DEEP LEARNING FOR INFORMATION EXTRACTION

by

Thien Huu Nguyen

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

January, 2018

Professor Ralph Grishman

Dedication

To my beloved mother

Acknowledgments

People have different stories to tell in their PhD time. My PhD journey began at a day of Fall 2012. I was supposed to meet my advisor, Professor Ralph Grishman, for the first time in his office. However, on my road to his office that day, I accidentally ran into him in the Broadway avenue. It was the first time I talked to Ralph in person after the many discussions via emails. The unexpected meeting with Ralph impressed me so much that I forgot my nerve and worry of the first days I entered United States to pursue a PhD degree. Ralph was so nice and approachable that I could immediately feel the trust and confidence, the values I have relied on throughout my PhD. The more I work with Ralph, the more fortunate I find myself. Ralph gave me the courage and freedom to explore deep learning for information extraction at the very early days of the field. He provided me with valuable advice and suggestion whenever I need to deal with research challenges and career decisions. It is his advice and enthusiasm that made me more determined to purse an academic career that I have always dreamed of. His work responsibility and positive attitudes on research would be the principles I employ in the rest of my life. Ralph is the first and foremost person I would like to thank in my academic career.

ACKNOWLEDGMENTS

I own special thanks to Professor Kyunghyun Cho, an outstanding advisor and a great friend who taught me much about deep learning and its potentials to transform our life. Kyunghyun has always been generous on spending his time discussing with me and providing me with great helps and encouragement. I also would like to thank the members of the Proteus project at New York University of which I am always proud to be a part. Dr. Adam Meyers gave me much insights into linguists. Yifan He and Lisheng Fu were always willing to discuss with me on new ideas. Xiang Li was a great listener who encouraged me whenever I have any problem in research and life. Masha Pershina had been a great collaborator on several projects. Kai Cao was always generous to share with me research resources. Bonan Min and Wei Xu were the great academic brother and sister who showed me great tips and advice. I have also learned much useful information from the discussion with Professor Satoshi Sekine and Dr. Angus Grieve-Smith.

During my PhD, I was fortunate to have two outstanding internships with two different groups at IBM T.J. Watson Research Center. Some parts in this dissertation were conducted during such internships. My first internship was associated with the group on Statistical Multilingual Information Extraction from Text where I had the opportunity to work with Dr. Radu Florian, Dr. Avirup Sil, Dr. Georgiana Dinu, Dr. Salim Rukous, Dr. Vittorio Castelli and many other great people. I have obtained much research experience and received much support from that internship. I learned to implement my first recurrent neural networks there. I would like to especially thank Dr. Radu Florian for providing me so much guidance and support even after I completed the internship. My second internship was

within the Knowledge Induction team, working with Dr. Mariano Rodriguez muro, Dr. Oktie Hassanzadeh, Dr. Achille Fokoue, Dr. Mohammad Sadoghi Hamedani, Dr. Alfio M Gliozzo and Dr. Lisa Amini. This was another fruitful internship at IBM with one paper and one patent published that I would not be able to make without such great mentors. I also would like to thank Dr. Barbara Plank who collaborated with me and taught me how to write a good paper.

Last, but certainly not least, I would like to thank my parents and my wife for their tremendous emotional support of my PhD study. Their infinite love and unconditional care have served as the strong basis that I could always resort to in my life. Their trust, encouragement and sympathy have been the major forces that drive me forward and conquer new challenges. This dissertation was written while my wife was expecting our first child. We went to the hospital for delivery few hours after I defended this dissertation in November 28, 2017. An angel, named An Quynh Nguyen, was born two days later. I am grateful to her for bringing this miracle to my PhD career.

My PhD study was gratefully supported by Vietnam Education Foundation, the Ph.D Fellowship from IBM, and the Dean's Dissertation Fellowship and the Henry MacCracken Fellowship from the Graduate School of Arts and Science at New York University.

Abstract

The explosion of data has made it crucial to analyze the data and distill important information effectively and efficiently. A significant part of such data is presented in unstructured and free-text documents. This has prompted the development of the techniques for information extraction that allow computers to automatically extract structured information from the natural free-text data. Information extraction is a branch of natural language processing in artificial intelligence that has a wide range of applications, including question answering, knowledge base population, information retrieval etc. The traditional approach for information extraction has mainly involved hand-designing large feature sets (feature engineering) for different information extraction problems, i.e, entity mention detection, relation extraction, coreference resolution, event extraction, and entity linking. This approach is limited by the laborious and expensive effort required for feature engineering for different domains, and suffers from the unseen word/feature problem of natural languages.

This dissertation explores a different approach for information extraction that uses deep learning to automate the representation learning process and generate more effective features. Deep learning is a subfield of machine learning that uses

multiple layers of connections to reveal the underlying representations of data. I develop the fundamental deep learning models for information extraction problems and demonstrate their benefits through systematic experiments.

First, I examine word embeddings, a general word representation that is produced by training a deep learning model on a large unlabelled dataset. I introduce methods to use word embeddings to obtain new features that generalize well across domains for relation extraction. This is done for both the feature-based method and the kernel-based method of relation extraction.

Second, I investigate deep learning models for different problems, including entity mention detection, relation extraction and event detection. I develop new mechanisms and network architectures that allow deep learning to model the structures of information extraction problems more effectively. Some extensive experiments are conducted on the domain adaptation and transfer learning settings to highlight the generalization advantage of the deep learning models for information extraction.

Finally, I investigate the joint frameworks to simultaneously solve several information extraction problems and benefit from the inter-dependencies among these problems. I design a novel memory augmented network for deep learning to properly exploit such inter-dependencies. I demonstrate the effectiveness of this network on two important problems of information extraction, i.e, event extraction and entity linking.

Table of contents

List of Figures

List of Figures

List of Tables

List of Tables

List of Tables

List of Tables

Chapter 1

Introduction

Entities and events are the central objects in the languages we produce in discussion and communication everyday. For instance, the articles from news might describe some recent attacks (events) while we might talk about celebrities or politicians (entities) in our casual discussion. It is therefore crucial for computers to recognize such entities and events so that they can come closer to the understanding of human languages. This is essentially the target of Information Extraction (IE), a branch of research in Natural Language Processing (NLP), that aims at identifying entities, events and the inter-connections between them within text. The ultimate goal is to transfer information in text into a more accessible form for other computer applications such as question answering, information retrieval, knowledge base population, knowledge reasoning, to name a few.

In order to solve the problems of IE, the traditional systems have employed different pipelines to generate feature representations (feature sets) that are then fed into machine learning models to perform classification or labeling. These feature

pipelines often involve various NLP supervised modules and resources to extract different linguistic characteristics, hopefully capturing important features for the IE tasks. The determination of which NLP modules and resources are used and which features should be extracted is called "*feature engineering*". For convenience, we will also call these traditional feature engineering models "*feature-based*" models. There are three major problems with this feature-based approach:

1. Feature engineering for IE is a manual and expensive process that requires much linguistic intuition as well as domain expertise. The feature representations are often customized for some specific domains, thus necessitating additional investigations whenever a new domain of data is presented.

2. Despite much effort on hand-designing feature representations for IE, the resulting feature sets might be not necessarily optimal. This issue can be seen in three aspects. First, as our understanding about the IE tasks and their domains (domain knowledge) is often incomplete, the feature designer might still miss some important features for his tasks. Second, it is challenging to realize and capture the interactions between the designed features, potentially causing information redundancy in the feature sets. Third, the NLP supervised modules and resources for feature generation might involve errors, leading to errors of the features they generate. All these aspects would eventually impair the performance of the IE systems.

3. The feature-based models suffer from the data sparsity or unseen word/feature problem. In this problem, some important words/features of the machine

learning models do not appear in the new data to which we want to apply the models, causing the failure of the models on such new data. The unseen word/feature problem stems from the representations of words as symbolic items and the computation of the feature values as the discrete compositions of words. Such discrete nature implies the hard matches of feature appearance that is very likely to fail in the context of the new data.

This dissertation introduces a new approach for IE problems that is based on Deep Learning to address the three aforementioned problems of the feature-based models. Deep learning or artificial neural networks (NN) is a branch of machine learning whose major advantage is the capacity to automatically induce effective feature representations from data. In the deep learning models for IE, multiple layers of hidden vectors are put on top of word embeddings, the general representations of words that can capture their hidden syntactic and semantic properties. Word embeddings replace the hard matches of words in the feature-based approach with the soft matches of continuous word vectors while the multiple layers of hidden vectors further abstracts the word embeddings to automatically obtain underlying feature representations from data. Consequently, word embeddings mitigate the unseen word/feature problem of the feature-based models while the whole deep learning models help to avoid feature engineering and/or provide effective feature representations. To the best of my knowledge, this dissertation is among the first works that develop the fundamental deep learning models for information extraction.

In the next section of this chapter, I will first describe the IE problem in detail

and then review some background on machine learning and deep learning that is necessary for the next chapters. Afterward, I will present some of the representative previous work for IE. Finally, I will sketch an outline for this dissertation.

1.1 Information Extraction

Information Extraction is the process of extracting structured information from unstructured text (i.e, online news, government documents, social media text, medical alerts and records, etc.). The structured information is often organized in the table format of databases. In information extraction, we are mainly interested in two following types of databases:

- Relation Database: storing entities and the semantic relations (connections) between those entities.
- Event Database: recording events and the entities that participate in such events.

For this dissertation, we only focus on the entities, relations and events that are mentioned explicitly in text. The possible reasoning or inference over these objects at the database level should be considered as the next important steps and is beyond the scope of the current work.

Practically, given a set of documents, we want to distill the entities, relations and events of interest, and utilize them to populate relation and event databases. An illustration of this process is shown in Figure 1.1. The left side of this figure

presents a piece of unstructured text from some set of documents while the right side shows the relation and event databases we want to create from the left side's text. As we can see in the figure, the relation database contains the semantic relation "*leaderOf*" between Giuliani (the former mayor of New York City) and New York City (a city in the United States). The event database, on the other hand, includes a "*divorce*" event in July between Giuliani (again, the former mayor of New York City) and Donna Hanover (the second wife of Giuliani). All these information and facts are mentioned explicitly in the text of Figure 1.1 and our IE task is to automatically extract them.

The construction of relation and event databases from a set of documents can be divided into several tasks that together constitute the information extraction pipeline (Grishman, 2012). This pipeline involves the tasks of Entity Mention Detection, Relation Extraction, Coreference Resolution, Entity Linking, Trigger Prediction and Argument Prediction. The tasks and the flow of information in the information extraction pipeline is shown in Figure 1.2.

In the following, we always assume that documents have been split into sentences and sentences are already segmented into words. Sentence splitting and word segmentation are the preprocessing steps that can be done very well for English using the existing toolkits such as NTLK[1] or Stanford CoreNLP[2].

In order to describe the IE tasks in more details, let us consider the text in Figure 1.1 as an example:

1. `http://www.nltk.org`
2. `http://stanfordnlp.github.io/CoreNLP`

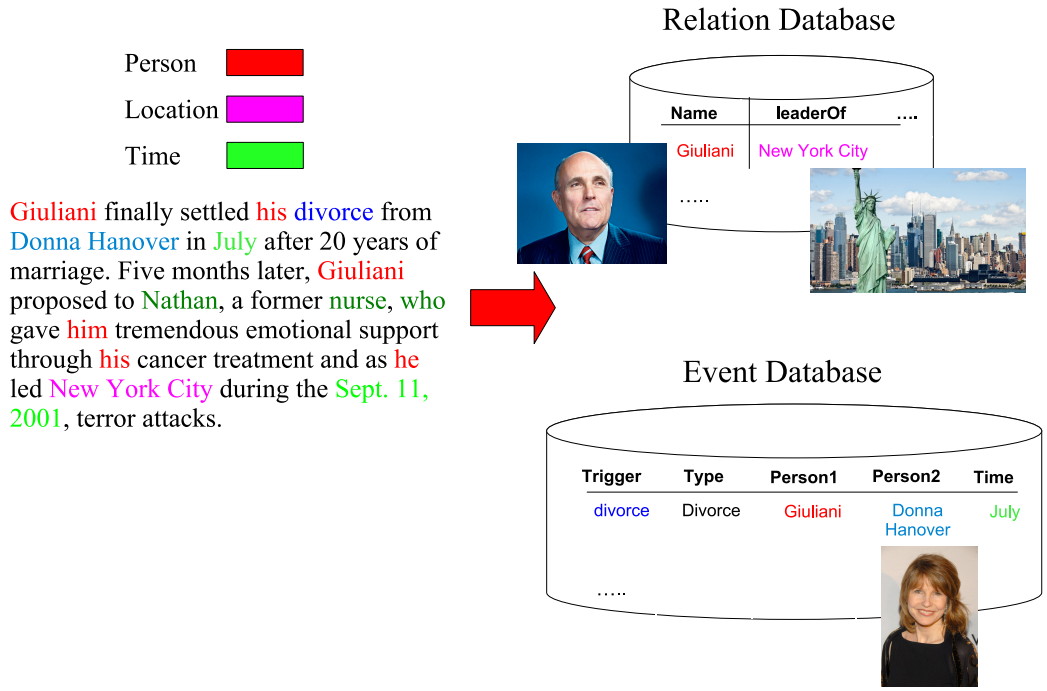

Figure 1.1: Information extraction system.

> "Giuliani finally settled his divorce from Donna Hanover in July after 20 years of marriage. Five months later, Giuliani proposed to Nathan, a former nurse, who gave him tremendous emotional support through his cancer treatment and as he led New York City during the Sept. 11, 2001, terror attacks."

