-level Word Embedding
Character-level Bi-LSTM/CNN Encoder
LSTM / CNN
Input Word
"Ankara"

FIGURE 4.1. Overview of the final word embeddings. After concatenating embeddings obtained from *fasttext*, *word2vec*, *morph2vec* and orthographic character-level embeddings, we also apply dropout for better generalization and consequently, obtain the final word embedding for a word.

All 3 of them say "Jina did it"...

is encoded as

Ccc n cc cccc ccc pCccc ccc ccPppp

This allow us to reduce sparsity and capture shapes and patterns of words. Additionally, an embedding lookup table where each orthographic character (i.e. "c", "C", "p", "x") has an embedding vector is initialized using *uniform Glorot* initialization. This initialization method ensures that prior to training the embeddings are uniformly distributed between $[-n, +n]$ where n is the $sqrt(6/(n_{inputdimension} + n_{outputdimension}))$. After generating the orthographic character-level embeddings of words, these embeddings are fed into a bidirectional LSTM (Long-Short Term Memory) which is simply two different LSTM networks where one of them takes input sequence in reverse order. During training, both the parameters of these LSTM networks and the embeddings are updated accordingly. Output of these forward and backward LSTM networks are then concatenated. We argue that this approach ensures the model can learn better representations of orthographic (i.e. capitalization, punctuation marks, numerical characters) and character-level features (i.e. prefix, suffix) without using any hand-crafted features. We set the dimension of hidden layers of the bidirectional LSTM as 30 which results in embeddings with a dimension of 60. Figure 4.2. presents the overview of the approach.

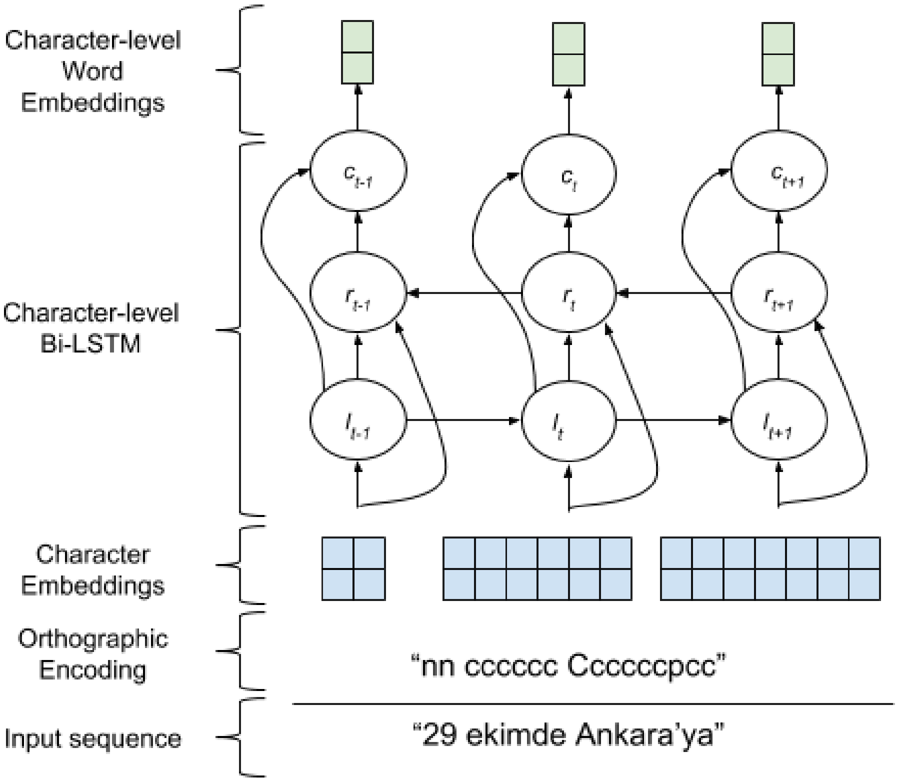


FIGURE 4.2. Character-level word embedding using a bidirectional LSTM

As an alternative approach, we also trained a character-level Convolutional Neural Network

(CNN), following the example of Aguilar et al. [1]. We used the same character embedding size similar to that of bidirectional LSTM and set maximum word length as 20. We padded shorter words with zeros and truncated the longer ones. Then, we apply two-stacked convolutional layers and perform global average pooling to the output. Finally, we use a fully-connected feed-forward layer with a Rectifier Linear Unit (ReLU) activation function. Hidden dimension of this layer is 32. An overview of this alternate approach can be seen in Figure 4.3.. Although CNNs are networks that are designed to extract position-invariant features, we argue that orthographic and character-level features are not position-dependent so that CNNs can also be effectively used as much as LSTMs to learn valuable character-level features.

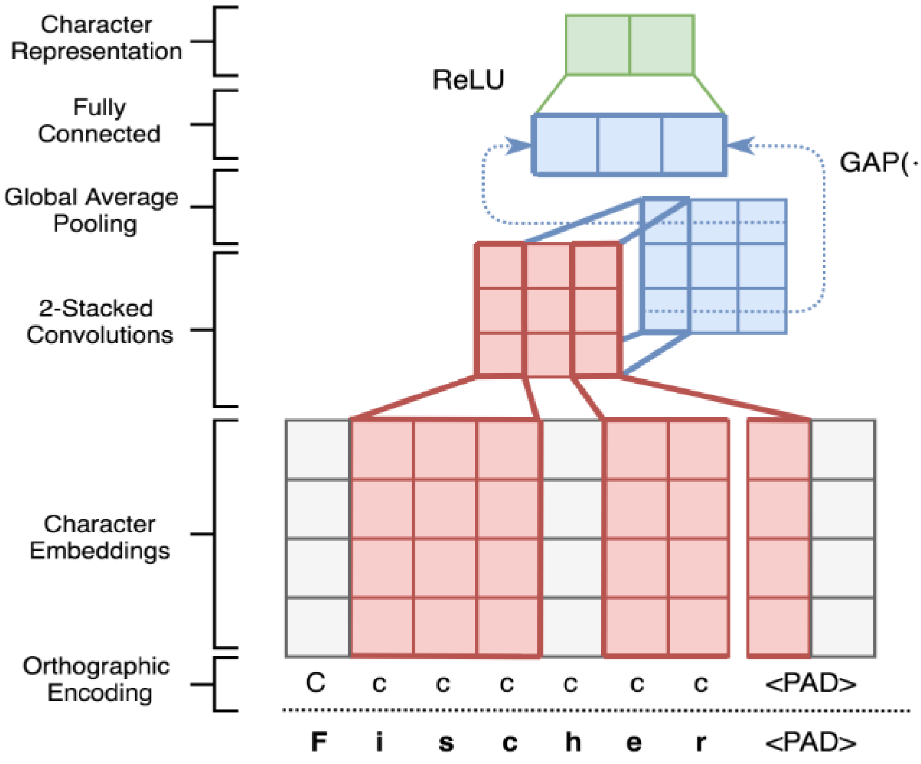


FIGURE 4.3. Character-level word embedding using CNN. Source: Aguilar et al. [1]

4.1.2. Word2vec

Proposed by Mikolov et al. [7], word2vec is an efficient word representation algorithm where the syntactic and semantic relations between words can be effectively captured. In order to represent the words in dense word embeddings with this syntactic and semantic knowledge, we use pre-trained embeddings trained on a corpus of 400M English tweets [46]. Similarly, we use pre-trained embeddings that is trained on a news corpus (BOUN web corpus) of 423M words [27], [28] and 20M Turkish tweets [29]. Embedding dimension of words is 400 for both English and Turkish.

4.1.3. FastText

As we have already discussed in Chapter 2, FastText [8] is another word representation technique that is similar to word2vec but comparably better at capturing word representation ”for morphologically-rich languages such as Turkish” [8]. This is due to its ability to form vectorial representation of words from their vectors of character n-grams. As a result, this allows us to generate word embeddings using n-grams even for out-of-vocabulary words which is a common case for noisy data and agglutinative languages, namely sparse data.

In order to obtain Turkish word embeddings, we trained Skipgram model of FastText on a corpus of 20M Turkish tweets[1]. During training, we used the default settings except for an embedding dimension of 200, learning rate of 0.025 and trained for 4 epochs. A complete list of the training settings can be seen in Table 4.1. On the other hand, we used pre-trained English word embeddings that is provided by FastText[2]. These word embeddings are obtained from a CBOW model of FastText that is trained on Common Crawl[3], a web site that provides web crawl data. Embedding dimension for English is 300.

[1] http://www.kemik.yildiz.edu.tr/data/File/20milyontweet.rar

[2] https://s3-us-west-1.amazonaws.com/fasttext-vectors/crawl-300d-2M-subword.zip

[3] https://commoncrawl.org/

TABLE 4.1. FastText training settings for Turkish word embeddings

Argument	Value
learning rate	0.025
lr update rate	100
dimension	200
size of the context window	5
number of epochs	4
nnumber of negatives sampled	5
loss function	ns
number of threads	12

Using these pre-trained models, prior to training, we compute word embeddings for all out-of-vocabulary (OOV) words which are generated by using their character n-grams. Then we initialize an embedding lookup table where each word has a corresponding word embedding.

4.1.4. Morph2vec

Morph2vec [9] is another representation learning algorithm that utilizes sub-word information to form the word embeddings. The algorithm takes a list of candidate morphological segmentations of all words in the training data and computes the word embeddings from their related morpheme embeddings. Given that each word has multiple sequences of candidate morphological segmentations, an attention mechanism is used on top of the model in order to give more weight to the correct sequence of segmentations. Finally, vectorial representation of words from a pre-trained word2vec model is used in order to allow the model compute distance between the learned and pre-trained embeddings.

We also incorporate morpheme embeddings that we obtain from a pre-trained morph2vec model. Embedding dimensions are for English and Turkish are 75 and 50 respectively. We argue that using morphemes makes it easier to distinguish words with inflectional endings that mostly made up of verbs, so that these words can contribute to the model as negative samples.