1.1.1 Entity Mention Detection

In the IE pipeline, the first task in building the relation database is Entity Mention Detection (EMD) that locates and classifies entity mentions in text into predefined classes (types). Entity mentions are continuous sequences of words in

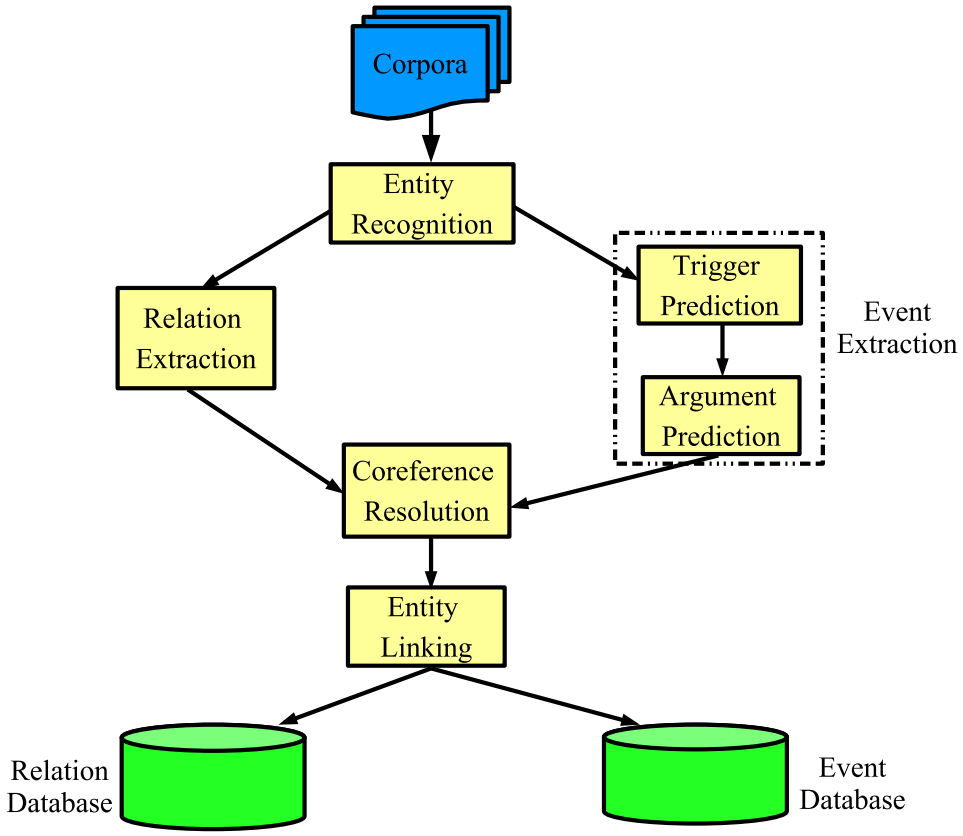

Figure 1.2: Information extraction pipeline.

the sentences that mention some objects (entities) in reality. Entity mentions can appear in various forms, including names, pronouns (i.e, "*he*", "*she*", "*it*", "*her*", "*who*", etc), and nominals (i.e, nouns, noun phrases, etc). In our text example above, for entity names, an EMD system should be able to recognize "*Giuliani*", "*Donna Hanover*" and "*Nathan*" as person names, "*New York City*" as a city name, and "*July*" and "*Sept. 11, 2011*" as times[3]. Besides, an EMD system should realize

3. In the applications, the detection of times is often done separately from the detection of other entity types, thus differentiating times from entity mentions in terms of concepts. However,

that the pronouns "*he*", "*his*", "*him*" and "*who*" as well as the nominal "*a former nurse*" are also entity mentions of some persons in this text. A reduced form of EMD is Named Entity Recognition where we only need to extract entity names (i.e, ignoring pronouns and nominals).

Note that for each entity mention, we often designate a single word (called the head word) that is most responsible for its meaning in the corresponding word sequence. For instance, the head word of the entity mention "*a former nurse*" is "*nurse*" while the head word for the pronoun entity mentions (i.e, "*he*", "*his*", etc) are the pronouns themselves. There is one exception for names as we consider their head words as comprising every word in their word sequences, thus possibly spanning several consecutive words in the sentences. "*Donna Hanover*" is an example for named entity mentions that have more than one head word (i.e, "*Donna*" and "*Hanover*"). In the applications, it is often sufficient for the EMD systems to recognize only the head words of the entity mentions.

1.1.2 Relation Extraction

Given the entity mentions from the previous step of EMD, in the next step of Relation Extraction, we want to detect and classify the semantic relationships between two entity mentions within in the same sentence. For convenience, we often call any pairs of entity mentions appearing in the same sentence as relation mentions. In our example text, a relation extraction system is expected to identify the relation of type "*leader of*" between the entity mention "*he*" (for a person)

in this dissertation, for convenience, we will consider times as a type of entity mentions unless a clarification is needed.

and the entity mention "*New York City*" (for a city) in the phrase "*as he led New York City during Sept. 11, 2001, terror attacks.*". Note that the relation classes of interest for classification (i.e, "*leader of*") are often given as an input by users.

1.1.3 Coreference Resolution

From the relation extraction component, we know that "*he*" is the leader of "*New York City*". However, who is "*he*" in this case? By looking at the previous context of the phrase "*as he led New York City during Sept. 11, 2001, terror attacks.*" in the example text, we know that the pronoun "*he*" is referring to the entity mention "*Giuliani*" appearing at the beginning of its sentence (i.e, "*Five months later, Giuliani proposed to Nathan …*"). This is essentially the task we want to perform in Coreference Resolution (i.e, recognizing the coreference of the pronoun "*he*" and the entity mention "Giuliani"). More generally, the goal of Coreference Resolution is to group entity mentions corresponding to the same entity in a document into the same cluster. We generate one cluster for each entity, forming a set of entity clusters for each document.

1.1.4 Entity Linking

The name "*Giuliani*" provides some identity information for the the pronouns "*he*", "*his*", and "*him*", and their corresponding entity cluster. However, it is not sufficient to determine unique person (entity) in reality that is necessary to find the entry for this entity in the database. This problem is due to our analysis of multiple documents that might contain several entities with the same name

"*Giuliani*". In order to overcome this problem, we need to do Entity Linking or Entity Disambiguation that seeks to link entity mentions or entities in documents into some real world entities. For instance, in our text example, an entity linking system should understand that "*Giuliani*" is the former mayor of New York City while "*Donna Hanover*" is his second wife based on the context of such entity mentions.

In practice, we often model the real world by some graph (called knowledge base) whose nodes represent real world entities and edges capture the connections among the entities. Each node of the knowledge base might contain different information about the corresponding entity. Wikipedia[4] is a typical example for such modeling effort. Given the knowledge base, Entity Linking is often defined as mapping entity mentions or entities in documents into the corresponding entities (nodes) of the knowledge base. The identities of the nodes (entities) in the knowledge base now serve as the identities for the entities and entity mentions in the input documents.

The combination of all the information gathered in the previous components (i.e, Entity Mention Detection, Relation Extraction, Coreference Resolution and Entity Linking) allows us to populate the relation database as we can see in Figure 1.1. In the next section, we discuss the necessary steps to fill in the event database.

Similar to the relation database, the first step for the event database is detecting entity mentions in the input documents (EMD). This is then followed by a new task of Event Extraction.

4. `https://en.wikipedia.org/wiki/Main_Page`

1.1.5 Event Extraction

An event such as marriage, death, election, etc, can be narrated several times in different sentences of a document. Each such sentence is called an event mention for that event and contains a trigger, the main word expressing the event. The event mentions of an event share the same event type and involve several arguments (i.e, entity mentions) as properties. These properties often concern the participants of the event mentions and their times and locations. For instance, according to the ACE 2005 guideline[5], the first sentence of the example text "*Giuliani finally settled his divorce from Donna Hanover in July after 20 years of marriage*" is an event mention of type "*Divorce*" whose trigger is "*divorce*" and arguments are "*Giuliani*" and "*Donna Hanover*" for the participating people and "*July*" for the time. Consequently, each event in a document is a cluster of coreferring event mentions with the same type while each sentence in a document might correspond to multiple event mentions (possibly from different events) that have different trigger words.

The Event Extraction task is to identify and classify the trigger words into some types of interest as well as locate arguments for the detected event types within the same sentence. Traditionally, event extraction systems have sequentially performed the two following steps: (i) recognizing event triggers in sentences (Trigger Prediction, Trigger Labeling or Event Detection), and (ii) assigning roles to the entity mentions as arguments for the recognized event triggers in step (i) (Argument Prediction or Argument Labeling).

5. https://www.ldc.upenn.edu/sites/www.ldc.upenn.edu/files/english-events-guidelines-v5.4.3.pdf

Finally, we also need to perform coreference resolution and entity linking over the entity mention arguments of the detected event mentions so we can group and link them to their real world entities (Figure 1.2). The resulting event mentions and linked entity mentions can be then inserted into the event knowledge base as in Figure 1.1. This concludes our description of the information extraction tasks. In the next section, I will present some background on machine learning and deep learning that is important to our following discussion on IE.

1.2 Machine Learning Background

Most of the IE tasks can be formulated as a classification problem over some appropriate objects or instances. In this problem, given a set of K predefined classes $\mathcal{Y}$ ($|\mathcal{Y}| = K$) and an object X, we need to choose a class $Y \in \mathcal{Y}$ that captures the nature of X. For instance, in relation extraction, the predefined classes $\mathcal{Y}$ involve the semantic relations of interest (i.e, "*leader of*", "*employed by*", etc) while the objects correspond to the relation mentions consisting of two target entity mentions within a sentence.

In order to solve this classification problem for IE, the first step is to design some function R to transform the initial input object X into some mathematical representation $R(X)$ that is more convenient for the mathematical analysis later. In practice, $R(X)$ is often a binary or continuous vector that is expected to involve the most representative and important features for the classification task. The design of the transformation function R is an art whose effectiveness is crucial to the classification performance.

Once the feature representation $R(X)$ has been computed, the next step is to use a function S to map $R(X)$ into some scores $S(R(X))$ whose values can be used to decide the class for the initial instance X in $\mathcal{Y}$. In most of the cases, $S(R(X))$ is a vector of real-valued numbers that assign a likelihood score for each class in $\mathcal{Y}$. The task of machine learning is to build the function S to effectively predict the classes for the input instances X.

For information extraction, the score function S is often parameterized by some parameter θ, written as $S(R(X), \theta)$ or simply as $S(X, \theta)$ if we do not want to emphasize the representation function R. This parameterization converts the problem of determining S into the problem of finding the suitable value for the parameter θ. In machine learning for information extraction, we often seek to find such value (called θ^*) by minimizing the expected risk:

$$\theta^* = \text{argmin}_\theta \mathbf{E}_{(X,Y)\sim P(X,Y)}[L(S(X,\theta), Y)] \tag{1.1}$$

In this formula, X and Y are random variables to denote the initial input instances and their corresponding *correct* classes in $\mathcal{Y}$ while $P(X, Y)$ represents the joint probability distribution over these two variables. In addition, $L(S(X, \theta), Y)$ is the cost function or the objective function that evaluates the loss of using $S(X, \theta)$ to determine the class for X (the predicted class) given that Y is the correct class for X in this case.

Unfortunately, the evaluation of the expectation in Equation 1.1 is often intractable as we need to enumerate over all the exponentially possible values for (X, Y). In practice, we can only try to obtain a finite set of samples D =

$\{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\}$ from $P(X, Y)$ (i.e, $|D| = n$ and $(X_i, Y_i) \sim P(X, Y)$ for all $i \in [1..n]$). D is also known as the training dataset in supervised learning. We can then use D to approximate the probability distribution $P(X, Y)$, leading to the empirical distribution $\hat{P}(X, Y)$ and the approximation of the expected risk by the empirical risk from the Monte Carlo sampling principle:

$$\begin{aligned} \mathbf{E}_{(X,Y)\sim P(X,Y)}[L(S(X,\theta),Y)] &\approx \mathbf{E}_{(X,Y)\sim \hat{P}(X,Y)}[L(S(X,\theta),Y)] \\ &= \frac{1}{n}\sum_{i=1}^{n} L(S(X_i,\theta),Y_i) \end{aligned} \tag{1.2}$$

Thus, θ^* can be computed by:

$$\begin{aligned} \theta^* &= \text{argmin}_\theta \mathbf{E}_{(X,Y)\sim P(X,Y)}[L(S(X,\theta),Y)] \\ &\approx \text{argmin}_\theta \frac{1}{n}\sum_{i=1}^{n} L(S(X_i,\theta),Y_i) \end{aligned} \tag{1.3}$$

In order to avoid overfitting to the training dataset D, we often add a regularizer $\Omega(\theta)$ to the right hand side of Equation 1.3 to penalize the large values of θ:

$$\theta^* \approx \text{argmin}_\theta \left[\frac{1}{n}\sum_{i=1}^{n} L(S(X_i,\theta),Y_i) + \lambda\Omega(\theta)\right] \tag{1.4}$$

where λ is a tradeoff between the loss on the training dataset D and the complexity of the model measured by $\Omega(\theta)$. The regularizer $\Omega(\theta)$ is often some norm of θ such as the L_1 norm $||\theta||_1$ or the L_2 norm $||\theta||_2$.

The optimization problem in Equation 1.4 can be solved by optimization techniques such as gradient descent, coordinate descent or Newton's methods (Good-

fellow et al., 2016). Once θ^* is obtained, we can combine the function (or model) $S(X, \theta^*)$ with our decision rule to predict the class for any new input instance X.

This framework for classification belongs to the class of the supervised learning methods in machine learning as we assume the correct labels Y_i for the input instances X_i in the training dataset D. Two important elements in this framework are the feature representation function $R(X)$ and the score function $S(R(X))$ that will significantly affect the performance of a classification model. The options for these two functions amount to different classification models for information extraction. Note that the score function $S(R(X))$ is often associated with a decision rule to infer the class for the input instances X.

There are two possible ways to form the representation function $R(X)$, i.e, *feature engineering* and *representation learning*. In feature engineering, $R(X)$ is manually designed by the domain experts who rely on their domain knowledge and linguistic intuition to specify the most important characteristics or features for some IE task. The researchers then determine the toolkits, resources and mechanisms to compute such characteristics for the input instances X. Features in this approach often have binary values to indicate the presence or absence of some discrete linguistic structures (i.e, the appearance of some word in some gazetteer, the occurrence of some word or syntactic relation in the context, etc).