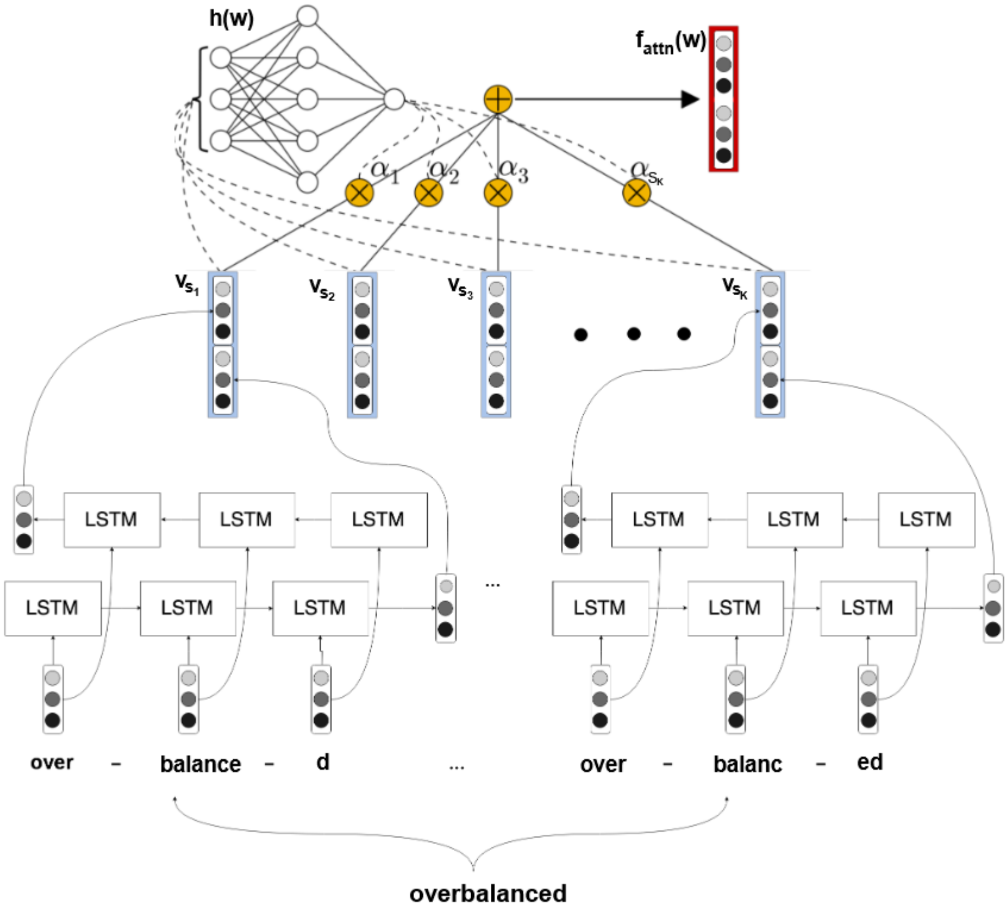


FIGURE 4.4. Architecture of morph2vec with an attention mechanism on top of bi-LSTMs. [9]

Since we could not find any morpheme-level embeddings for more than 20% words in Turkish dataset, *DS-1* due to its noisy nature, we also tried to extend the morph2vec model by additionally learning character-level embeddings so that if a word does not have a morpheme-level embedding, we can build an embedding using its characters. Results of this alternate approach can also be seen in Chapter 5, although it failed to improve the results any further.

4.1.5. Dropout

Finally, the resulting word embeddings that we used as input to the baseline and transfer learning models are the concatenation of FastText, morph2vec, word2vec and orthographic character-level embeddings.

After concatenating the aforementioned word embeddings, we also apply *dropout* on this final embedding vector which randomly sets a ratio r of the embeddings as zero. In addition, outside of these zeroed ratio are also scaled by a factor of $1/(1-r)$, so that the sum of the embedding remains unchanged. Dropout operation, thus, prevents the model to solely depend on one type of word embeddings and, consequently, ensures better generalization. We use dropout rate $r = 0.5$ during the experiments.

4.2. LSTM-CRF Model

Our baseline model is a basic *LSTM-CRF* model that is similar to that of Lample et al. [4], Collobert et al [33], Huang et al. [32], Chiu et al. [34], Ma et al. [35]. In this model, we use a word-level LSTM to learn higher-order features and then use them to feed a linear-chain CRF to output a prediction of label sequence. Overview of the baseline model can be seen in Figure 4.5.. The model consists of LSTM and CRF components.

4.2.1. LSTM Component

Given an input sentence (i.e. tweet) $W = (w_1, w_2, ..., w_N)$ where the length is N, we first compute the vectorial representation x_n of each word w_n. This word representation is the concatenation of different word embeddings techniques that we describe previously. The sequence of these word representations $(x_1, x_2, ..., x_N)$ for a word w_n is then used as input to bidirectional LSTM layer so that forward and backward LSTM layers can capture the context of words on both sides. A bidirectional LSTM (Bi-LSTM) comprises two separate LSTM networks. While one of them is fed with vector representation of words in order, the other one is fed with vector representation of words in *reverse order*. Thus, during training, they learn forward and backward word representations independently and this allow us to capture information semantically better. An architecture of this network can be seen in Figure 4.6..

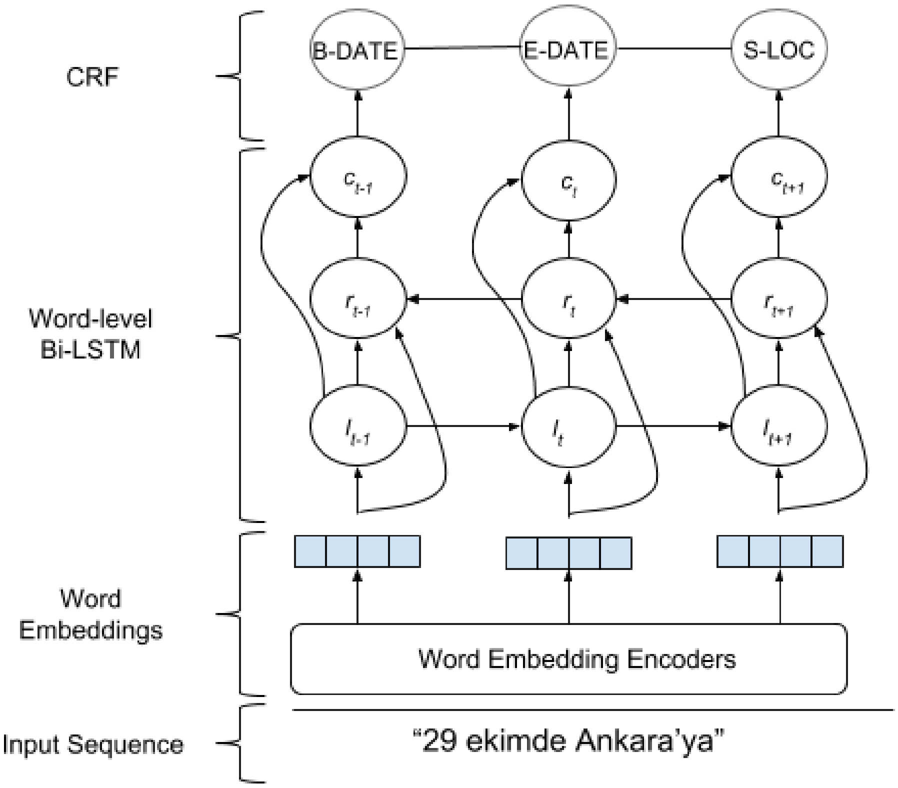


FIGURE 4.5. Our baseline LSTM-CRF model that we learn higher-order word representations by using a Bi-LSTM and then fed the concatenated output to CRF in order to predict the label sequence. Here, word embedding encoders are namely *fasttext*, *word2vec*, *morph2vec* and orthographic character-level embedding techniques that we have presented earlier.

We use the following LSTM implementation [14]:

$$f_t = \sigma(W_f.[h_{t_1}, x_t] + b_f) \tag{105}$$

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{106}$$

$$\widetilde{c_t} = tanh(W_{\widetilde{c_t}}.[h_{t-1}, x_t] + b_{\widetilde{c_t}}) \tag{107}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c_t} \tag{108}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{109}$$

$$h_t = tanh(c_t) \circ o_t, \tag{110}$$

where $\circ$ is the element-wise product and σ is the element-wise sigmoid function which squashes the values between zero and one, thus, deciding how much of the old information it wants to keep. As we can see in the equations, input i, forget f and output o gates are almost the same sigmoid functions except for their different weights and biases. Forget gate f_t determines the proportion of the previous information c_{t-1} to be kept for new cell state c_t. Input gate i_t and candidate values $\widetilde{c_t}$ decides how much of the information will be updated for the new cell state c_t. Following this operation, output gate o_t determines how much of the cell state c_t will be outputted after the $tanh$ operation on the cell state which squashes the values between -1 and 1. The important thing to notice here is the gating mechanisms that allow to forget/keep some of the old hidden states in order to prevent vanishing gradients during matrix multiplication. Finally, we obtain the output of forward and backward LSTM layers; $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ respectively which are concatenated to form the final word representation $h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$.

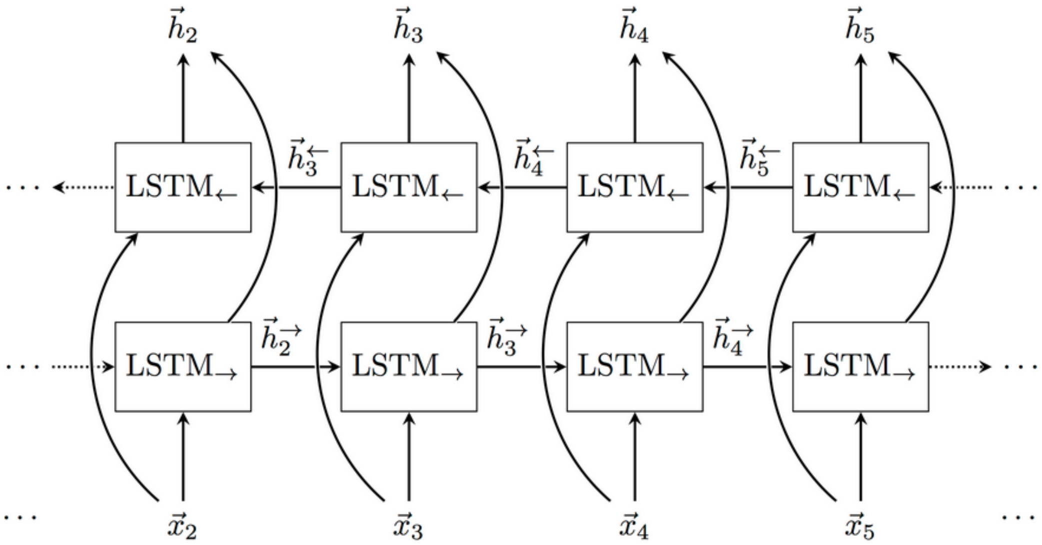


FIGURE 4.6. Architecture of a bidirectional LSTM network. Source: University of Cambridge Research Students Lecture Series. Source: https://www.cl.cam.ac.uk

4.2.2. CRF Component

We apply dropout operation on the output h_t of the bidirectional LSTM for better generalization and then use a linear-chain CRF to jointly predict sequence of labels. Given an input sequence of $X = (x_1, x_2, ..., x_N)$, CRF computes an output sequence of predictions $Y = (y_1, y_2, ..., y_N)$. The prediction score of a sequence can be defined as:

$$S(X, y) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} P_{i, y_i}, \tag{111}$$

where P is the matrix that we obtained from the bidirectional LSTM and A is the transition matrix that is a computation of transition from a previous state (i.e. label) to the next state. P matrix has the size of $N.k$ where k is number of distinct tags. Thus, P_{i,y_i} is the score of y_i given a word w_i. As a log-linear model, the probability of the output sequence of y then becomes:

$$p(y|X) = \frac{\epsilon^{s(X,y)}}{\sum_{\widetilde{y} \in Y_x} \epsilon^{s(X,y)}}, \tag{112}$$

where Y_x is the all possible label sequences. Finally, the goal becomes to maximize the log-probability of the correct prediction sequence. Building the log-linear model gives us the form:

$$log(p(y|X)) = s(X, y) - log(\sum_{\widetilde{y} \in Y_x} \epsilon^{s(X, \widetilde{y})}) \tag{113}$$

The correctly-predicted sequence of labels is the one that maximizes the equation above such that:

$$\arg\max_{\widetilde{y} \in Y_x} s(X, \widetilde{y}). \tag{114}$$

As we have already discussed in Chapter 2. both the parameter estimation and decoding of this equation can be solved by using dynamic programming. Weights of bidirectional LSTM and CRF layers are initialized using *uniform Glorot initialization.*

4.2.2.1. Tagging Scheme

There are different tagging schemes that can be used for the CRF output. When it is thought that a named entity may span multiple consecutive words, a tagging scheme that impose some constraints on determining the possible label of a word is highly useful. IOB format is such tagging scheme which uses *B* prefix for a word at the beginning of an named entity, *I* prefix for a word that inside a named entity, and *O* prefix for others.

IOBES is a variant of IOB format that further restricts the possible label of a word with the help of additional prefixes such as *E* prefix that is used for specifying the ending of a named entity and *S* prefix that is used for named entities with only one word. Here is an example sentence tagged with the IOBES format:

*Mustafa/**B-PERSON** Kemal/**I-PERSON** Ataturk/**E-PERSON** was born in 1881/**S-DATE** in the former Ottoman/**B-ORGANIZATION** Empire/**E-ORGANIZATION**.*

4.3. Transfer Learning Model

As the results presented in Chapter 5 also indicate, the baseline model fails to assign labels for some of the named entity types if the types are rarely seen in the training data. In order to learn these rare types better, we have incorporated another CRF layer that we trained on a different, preferably larger dataset named *source dataset*. The key idea is that the model can learn from both datasets while optimized only for the *target dataset*, so that it can also predict rare entity types seen in the target dataset. In order to achieve this goal, we have adapted the cross-domain transfer learning model proposed by Yang et al. [17] as an extension to our baseline model. Figure 4.7. presents the overview of this model.

Furthermore, following the example of von Däniken et al. [43], we also add Rectified Linear Unit (ReLU) layers and linear layers between the bidirectional LSTM and CRF layers which increased the overall performance of the model considerably. A linear layer is simply a fully-connected feed-forward network (FFNN) with one hidden layer. Similarly, A ReLU

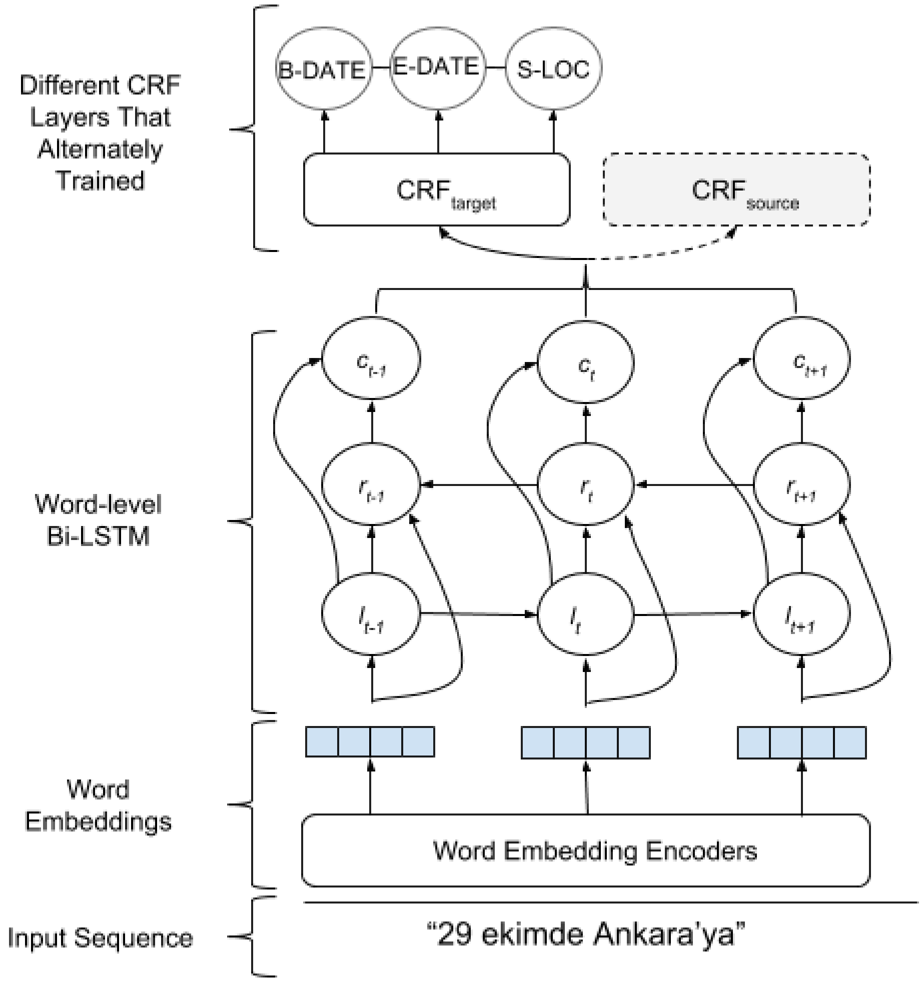


FIGURE 4.7. Overview of the initial transfer learning model which incorporates additional CRF layer. CRF layers are alternately trained on different datasets so that the shared layers learn from both datasets and, therefore, learning can be transferred from source dataset to target dataset.

layer is a fully-connected FFNN with ReLU activation function. As we can see in Figure 4.8., all these layers and their parameters up to the last linear layer are shared by both of the CRF layers. Weights of this last linear layer are initialized using uniform Glorot initializer, whereas, biases of the layer are initialized as zeros.

We alternately trained on one of the two datasets for each epoch. Unlike the model proposed by [43], we have used the same word embeddings as our base model and, in this regard, we include the concatenation of word embeddings obtained from fasttext, morph2vec, word2vec and orthographic character-level embeddings in our architecture.

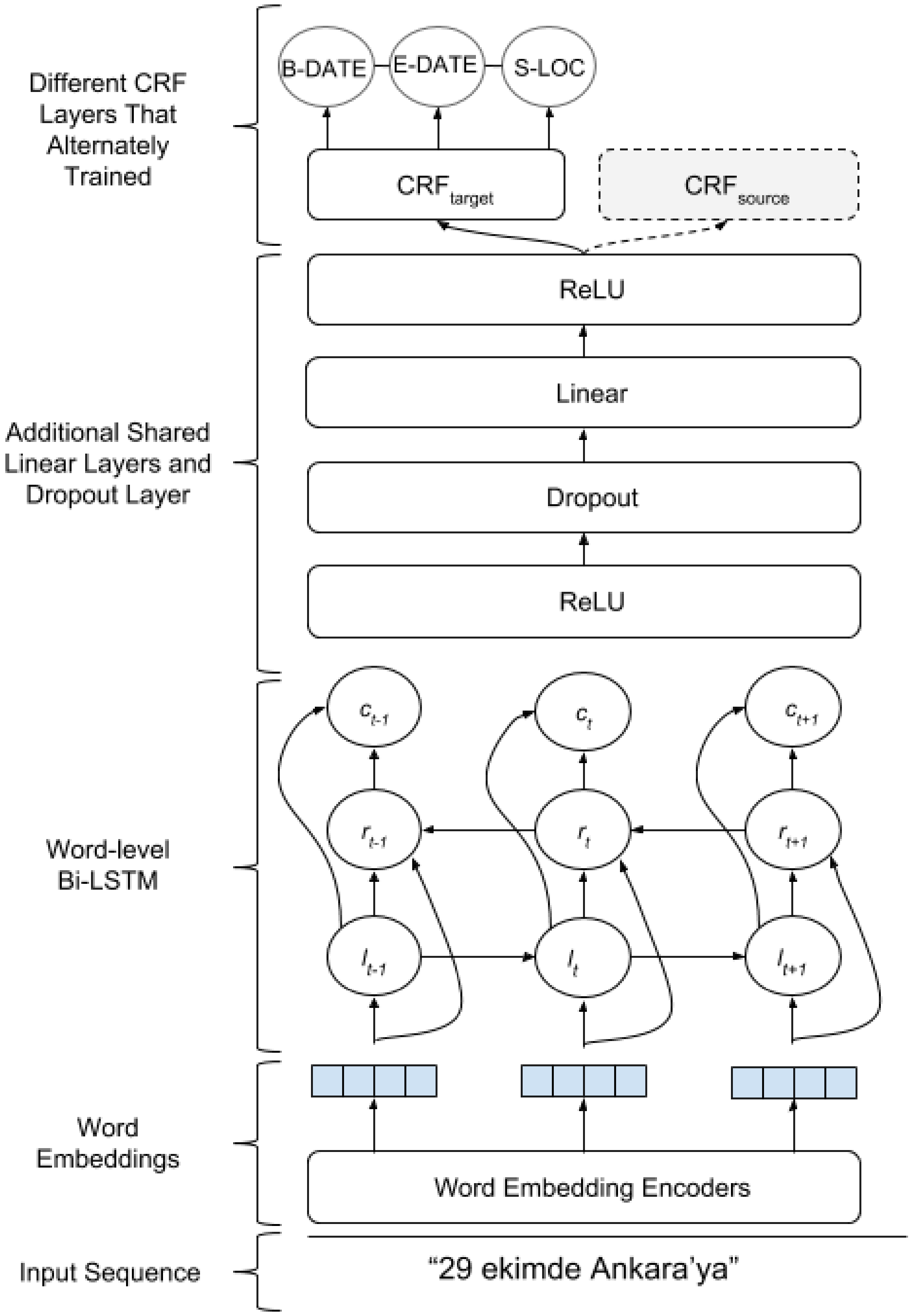


FIGURE 4.8. Overview of the final transfer learning model which incorporates additional CRF layer and additional linear and ReLU layer to further improve model performance.

4.4. Implementation Details

The models presented in the thesis are implemented using Python3.5 and Tensorflow 1.8.0. All source code and related material can also be accessed openly on https://github.com/emrekgn/turkish-ner

5. EXPERIMENTS & RESULTS

In this chapter, we first introduce the Turkish and English datasets which are used throughout the experiments. Then, we present the results of our experiments on Turkish and English datasets while mainly focusing on the results of our baseline and transfer learning models. Additionally, we present the contribution of each embedding type to our models and give a brief overview of all experiments conducted in order to better understand how the changing parts of the models affect the results.

5.1. Datasets

We experimented on three different datasets in English and Turkish that can be seen in Table 5.1.

TABLE 5.1. Datasets that are used throughout the experiments

#	Dataset	NE types	Noisy	# of tokens	# of NEs
DS-1	TR-tweet	ENAMEX, TIMEX, NUMEX	Yes	55K	1.4K
DS-2	CoNLL'03	ENAMEX	No	302K	35K
DS-3	WNUT'17	corporation, creative-work, group, location, person, product	Yes	104K	3.8K

The first one, **DS-1** [6] is the re-annotated version of the initial Turkish tweet (noisy) dataset [20] that consists of ENAMEX, TIMEX and NUMEX types. ”ENAMEX types include *organization, location and person* names, TIMEX contains *date and time* and NUMEX contains *money and percentage*” [3]. It is also stated in [47] that re-annotation according to the sixth of Message Understanding Conferences, MUC-6, guidelines [3] also provides better consistency and quality. We can see the NE type distribution of the dataset in Table 5.2.. Since the dataset does not have training and test splits, we split the dataset into training, test and validation sets with a ratio of 80%, 10%, 10% respectively and applied 10-fold cross validation.