Feature engineering allows the incorporation of our intuition (for the classification problem) into the models, generating highly interpretable classification models. These advantages cause the prevalence of feature engineering to IE tasks that has significantly advanced our performance for such tasks in the last decade. However,

as we discussed at the beginning of this chapter, feature engineering has several limitations due to its use of binary features and manual construction of feature sets. Such limitations are the major motivation of this dissertation that explores the representation learning approach to generate the feature representation $R(X)$. Representation learning uses neural networks to automatically induce $R(X)$ from data and mitigates the unseen word/feature problem of binary features. In the following, I will review methods to build the score function $S(X)$ in the feature engineering approach as well as present some background on neural networks to facilitate our discussion later.

1.2.1 Feature Engineering

Once the feature representation vector $R(X)$ has been hand-designed and computed for the input instances X, we can apply different methods to model the score function $S(R(X))$. The two most popular methods for $S(R(X))$ in information extraction are Maximum Entropy and Support Vector Machines (SVM). For convenience, we denote d as the size of the vector $R(X)$ ($|R(X)| = d$). Note that this size d is fixed for all the representation vectors $R(X)$ of all the possible input instances X.

1.2.1.1 Maximum Entropy

In Maximum Entropy (Kambhatla, 2004), we parameterize $S(R(X))$ by a parameter matrix B ($B \in \mathbb{R}^{d \times K}$) to assign importance weights for the features (elements) in the representation vector $R(X)$ with respect to different possible classes

in $\mathcal{Y}$ ($|\mathcal{Y}| = K$), i.e, the i-th column of B corresponds to the feature weights for the i-th class in $\mathcal{Y}$. The product $B^T R(X)$ is then added by a bias vector b ($b \in \mathbb{R}^K$) to obtain the likelihood vector[6] $A = B^T R(X) + b$ of size K for every class in $\mathcal{Y}$. Finally, we normalize the likelihoods in $A = [a_1, a_2, \ldots, a_K]$ via the *softmax* function to obtain a probability distribution over the classes in $\mathcal{Y}$, severing as our score function $S(R(X))$ in this method:

$$S(R(X)) = S(X, \theta) = softmax(A) = \left[\frac{e^{a_1}}{Z}, \frac{e^{a_2}}{Z}, \ldots, \frac{e^{a_K}}{Z}\right] \tag{1.5}$$

where Z is a normalizing constant:

$$Z = \sum_{i=1}^{K} e^{a_i} \tag{1.6}$$

The parameter we need to learn in this case is $\theta = [B, b]$ while the loss function $L(S(X, \theta), Y)$ is often the negative log-likelihood:

$$L(S(X, \theta), Y) = -\log S(X, \theta)[Y] = -\log\left[\frac{e^{a_Y}}{\sum_{i=1}^{K} e^{a_i}}\right] \tag{1.7}$$

The class Y^* for a new instance X is determined by the class with the highest score in $S(X, \theta)$:

$$Y^* = \text{argmax} S(X, \theta) \tag{1.8}$$

6. We assume $R(X)$ and b are column vectors in this case for convenience

1.2.1.2 Support Vector Machines

For simplicity, we assume that there are only two classes, denoted by -1 and 1, in $\mathcal{Y}$ in this case (i.e, $\mathcal{Y} = \{-1, 1\}$). The extension from the binary classification setting to the multiple class setting (i.e, $|\mathcal{Y}| > 2$) for Support Vector Machines (SVM) (Cristianini and Taylor, 2000) can be done by considering multiple ($|\mathcal{Y}|$) binary classification problems. Each of such problem corresponds to a class in $\mathcal{Y}$ that tries to predict whether an input instance X belongs to that class or not (Cristianini and Taylor, 2000).

SVM considers each input instance X as one point in the d-dimensional space defined by its vector $R(X)$. In this space, the goal of SVM is to find a hyperplane that divides the groups of training instances X_i with $Y_i = 1$ and the groups of training instances X_i with $Y_i = -1$. As there might be multiple satisfying hyperplanes, SVM seeks to find two parallel hyperplanes that separate the instances in the training data D and have the largest distance between them. The final hyperplane of SVM is then the hyperplane that stands in the middle of the such two hyperplanes. This process translates into the score function $S(R(X))$ that are parameterized by a weight vector B ($|B| = |R(X)|$) and a bias b ($\theta = [B, b]$):

$$S(R(X)) = B^T R(X) - b \tag{1.9}$$

The loss function in this case is the hinge loss function:

$$L(S(X, \theta), Y) = \max(0, 1 - Y(B^T R(X) - b)) \tag{1.10}$$

Finally, the decision rule for the prediction class Y^* is:

$$Y^* = \text{sgn}(S(R(X))) = \text{sgn}(B^T R(X) - b) \tag{1.11}$$

where *sgn* is the sign function so that $\text{sgn}(x) = 1$ if $x \geq 0$ and $\text{sgn}(x) = -1$ if $x < 0$.

We can replace $L(S(X, \theta))$ in Equation 1.4 with that in Equation 1.10 to obtain the optimization problem for the *soft-margin* version of SVM. As this version of SVM aims at learning a hyperplane, it is only suitable for the problems where the two classes of data can be approximately separated well by hyperplanes (linearly separable).

In order to deal with nonlinear separation, we need to incorporate the kernel trick into SVM (Cristianini and Taylor, 2000). The general idea is to map the input instances from the original space for $R(X)$ (called $\mathcal{X}$) into another space $\mathcal{V}$ by some nonlinear transformation so that they become linearly separable in $\mathcal{V}$. We can then apply our original SVM algorithm in the new space. In stead of explicitly mapping the instances in the original space, the kernel trick suggests that we only need to build a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ to compute a score for every pair of instances (X, X') in $\mathcal{X}$. The expectation is that this score will correspond to the dot product between the images of X and X' in the new space $\mathcal{V}$, and that the kernel function can implicitly capture the nonlinear transformation. Consequently, we only need to replace all the dot product between two instances X and X' in the linear SVM algorithm[7] with the kernel function $k(X, X')$ so that we can convert the linear SVM algorithm into a nonlinear algorithm for classification. In general,

7. The dot products will naturally appear when we form the dual problem to solve the optimization problem in Equation 1.4 for SVM.

the construction of the kernel function k is much easier than the formation of the direct mapping, leading to the popularity of the kernel methods for nonlinear classification in SVM.

1.2.2 Representation Learning

In representation learning, we aim at automatically inducing the representation function $R(X)$ from data. The major tools for such autonomous feature learning is neural networks (NN) or deep learning. This section reviews feed-forward neural networks, the most basic network architecture for the deep learning models in this dissertation.

1.2.2.1 Feed-forward Neural Networks

The first important element in NN is the input instances X that should be in the form of real-valued vectors, matrices or tensors. In information extraction problems, the input instances X are not readily presented in this format as they often involve sentences, sequences of discrete symbols for words. I will show how to convert the discrete instances X in IE into the continuos tensors when we discuss the specific IE problems. For convenience, in this section, we will assume that the input instances X are already given as real-valued column vectors.

In order to learn $R(X)$ for X, feed-forward neural networks use a stack of multiple hidden layers that connected to each other via linear and nonlinear transformations. Each hidden layer is expected to capture some representation at some abstract level of the input X. The deeper the hidden layer is, the more abstract

the representation is. Formally, let m be the number of hidden layers in some feed-forward network, H_i be the the hidden vector at the i-th layer ($1 \leq i \leq m$) and l_i be the dimensionality of the column vector H_i. The elements of the hidden vectors are called the hidden units in the literature. The computation of H_i and $R(X)$ in the feed-forward neural network is then given by:

$$H_{i+1} = g(U_{i+1}^T H_i + V_{i+1}) \text{ for } i = 0 \text{ to } m \tag{1.12}$$

where H_0 is the input vector X, H_{m+1} is the representation vector $R(X)$ we have learned (i.e, $H_0 = X$ and $H_{m+1} = R(X)$), and $U_i \in \mathbb{R}^{l_i \times l_{i+1}}$ and $V_i \in \mathbb{R}^{l_{i+1}}$ are the parameters to be optimized from data. In this case, $l_0 = d$ is the dimensionality of the input X while l_{m+1} is the expected dimensionality of $R(X)$ that should be chosen in advance as a hyperparameter. Finally, g is a differential and nonlinear function whose application on a vector amounts to the applications on each element of that vector (i.e, element-wise):

$$g([v_1, v_2, \ldots, v_h]) = [g(v_1), g(v_2), \ldots, g(v_h)] \tag{1.13}$$

Some typical nonlinear functions for deep learning include *sigmoid*, *tanh* and *rectifier*:

$$sigmoid(t) = \frac{1}{1 + e^{-t}} \tag{1.14}$$

$$tanh(t) = \frac{1 - e^{-2t}}{1 + e^{-2t}} \tag{1.15}$$

$$\textit{rectifier}(t) = \max(0, t) \tag{1.16}$$

A graphical illustration of a feed-forward natural network is shown in Figure 1.3. As we can see from the figure, the hidden units of the i-th layer are fully connected to those of the $(i + 1)$-th layer, forming a flow of information as presented in Equation 1.12. This direction of the information flow from X to $R(X)$ is called a *forward* pass to differentiate it from a *backward* pass (from $R(X)$ to X) that updates the parameters later.

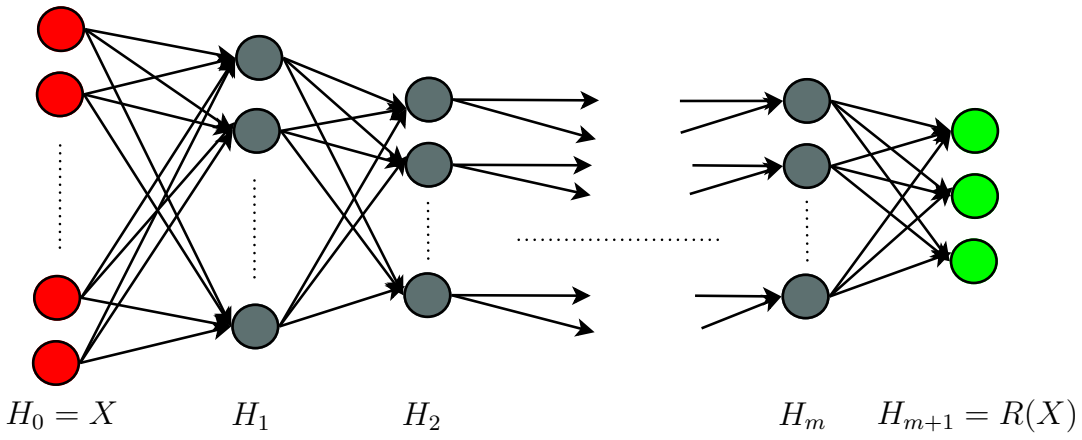

Figure 1.3: Feed-forward Neural Networks.

1.2.2.2 Training

From our previous description, feed-forward neural networks learn $R(X)$ for X by treating $R(X)$ as a parameterized function of X using the parameter $\gamma = [U_1, V_1, U_2, V_2, \ldots, U_{m+1}, V_{m+1}]$ and the network architecture in Figure 1.3. In feed-forward neural networks, and more generally, in deep learning, γ will be optimized

jointly with the parameter θ of the score function $S(R(X))$ by solving Equation 1.4 over the training data D. Eventually, it helps to capture the underlying representation of data via optimization, potentially introducing useful features for the classification tasks for such data. Unfortunately, this joint optimization creates much more complicated functions in Equation 1.4 than those in the feature engineering method. In particular, the joint function of γ and θ in representation learning is often non-convex and contain many local minima while their counterparts in feature engineering are convex and involve only global minima. In addition, the joint optimization requires to search in a much larger space for the optimal values of parameters, making it a much more expensive computation (Goodfellow et al., 2016).

Fortunately, it is now believed and demonstrated in practice that the local minima in representation learning often corresponds to the points with small values of the optimization functions in Equation 1.4 (Goodfellow et al., 2016). Consequently, searching for a local minima often provides a practically good solution. Regarding the computation expense, the use of *Graphical Processing Unit* (GPU) allows us to complete the operations in representation learning (deep learning) much faster via the use of massively parallel graphics processors. This has significantly accelerated the training process and made it possible to run representation learning models recently.

In addition to the use of GPU, deep learning models seek to reduce the computation time by solving the optimization problem in Equation 1.4 approximately. In particular, rather than using the whole training data D to evaluate the opti-

mization function at a time, deep learning randomly initializes the parameters, and then iteratively consumes a small subset of D and updates the parameters based on that subset. Typically, an update is estimated from the gradient of the objective function (gradient descent). This online learning method is known as stochastic gradient descent (SGD) and the small subsets of D are called batches or minibatches. The size of a batch is often fixed during training. The SGD algorithm for learning parameters (training) is shown in Algorithm 1:

Algorithm 1: Stochastic Gradient Descent for Learning Parameters

1 Initialize the parameters in representation learning $\psi = [\gamma, \theta]$ randomly
while *stopping criterion not met* **do**
2 Sample a mini batch of J training instances from the training dataset D: $\{(X^1, Y^1), (X^2, Y^2), \ldots, (X^J, Y^J)\}$
3 Compute the gradient G for the optimization function in Equation 1.4 over the sampled minibatch: $G \leftarrow \frac{1}{J} \nabla_\psi \left[\sum_{i=1}^{J} L(S(X^i, \psi), Y^i) + \lambda \Omega(\psi) \right]$
4 Compute update: $\Delta\psi = -\epsilon \odot G$ (element-wise multiplication of vectors)
5 Apply update: $\psi \leftarrow \psi + \Delta\psi$
6 **end**

The parameter ϵ is called *the learning rate* that determines the size of the step we move with each update. In SDG, ϵ is fixed during the training process and often chosen via some development dataset[8]. In practice, fixing the learning rate might not work very well as it might slow down the convergence of SGD (i.e, long training

8. Development dataset is similar to the training dataset D as both of them include a set of input instances X and their corresponding classes Y. However, the development dataset is disjoint with the the training dataset. The former is used to evaluate the effectiveness of the hyperparameters in the models (i.e, number of hidden layers m, and their dimensionality l_i, regularizer weight λ, etc) while the latter is employed to estimate the parameters in the representation and score functions (i.e $R(X)$ and $S(R(X))$). The hyperparameters are typically selected by taking the values the produces the highest model performance on the development dataset.

time) or render a poor approximation of the optimal values for the parameters ψ. There have been several efforts to update the learning rate ϵ during the training process so that the convergence is achieved faster or the solution is more effective. These methods include *AdaGrad* (Duchi et al., 2011), *AdaDelta* (Zeiler, 2012) and *Adam* (Kingma and Ba, 2014).

1.2.2.3 Dropout

Deep learning models with multiple hidden layers can learn complicated relationships between the input X and the class output Y via the representation and score functions $R(X)$ and $S(R(X))$. However, they tend to overfit the training data if such data only has a limited size. The consequence is the poor performance on real test data as the relationships induced from the small training dataset are likely due to the sampling noise. There have been many approaches to address the overfitting problem in machine learning. One of the common methods is to introduce parameter penalties into the objective function as in Equation 1.4. In deep learning, a more common method to overcome overfitting is *dropout* that randomly drops (i.e, sets to zero) units along with their connections in the network architecture during training. The key idea is to prevent the co-adaptation among the units (Srivastava et al., 2014). Formally, consider the hidden vector H_i in the feed-forward neural network above. In order to inject dropout into H_i, we first sample a binary vector r_i of the same size with H_i (i.e, $|r_i| = |H_i|$) from the Bernoulli distribution with the mean p:

$$r_i \sim \text{Bernoulli}(p) \tag{1.17}$$

This is then combined with H_i to generate a new hidden vector $\widetilde{H}_i$:

$$\widetilde{H}_i = r_i \odot H_i \tag{1.18}$$

where $\odot$ is the element-wise multiplication operation.

For dropout, H_i is replaced by $\widetilde{H}_i$ in all the following computations in the neural network models. The decision of which hidden layers should be dropped depends on network architectures and the target problems (Zaremba et al., 2014). In information extraction, we often find that dropping the representation vector (i.e, the last layer $R(X)$ in the network architectures) produces good performance.

1.3 Prior Work

In this section, I review the most representative work for information extraction that are related to the work in this dissertation. I will only focus on the feature engineering approach in this section and leave the related work on representation learning for discussion in the later chapters.

Information extraction has been an active area of research in natural language processing. A large portion of the previous research effort has been spent on developing effective feature sets for different tasks of IE, such as named entity recognition, relation extraction, event extraction. For instance, the investigated features for named entity recognition and mention detection include the orthographic fea-

tures, gazetter features, cache features, word clusters, word embeddings etc (Ando and Zhang, 2005; Bikel et al., 1997; Borthwick et al., 1997; Cherry and Guo, 2015; Florian et al., 2003; Florian et al., 2004; Florian et al., 2006, 2010; Lin and Wu, 2009; Miller et al., 2004; Passos et al., 2014; Ratinov and Roth, 2009; Ritter et al., 2011; Sam et al., 2011; Sang and Meulder, 2003; Suzuki and Isozaki, 2008; Turian et al., 2010). The typical machine learning models for such sequential labeling tasks are Hidden Markov Models (HMMs), Maximum Entropy Markov Models (MEMMs) or Conditional Random Fields (CRFs) (Lafferty et al., 2001). For relation extraction, it is very common to use the Maximum Entropy (MaxEnt) model or SVM (Cristianini and Taylor, 2000) to learn the weights for various hand-designed features, including the entity features, syntactic features, and semantic features (Chan and Roth, 2010; Grishman et al., 2005; Jiang and Zhai, 2007a; Kambhatla, 2004; Nguyen et al., 2014b; Sun et al., 2011; Zhou et al., 2005). This is also the main approach for other IE tasks such as event extraction (Ahn, 2006; Grishman et al., 2005; Gupta and Ji, 2009; Hong et al., 2011; Huang and Riloff, 2012; Ji and Grishman, 2008; Li et al., 2015; Liao and Grishman, 2010a, 2011; McClosky et al., 2011; Patwardhan and Rilof, 2009), entity linking (Bunescu and Pasca, 2006; Cassidy et al., 2011; Ji and Grishman, 2011; Mendes et al., 2011; Milne and Witten, 2008; Shen et al., 2014b; Zheng et al., 2010) or coreference resolution (Clark and Manning, 2016; Durrett and Klein, 2013; Raghunathan et al., 2010; Wiseman et al., 2015). In addition, kernel tricks are also applied in SVM for information extraction to avoid explicit feature engineering. For example, string kernels (Bunescu and Mooney, 2005b) and tree kernels (Bunescu and Mooney,

2005a; Nguyen et al., 2009; Plank and Moschitti, 2013; Qian et al., 2008; Zelenko et al., 2003; Zhang et al., 2006) are the powerful techniques for relation extraction.

Another important line of research in information extraction is joint modeling that attempts to perform several IE tasks simultaneously (joint inference). The rationale is to capture the inter-dependencies among these tasks to improve the performance of the individual tasks. In the literature, the inter-dependences are often exploited between trigger prediction and argument prediction in event extraction (Li et al., 2013b), entity mention detection and relation extraction (Kate and Mooney, 2010; Li and Ji, 2014a; Li et al., 2014b; Miwa et al., 2014; Roth and Yih, 2004; Roth and Yih, 2007), and among name tagging, coreference resolution and relation extraction (Ji and Grishman, 2005; Ji et al., 2005; Singh et al., 2013). All of these work involves some level of feature engineering that are fed into structured prediction models (i.e, structured perceptron, graphical models, etc.) to perform the tasks.

So far, we have mainly discussed the IE work in the supervised learning paradigm that assumes the availability of a vast amount of training data (labeled data). This assumption does not hold often in reality as the training data for some tasks and domains might be very expensive and difficult to obtain. How can we relax this assumption so we can build good models for IE without requiring much training data? This is the key target of semi-supervised learning that employs large amounts of unlabeled data in addition to limited amounts of labeled data to build effective IE systems. The semi-supervised models have been applied to various IE tasks, including named entity recognition via co-training (Collins and Singer,

1999; Yangarber et al., 2002), relation extraction (Agichtein and Gravano, 2000; Brin, 1998), and event extraction via boostrapting (Liao and Grishman, 2010a; Riloff, 1996; Stevenson and Greenwood, 2005; Yangarber et al., 2000; Yangarber, 2003). In order to go further, unsupervised learning aims at removing the need for training data and only relying on unlabeled data to create IE systems. The main technologies in such unsupervised approach are unsupervised clustering or topic modeling methods to identify the major patterns for IE (Hasegawa et al., 2004; Min et al., 2012; Shinyama and Sekine, 2006; Yao et al., 2011; Yates and Etzioni, 2007). In addition, distant supervision is an alternative to avoid the need for a training dataset. It aligns the facts in some knowledge bases (i.e, Freebase, YAGO, etc.) with a large amount of unlabeled text to automatically generate large training dataset for IE models. Distant supervision is especially successful in relation extraction; that has been one of the main breakthroughs in IE over the last decades (Craven and Kumlien, 1999; Hoffmann et al., 2011; Mintz et al., 2009; Surdeanu et al., 2012). Finally, there have been also studies in active learning that try to request classes or labels of some input instances from users and use these labeled data to guide the learning process. The goal of active learning is to minimize the number of requests from users so an effective IE system can be built quickly without much annotation effort (Becker et al., 2005; Fu and Grishman, 2013; Sun and Grishman, 2012).

Semi-supervised learning, distant supervision and unsupervised learning are beyond the scope of this work that focuses on supervised learning and deep learning for information extraction. Exploring such learning paradigms for IE with deep

learning is a very promising research area in the future.

1.4 Outline of Thesis

The rest of this dissertation is organized as follows: Chapter 2 introduces the use of word embeddings for domain adaptation of relation extraction. Chapters 3, 4 and 5 develop deep learning models for entity mention detection, relation extraction and event detection respectively. Chapter 6 proposes memory-augmented neural networks for information extraction and demonstrate their applications on event extraction and entity linking. I conclude this dissertation and discuss some future work in Chapter 7.

Chapter 2

Word Embeddings for Domain Adaptation of Relation Extraction

The previous research on supervised learning has mainly approached relation extraction (RE) in two directions: feature-based (Boschee et al., 2005; Chan and Roth, 2010; Grishman et al., 2005; Jiang and Zhai, 2007a; Kambhatla, 2004; Sun et al., 2011; Zhou et al., 2005) and kernel-based (Bunescu and Mooney, 2005a,b; Nguyen et al., 2009; Qian et al., 2008; Zelenko et al., 2003; Zhang et al., 2006). Both approaches attempt to improve performance by enriching the RE representations from multiple sentence analyses and knowledge resources. The fundamental assumption of the supervised systems in such research is that the training data and the data to which the systems are applied are sampled independently and identically from the same distribution. When there is a mismatch between the data distributions (domain shifts), the RE performance of these systems tends to degrade dramatically (Plank and Moschitti, 2013). This is where we need to resort

to domain adaptation techniques (DA) to adapt a model trained on one domain (the source domain) into a new model which can perform well on new domains (the target domains). To make it clear, we assume the same relation classes (types), thus the same extraction task in both source and target domains but a shift in the underlying data distributions.

This chapters introduce methods to incorporate word embeddings into models as domain adaptation techniques for RE. We will examine both approaches (i.e, feature-based and kernel-based) to demonstrate the benefits of word embeddings for DA of RE. Note that we actually design features for DA of RE based on word embeddings in this chapter. However, as word embeddings are automatically induced by deep learning models on some unlabeled corpus, we would still consider the techniques in this chapter as semi-automatically learning feature representations from data. The work in this chapter has been published in (Nguyen and Grishman, 2014a) and (Nguyen et al., 2015c).

2.1 The Feature-based Approach

The consequences of linguistic variation between training and testing data on NLP tools (domain shifts) have been studied extensively in the last couple of years for various NLP tasks such as Part-of-Speech tagging (Blitzer et al., 2006; Huang and Yates, 2010; Schnabel and Schütze, 2014), named entity recognition (Daume, 2007) and sentiment analysis (Blitzer et al., 2007, 2011; Daume, 2007; Daume et al., 2010), etc. Unfortunately, there is very little work on domain adaptation for RE. The only study explicitly targeting this problem before the current work is by Plank

and Moschitti, 2013 who find that the out-of-domain performance of kernel-based relation extractors can be improved by embedding semantic similarity information generated from word clustering and latent semantic analysis (LSA) into syntactic tree kernels. Although this idea is interesting, it suffers from two major limitations:

1. It does not incorporate word cluster information at different levels of granularity. In fact, Plank and Moschitti, 2013 only use the 10-bit cluster prefix in their study. We will demonstrate later that the adaptability of relation extractors can benefit significantly from the addition of word cluster features at various granularities.

2. It is unclear if this approach can encode real-valued features of words (such as word embeddings (Collobert and Westion, 2008; Mnih and Hinton, 2007)) into the syntactic trees effectively. As the real-valued features are able to capture latent yet useful properties of words, the augmentation of lexical terms with these features is desirable to provide a more general representation, potentially helping relation extractors perform more robustly across domains.

In this work, we propose to avoid these limitations by applying a feature-based approach for RE which allows us to integrate various word features of generalization into a single system more naturally and effectively.

The application of word representations such as word clusters in domain adaptation of RE (Plank and Moschitti, 2013) is motivated by its successes in semi-supervised methods (Chan and Roth, 2010; Sun et al., 2011) where word representations help to reduce data-sparseness of lexical information in the training data. In DA terms, since the vocabularies of the source and target domains are usually

different, word representations would mitigate the lexical sparsity by providing general features of words that are shared across domains, hence bridge the gap between domains. The underlying hypothesis here is that the absence of lexical target-domain features in the source domain can be compensated by these general features to improve RE performance on the target domains.

We extend this motivation by further introducing word embeddings (Bengio et al., 2003; Collobert and Westion, 2008; Mnih and Hinton, 2007; Turian et al., 2010) into feature-based methods to adapt RE systems to new domains. Word embedding is another type of word representations assisting generalization of terms that do not appear in the training data, but are similar to those in training data with respect to their distributed representations (Bengio et al., 2003). We explore the embedding-based features in a principled way and demonstrate that word embedding itself is also an effective representation for domain adaptation of RE. More importantly, we show empirically that word embeddings and word clusters capture different information and their combination would further improve the adaptability of relation extractors.

2.1.1 Regularization

Given the more general representations provided by word representations above, how can we learn a relation extractor from the labeled source domain data that generalizes well to new domains? In traditional machine learning where the challenge is to utilize the training data to make predictions on unseen data points (generated from the same distribution as the training data), the classifier with a

good generalization performance is the one that not only fits the training data, but also avoids overfitting over it. This is often obtained via regularization methods to penalize complexity of classifiers. Exploiting the shared interest in generalization performance with traditional machine learning, in domain adaptation for RE, we would prefer the relation extractor that fits the source domain data, but also circumvents the overfitting problem over this source domain[1] so that it could generalize well to new domains. Eventually, regularization methods can be considered naturally as a simple yet general technique to cope with DA problems.

To our knowledge, there have not been any studies assessing the impact of regularization on RE in general and on domain adaptation of RE specifically. It is also worth pointing out that some studies in this area do not apply regularization in their models. For example, Sun et al., 2011 use the MaxEnt package of OpenNLP[2] which did not support regularization.

Following Plank and Moschitti, 2013, we assume that we only have labeled data in a single source domain but no labeled nor unlabeled target data. Moreover, we consider the single-system DA setting where we construct a single system able to work robustly with different but related domains (multiple target domains). This setting differs from most previous studies (Blitzer et al., 2006) on DA which have attempted to design a specialized system for every specific target domain. In our view, although this setting is more challenging, it is more practical for RE. In fact, this setting can benefit considerably from our general approach of applying word representations and regularization. Finally, due to this setup, the best way to set

1. *domain overfitting* (Jiang and Zhai, 2007c)
2. `http://opennlp.apache.org`

up the regularization parameter is to impose the same regularization parameter on every feature rather than a skewed regularization (Jiang and Zhai, 2007c).