As we have earlier stated that our bidirectional LSTM-CRF model is based on the work of Lample et al. [4] and the second dataset, **DS-2**, is the English CoNLL'03 dataset [48] used

TABLE 5.2. Number of different entity types in Turkish noisy dataset, *DS-1*

Entity Type	Amount
person	699
location	230
organization	363
date	56
time	20
money	12
percentage	3
Total	**1383**

by Lample et al. [4] to evaluate their model. The dataset consists of person, organization, location and miscellaneous types. Miscellaneous type is used to indicate entities that do not belong to other three types. Although the dataset contains both NE tags and POS tags, we did not use POS tags for our model. The dataset has training, test and development sets with a size of 203K, 51K, and 46K tokens.

TABLE 5.3. Number of different entity types in English formal dataset, *DS-2*

Entity Type	Train	Development	Test	Total
person	6600	1842	1617	10059
location	7140	1837	1668	10645
organization	6321	1341	1661	9323
miscellaneous	3438	922	702	5062
Total	**23499**	**5942**	**5648**	**35089**

The last one, **DS-3** [49], is the English dataset prepared for **Workshop on Noisy User-generated Text (WNUT'17) emerging and rare entity recognition task** [50] which includes: *person, location, corporation, product (consumer goods, service), creative work (song, movie, tv series, book), group (music band, sports team, non-corporate organizations)*. According to [49], the text was collected from Youtube comments, StackExchange comments/posts, tweets and Reddit comments with the goal ”to provide high-variance data, with very few repeated surface forms” [50]. Similar to the dataset for WNUT'15 task [51], this dataset also contains Twitter NER dataset of [36] as its training data. While development data contains Youtube comments, test data consists of comments/posts from other sources stated earlier. Distribution of NE types can be seen below in Table 5.4. The dataset has training, test and development sets with a size of 65K, 23K, and 16K tokens.

TABLE 5.4. Number of different entity types in English noisy dataset, *DS-3*

Entity Type	Train	Development	Test	Total
person	660	470	429	1559
location	548	74	150	772
corporation	221	34	66	321
product	142	114	127	383
creative-work	140	104	142	386
group	264	39	165	468
Total	**1975**	**835**	**1079**	**3889**

5.2. Experiments

We trained and evaluated models for both **English** and **Turkish** separately. We used the provided data (training, development and test) splits for the English datasets. On the other hand, due to being a relatively small dataset and having imbalanced distribution of NE types, we used **10-fold cross validation** for all of our experiments that we conducted on Turkish dataset. Turkish dataset, *DS-1*, is split into training, test and development data with a ratio of 80%, 10%, 10% respectively.

As we can see in the results, two different models are used throughout the experiments, namely:

- **Baseline Model** which is a bidirectional LSTM-CRF model that is similar to that of model presented by Lample et al. [4].
- **Transfer Learning Model** which is an extension of the baseline model that is based on the models presented by [17] and [43] and trained on two different datasets simultaneously.

5.2.1. Preprocessing

Prior to tokenization of all of the datasets;

- We replaced URLs (tokens starting with *http*) with a special token. This allows us to reduce sparsity and our model to converge relatively faster.
- We also replaced Twitter mentions (Twitter usernames starting with @ sign) with another special token for DS-1. As a result, this greatly reduced the number of *PERSON* entities from 4256 to 699.

5.2.2. Experimental Setting & Training

During all experiments, both of the models are trained using backpropagation algorithm and their related parameters are optimized using Stochastic Gradient Descent. We trained for 100 epochs with a learning rate of 0.005 in addition to using gradient clipping of 5.0. Dropout rate for all our dropout layers are set to 0.5. Hidden dimension of character-level bidirectional LSTM and word-level bidirectional LSTM layers are set to 30 and 250 respectively. Tuning these dimensions or any other hyperparameter did not significantly improve the performance of the models. An overview of these hyperparameters can be seen in Table 5.5.

We use an embedding dimension of 400 for English and Turkish word embeddings that are obtained from pre-trained word2vec models. English word embeddings are trained on a corpus of 400M English tweets [46], whereas Turkish word embeddings are trained on BOUN web corpus of 423M words and 20M Turkish tweets. For morpheme-level embeddings, we use pre-trained morph2vec models for English and Turkish with dimensions of 75 and 50 respectively. In order to obtain character n-gram level embeddings, we trained fasttext on a corpus of 20M Turkish tweets[1] by Bolat and Amasyalı. On the other hand, we used pre-trained English character n-gram level embeddings that is provided by FastText[2] itself. These

[1] http://www.kemik.yildiz.edu.tr/data/File/20milyontweet.rar

[2] https://s3-us-west-1.amazonaws.com/fasttext-vectors/crawl-300d-2M-subword.zip

embeddings are obtained from a model that is trained on web crawl data with a dimension of 300.

TABLE 5.5. Hyperparameters

Hyperparameter	Value
gradient clip	5.0
learning rate	0.005
lr optimizer	sgd
batch size	10
dropout	0.5
epochs	100
hidden size$_{\text{BiLSTM(char)}}$	30
hidden size$_{\text{BiLSTM(word)}}$	250
dimension$_{\text{fasttext(en)}}$	300
dimension$_{\text{morph2vec(en)}}$	75
dimension$_{\text{fasttext(tr)}}$	200
dimension$_{\text{morph2vec(tr)}}$	50
dimension$_{\text{word2vec}}$	400
dimension$_{\text{char}}$	30

5.2.3. Evaluation

During evaluation of the experiments on *DS-1* and *DS-2*, we used Python re-implementation of the CoNLL'00 evaluation script [48] for NER[3] that is the *de facto* choice in the literature. Different measures are provided by CoNLL'00 evaluation script such as *accuracy*, *precision*, *recall* and *F1 score*. Here, accuracy measures the overall performance of the model by computing the ratio of correctly labeled labels to the total number of tokens. But this results in a highly imbalanced value since most of the tokens are not part of a named entity and, therefore, labeled as '0'. Precision gives the ratio of correctly labeled named entities (chunks) to the total label predictions. Similarly, recall measures the ratio of correctly labeled named entities (chunks) to the total correct predictions. Finally, F1 score is computed by taking "harmonic mean of precision and recall $(2 * precision * recall/(precision + recall))$" [48]. In order to measure the overall performance of any given model for a task of sequence labeling, F1 score is commonly chosen over accuracy since it intuitively defines a good measure

[3] https://github.com/spyysalo/conlleval.py

of the model by taking false negatives and false positives into account and accuracy gives imbalanced results due to highly-skewed entity type distribution (i.e. most of the tokens does not have any entity label).

However, for the experiments on *DS-3*, we used WNUT'17 evaluation script[4] that is also a re-implementation of CoNLL'00 evaluation script with stricter rules for ill-formed entities. This is done by measuring both entity-level and surface forms measures (precision, recall and F1 score). Entity-level measures are similar to the CoNLL'00 script but surface form measures are measured by using the set of unique surface forms so that a correctly-labeled entity is "*only counted once* no matter how many times it appears in the dataset" [49]. This way a model will not be rewarded if it can only learn the frequent entities and, thus, it can be evaluated on how good it is to detect and correctly label infrequent entities.

5.2.4. Experimental Results on Turkish

We have already discussed that named entity recognition for Turkish is still a challenging problem due to being a morphologically-rich language and having scarce annotated data.

Experiments for Turkish have been conducted on *DS-1* which is a relatively small dataset with highly imbalanced NE type distribution as we can see in Table 5.2. An overview of the experimental results can be seen in Table 5.8. Embeddings column of the table represents the word embeddings obtained from different word (and sub-word) level representation learning algorithms that we have discussed earlier. In this regard fasttext, word2vec, morph2vec, character-level and orthographic character-level embeddings are denoted in the table by *ft*, *w2v*, *m2v*, *char*, *ortho* respectively. For brevity, only the notable experiments are shown and detailed here. Complete list of the experiment results on Turkish noisy dataset, *DS-1* can be seen in **Appendix A**.

[4] https://noisy-text.github.io/2017/emerging-rare-entities.html

5.2.4.1. Baseline Model

Our baseline model is the bidirectional LSTM-CRF architecture that incorporates different word embeddings obtained from fasttext and orthographic character-level embeddings that are trained on another bidirectional LSTM layer. Alternately, we have also experimented with an orthographic character-level convolutional neural network (CNN) in order to obtain orthographic embeddings but the results are comparably worse than the results of this variant. The results for the CNN experiments can also be analyzed in Appendix A.

TABLE 5.6. Experiment results of the baseline model with fasttext and orthographic character-level embeddings on Turkish noisy dataset, *DS-1*

Entity Type	Precision (%)	Recall (%)	F1 score (%)
person	69.74	52.95	60.04
organization	82.87	59.38	68.54
location	64.47	45.76	52.75
date	46.66	19.91	26.26
time	11.11	11.11	11.11
money	0	0	0
percentage	0	0	0
overall	**71.96**	**51.1**	**59.7**

We have achieved an F1 score of 59.7% with the baseline model although the model failed to label any entities with *MONEY* and *PERCENTAGE* types. We believe this is mainly because of the low number of these entity types. There are only 12 examples of *MONEY* and 3 examples of *PERCENTAGE*. Furthermore, the model did not even encounter examples of some of these entity types in most of the iterations of the cross-validation during training. As a result, the model expectedly did not manage to learn valuable features for these types. Instead of 10-fold cross validation, choosing different number of splits (folds) also did not improve the results.

5.2.4.2. Transfer Learning Model

Our proposed model which we apply transfer learning, is an extension of the baseline model by incorporating another CRF layer that we train on a different preferably larger dataset, named *source dataset*. It is important to emphasize again that our aim is to train the model on two different datasets so that it can learn from both of them, but the model is evaluated only for the *target dataset*. As source dataset, we used the re-annotated version of the Turkish news corpus with 492K tokens originally provided by Tür, Gökhan et al. [19] and re-annotated by Şeker, Gökhan Akın, and Gülşen Eryiğit [6]. In this transfer learning model, we used pre-trained embeddings of fasttext and morph2vec. Additionally, we also used orthographic character-level embeddings that we trained on a character-level CNN. As we can see in the results of other variants in Appendix A, character-level CNN performed better than the character-level LSTM for the transfer learning model on DS-1.

TABLE 5.7. Experiment results of the transfer learning model with fasttext, morph2vec and ortographic character-level embeddings on Turkish noisy dataset, *DS-1*

Entity Type	Precision (%)	Recall (%)	F1 score (%)
person	72.69	57.9	64.34
organization	80.56	67.52	73.08
location	75.38	60.21	66.24
date	50.83	29.59	36.57
time	30	18.66	21.33
money	20	13.33	15
percentage	0	0	0
overall	**74.45**	**58.94**	**65.72**

As we can see in Table 5.7., transfer learning approach improves upon the results of the baseline model with an F1 score 65.72%. Although, the overall result and the results on rare entity types (such as *DATE*, *TIME*, *MONEY*) are higher compared to the baseline model, the model still failed to label the entity type *PERCENTAGE* but we believe this is an expected outcome given that it has only 3 examples in the whole dataset.