2.1.2 Word Representations

Word representations are high-dimensional vectors that are associated with words in an unlabeled corpus. In this work, we consider two types of word representations and use them as additional features in our DA system, namely Brown word clustering (Brown et al., 1992) and word embeddings. While word clusters can be recognized as an one-hot vector representation over a small vocabulary, word embeddings are dense, low-dimensional, and real-valued vectors (distributed representations). Each dimension of the word embeddings expresses a latent feature of the words, hopefully reflecting useful semantic and syntactic regularities (Turian et al., 2010). We investigate word embeddings induced by two typical language models: Collobert and Weston (2008) embeddings (C&W) (Collobert and Westion, 2008; Turian et al., 2010) and Hierarchical log-bilinear embeddings (HLBL) (Mnih and Hinton, 2007, 2008; Turian et al., 2010).

2.1.3 Feature Set

2.1.3.1 Baseline Feature Set

Sun et al., 2011 utilize the full feature set from (Zhou et al., 2005) plus some additional features and achieve the state-of-the-art feature-based RE system. Unfortunately, this feature set includes the *human-annotated* (gold-standard) information on entity and mention types which is often missing or noisy in the test

time (Plank and Moschitti, 2013). This issue becomes more serious in our setting of single-system DA where we have a single source domain with multiple dissimilar target domains, and an automatic system able to recognize entity and mention types very well in different domains may not be available. Therefore, following the settings of (Plank and Moschitti, 2013), we will only assume entity boundaries and not rely on the gold standard information in the experiments. We apply the same feature set as (Sun et al., 2011) but remove the entity and mention type information[3]. Clearly, evaluating the system on predicted mentions (Giuliano et al., 2007) is also an important topic but out of the scope of this work.

2.1.3.2 Lexical Feature Augmentation

While Sun et al., 2011 show that adding word clusters to the heads of the two mentions is the most effective way to improve the generalization accuracy, the right lexical features into which word embeddings should be introduced to obtain the best adaptability improvement are unexplored. Also, which dimensionality of which word embedding should we use with which lexical features? In order to answer these questions, following (Sun et al., 2011), we first group lexical features into 4 groups and rank their importance based on linguistic intuition and illustrations of the contributions of different lexical features from various feature-based RE systems. After that, we evaluate the effectiveness of these lexical feature groups for word embedding augmentation individually and incrementally according to the

3. We have the same observation as (Plank and Moschitti, 2013) that when the gold-standard labels are used, the impact of word representations is limited since the gold-standard information seems to dominate. However, whenever the gold labels are not available or inaccurate, the word representations would be useful for improving adaptability performance. Moreover, in all the cases, regularization methods are still effective for domain adaptation of RE.

rank of importance. For each of these group combinations, we assess the system performance with different numbers of dimensions for both C&W and HLBL word embeddings. Let M1 and M2 be the first and second entity mentions in the relation mention. Table 2.1 describes the lexical feature groups.

Rank	Group	Lexical Features
1	**HM**	HM1 (head of M1)
		HM2 (head of M2)
2	**BagWM**	WM1 (words in M1)
		WM2 (words in M2)
3	**HC**	heads of chunks between M1 and M2
4	**BagWC**	words between M1 and M2

Table 2.1: Lexical feature groups ordered by importance.

2.1.4 Experiments

2.1.4.1 Tools and Data

Our relation extraction system is hierarchical and includes a relation detector as well as a relation classifier (Bunescu and Mooney, 2005b; Sun et al., 2011). We use maximum entropy (MaxEnt) in the MALLET[4] toolkit as our machine learning tool. For Brown word clusters, we directly apply the clustering trained by (Plank and Moschitti, 2013) to facilitate system comparison later. We evaluate C&W word embeddings with 25, 50 and 100 dimensions as well as HLBL word embeddings with 50 and 100 dimensions that are introduced in (Turian et al., 2010) and can be downloaded here[5]. The fact that we utilize the large, general and unbiased

4. http://mallet.cs.umass.edu/
5. http://metaoptimize.com/projects/wordreprs

resources generated from previous work for evaluation not only helps to verify the effectiveness of the resources across different tasks and settings but also supports our setting of single-system DA.

We use the ACE 2005 corpus for DA experiments (as in (Plank and Moschitti, 2013)). It involves 6 relation types and 6 domains: broadcast news (`bn`), newswire (`nw`), broadcast conversation (`bc`), telephone conversation (`cts`), weblogs (`wl`) and usenet (`un`). We follow the standard practices on ACE (Plank and Moschitti, 2013) and use `news` (the union of `bn` and `nw`) as the source domain and `bc`, `cts` and `wl` as our target domains. We take half of `bc` as the only target development set, and use the remaining data and domains for testing purposes (as they are small already). The domain `un` is not considered in the experiments. As noted in (Plank and Moschitti, 2013), the distributions of relations as well as the vocabularies of the domains are quite different.

2.1.4.2 Evaluation of Word Embedding Features

We investigate the effectiveness of word embeddings on lexical features by following the procedure described in Section 2.1.3.2. We test our system on two scenarios: In-domain: the system is trained and evaluated on the source domain (`bn+nw`, 5-fold cross validation); Out-of-domain: the system is trained on the source domain and evaluated on the target development set of `bc` (`bc` dev). Table 2.2 presents the F measures of this experiment[6] (The cells in bold are the best results, and the suffix `ED` in lexical group names is to indicate the embedding features).

6. All the in-domain improvement in rows 2, 6, 7 of Table 2.2 are significant at confidence levels $\geq 95\%$.

	System	In-domain (bn+nw)					Out-of-domain (bc development set)				
		C&W,25	C&W,50	C&W,100	HLBL,50	HLBL,100	C&W,25	C&W,50	C&W,100	HLBL,50	HLBL,100
1	Baseline	51.4	51.4	51.4	51.4	51.4	49.0	49.0	49.0	49.0	49.0
2	1+HM_ED	54.0(+2.6)	54.1(+2.7)	**55.7(+4.3)**	53.7(+2.3)	55.2(+3.8)	51.5(+2.5)	**52.7(+3.7)**	52.5(+3.5)	50.2(+1.2)	50.6(+1.6)
3	1+BagWM_ED	52.3(+0.9)	50.9(-0.5)	51.5(+0.1)	51.8(+0.4)	52.5(+1.1)	48.5(-0.5)	48.9(-0.1)	48.6(-0.4)	48.7(-0.3)	49.0(+0.0)
4	1+HC_ED	51.3(-0.1)	50.9(-0.5)	48.3(-3.1)	50.8(-0.6)	49.8(-1.6)	44.9(-4.1)	45.8(-3.2)	45.8(-3.2)	48.7(-0.3)	47.3(-1.7)
5	1+BagWC_ED	51.5(+0.1)	50.8(-0.6)	49.5(-1.9)	51.4(+0.0)	50.3(-1.1)	48.3(-0.7)	46.3(-2.7)	44.0(-5.0)	46.6(-2.4)	44.8(-4.2)
6	2+BagWM_ED	54.3(+2.9)	53.2(+1.8)	53.2(+1.8)	54.0(+2.6)	53.8(+2.4)	52.5(+3.5)	51.4(+2.4)	50.6(+1.6)	50.0(+1.0)	48.6(-0.4)
7	6+HC_ED	53.4(+2.0)	52.3(+0.9)	52.7(+1.3)	54.2(+2.8)	53.1(+1.7)	50.5(+1.5)	50.9(+1.9)	48.4(-0.6)	50.0(+1.0)	48.9(-0.1)
8	7+BagWC_ED	53.4(+2.0)	52.2(+0.8)	50.8(-0.6)	53.5(+2.1)	53.6(+2.2)	49.2(+0.2)	50.7(+1.7)	49.2(+0.2)	47.9(-1.1)	49.5(+0.5)

Table 2.2: In-domain and Out-of-domain performance for different embedding features.

From the table, we find that for C&W and HLBL embeddings of 50 and 100 dimensions, the most effective way to introduce word embeddings is to add embeddings to the heads of the two mentions (row 2; both in-domain and out-of-domain) although it is less pronounced for HLBL embedding with 50 dimensions. Interestingly, for C&W embedding with 25 dimensions, adding the embedding to both heads and words of the two mentions (row 6) performs the best for both in-domain and out-of-domain scenarios. This is new compared to the word cluster features where the heads of the two mentions are always the best places for augmentation (Sun et al., 2011). It suggests that a suitable amount of embeddings for words in the mentions might be useful for the augmentation of the heads and inspires further exploration. Introducing embeddings to words of mentions alone has mild impact while it is generally a bad idea to augment chunk heads and words in the contexts.

Comparing C&W and HLBL embeddings is somehow more complicated. For both in-domain and out-of-domain settings with different numbers of dimensions, C&W embedding outperforms HLBL embedding when only the heads of the mentions are augmented while the degree of negative impact of HLBL embedding on

chunk heads as well as context words seems less serious than C&W's. Regarding the incremental addition of features (rows 6, 7, 8), C&W is better for the out-of-domain performance when 50 dimensions are used, whereas HLBL (with both 50 and 100 dimensions) is more effective for the in-domain setting. For the next experiments, we will apply the C&W embedding of 50 dimensions to the heads of the mentions for its best out-of-domain performance.

2.1.4.3 Domain Adaptation Experiments with Word Embeddings

This section examines the effectiveness of word representations for RE across domains. We evaluate word cluster and embedding (denoted by ED) features by adding them individually as well as simultaneously into the baseline feature set. For word clusters, we experiment with two possibilities: (i) only using a single prefix length of 10 (as Plank and Moschitti, 2013 did) (denoted by WC10) and (ii) applying multiple prefix lengths of 4, 6, 8, 10 together with the full string[7] (denoted by WC). Table 2.3 presents the system performance (F measures) for both in-domain and out-of-domain settings[8].

System	In-domain	bc	cts	wl
Baseline(B)	51.4	49.7	41.5	36.6
B+WC10	52.3(+0.9)	50.8(+1.1)	45.7(+4.2)	39.6(+3)
B+WC	53.7(+2.3)	52.8(+3.1)	46.8(+5.3)	41.7(+5.1)
B+ED	54.1(+2.7)	52.4(+2.7)	46.2(+4.7)	42.5(+5.9)
B+WC+ED	**55.5(+4.1)**	**53.8(+4.1)**	**47.4(+5.9)**	**44.7(+8.1)**

Table 2.3: Domain adaptation results with word representations.

7. This set of prefix lengths is shown to produce the best results experimentally.

8. All the improvements over the baseline in Table 2.3 are significant at confidence level $\geq$ 95%.

The key observations from the table are:

(i): The baseline system achieves a performance of 51.4% within its own domain while the performance on target domains `bc`, `cts`, `wl` drops to 49.7%, 41.5% and 36.6% respectively. Our baseline performance is worse than that of (Plank and Moschitti, 2013) only on the target domain `cts` and better in the other cases. This might be explained by the difference between our baseline feature set and the feature set underlying their kernel-based system. However, the performance order across domains of the two baselines are the same. Besides, the baseline performance is improved over the target domains when the system is enriched with word cluster features of the 10 prefix length only (row 2).

(ii): Over all the target domains, the performance of the system augmented with word cluster features of various granularities (row 3) is superior to that when only cluster features for the prefix length 10 are added (row 2). This is significant (at confidence level $\geq 95\%$) for domains `bc` and `wl` and verifies our assumption that various granularities for word cluster features are more effective than a single granularity for domain adaptation of RE.

(iii): Row 4 shows that word embedding itself is also very useful for domain adaptation in RE since it improves the baseline system for all the target domains.

(iv): In row 5, we see that the addition of both word cluster and word embedding features improves the system further and results in the best performance over all target domains (this is significant with confidence level $\geq 95\%$ in domains `bc` and `wl`). The result suggests that word embeddings seem to capture different information from word clusters and their combination would be effective to generalize

relation extractors across domains. However, in domain `cts`, the improvement that word embeddings provide over word clusters is modest. This is because the RCV1 corpus used to induce the word embeddings (Turian et al., 2010) does not cover spoken language words in `cts` very well.

(v): Finally, the in-domain performance is also improved consistently demonstrating the robustness of word representations (Plank and Moschitti, 2013).

2.1.4.4 Domain Adaptation Experiments with Regularization

All the experiments we have conducted so far do not apply regularization for training (like some previous research on RE (Sun et al., 2011)). In this section, in order to evaluate the effect of regularization on the generalization capacity of relation extractors across domains, we replicate all the experiments in Section 2.1.4.3 but apply regularization when relation extractors are trained[9]. Table 2.4 presents the results[10].

System	In-domain	bc	cts	wl
Baseline(B)	56.2	55.5	48.7	42.2
B+WC10	57.5(+1.3)	57.3(+1.8)	52.3(+3.6)	45.0(+2.8)
B+WC	58.9(+2.7)	58.4(+2.9)	52.8(+4.1)	47.3(+5.1)
B+ED	58.9(+2.7)	59.5(+4.0)	52.6(+3.9)	48.6(+6.4)
B+WC+ED	**59.4(+3.2)**	**59.8(+4.3)**	**52.9(+4.2)**	**49.7(+7.5)**

Table 2.4: Domain adaptation results with regularization.

For this experiment, every statement in (ii), (iii), (iv) and (v) of Section 2.1.4.3

9. We use a L2 regularizer with the regularization parameter of 0.5 for its best experimental results.

10. All the improvements over the baseline in Table 2.4 are significant at confidence level $\geq$ 95%.

also holds. More importantly, the performance in every cell of Table 2.4 is significantly better than the corresponding cell in Table 2.3 (5% or better gain in F measure, a significant improvement at confidence level $\geq 95\%$). This demonstrates the effectiveness of regularization for RE in general and for domain adaptation of RE specifically.