5.2.4.3. Discussion

As we can see in the results, transfer learning model that incorporates word embeddings obtained from fasttext, morph2vec and orthographic character-level embeddings achieves the best results out of all the experiments with an F1 score of 65.72%. Although it fails to

TABLE 5.8. Overview of the results on Turkish noisy dataset, *DS-1*. *Transfer learning-1* is the initial transfer learning model that is simply an extension of the baseline model by adding additional CRF. *Transfer learning-2* represents the final transfer learning model with the additional ReLU and linear layers.

Model	Embeddings	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
baseline	m2v, char	96.70	47.78	14.29	21.88
baseline	m2v, ortho	96.72	60.6	14.11	22.81
baseline	w2v, char	96.77	53.28	19.57	28.45
baseline	w2v, ortho	96.69	60.64	16.57	25.8
baseline	ft, char	97.7	74.25	48.69	58.69
baseline	ft, ortho	97.73	71.96	51.1	59.7
baseline	ft, char, ortho	97.71	74.79	48.1	58.42
baseline	ft, m2v	97.67	71.36	50.02	58.73
baseline	ft, m2v, char	97.72	74.48	49.11	59.03
baseline	ft, m2v, ortho	97.68	73.12	48.7	58.32
baseline	ft, m2v, ortho (cnn)	97.69	74	49.46	59.07
transfer learning-1	ft, ortho	97.82	74.88	51.57	60.89
transfer learning-2	ft, m2v, ortho	97.87	70.78	60.35	65.12
transfer learning-2	ft, m2v, ortho (cnn)	97.95	74.45	58.94	65.72
transfer learning-2	ft, ortho (cnn)	97.89	69.89	60.56	64.73
transfer learning-2	ft, ortho	97.88	68.09	63.04	65.37

predict infrequent named entities that occur less than 20 times, we believe this is a reasonable outcome since the dataset, *DS-1* is arguably small and highly imbalanced.

As we have reviewed in Chapter 3, state-of-the-art research [6] for Turkish noisy dataset, *DS-1* achieved an F1 score of 67.96% which is a CRF-based model incorporating hand-crafted features and domain-specific knowledge (gazetteers). These include morphological features (such as stem, POS tags, noun case, proper noun, inflectional features), lexical features (case feature, is-start-of-the-sentence), gazetteers lookup features, TIMEX and NUMEX related features (is-numeric-value, has-percentage-sign, is-oclock-term, has-column-indicator, month gazetteer, currency gazetteer). In oder to adapt the model for user-generated content (noisy data), they also used: auto capitalization gazetteer (a list of names with little chance of being used as common nouns) as feature. Hence, they aim to differentiate proper names from common nouns. Additionally, they used Twitter mentions (Twitter usernames starting with a handle character, @) as feature. In this regard, using hand-crafted features heavily relies on the domain at hand. For example, the existence of a handle character as an indication to label tokens as *PERSON* is only applicable for Twitter domain and without the use of this feature, they obtained an F1 score of 63.63%. Moreover, they only obtained 47.15% F1 score without using capitalization feature.

Similarly, other related work on this matter also uses CRF-based models with hand-crafted features and gazetteers. Çelikkaya et al. [20] use morphological and lexical features such as stem, POS tag, noun case, lower/upper case are used along with gazetteers. Eken, Beyza and Tantug, Cüneyd [25] use the existing of apostrophe character, case of the word, start of sentence, gazetteers for NE types of PLOs (person, location, organization names) with optional Levenshtein distance-based matching. Different from the others, Küçük, Dilek and Steinberger, Ralf [23] use a rule-based NER system that incorporates list of person, location, organization (PLOs) names and patterns for NE types time, date, money, percent and PLOs, they also apply normalization scheme before NER phase. We present a comparison of our baseline and transfer learning models with these related work on Turkish noisy dataset, *DS-1* in Table 5.9.. Note that these related work uses different (re-annotated) versions of the same dataset, so that named entity distributions of them may differ slightly. Şeker and Eryiğit [6] present the latest version (*v4*) of the dataset, which is also used in our experiments.

TABLE 5.9. Comparison of our models with the related work on Turkish noisy dataset, DS-1

Related Work	F1 score (%)	Dataset
Şeker and Eryiğit [6]	63.63	DS-1 v4
Çelikkaya et al. [20]	19.28	DS-1 v1
Küçük and Steinberger [23]	46.93	DS-1 v2
Eken and Tantug [25]	28.53	DS-1 v3
baseline model (ft, ortho)	**59.7**	**DS-1 v4**
transfer learning model-1 (ft, ortho)	**65.12**	**DS-1 v4**
transfer learning model-2 (ft, m2v, ortho-cnn)	**65.72**	**DS-1 v4**

5.2.5. Experimental Results on English

Experiments for English have been conducted on *DS-2* and *DS-3* datasets. We experimented on *DS-2* for the sole purpose of proving that our baseline model is a similar implementation of the model proposed by Lample et al. [4]. The complete list of the experiment results on English datasets *DS-2* and *DS-3* can be further analyzed in **Appendix B**.

5.2.5.1. Baseline Model

Our baseline, bidirectional LSTM-CRF, model that is trained on *DS-2* dataset with pre-trained embeddings from word2vec and character-level embeddings obtains an F1 score of **89.95%** on English news dataset, *DS-2*. This proves that our model is competitive to that of the BiLSTM-CRF model presented in Lample et al. [4]. They report an F1 score of 90.94% on the same dataset. Results of our experiment is detailed on Table 5.11..

Furthermore the baseline model that is trained on *DS-3* dataset with embeddings from fast-text, word2vec, morph2vec and orthographic character-level embeddings obtains an F1 score of **39.84%** on entity-level and **37.74%** on surface form. Additionally, we also incorporated pre-trained embeddings from word2vec, following the example of Aguilar et al. [1]. Since English is a comparably morphologically-poor language, we observed that using word-level

TABLE 5.10. Overview of the results on English noisy dataset, *DS-3*. *Transfer learning-1* is the initial transfer learning model that is simply an extension of the baseline model by adding additional CRF. *Transfer learning-2* represents the final transfer learning model with the additional ReLU and linear layers.

Model	Embeddings	Entity Level (%)				Surface Form (%)			
		Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
baseline	m2v, char	92.51	26.67	5.57	9.21	92.51	25.7	5.77	9.42
baseline	m2v, ortho	92.56	24.2	4.92	8.17	92.56	23.7	5.24	8.58
baseline	w2v, char	94	64.9	26.07	37.19	94	64.57	28.21	39.27
baseline	w2v, ortho	94.11	66.43	26.44	37.82	94.11	66.29	29.21	40.55
baseline	ft, char	93.14	52.87	7.7	13.44	93.14	52.38	8.07	13.99
baseline	ft, ortho	93.31	47.54	15.21	23.05	93.31	45.02	14.68	22.13
baseline	ft, char, ortho	93.32	48.24	14.01	21.71	93.32	46.69	14.05	21.6
baseline	ft, m2v	93.07	54.41	6.86	12.19	93.07	52.85	6.81	12.07
baseline	ft, m2v, char	93.1	46.67	7.79	13.35	93.1	46.11	8.07	13.74
baseline	ft, m2v, ortho	93.39	48.39	15.31	23.26	93.39	45.45	14.68	22.19
baseline	ft, m2v, ortho, w2v	94.19	66.81	28.39	39.84	94.19	66.76	26.31	37.74
baseline	ft, m2v, ortho (cnn), w2v	94.14	66.23	28.39	39.74	94.14	65.87	26.1	37.39
transfer learning-1	ft, ortho	91.4	24.47	21.24	22.74	91.4	26.38	20.02	22.77
transfer learning-2	ft, m2v, ortho	93.63	48.81	22.91	31.19	93.63	46.98	21.17	29.19
transfer learning-2	ft, ortho, w2v	94.22	55.67	33.67	41.97	94.22	54.45	31.45	39.87
transfer learning-2	ft, m2v, ortho, w2v	94.17	57.01	33.21	41.97	94.17	56.14	31.13	40.05

TABLE 5.11. Experiment results of the baseline model with word2vec and ortographic character-level embeddings on English formal dataset, *DS-2*

Entity type	Precision	Recall	F1
person	96.11	96.17	96.14
organization	87.75	87.96	87.85
location	92.04	90.83	91.43
miscellaneous	78.04	75.93	76.97
overall	**90.24**	**89.66**	**89.95**

embeddings such as word2vec can indeed increase the accuracy of the model. The model processed 23376 tokens with 1078 phrases and found 458 phrases on entity-level where 306 of them are correct. Additionally, it found 376 phrases on surface-form where 251 of them are correct. Results of this experiment are presented in Table 5.12..

TABLE 5.12. Experiment results of the baseline model with fastText, word2vec, morph2vec and orthographic character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	35.71	15.15	21.28	33.33	11.67	17.28
creative-work	56.25	6.34	11.39	60	6.62	11.92
group	43.75	12.73	19.72	40.48	12.06	18.58
location	72	48	57.6	71.79	44.8	55.17
person	74.9	43.93	55.38	75	41.6	53.52
product	40	4.72	8.45	50	5.13	9.3
overall	**66.81**	**28.39**	**39.84**	**66.76**	**26.31**	**37.74**

5.2.5.2. Transfer Learning Model

On the other hand, our proposed approach - transfer learning model - obtains an F1 score of **41.97%** on entity-level and **40.05%** on surface forms. The model, similar to the baseline model, processes a total of 23376 tokens with 1078 phrases and found 628 phrases on entity-level where 358 of them are correct. Moreover, it found 529 phrases on surface forms where 297 of them are correctly labeled. Details on different entity types can be found in Table 5.13.. We can see that transfer learning helps the model to learn rarely-seen entity types compared to the baseline model, thus the results are significantly improved.

TABLE 5.13. Experiment results of the transfer learning model with fasttext, morph2vec, word2vec and orthographic character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	32.43	18.18	23.3	34.62	15	20.93
creative-work	37.5	6.34	10.84	40.91	6.62	11.39
group	56.36	18.79	28.18	54	19.15	28.27
location	42.86	54	47.79	39.26	51.2	44.44
person	70.72	50.23	55.74	71.6	47.73	57.28
product	52.63	7.87	13.7	50	7.69	13.33
overall	**57.01**	**33.21**	**41.97**	**56.14**	**31.13**	**40.05**

5.2.5.3. Discussion & Comparison

As we can see in the results, transfer learning model that incorporates embeddings obtained from fasttext, morph2vec, word2vec and orthographic character-level embeddings achieves the best results out of all experiments with an F1 score of 41.97% on entity-level and 40.05% on surface forms.