2.2 The Kernel-based Approach

In parallel to our work in the previous section on DA for RE (Nguyen and Grishman, 2014a), Nguyen et al., 2014d present a supervised DA algorithm for RE that assumes some labeled data in the target domains. This differs from the work in (Nguyen and Grishman, 2014a) and (Plank and Moschitti, 2013) in that the latter belongs to the unsupervised domain adaptation techniques (i.e, requiring no labeled data in the target domains). In our view, unsupervised DA is more challenging, but more realistic and practical for RE as we usually do not know which target domains we need to work on in advance, thus cannot expect to possess labeled data of the target domains. Our work in this section therefore also focuses on the single-system *unsupervised* DA. The *single-system* setting implies the construction of a single system that can work robustly with different but related domains (multiple target domains) as in the previous section.

Plank and Moschitti, 2013 propose to embed word clusters and latent semantic analysis (LSA) of words into tree kernels for DA of RE, while (Nguyen and Grishman, 2014a) studies the application of word clusters and word embeddings for DA of RE on the feature-based method. Although word clusters have been employed

by both studies to improve the performance of relation extractors across domains, the application of word embeddings (Bengio et al., 2003; Mnih and Hinton, 2008; Turian et al., 2010) for DA of RE is only examined in the feature-based method and never explored in the tree kernel-based method so far, giving rise to the first question we want to address in this section:

(i) Can word embeddings help the tree kernel-based methods on DA for RE and more importantly, in which way can we do it effectively?

This question is important as word embeddings are real valued vectors, while the tree kernel-based methods rely on the symbolic matches or mismatches of concrete labels in the parse trees to compute the kernels. It is unclear at the first glance how to encode word embeddings into the tree kernels effectively so that word embeddings could help to improve the generalization performance of RE.

One way is to use word embeddings to compute similarities between words and embed these similarity scores into the kernel functions, e.g., by resembling the method of (Plank and Moschitti, 2013) that exploited LSA (in the semantic syntactic tree kernel (SSTK), cf. §2.2.1.1). We explore various methods to apply word embeddings to generate the semantic representations for DA of RE and demonstrate that semantic representations are very effective to significantly improve the portability of the relation extractors based on the tree kernels, bringing us to the second question:

(ii) Between the feature-based method in (Nguyen and Grishman, 2014a) and the SSTK method in (Plank and Moschitti, 2013), which method is better for DA of RE, given the recent discovery of word embeddings for both methods?

It is worth noting that besides the approach difference, these two works employ rather different resources and settings in their evaluation, making it impossible to directly compare their performance. In particular, while Plank and Moschitti, 2013 only use the path-enclosed trees induced from the constituent parse trees as the representation for relation mentions, Nguyen and Grishman, 2014a include a rich set of features extracted from multiple resources such as constituent trees, dependency trees, gazetteers, semantic resources in the representation. Besides, Plank and Moschitti, 2013 consider the direction of relations in their evaluation (i.e, distinguishing between relation classes and their inverses) but Nguyen and Grishman, 2014a disregard this relation direction. Finally, we note that although both studies evaluate their systems on the ACE 2005 dataset, they actually have different dataset partitions. In order to overcome this limitation, we conduct an evaluation in which the two methods are directed to use the same resources and settings, and are thus compared in a *compatible* manner to achieve an insight on their effectiveness for DA of RE. In fact, the problem of incompatible comparison is unfortunately very common in the RE literature (Plank and Moschitti, 2013; Wang, 2008) and we believe there is a need to tackle this increasing confusion in this line of research. Therefore, this is actually the first attempt to compare the two methods (tree kernel-based and feature-based) on the same settings. To ease the comparison for future work and circumvent the *Zigglebottom* pitfall (Pedersen, 2008), the entire setup and package is available[11].

11. `https://bitbucket.org/nycphre/limo-re`

2.2.1 Relation Extraction Approaches

In the following, we review and compare the two relation extraction systems with greater detail to facilitate the later discussion.

2.2.1.1 Tree kernel-based Method

In the tree kernel-based method (Moschitti, 2006, 2008; Plank and Moschitti, 2013), a relation mention (the two entity mentions and the sentence containing them) is represented by the path-enclosed tree (PET), the smallest constituency-based subtree including the two target entity mentions (Zhang et al., 2006). The syntactic tree kernel (STK) is then defined to compute the similarity between two PET trees (where target entities are marked) by counting the common sub-trees, without enumerating the whole fragment space (Moschitti, 2006, 2008). STK is then applied in the support vector machines (SVMs) for RE. The major limitation of STK is its inability to match two trees that share the same substructure, but involve different, though semantically related, terminal nodes (words). This is caused by the hard matches between words, and consequently between sequences containing them. For instance, in the following example taken from (Plank and Moschitti, 2013), the two fragments "*governor from Texas*" and "*head of Maryland*" would not match in STK although they have very similar syntactic structures and basically convey the same relationship.

Plank and Moschitti, 2013 propose to resolve this issue for STK using the semantic syntactic tree kernel (SSTK) (Bloehdorn and Moschitti, 2007) and apply it to the domain adaptation problem of RE. The two following techniques are

utilized to activate the SSTK: (i) replace the part-of-speech nodes in the PET trees by the new ones labeled by the word clusters of the corresponding terminals (words); (ii) replace the binary similarity scores between words (i.e, either 1 or 0) by the similarities induced from the latent semantic analysis (LSA) of large corpus. The former generalizes the part-of-speech similarity to the semantic similarity on word clusters; the latter, on the other hand, allows soft matches between words that have the same latent semantic but differ in symbolic representation. Both techniques emphasize the invariants of word semantics in different domains, thus being helpful to alleviate the vocabulary difference across domains.

2.2.1.2 Feature-based Method

In the feature-based method (Nguyen and Grishman, 2014a; Sun et al., 2011; Zhou et al., 2005), relation mentions are first transformed into rich feature vectors that capture various characteristics of the relation mentions (i.e, lexicon, syntax, semantics etc). The resulting vectors are then fed into the statistical classifiers such as Maximum Entropy (MaxEnt) to perform classification for RE.

The main reason for the performance loss of the feature-based systems on new domains is the behavioral changes of the features when domains shift. Some features might be very informative in the source domain but become less relevant in the target domains. For instance, some words, that are very indicative in the source domain might not appear in the target domains (lexical sparsity). Consequently, the models putting high weights on such words (features) in the source domain will fail to perform well on the target domains. Nguyen and Grishman, 2014a

address this problem for the feature-based method in DA of RE by introducing word embeddings as additional features. The rationale is based on the fact that word embeddings are low dimensional and real valued vectors, capturing latent syntactic and semantic properties of words (Bengio et al., 2003; Mnih and Hinton, 2008; Turian et al., 2010). The embeddings of symbolically different words are often close to each other if they have similar semantic and syntactic functions. This again helps to mitigate the lexical sparsity or the vocabulary difference between the domains and has proven helpful for, amongst others, the feature-based method in DA of RE.

2.2.1.3 Tree Kernel-based vs Feature-based

The feature-based method explicitly encapsulates the linguistic intuition and domain expertise for RE into the features, while the tree kernel-based method avoids the complicated feature engineering and implicitly encodes the features into the computation of the tree kernels. Which method is better for DA of RE?

In order to ensure the two methods (Nguyen and Grishman, 2014a; Plank and Moschitti, 2013) are compared compatibly on the same resources, we make sure the two systems have access to the same amount of information. Thus, we follow (Plank and Moschitti, 2013) and use the PET trees (beside word clusters and word embeddings) as the only resource the two methods can exploit.

For the feature-based method, we utilize all the features extractable from the PET trees that are standard in the state-of-the-art feature-based systems for DA of RE (Nguyen and Grishman, 2014a). Specifically, the feature set employed in

this work (denoted by FET) includes: the lexical features, i.e., the context words, the head words, the bigrams, the number of words, the lexical path, the order of mention (Sun et al., 2011; Zhou et al., 2005); and the syntactic features, i.e., the path connecting the two mentions in PET and the unigrams, bigrams, trigrams along this path (Jiang and Zhai, 2007a; Zhou et al., 2005).

Hypothesis: Assuming identical settings and resources, we hypothesize that the tree kernel-based method is better than the feature-based method for DA of RE. This is motivated because of at least two reasons: (i) the tree kernel-based method implicitly encodes a more comprehensive feature set (involving all the sub-trees in the PETs), thus potentially captures more domain-independent features to be useful for DA of RE; (ii) the tree kernel-based method avoids the inclusion of fine-tuned and domain-specific features originating from the excessive feature engineering (i.e., hand-designing feature sets based on the linguistic intuition for specific domains) of the feature-based method.

2.2.2 Word Embeddings & Tree Kernels

In this section, we first give the intuition that guides us in designing the proposed methods. In particular, one limitation of the syntactic semantic tree kernel presented in (Plank and Moschitti, 2013) (§2.2.1.1) is that semantics is highly tied to syntax (the PET trees) in the kernel computation, limiting the generalization capacity of semantics to the extent of syntactic matches. If two relation mentions have different syntactic structures, the two relation mentions will not match, although they share the same semantic representation and express the same relation

class. For instance, the two fragments "*Tom is the CEO of the company*" and "*the company, headed by Tom*" express the same relationship between "*Tom*" and "*company*" based on the semantics of their context words, but cannot be matched in SSTK as their syntactic structures are different. In such a case, it is desirable to have a representation of relation mentions that is grounded on the semantics of the context words and reflects the latent semantics of the whole relation mentions. This representation is expected to be general enough to be effective on different domains. Once the semantic representation of relation mentions is established, we can use it in conjunction with the traditional tree kernels to extend their coverage. The benefit is mutual as both semantics and syntax help to generalize relation mentions to improve the recall, but also constrain each other to support precision. This is the basic idea of our approach, which we compare to the previous methods.

2.2.2.1 Methods

We propose to utilize word embeddings of the context words as the principal components to obtain semantic representations for relation mentions in the tree kernel-based methods. Besides more traditional approaches to exploit word embeddings, we investigate representations that go beyond the word level and use compositionality embeddings for domain adaptation for the first time.

In general, suppose we are able to acquire an additional real-valued vector $VECT_i$ from word embeddings to semantically represent a relation mention REL_i (along with the PET tree $TREE_i$), leading to the new representation of $REL_i =$

($TREE_i$, $VECT_i$). The new kernel function in this case is then defined by:

$$K_{new}(REL_i, REL_j) = (1 - \alpha)\text{SSTK}(TREE_i, TREE_j) + \alpha K_{vec}(VECT_i, VECT_j) \tag{2.1}$$

where $K_{vec}(VECT_i, VECT_j)$ is some standard vector kernel such as the polynomial kernels. α is a trade-off parameter and indicates whether the system attributes more weight to the traditional SSTK or the new semantic kernel K_{vec}.

In this work, we consider the following methods to obtain the semantic representation $VECT_i$ from the word embeddings of the context words of REL_i (assuming m_e is the dimensionality of the word embeddings):

HEAD: $VECT_i$ = the concatenation of the word embeddings of the two entity mention heads of REL_i. This representation is inherited from (Nguyen and Grishman, 2014a) that only examines embeddings at the word level separately for the feature-based method without considering the compositionality embeddings of relation mentions. The dimensionality of HEAD is $2m_e$.

According to the principle of compositionality (Baroni and Zamparelli, 2010; Paperno et al., 2014; Werning et al., 2006), the meaning of a complex expression is determined by the meanings of its components and the rules to combine them. We study the following two compositionality embeddings for relation mentions that can be generated from the embeddings of the context words:

PHRASE: $VECT_i$ = the mean of the embeddings of the words contained in the PET tree T_i of REL_i. Although this composition is simple, it is in fact competitive to the more complicated methods based on recursive neural networks (Blacoe and Lapata, 2012; Socher et al., 2012b; Sterckx et al., 2014) on representing phrase

semantics.

TREE: This is motivated by the training of recursive neural networks (Socher et al., 2012a) for semantic compositionality and attempts to aggregate the context words embeddings syntactically. In particular, we compute an embedding for every node in the PET tree in a bottom-up manner. The embeddings of the leaves are the embeddings of the words associated with them while the embeddings of the internal nodes are the means of the embeddings of their children nodes. We use the embeddings of the root of the PET tree to represent the relation mention in this case. Both PHRASE and TREE have m_e dimensions.

It is also interesting to examine combinations of these three representations (cf., Table 2.5).

SIM: Finally, for completeness, we experiment with a more obvious way to introduce word embeddings into tree kernels, resembling more closely the approach of (Plank and Moschitti, 2013). In particularly, the SIM method simply replaces the similarity scores between word pairs obtained from LSA by the cosine similarities between the word embeddings to be used in the SSTK kernel.

2.2.3 Experiments

2.2.3.1 Dataset, Resources and Parameters

We use the word clusters trained by (Plank and Moschitti, 2013) on the ukWaC corpus (Baroni et al., 2009) with 2 billion words, and the C&W word embeddings from (Turian et al., 2010)[12] with 50 dimensions following (Nguyen and Grishman,

12. http://metaoptimize.com/projects/wordreprs

2014a). In order to make the comparisons compatible, we introduce word embeddings into the tree kernel by extending the package provided by (Plank and Moschitti, 2013), which uses the Charniak parser to obtain the constituent trees, the SVM-light-TK for the SSTK kernel in SVM, the directional relation classes, etc. We utilize the default vector kernel in the SVM-light-TK package (`d=3`). For the feature-based method, we apply the MaxEnt classifier in the MALLET[13] package with the L2 regularizer on the hierarchical architecture for relation extraction as in (Nguyen and Grishman, 2014a).

Following prior work, we evaluate the systems on the ACE 2005 dataset which involves 6 domains: broadcast news (`bn`), newswire (`nw`), broadcast conversation (`bc`), telephone conversation (`cts`), weblogs (`wl`) and usenet (`un`). The union of `bn` and `nw` (`news`) is used as the source domain while `bc`, `cts` and `wl` play the role of the target domains. We take half of bc as the only target development set, and use the remaining data and domains for testing. The dataset partition is exactly the same as in (Plank and Moschitti, 2013).