In Table 5.14., we present the comparison our baseline and transfer learning models with the related work on English noisy dataset, *DS-3*. Aguilar et al. [1] reports the state-of-the-art results on this dataset with an F1 score of 41.86% entity level and 40.24% on surface forms. They apply multi-task learning with a CRF-based model that incorporates pre-trained word embeddings obtained from word2vec and orthographic character-level embeddings trained on a CNN with 2-stacked convolutional layers. They also make use of gazetteers for well-known entities of all of the entity types. von Däniken and Cieliebak [43] use a transfer learning model on which our proposed model is based. But unlike ours model, their model incorporates sentence-level embeddings (sent2vec) and capitalization features in addition to character-level embeddings trained on a CNN and pre-trained word embeddings obtained from FastText. Lin et al. [39] follow a similar approach as a CRF-based model and use word embeddings that are made of pre-trained word embeddings and character-level embeddings obtained from another bidirectional LSTM. They also incorporate syntactic information by using POS tags, dependency roles and word position in the sentence and head position. Sikdar and Gambäck [40] use a CRF as an ensemble-based approach which uses features learned from CRF, support vector machine (SVM) and an LSTM. They also use hand-crafted features such as POS tags, local context, chunk, suffix and prefix, word frequency and a collection of flags (is-word-length-less-than-5, is-all-digit etc.). Williams and Santia [41] introduce a statistical approach, context-sensitivity, where each word is associated with its contexts and context conditional probabilities are used to figure out NE tag probabilities. In addition to WNUT'17 dataset, they also used WNUT'16 dataset and a group of lexicon and external dictionaries. Jansson and Liu [42], inspired by the work of [38], use a bidirectional LSTM-CRF model that is similar to our baseline model but instead of orthographic features from the original work, LDA topic modeling and POS tags are chosen to feed the LSTM layer.

TABLE 5.14. Comparison of our models with the related work on English noisy dataset, DS-3

Related Work	F1 score (%)	
	Entity Level	Surface Form
Jansson and Liu [42]	39.98	37.77
Williams and Santia [41]	26.3	25.26
Sikdar and Gamback [40]	38.35	36.31
Lin et al. [39]	40.42	37.62
von Daniken and Cieliebak [43]	40.78	39.33
Aguilar et al. [1]	41.86	40.24
baseline model (ft, ortho, m2v, w2v)	**39.84**	**37.74**
transfer learning model-1 (ft, ortho)	**22.74**	**22.77**
transfer learning model-2 (ft, ortho, m2v, w2v)	**41.97**	**40.05**

Our final transfer learning model achieves competitive results on surface forms. However it achieves the best result on entity-level compared to the first two best models in WNUT'17; von Daniken and Cieliebak [43] and Aguilar et al. [1] although both of them make use of hand-crafted features such as capitalization or domain-specific knowledge such as gazetteers. Only model without the use of hand-crafted features or external resources is the one presented in Jansson and Liu [42]. Compared to this model, our transfer learning model predicts both common and infrequent NE types better which we believe this is mostly a result of using additional CRF-layer that is trained on another noisy dataset.

6. CONCLUSION

6.1. Concluding Remarks

We have researched the question of effectively using neural networks, and deep learning for that matter, instead of rule-based approaches in order to achieve the named entity recognition task for morphologically-rich languages such as Turkish and, more importantly, for noisy data. We have also investigated the effects of different word and sub-word level representation learning methods such as character n-gram level embeddings of fasttext, morpheme level embeddings of morph2vec and orthographic character-level embeddings. The existing studies in the literature on named entity recognition on Turkish still make use of hand-crafted features (e.g. capitalization, numerical/date/time patterns or other rule-based features) and external resources (e.g. gazetteers, lexicons). We believe this approach makes them domain-specific and language-specific whereas our aim is to provide a neural network architecture and in the process obtain valuable features without using any hand-crafted features or external knowledge resources. Due to having scarce annotated data, we have also researched the means in transfer learning from formal data to noisy (informal) data. So that, the resulting Turkish NER model can also be used in different applications on different domains without the need for domain-specific knowledge. We also argue that our proposed model can also be effectively used for other morphologically-rich languages.

In this regard, we have experimented with a bidirectional LSTM-CRF architecture as our baseline model that is widely adopted for various sequence labeling tasks in the literature. As our feature set, we have also made use of pre-trained embedding that we obtained from fasttext and morph2vec. Additionally, we have also trained orthographic character-level bidirectional LSTM in order to learn orthographic features of words. For the experiments on English dataset, we have also used pre-trained word embeddings that we obtained from word2vec. Although the results we obtained during experimentation of this baseline model seem promising both for Turkish and English, it performs comparably poor for rarely-seen entity types (e.g. PERCENTAGE, TIME, MONEY in Turkish dataset and CORPORATION in English dataset). We believe this is an expected outcome since, for example, there are only 5 samples of PERCENTAGE, 20 samples of TIME and 24 samples of MONEY types out of total 455 entities in Turkish noisy dataset, DS-1. Similarly, there are 312 samples of CORPORATION out of total 3889 entities in English noisy dataset, DS-3. Furthermore, due to applying 10-fold cross-validation on Turkish noisy dataset, DS-1, the frequency of seeing these infrequent examples is quite small. In order to overcome the problem of labeling these infrequent NE types, we have extended the baseline model by incorporating another CRF layer that we trained on a different larger dataset. While this proposed model is alternately trained on two datasets, the neural layers up to the last CRF layers are shared so that the overall model can learn from both datasets but optimized and evaluated only for the target, noisy dataset. This allows us to successfully learn meaningful features to correctly label some of the infrequent types and consequently it improved the overall results. The transfer learning model, similar to the baseline model, also makes use of pre-trained embeddings obtained from fasttext, morph2vec and orthographic character-level embeddings that are trained on another bidirectional LSTM. As a result, the proposed model for Turkish achieved an F1 score of 65.72% on Turkish noisy dataset, DS-1. Additionally, similar to its baseline counterpart, the proposed model for English also uses pre-trained embeddings of word2vec. Consequently, we obtained an F1 score of 41.97% on entity-level and 40.05% on surface forms for English.

These experimental results show that sub-word level representation techniques such as orthographic character-level and character n-gram level embeddings play a vital role for named entity recognition on morphologically-rich languages. More importantly, we can successfully learn valuable information without using hand-crafted features or domain-specific external resources. Furthermore, it is also proven that transfer learning approach can indeed effectively be used to tackle the problem of data scarcity. The proposed Turkish model obtains the highest results when compared to the state-of-the-art result of Şeker and Eryiğit [6] when their *Twitter mention* feature is excluded. As we have stated earlier, we believe that Twitter mentions should not be included as PERSON types and therefore should not be labeled at all which is also the case for other named entity recognition studies in the literature.

Moreover, the proposed model for English achieves competitive results when compared to state-of-the-art studies on English noisy data such as Aguilar et al. [1], and von Däniken [43].

6.2. Future Work

Given that one of our main goals is to perform named entity recognition on morphologically-rich languages, we plan to investigate the means to further improve the contribution of subword level representation techniques, specifically character level and morpheme level embeddings. In this regard, focusing on the distinction between derivational and inflectional morphemes can be used since majority of named entities are proper nouns that can only take inflectional suffixes. We also aim to achieve better when it comes to transfer learning by investigating the effects of different neural architectures. Additionally, we also intend to experiment with different morphologically-rich languages such as other Turkic languages.

A APPENDIX: EXPERIMENTAL RESULTS ON TURKISH NOISY DATASET

TABLE 1.1. Experiment of baseline model with fasttext and character-level embeddings on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	69.37	49.09	57.47
organization	85.74	58.96	69.4
location	74.78	46.26	56.17
date	46	20.23	27.52
time	0	0	0
money	0	0	0
percentage	0	0	0
overall	**74.25**	**48.69**	**58.69**

TABLE 1.2. Experiment of baseline model with morph2vec and character-level embeddings on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	40.25	14.29	20.83
organization	57.48	28.38	36.49
location	0	0	0
date	0	0	0
time	0	0	0
money	0	0	0
percentage	0	0	0
overall	**47.78**	**14.29**	**21.88**

TABLE 1.3. Experiment of baseline model with word2vec and character-level embeddings on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	55.1	23.11	32.21
organization	51.73	29.66.24	36.81
location	5	0.37	0.69
date	20	4.5	7.33
time	0	0	0
money	0	0	0
percentage	0	0	0
overall	**53.28**	**19.57**	**28.45**

TABLE 1.4. Experiment of baseline model with fasttext and orthographic character-level embeddings on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	69.74	52.95	60.04
organization	82.87	59.38	68.54
location	64.47	45.76	52.75
date	46.66	19.91	26.26
time	11.11	11.11	11.11
money	0	0	0
percentage	0	0	0
overall	**71.96**	**51.1**	**59.7**

TABLE 1.5. Experiment of baseline model without CRF layer and with fasttext and orthographic character-level embeddings on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	39.1	10.67	16.52
organization	68.64	31.21	42.44
location	0	0	0
date	0	0	0
time	0	0	0
money	0	0	0
percentage	0	0	0
overall	**53.81**	**14.04**	**22.09**

TABLE 1.6. Experiment of baseline model with fasttext, character-level and orthographic character-level embeddings on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	70.84	48.92	57.79
organization	85.96	56.88	67.7
location	70.51	43.35	53.14
date	60	21.06	29.22
time	10	3.33	5
money	0	0	0
percentage	0	0	0
overall	**74.79**	**48.1**	**58.42**

TABLE 1.7. Experiment of baseline model with fasttext, morph2vec and orthographic character-level embeddings that is trained on CNN, on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	69.36	49.62	57.49
organization	82.78	59.23	68.72
location	76.84	47.03	56.59
date	56.5	23.1	28.78
time	0	0	0
money	0	0	0
percentage	0	0	0
overall	**74.00**	**49.46**	**59.07**

TABLE 1.8. Experiment of baseline model with fasttext, morph2vec that is trained for morpheme and character embeddings and orthographic character-level embeddings that is trained on CNN, on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	72.09	45.81	55.62
organization	85.8	60.99	70.1
location	78.9	50.04	59.75
date	43	20.49	26.48
time	0	0	0
money	0	0	0
percentage	0	0	0
overall	**76.37**	**48**	**58.78**

TABLE 1.9. Experiment of baseline model with morph2vec and orthographic character-level embeddings on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	52.05	13.71	21.58
organization	72.53	27.7	39.39
location	10	0.32	0.62
date	0	0	0
time	0	0	0
money	0	0	0
percentage	0	0	0
overall	**60.6**	**14.11**	**22.81**

TABLE 1.10. Experiment of baseline model with word2vec and orthographic character-level embeddings on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	59.77	24.07	33.91
organization	68.3	15.95	25.28
location	0	0	0
date	20	2.76	4.76
time	0	0	0
money	0	0	0
percentage	0	0	0
overall	**60.64**	**16.57**	**25.8**

TABLE 1.11. Experiment of baseline model with fasttext and morph2vec embeddings on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	67.07	51.43	58.02
organization	83.17	60.41	69.38
location	66.8	45.33	53.56
date	35.83	24.77	28.26
time	0	0	0
money	0	0	0
percentage	0	0	0
overall	**71.36**	**50.02**	**58.73**

TABLE 1.12. Experiment of baseline model with fasttext, morph2vec and character-level embeddings on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	71.04	51.52	59.60
organization	86.09	56.45	67.69
location	72.56	43.90	53.70
date	37.66	20.78	26.45
time	0	0	0
money	0	0	0
percentage	0	0	0
overall	**74.48**	**49.11**	**59.03**