2.2.3.2 Word Embeddings for Tree Kernel

We investigate the effectiveness of different semantic representations (§2.2.2.1) in tree kernels by taking the PET tree as the baseline[14], and evaluate the performance of the representations when combined with the baseline on the `bc` development set.

Table 2.5 shows the results. The main conclusions include:

13. `http://mallet.cs.umass.edu`
14. By using their system we obtained the same results.

Method	P	R	F1
PET (Plank and Moschitti, 2013)	52.2	41.7	46.4
PET+SIM	39.4	37.2	38.3
PET+HEAD	60.4	44.9	51.5
PET+PHRASE	58.4	40.7	48.0
PET+TREE	59.8	42.2	49.5
PET+HEAD+PHRASE	63.2	46.2	**53.4**
PET+HEAD+TREE	61.0	45.7	52.3
PET+PHRASE+TREE	59.2	42.4	49.4
PET+HEAD+PHRASE+TREE	60.8	45.2	51.9

Table 2.5: Performance on the `bc` dev set for PET.

(i) The substitution of LSA similarity scores with the word embedding cosine similarities (SIM) substantially degrades the performance of the tree kernel method.

(ii) When employed independently, both the word level embeddings (HEAD) and the compositionality embeddings (PHRASE, TREE) are effective for the tree kernel-based method on DA for RE, showing a slight advantage for HEAD.

(iii) Thus, the compositionality embeddings PHRASE and TREE seem to capture different information with respect to the word level embeddings HEAD. We expect the combination of HEAD with either PHRASE or TREE to further improve performance. This is the case when adding one of them at a time. PHRASE and TREE seem to capture similar information, combining all (last row in Table 2.5) is not the overall best system. The best performance is achieved when the HEAD and PHRASE embeddings are utilized at the same time, reaching an F1 of 53.4% (compared to 46.4% of the baseline) on the development set.

The results in Table 2.5 are obtained using the trade-off parameter $\alpha = 0.7$.

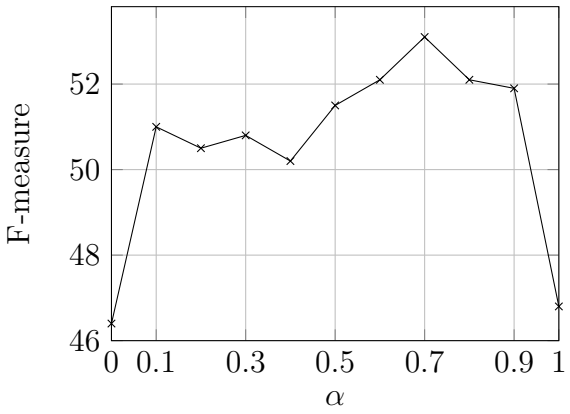

Figure 2.1: α vs F-measure on PET+HEAD+PHRASE

Figure 2.1 additionally shows the variation of the performance with changing α (for the best system on dev, i.e., for the representation PET+HEAD+PHRASE). As we can see, the performance for $\alpha > 0.5$ is in general better, suggesting a preference for the semantic representation over the syntactic representation in DA for RE. The performance reaches its peak when the suitable amounts of semantics and syntax are combined (i.e, $\alpha = 0.7$).

In the following experiments, we use the embedding combination (HEAD+PHRASE) with $\alpha = 0.7$ for the tree kernels, denoted WED.

2.2.3.3 Domain Adaptation Experiments

In this section, we examine the semantic representation for DA of RE in the tree kernel-based method. In particular, we take the systems using the PET trees, word clusters and LSA in (Plank and Moschitti, 2013) as the baselines and augment them with the embeddings WED = HEAD+PHRASE. We report the performance

of these augmented systems in Table 2.6 for the two scenarios: (i) in-domain: both training and testing are performed on the source domain via 5-fold cross validation and (ii) out-of-domain: models are trained on the source domain but evaluated on the three target domains. To summarize, we find:

		nw+bn (in-dom.)			bc			cts			wl		
#	**System:**	P:	R:	F1:	P:	R:	F1:	P:	R:	F1:	P:	R:	F1:
1	PET (Plank and Moschitti, 2013)	50.6	42.1	46.0	51.2	40.6	45.3	51.0	37.8	43.4	35.4	32.8	34.0
2	PET+WED	55.8	48.7	52.0	57.3	45.7	50.8	54.0	38.1	44.7	40.1	36.5	38.2
3	PET_WC	55.4	44.6	49.4	54.3	41.4	47.0	55.9	37.1	44.6	40.0	32.7	36.0
4	PET_WC+WED	56.3	48.2	51.9	57.0	44.3	49.8	56.1	38.1	45.4	40.7	36.1	38.2
5	PET_LSA	52.3	44.1	47.9	51.4	41.7	46.0	49.7	36.5	42.1	38.1	36.5	37.3
6	PET_LSA+WED	55.2	48.5	51.6	58.8	45.8	51.5	54.1	38.1	44.7	40.9	38.5	39.6
7	PET+PET_WC	55.0	46.5	50.4	54.4	43.4	48.3	54.1	38.1	44.7	38.4	34.5	36.3
8	PET+PET_WC+WED	56.3	50.3	53.1	57.5	46.6	51.5	55.6	39.8	**46.4**	41.5	37.9	39.6
9	PET+PET_LSA	52.7	46.6	49.5	53.9	45.2	49.2	49.9	37.6	42.9	37.9	38.3	38.1
10	PET+PET_LSA+WED	55.5	49.9	52.6	56.8	45.8	50.8	52.5	38.6	44.5	41.6	39.3	**40.5**
11	PET+PET_WC+PET_LSA	55.1	45.9	50.1	55.3	43.1	48.5	53.1	37.0	43.6	39.9	35.8	37.8
12	PET+PET_WC+PET_LSA+WED	55.0	48.8	51.7	58.5	47.3	**52.3**	52.6	38.8	44.7	42.3	38.9	**40.5**

Table 2.6: In-domain (first column) and out-of-domain performance (columns two to four) on ACE 2005. Systems of the rows not in gray come from (Plank and Moschitti, 2013) (the baselines). WED means HEAD+PHRASE.

First, word embeddings seem to subsume word clusters in the tree kernel-based method (comparing rows 2 and 4, and except domain cts) while word embeddings and LSA actually encode different information (comparing rows 2 and 6 for the out-of-domain experiments) and their combination would be helpful for DA of RE.

Second, regarding composite kernels, given word embeddings, the addition of the baseline kernel (PET) is in general useful for the augmented kernels PET_WC and PET_LSA (comparing rows 4 and 8, rows 6 and 10) although it is less pronounced for PET_LSA.

Third and most importantly, for all the systems in (Plank and Moschitti, 2013) (the baselines) and for all the target domains, whether word clusters and

LSA are utilized or not, we consistently witness performance improvement over the baselines when combined with word embedding (comparing systems X and X+WED where X is some baseline system). The best out-of-domain performance is achieved when word embeddings are employed in conjunction with the composite kernels (PET+PET_WC+PET_LSA for the target domains `bc` and `wl`, and PET+PET_WC for the target domain `cts`). To be more concrete, the best system with word embeddings (row 12 in Table 2.6) significantly outperforms the best system in (Plank and Moschitti, 2013) with $p < 0.05$, an improvement of 3.7%, 1.1% and 2.7% on the target domains `bc`, `cts` and `wl` respectively, demonstrating the benefit of word embeddings for DA of RE in the tree kernel-based method.

2.2.3.4 Tree Kernel-based vs Feature-based DA of RE

This section aims to compare the tree kernel-based method in (Plank and Moschitti, 2013) and the feature-based method in (Nguyen and Grishman, 2014a) for DA of RE on the same settings (i.e, same dataset partition, the same pre-processing procedure, the same model of directional relation classes, the same PET trees for tree kernels and feature extraction, the same word clusters and the same word embeddings). We first evaluate the feature-based system with different combinations of embeddings (i.e, HEAD, PHRASE and TREE) on the bc development set. Based on the evaluation results, we then discuss the effect of the semantic representations on the feature-based system and the tree kernel-based system, and then compare the performance of the two methods when they are augmented with their best corresponding embedding combinations.

Table 2.7 presents the evaluation results on the bc development for the feature-based system where B is the baseline feature set consisting of FET and word clusters (WC) (Nguyen and Grishman, 2014a).

System	P	R	F1
B	51.2	49.4	50.3
B+HEAD	55.8	52.4	**54.0**
B+PHRASE	50.7	46.2	48.4
B+TREE	53.6	51.1	**52.3**
B+HEAD+PHRASE	53.2	50.1	51.6
B+HEAD+TREE	54.9	51.4	**53.1**
B+PHRASE+TREE	50.7	48.4	49.5
B+HEAD+PHRASE+TREE	52.7	49.4	51.0

Table 2.7: Performance of the feature-based method (dev).

The Role of Semantic Representations

Considering Table 2.7 for the feature-based method and Table 2.5 for the tree kernel-based method, we see that when combined with the HEAD embeddings, the compositionality embedding TREE is more effective for the feature-based method, in contrast to the tree kernel-based method, where the PHRASE embeddings are better. This can be partly explained by the fact that the tree kernel-based method emphasizes the syntactic structure of the relation mentions, while the feature-based method exploits the sequential structure more. Consequently, the syntactic semantics of TREE are more helpful for the feature-based method, whereas the sequential semantics of PHRASE are more useful for the tree kernel-based method.

Performance Comparison

The three best embedding combinations for the feature-based system in Table 2.7 are (listed by performance order): (HEAD), (HEAD+TREE) and (TREE),

where (HEAD) is also the best word level method employed in (Nguyen and Grishman, 2014a). In order to enable a fairer and clearer evaluation, when doing comparison, we use both the three best embedding combinations in the feature-based method and the best embedding combination (HEAD+PHRASE) in the tree kernel-based method. In the tree kernel-based method, we do not employ the LSA information as it comes in the form of similarity scores between pairs of words, and it is not clear how to encode this information into the feature-based method effectively. Finally, we utilize the composite kernel for its demonstrated effectiveness in Section 2.2.3.3.

	`nw+bn` (in-dom.)			`bc`			`cts`			`wl`		
System:	P:	R:	F1:	P:	R:	F1:	P:	R:	F1:	P:	R:	F1:
Tree kernel-based:												
PET+PET_WC+HEAD+PHRASE	56.3	50.3	**53.1**	57.5	46.6	**51.5**	55.6	39.8	**46.4**	41.5	37.9	**39.6**
Feature-based:												
FET+WC+HEAD	44.5	51.0	47.5	46.5	49.3	47.8	44.5	40.0	42.1	35.4	39.5	37.3
FET+WC+TREE	44.4	50.2	47.1	46.4	48.7	47.6	43.7	40.3	41.9	32.7	36.7	34.6
FET+WC+HEAD+PHRASE	44.9	51.6	48.0	46.0	49.1	47.5	45.2	41.5	43.3	34.7	39.2	36.8
FET+WC+HEAD+TREE	45.1	51.0	47.8	46.9	48.4	47.6	43.8	39.5	41.5	34.7	38.8	36.6

Table 2.8: Tree kernel-based in (Plank and Moschitti, 2013) vs feature-based in (Nguyen and Grishman, 2014a). All the comparisons between the tree kernel-based method and the feature-based method in this table are significant with $p < 0.05$.

The most important observation from the experimental results (shown in Table 2.8) is that over all the target domains, the tree kernel-based system is significantly better than the feature-based systems with $p < 0.05$ (assuming the same resources and settings mentioned above). In fact, there are large margins between the tree kernel-based and the feature-based methods in this case (i.e, about 3.7% for `bc`, 3.1% for `cts` and 2.3% for `wl`), clearly confirming the hypothesis about the advantage of the tree kernel-based method over the feature-based method on DA for RE

in Section 2.2.1.3.

2.2.4 Analysis

This section analyzes the output of the systems to gain more insights into their operation.

Word Embeddings for the Tree-kernel based Method

We focus on the comparison of the best model in (Plank and Moschitti, 2013) (row 11 in Table 2.6) (called P) with the same model but augmented with the embedding WED (row 12 in Tabel 2.6) (called P+WED). One of the most interesting insights is that the embedding WED helps to semantically generalize the phrases connecting the two target entity mentions beyond the syntactic constraints. For instance, model P fails to discover the relation between "*Chuck Hagel*" and "*Vietnam*" in the sentence (of the target domain bc): "*Sergeant Chuck Hagel was seriously wounded twice in Vietnam.*" (i.e, it returns the NONE relation as the prediction) as the substructure associated with "*seriously wounded twice*" does not appear with any relation in the source domain. Model P+WED, on the other hand, correctly predicts the PHYS (Located) relation between the two entities as the PHRASE embedding of "*Chuck Hagel was seriously wounded twice in Vietnam.*" (phrase X1) is very close to the embedding of the source domain phrase: "*Stewart faces up to 30 years in prison*" (phrase X2) (annotated with the PHYS relation between "*Stewart*" and "*prison*").

In fact, X2 is only the 9th closest phrase in the source domain of X1. The closest phrase of X1 in the source domain is X3: the phrase between "*Iraqi sol-*

diers" and "*herself*" in the sentence "*The Washington Post is reporting she shot several* ***Iraqi soldiers before she was captured and she was shot herself****, too.*". However, as the syntactical structure of X1 is more similar to X2's, and is remarkably different from X3 as well as the other closest phrases (ranked from 2nd to 8th), the new kernel function K_{new} would still prefer X2 due to its trade-off between syntax and semantics.