TABLE 1.13. Experiment of baseline model with fasttext, morph2vec and orthographic character-level embeddings on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	70.15	49.45	57.71
organization	83.31	59.53	69.07
location	69.09	44.23	52.8
date	53.66	18.5	25.85
time	0	0	0
money	0	0	0
percentage	0	0	0
overall	**73.12**	**48.7**	**58.32**

TABLE 1.14. Experiment of alternate transfer learning model without additional ReLU and Linear layers and with fasttext, orthographic character-level embeddings on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	72.72	53.2	61.09
organization	84.17	59.51	69.44
location	69.61	47.76	56.31
date	56.66	25.27	29.95
time	20	6.66	10
money	10	3.33	5
percentage	0	0	0
overall	**74.88**	**51.57**	**60.89**

TABLE 1.15. Experiment of transfer learning model with fasttext, and orthographic character-level embeddings on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	65.06	63.92	64.24
organization	78.25	68.57	72.77
location	67.59	66.56	66.55
date	50.33	39.7	42.26
time	30	20	23
money	2.5	3.33	2.85
percentage	0	0	0
overall	**68.09**	**63.04**	**65.37**

TABLE 1.16. Experiment of transfer learning model with fasttext and orthographic character-level embeddings that is trained on CNN on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	68.42	59.99	63.49
organization	78.04	68.96	72.91
location	68.93	64.59	66.37
date	48.27	31.85	36.64
time	20	8.33	11.66
money	6.66	10	8
percentage	0	0	0
overall	**68.09**	**63.04**	**65.37**

TABLE 1.17. Experiment of transfer learning model with fasttext, morph2vec that is trained for morpheme and character embeddings and orthographic character-level embeddings that is trained on CNN, on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	65.9	23.78	34.7
organization	65.72	30.74	41.39
location	70.84	22.48	32.87
date	0	0	0
time	0	0	0
money	0	0	0
percentage	0	0	0
overall	**65.63**	**23.85**	**34.88**

TABLE 1.18. Experiment of transfer learning model with fasttext, morph2vec and orthographic character-level embeddings on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	68.52	62.19	65.11
organization	81.88	68.42	74.25
location	68.25	58.13	61.94
date	47.66	24.44	29.96
time	10	6.66	8
money	0	0	0
percentage	0	0	0
overall	**70.78**	**60.35**	**65.12**

TABLE 1.19. Experiment of transfer learning model with fasttext, morph2vec and orthographic character-level embeddings that is trained on CNN on Turkish noisy dataset, *DS-1*

Entity type	Precision	Recall	F1
person	72.69	57.9	64.34
organization	80.56	67.52	73.08
location	75.38	60.21	66.24
date	50.83	29.59	36.57
time	30	18.66	21.33
money	20	13.33	15
percentage	0	0	0
overall	**74.45**	**58.94**	**65.72**

TABLE 1.20. Experiment of transfer learning model with fasttext, morph2vec and orthographic character-level embeddings that is trained on CNN on Turkish noisy dataset, *DS-1* without preprocessing of Twitter mentions (same as the original dataset where Twitter mentions are also labelled as *PERSON*)

Entity type	Precision	Recall	F1
person	95.93	92.98	94.42
organization	84.36	63.75	72.08
location	72.69	63.9	67.15
date	56.33	37.54	41.91
time	26.66	10	14.3
money	0	0	0
percentage	0	0	0
overall	**93.71**	**88.24**	**90.88**

B APPENDIX: EXPERIMENTAL RESULTS ON ENGLISH NOISY AND FORMAL DATASETS

TABLE 2.1. Experiment of baseline model with word2vec and character-level embeddings on English formal dataset, *DS-2*

Entity type	Precision	Recall	F1
person	96.11	96.17	96.14
organization	87.75	87.96	87.85
location	92.04	90.83	91.43
miscellaneous	78.04	75.93	76.97
overall	**90.24**	**89.66**	**89.95**

TABLE 2.2. Experiment of baseline model with fasttext and character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	0	0	0	0	0	0
creative-work	28.57	1.41	2.68	28.57	1.47	2.8
group	0	0	0	0	0	0
location	42.5	11.33	17.89	43.59	13.6	20.73
person	61.54	14.95	24.06	61.05	15.47	24.68
product	0	0	0	0	0	0
overall	**52.87**	**7.7**	**13.44**	**52.38**	**8.07**	**13.99**

TABLE 2.3. Experiment of baseline model with morph2vec and character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	0	0	0	0	0	0
creative-work	0	0	0	0	0	0
group	100	0.61	1.2	100	0.71	1.41
location	22.22	13.33	16.67	18.6	12.8	15.17
person	30.71	9.11	14.05	31.67	10.13	15.35
product	0	0	0	0	0	0
overall	**26.67**	**5.57**	**9.21**	**25.7**	**5.77**	**9.42**

TABLE 2.4. Experiment of baseline model with word2vec and character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	33.33	16.67	22.22	32	17.02	22.22
creative-work	46.67	4.93	8.92	53.85	5.83	10.53
group	47.37	10.91	17.73	44.12	11.81	18.63
location	74.42	42.67	54.24	73.85	40	51.89
person	71.2	41.59	52.51	71.57	52.33	60.46
product	27.27	2.36	4.35	22.22	1.85	3.42
overall	**64.9**	**26.07**	**37.19**	**64.57**	**28.21**	**39.27**

TABLE 2.5. Experiment of baseline model with fasttext and orthographic character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	0	0	0	0	0	0
creative-work	36.36	2.82	5.23	36.36	2.94	5.44
group	29.41	3.03	5.49	29.41	3.55	6.33
location	31.01	26.67	28.67	27.59	25.6	26.56
person	62.16	26.87	37.52	60.37	26.4	36.73
product	0	0	0	0	0	0
overall	**47.54**	**15.21**	**23.05**	**45.02**	**14.68**	**22.13**

TABLE 2.6. Experiment of baseline model with morph2vec and orthographic character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	0	0	0	0	0	0
creative-work	0	0	0	0	0	0
group	0	0	0	0	0	0
location	18.1	12.67	14.9	16	12.8	14.22
person	30.91	7.94	12.64	31.78	9.04	14.11
product	0	0	0	0	0	0
overall	**24.2**	**4.92**	**8.17**	**23.7**	**5.24**	**8.58**

TABLE 2.7. Experiment of baseline model with word2vec and orthographic character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	29.41	15.15	20	28	14.89	19.44
creative-work	53.85	4.93	9.03	53.85	5.83	10.53
group	48.72	11.52	18.63	47.06	12.6	19.88
location	69.57	42.67	52.89	69.01	40.83	51.31
person	74.79	41.59	53.45	75.25	53.41	62.47
product	53.85	5.51	10	50	5.56	10
overall	**66.43**	**26.44**	**37.82**	**66.29**	**29.21**	**40.55**

TABLE 2.8. Experiment of baseline model with fasttext, morph2vec and character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	0	0	0	0	0	0
creative-work	14.29	0.7	1.34	14.29	0.74	1.4
group	8.33	0.61	1.13	8.33	0.71	1.31
location	36.54	12.67	18.81	37.25	15.2	21.59
person	59.43	14.72	23.6	59.57	14.93	23.88
product	0	0	0	0	0	0
overall	**46.67**	**7.79**	**13.35**	**46.11**	**8.07**	**13.74**

TABLE 2.9. Experiment of baseline model with fasttext, morph2vec and orthographic character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	0	0	0	0	0	0
creative-work	36.36	2.82	5.23	36.36	2.94	5.44
group	40	4.85	8.65	33.33	4.26	7.55
location	29.51	24	26.47	27.43	24.8	26.05
person	63.59	27.34	38.24	61.11	26.4	36.87
product	0	0	0	0	0	0
overall	**48.39**	**15.31**	**23.26**	**45.45**	**14.68**	**22.19**

TABLE 2.10. Experiment of baseline model with fasttext and morph2vec embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	0	0	0	0	0	0
creative-work	33.33	1.41	2.7	33.33	1.47	2.82
group	0	0	0	0	0	0
location	57.14	10.67	17.98	59.26	12.8	21.05
person	60.22	13.08	21.5	59.02	12.53	20.61
product	0	0	0	0	0	0
overall	**54.41**	**6.86**	**12.19**	**52.85**	**6.81**	**12.07**

TABLE 2.11. Experiment of baseline model with fasttext, character-level and orthographic character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	0	0	0	0	0	0
creative-work	36.36	2.82	5.23	36.36	2.94	5.44
group	30.43	4.24	7.45	27.27	4.26	7.36
location	32	21.33	25.6	29.35	21.6	24.88
person	60.67	25.23	35.64	60.25	25.87	36.19
product	0	0	0	0	0	0
overall	**48.24**	**14.01**	**21.71**	**46.69**	**14.05**	**21.6**

TABLE 2.12. Experiment of baseline model with fasttext, morph2vec, word2vec and orthographic character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	35.71	15.15	21.28	33.33	11.67	17.28
creative-work	56.25	6.34	11.39	60	6.62	11.92
group	43.75	12.73	19.72	40.48	12.06	18.58
location	72	48	57.6	71.79	44.8	55.17
person	74.9	43.93	55.38	75	41.6	53.52
product	40	4.72	8.45	50	5.13	9.3
overall	**66.81**	**28.39**	**39.84**	**66.76**	**26.31**	**37.74**

TABLE 2.13. Experiment of alternate transfer learning model without additional ReLU and Linear layers and with fasttext, orthographic character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	0	0	0	0	0	0
creative-work	0	0	0	0	0	0
group	33.33	5.45	9.38	28	4.96	8.43
location	24.31	29.33	26.59	21.43	28.8	24.57
person	25.14	41.12	31.21	29.42	39.47	33.71
product	0	0	0	0	0	0
overall	**24.47**	**21.24**	**22.74**	**26.38**	**20.02**	**22.77**

TABLE 2.14. Experiment of transfer learning model with fasttext, morph2vec and orthographic character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	0	0	0	0	0	0
creative-work	50	1.41	2.74	50	1.47	2.86
group	14.29	12.73	13.46	13.97	13.48	13.72
location	57.32	31.33	40.52	52.86	29.6	37.95
person	68.6	41.36	51.6	68.9	38.4	49.32
product	0	0	0	0	0	0
overall	**48.81**	**22.91**	**31.19**	**46.98**	**21.17**	**29.19**

TABLE 2.15. Experiment of transfer learning model with fasttext, morph2vec, PoS tag embeddings and orthographic character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	20	1.52	2.82	20	1.67	3.08
creative-work	33.33	2.11	3.97	33.33	2.21	4.14
group	24.39	12.12	16.19	23.61	12.06	15.96
location	63.41	34.67	44.83	63.64	33.6	43.98
person	69.23	33.64	45.28	70.48	31.2	43.25
product	0	0	0	0	0	0
overall	**55.56**	**20.41**	**29.85**	**54.88**	**18.87**	**28.08**

TABLE 2.16. Experiment of transfer learning model with fasttext, word2vec and orthographic character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	32.5	19.7	24.53	31.03	15	20.22
creative-work	47.37	6.34	11.18	50	6.62	11.69
group	54.55	18.18	27.27	53.06	18.44	27.37
location	42.86	56	48.55	39.64	53.6	45.58
person	68.34	50.93	58.37	68.56	48.27	56.65
product	39.13	7.09	12	36.36	6.84	11.51
overall	**55.67**	**33.67**	**41.97**	**54.45**	**31.45**	**39.87**

TABLE 2.17. Experiment of baseline model with fasttext, morph2vec, word2vec and orthographic character-level embeddings that is trained on CNN on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	34.78	12.12	17.98	31.25	8.33	13.16
creative-work	50	4.93	8.97	58.33	5.15	9.46
group	47.62	12.12	19.32	44.74	12.06	18.99
location	66.97	48.67	56.37	64.77	45.6	53.52
person	73.85	44.86	55.81	74.76	41.87	53.68
product	42.86	4.72	8.51	42.86	5.13	9.16
overall	**66.23**	**28.39**	**39.74**	**65.87**	**26.1**	**37.39**

TABLE 2.18. Experiment of transfer learning model with fasttext, morph2vec, word2vec and orthographic character-level embeddings on English noisy dataset, *DS-3*

Entity Type	Entity Level (%)			Surface Form (%)		
	Precision	Recall	F1 score	Precision	Recall	F1 score
corporation	32.43	18.18	23.3	34.62	15	20.93
creative-work	37.5	6.34	10.84	40.91	6.62	11.39
group	56.36	18.79	28.18	54	19.15	28.27
location	42.86	54	47.79	39.26	51.2	44.44
person	70.72	50.23	55.74	71.6	47.73	57.28
product	52.63	7.87	13.7	50	7.69	13.33
overall	**57.01**	**33.21**	**41.97**	**56.14**	**31.13**	**40.05**

REFERENCES

[1] Gustavo Aguilar, Suraj Maharjan, Adrian Pastor López Monroy, and Thamar Solorio. A multi-task approach for named entity recognition in social media data. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 148–153. **2017**.

[2] Ralph Grishman and Beth Sundheim. Message understanding conference-6: A brief history. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, volume 1. **1996**.

[3] Ralph Grishman. Sixth message understanding conference MUC-6 task description, **1996**. [Online; accessed 25. Sep. 2018].

[4] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, **2016**.

[5] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370. Association for Computational Linguistics, **2005**.

[6] Gökhan Akın Şeker and Gülşen Eryiğit. Extending a CRF-based named entity recognition model for Turkish well formed text and user generated content 1. *Semantic Web*, 8(5):625–642, **2017**.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, **2013**.

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, **2017**. ISSN 2307-387X.

[9] Ahmet Üstün, Murathan Kurfalı, and Burcu Can. Characters or morphemes: How to represent words? In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 144–153. **2018**.

[10] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. **2001**.

[11] R Hecht-Nielsen. Neural network primer: part i. *AI Expert*, pages 4–51, **1989**.

[12] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91:1, **1991**.

[13] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, **2001**.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, **1997**.

[15] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, **2014**.

[16] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, **2015**.

[17] Zhilin Yang, Ruslan Salakhutdinov, and William W Cohen. Transfer learning for sequence tagging with hierarchical recurrent networks. *arXiv preprint arXiv:1703.06345*, **2017**.

[18] Onur Güngör, Eray Yıldız, Suzan Üsküdarlı, and Tunga Güngör. Morphological embeddings for named entity recognition in morphologically rich languages. *arXiv preprint arXiv:1706.00506*, **2017**.

[19] Gökhan Tür, Dilek Hakkani-Tür, and Kemal Oflazer. A statistical information extraction system for Turkish. *Natural Language Engineering*, 9(2):181–210, **2003**.

[20] Gökhan Çelikkaya, Dilara Torunoğlu, and Gülşen Eryiğit. Named entity recognition on real data: a preliminary investigation for Turkish. In *Application of Information and Communication Technologies (AICT), 2013 7th International Conference on*, pages 1–5. IEEE, **2013**.

[21] Tuğba Pamay, Umut Sulubacak, Dilara Torunoğlu-Selamet, and Gülşen Eryiğit. The annotation process of the ITU web treebank. In *Proceedings of The 9th Linguistic Annotation Workshop*, pages 95–101. **2015**.

[22] Gülşen Eryiğit. ITU Turkish NLP web service. In *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 1–4. **2014**.

[23] Dilek Küçük and Ralf Steinberger. Experiments to improve named entity recognition on Turkish tweets. *arXiv preprint arXiv:1410.8668*, **2014**.

[24] Dilek Küçük, Guillaume Jacquet, and Ralf Steinberger. Named entity recognition on Turkish tweets. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*. **2014**.

[25] Beyza Eken and Cüneyd Tantuğ. Recognizing named entities in Turkish tweets. In *Proceedings of the Fourth International Conference on Software Engineering and Applications, Dubai, UAE*. **2015**.

[26] Eda Okur, Hakan Demir, and Arzucan Özgür. Named entity recognition on Twitter for Turkish using semi-supervised learning with word embeddings. In *LREC*. **2016**.

[27] Haşim Sak, Tunga Güngör, and Murat Saraçlar. Turkish language resources: Morphological parser, morphological disambiguator and web corpus. In *Advances in natural language processing*, pages 417–427. Springer, **2008**.

[28] Haşim Sak, Tunga Güngör, and Murat Saraçlar. Resources for Turkish morphological processing. *Language resources and evaluation*, 45(2):249–261, **2011**.

[29] B Sezer, T Sezer, and Mersin Ünivesitesi. TS Corpus: Herkes için Türkçe derlem. In *Proceedings 27th National Linguistics Conference. May*, pages 3–4. **2013**.

[30] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics, **2010**.

[31] Dilara Torunoğlu and Gülşen Eryiğit. A cascaded approach for social media text normalization of Turkish. In *Proceedings of the 5th Workshop on Language Analysis for Social Media (LASM)*, pages 62–70. **2014**.

[32] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*, **2015**.

[33] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, **2011**.

[34] Jason PC Chiu and Eric Nichols. Named entity recognition with bidirectional LSTM-CNNs. *arXiv preprint arXiv:1511.08308*, **2015**.

[35] Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. *arXiv preprint arXiv:1603.01354*, **2016**.

[36] Alan Ritter, Sam Clark, Oren Etzioni, et al. Named entity recognition in tweets: an experimental study. In *Proceedings of the conference on empirical methods in*

natural language processing, pages 1524–1534. Association for Computational Linguistics, **2011**.

[37] Daniel Ramage, David Hall, Ramesh Nallapati, and Christopher D Manning. Labeled lda: A supervised topic model for credit attribution in multi-labeled corpora. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 248–256. Association for Computational Linguistics, **2009**.

[38] Nut Limsopatham and Nigel Henry Collier. Bidirectional LSTM for named entity recognition in Twitter messages. **2016**.

[39] Bill Y Lin, Frank Xu, Zhiyi Luo, and Kenny Zhu. Multi-channel BiLSTM-CRF model for emerging named entity recognition in social media. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 160–165. **2017**.

[40] Utpal Kumar Sikdar and Björn Gambäck. A feature-based ensemble approach to recognition of emerging and rare named entities. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 177–181. **2017**.

[41] Jake Williams and Giovanni Santia. Context-sensitive recognition for emerging and rare entities. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 172–176. **2017**.

[42] Patrick Jansson and Shuhua Liu. Distributed representation, lda topic modelling and deep learning for emerging named entity recognition from social media. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 154–159. **2017**.

[43] Pius von Däniken and Mark Cieliebak. Transfer learning and sentence level features for named entity recognition on tweets. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 166–171. **2017**.

[44] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. Unsupervised learning of sentence embeddings using compositional n-gram features. *arXiv preprint arXiv:1703.02507*, **2017**.

[45] Low-resource named entity recognition via multi-source projection: Not quite there yet? - W-NUT201825.pdf, **2018**. [Online; accessed 1. Oct. 2018].

[46] Fréderic Godin, Baptist Vandersmissen, Wesley De Neve, and Rik Van de Walle. Multimedia lab @ ACL WNUT NER shared task: Named entity recognition for Twitter microposts using distributed word representations. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 146–153. **2015**.

[47] Gökhan Akın Şeker and Gülşen Eryiğit. Initial explorations on using CRFs for Turkish named entity recognition. *Proceedings of COLING 2012*, pages 2459–2474, **2012**.

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

[49] Leon Derczynski, Eric Nichols, Marieke van Erp, and Nut Limsopatham. Results of the WNUT2017 shared task on novel and emerging entity recognition. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 140–147. **2017**.

[50] WNUT2017 Emerging entities dataset.

[51] Timothy Baldwin, Marie-Catherine de Marneffe, Bo Han, Young-Bum Kim, Alan Ritter, and Wei Xu. Shared tasks of the 2015 workshop on noisy user-generated text: Twitter lexical normalization and named entity recognition. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 126–135. **2015**.

CURRICULUM VITAE

Credentials

Name,Surname	:	EMRE KAĞAN AKKAYA
Place of Birth	:	Bursa,Turkey
Marital Status	:	Single
E-mail	:	emrekaganakkaya@gmail.com
Address	:	Computer Engineering Dept., Hacettepe University Beytepe-ANKARA

Education

BSc. :	Computer Engineering Dept., Hacettepe University, Turkey
MSc. :	Computer Engineering Dept., Hacettepe University, Turkey

Foreign Languages

English

Work Experience

Bor Yazılım (2012-2013) - Part-time #NET Web Developer

Agem Bilişim (2013-) - Full-time Java/Python Developer

Areas of Experiences

Machine Learning, NLP

Projects and Budgets

——

Publications

”Transfer Learning for Turkish Named Entity Recognition on Noisy Text”,

Natural Language Engineering, 2018 (**submitted**),

Emre Kağan Akkaya, Burcu Can Buğlalılar

Oral and Poster Presentations

——

HACETTEPE UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING
THESIS/DISSERTATION ORIGINALITY REPORT

HACETTEPE UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING
TO THE DEPARTMENT OF COMPUTER ENGINEERING

Date: 13/12/2018

Thesis Title / Topic: Deep Neural Networks for Named Entity Recognition on Social Media

According to the originality report obtained by my thesis advisor by using the *Turnitin* plagiarism detection software and by applying the filtering options stated below on 13/12/2018 for the total of 103 pages including the a) Title Page, b) Introduction, c) Main Chapters, d) Conclusion sections of my thesis entitled as above, the similarity index of my thesis is 8 %.

Filtering options applied:

1. Bibliography/Works Cited excluded
2. Quotes excluded
3. Match size up to 5 words excluded

I declare that I have carefully read Hacettepe University Graduate School of Sciene and Engineering Guidelines for Obtaining and Using Thesis Originality Reports; that according to the maximum similarity index values specified in the Guidelines, my thesis does not include any form of plagiarism; that in any future detection of possible infringement of the regulations I accept all legal responsibility; and that all the information I have provided is correct to the best of my knowledge.

I respectfully submit this for approval.

Date and Signature
13.12.2018

Name Surname: EMRE KAĞAN AKKAYA

Student No: N14128491

Department: COMPUTER ENGINEERING

Program: MASTER OF SCIENCE

Status: ☒ Masters □ Ph.D. □ Integrated Ph.D.

ADVISOR APPROVAL

APPROVED.

Asst. Prof. Dr. Burcu Can BUĞLALILAR

(Title, Name Surname, Signature)