Tree Kernel-based vs Feature-based

From the analysis of the systems in Table 2.8, we find that, among others, the tree kernel-based method improves the precision significantly via the semantic and syntactic refinement it maintains. Let us consider the following phrase of the target domain bc: "*troops have dislodged stubborn Iraqi soldiers*" (called Y1). The feature-based systems in Table 2.8 incorrectly predict the ORG-AFF relation (Employment or Membership) between "*Iraqi soldiers*" and "*troops*". This is mainly due to the high weights of the features linking the words "*troop*" and "*soldiers*" with the relation type ORG-AFF in the feature-based models, which is, in turn, originated from the high correlation of these words and the relation type in the training data of the source domain (domain bias). The tree kernel-based model in Table 2.8 successfully recognizes the NONE relation in this case. A closer examination shows that the phrase with the closest embedding to Y1 in the source domain is Y2: "*Iraqi soldiers abandoned their posts*"[15], which is annotated with the NONE relation between "*Iraqi soldiers*" and "*their posts*". As the syntactic structure of Y2 is also very similar to Y1, it is not surprising that Y1 is closest to Y2 in the new

15. The full sentence is: "*After today's air strikes, Iraqi soldiers abandoned their posts and surrendered to Kurdish fighters.*".

kernel function, consequently helping the tree kernel-based method work correctly in this case.

2.3 Related work

Word embeddings have been only applied to RE recently. Socher et al., 2012b use word embeddings as input for matrix-vector recursive neural networks in relation classification while Zeng et al., 2014, and Nguyen and Grishman, 2015a employ word embeddings in the framework of convolutional neural networks for relation classification and extraction, respectively. Sterckx et al., 2014 utilize word embeddings to reduce noise of training data in distant supervision. Kuksa et al., 2010 present a string kernel for bio-relation extraction with word embeddings, and Yu et al., 2014, 2015 study the factor-based compositional embedding models. However, none of this work examines word embeddings for domain adaptation as we do.

Regarding DA, in the unsupervised DA setting, Huang and Yates, 2010 attempt to learn multi-dimensional feature representations while Blitzer et al., 2006 introduce structural correspondence learning. Daume, 2007 propose an easy adaptation framework (EA) while Xiao and Guo, 2013 present a log-bilinear language adaptation technique in the supervised DA setting. Unfortunately, all of this work assumes some prior (in the form of either labeled or unlabeled data) on the target domains for the sequential labeling tasks, in contrast to our single-system unsupervised DA setting for relation extraction. An alternative method that is also popular for DA is instance weighting (Jiang and Zhai, 2007b). However, as shown

by (Plank and Moschitti, 2013), instance weighting is not very useful for DA of RE.

2.4 Conclusion

In order to improve the generalization (DA) for relation extractors, we propose several methods to incorporate word embeddings into the feature-based and the tree kernel-based approaches: (i) We have evaluated the effectiveness of word embedding and clustering features as well as regularization on tackling the portability of a feature-based relation extractor to new domains, and (ii) We augment the semantic syntactic tree kernels with the semantic representation of relation mentions, generated from the word embeddings of the context words. The methods demonstrates strong promise for the DA of RE, i.e, it significantly improves the best system of (Plank and Moschitti, 2013) (up to 7% relative improvement). Moreover, we perform a compatible comparison between the tree kernel-based method and the feature-based method on the same settings and resources, which suggests that the tree kernel-based method (Plank and Moschitti, 2013) is better than the feature-based method (Nguyen and Grishman, 2014a) for DA of RE. An error analysis is conducted to get a deeper comprehension of the systems.

Chapter 3

Deep Learning for Entity Mention Detection

The previous chapter has introduced the application of word embeddings to improve the robustness of relation extractors. The remainder of this dissertation deals exclusively with developing deep learning methods for IE tasks. We start with the entity mention detection task in this chapter and dedicate chapters 4 and 5 for relation extraction and event detection respectively.

Traditionally, both entity mention detection (or simply mention detection (MD)) and named entity recognition (NER) are formalized as sequential labeling problems, thereby being solved by some linear graphical models such as Hidden Markov Models (HMMs), Maximum Entropy Markov Models (MEMMs) or Conditional Random Fields (CRFs) (Lafferty et al., 2001). Although these graphical models have achieved the top performance for MD, there are still at least three problems we want to focus on in this work:

(i) The first problem is the performance loss of the mention detectors when they are trained on some domain (the source domain) and applied to other domains (the target domains). The problem might originate from various mismatches between the source and the target domains (domain shifts) such as the vocabulary difference, the distribution mismatches etc (Blitzer et al., 2006; Daume, 2007; Plank and Moschitti, 2013).

(ii) Second, in mention detection, we might need to capture a long context, possibly covering the whole sentence, to correctly predict the type for a word. For instance, consider the following sentence with the pronominal "*they*":

Now, the reason that France, Russia and Germany are against war is because they have suffered much from the past war.

In this sentence, the correct type GPE[1] for "*they*" can only be inferred from its GPE references: "*France*", "*Russia*" and "*Germany*" which are far from the pronominal "*they*" of interest. The challenge is to come up with the models that can encode and utilize these long-range dependency contexts effectively.

(iii) The third challenge is to be able to quickly adapt the current techniques for MD so that they can perform well on new languages.

In this chapter, we propose to address these problems for MD via recurrent neural networks (RNNs) which offer an effective recurrent mechanism to embed the sentence context into a distributed representation and employ it to decode the sentences. Besides, as RNNs replace the symbolic forms of words in the sentences with their word embeddings, the distributed representation that captures the general syntactic and semantic properties of words (Turian et al., 2010), they

1. Geographical Political Entity

can alleviate the lexical sparsity, induce more general feature representation, thus generalizing well across domains (Nguyen and Grishman, 2015b). This also helps RNNs to quickly and effectively adapt to new languages which just require word embeddings as the only new knowledge we need to obtain. Finally, we can achieve the task-specific word embeddings for MD to improve the overall performance by updating the initial pre-trained word embeddings during the course of training in RNNs.

The recent emerging interest in deep learning has produced many successful applications of RNNs for NLP problems such as machine translation (Bahdanau et al., 2015; Cho et al., 2014a), semantic role labeling (Zhou and Xu, 2015) etc. However, to the best of our knowledge, there has been no prior work employing RNNs for MD on the cross-domain and language settings so far. To summarize, the main contributions of this chapter are as follows:

1. We perform a systematic investigation on various RNN architectures and word embedding techniques that are motivated from linguistic observations for MD.

2. We achieve the state-of-the-art performance for MD in the cross-domain setting with the bidirectional modeling applied to RNNs.

3. We demonstrate the portability of the RNN models for MD to new languages by their significant improvement with large margins over the best reported system for named entity recognition in Dutch.

The work in this chapter is published in (Nguyen et al., 2016d).

3.1 Models

We formalize the mention detection problem as a sequential labeling task. Given a sentence $W = [w_1, w_2, \ldots, w_n]$, where w_i is the i-th word and n is the length of the sentence, we want to predict the label sequence $Y = [y_1, y_2, \ldots, y_n]$ for X, where y_i is the label for w_i. The labels y_i follow the BIO2 encoding to capture the entity mentions in W. Note that this work focuses on the extraction of the entity mention heads, following (Florian et al., 2006) and (Li and Ji, 2014a).

In order to prepare the sentence for RNNs, we first transform each word w_i into a real-valued vector vec_i using the concatenation of two vectors emb_i and fet_i: $vec_i = [emb_i, fet_i]$[2], where:

- emb_i is the word embedding vector of w_i, obtained by training a language model on a large corpus (discussed later).
- fet_i is a binary vector encompassing different features for w_i. In this work, we are utilizing four types of features: capitalization, gazetteers, triggers (whether w_i is present in a list of trigger words[3] or not) and cache (the label that is assigned to w_i sometime before in the document).

We then enrich this vector representation by including the word vectors in a context window of v_c for each word in the sentence to capture the short-range dependencies for prediction (Mesnil et al., 2013). This effectively converts w_i into the context window version of the concatenated vectors: $x_i = [vec_{i-v_c}, \ldots, vec_i, \ldots, vec_{i+v_c}]$.

2. For simplicity, we are using the word w_i and its real-valued vector representation interchangeably.

3. Trigger words are the words that are often followed by entity names in sentences such as "*president*", "*Mr.*" etc.

Given the new input representation, we describe the RNNs to be investigated in this work below.

3.1.1 The Basic Models

In standard recurrent neural networks, at each time step (word position in sentence) i, we have three main vectors: the input vector $x_i \in \mathbb{R}^{m_I}$, the hidden vector $h_i \in \mathbb{R}^{m_H}$ and the output vector $o_i \in \mathbb{R}^{m_O}$ (m_I, m_H and m_O are the dimensions of the input vectors, the dimension of the hidden vectors and the number of possible labels for each word respectively). The output vector o_i is the probabilistic distribution over the possible labels for the word x_i and obtained from h_i via the softmax function φ:

$$o_i = \varphi(Oh_i), \qquad \varphi(t_j) = \frac{e^{t_j}}{\sum_k e^{t_k}} \tag{3.1}$$

Regarding the hidden vectors or units h_i, there are two major methods to obtain them from the current input and the last hidden and output vectors, leading to two different RNN variants:

- In the Elman model, called **ELMAN**, the hidden vector from the previous step h_{i-1}, along with the input in the current step x_i, constitute the inputs to compute the current hidden state h_i:

$$h_i = \Phi(Ux_i + Vh_{i-1}) \tag{3.2}$$

- In the Jordan model, called **JORDAN**, the output vector from the previous step o_{i-1} is fed into the current hidden layer rather than the hidden vector

from the previous steps h_{i-1}. The rationale for this topology is to introduce the label from the preceding step as a feature for current prediction:

$$h_i = \Phi(Ux_i + Vo_{i-1}) \tag{3.3}$$

In the formula above, Φ is the sigmoid activation function: $\Phi(t) = \frac{1}{1+e^{-t}}$ and O, U, and V are the same weight matrices for all time steps, to be learned during training. The unfolded dependency graphs for the two models are given in Figure 3.1.

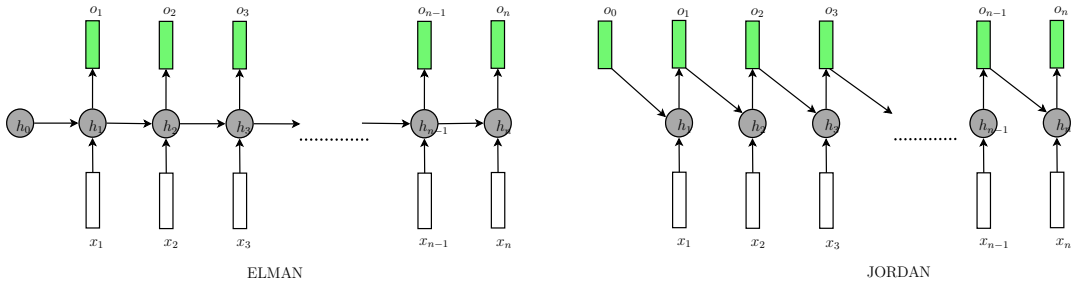

Figure 3.1: The ELMAN and JORDAN models

3.1.2 Gated Recurrent Units

The ELMAN and JORDAN models are basically the stacks of the standard feed-forward neural networks that share the same weight matrices. Unfortunately, this stacking mechanism is prone to the "*vanishing gradient*" problem (Bengio et al., 1994), making it challenging to train the networks properly in practice. This problem can be alleviated by long-short term memory units (LSTM) (Hochreiter and Schmidhuber, 1997) that propose the idea of memory cells to enable the information storage and access over a long period of time.

In this work, we use a variant of LSTM, called the *Gated Recurrent Units* (GRUs) by (Cho et al., 2014a). GRU is shown to be simpler than LSTM in terms of computation and implementation but still achieves comparable performance (Józefowicz et al., 2015).

The introduction of GRUs into the models ELMAN and JORDAN amounts to two new models, named **ELMAN_GRU** and **JORDAN_GRU** respectively, with two new methods to compute the hidden vectors h_i. The formula for ELMAN_GRU is adopted directly from (Cho, 2014b) and given below:

$$\begin{aligned} h_i &= z_i \odot \hat{h}_i + (1 - z_i) \odot h_{i-1} \\ \hat{h}_i &= \Phi(W_h x_i + U_h(r_i \odot h_{i-1})) \\ z_i &= \Phi(W_z x_i + U_z h_{i-1}) \\ r_i &= \Phi(W_r x_i + U_r h_{i-1}) \end{aligned} \tag{3.4}$$

where $W_h, W_z, W_r \in \mathbb{R}^{m_H \times m_I}$, $U_h, U_z, U_r \in \mathbb{R}^{m_H \times m_H}$ and $\odot$ is the element-wise multiplication operation.

We cannot directly apply the formula above to the JORDAN_GRU model since the dimensions of the output vectors o_i and the hidden vector h_i are different in general. For JORDAN_GRU, we first need to transform the output vector o_i into the hidden vector space, leading to the following formula:

$$
\begin{aligned}
h_i &= z_i \odot \hat{o}_i + (1 - z_i) \odot t_{i-1} \\
t_{i-1} &= T o_{i-1} \\
\hat{o}_i &= \Phi(W_o x_i + U_o(r_i \odot t_{i-1})) \\
z_i &= \Phi(W_z x_i + U_z t_{i-1}) \\
r_i &= \Phi(W_r x_i + U_r t_{i-1})
\end{aligned} \tag{3.5}
$$

where $T \in \mathbb{R}^{m_H \times m_O}$.

3.1.3 The Bidirectional Networks

One of the limitations of the four basic models presented above is their incapacity to incorporate the future context information that might be crucial to the prediction in the current step. For instance, consider the first word "*Liverpool*" in the following sentence:

Liverpool *suffered an upset first home league defeat of the season, beaten 1-0 by a Guy Whittingham goal for Sheffield Wednesday.*

In this case, the correct label ORGANIZATION can only be detected if we first go over the whole sentence and then utilize the context words after "*Liverpool*" to decide its label.

The limitation of the four models originates in their mechanism to perform a single pass over the sentences from left to right and make the prediction for a word when they first encounter it. Guided by this intuition, we propose to employ the bidirectional networks to solve the MD problem.

The bidirectional networks involve three passes over the sentence, in which the first two passes are designated to encode the sentence while the third pass is responsible for decoding. The procedure for the sentence $X = [x_1, x_2, \ldots, x_n]$ is below:

(i) Run the first RNN $\overrightarrow{RNN}$ from left to right over $[x_1, x_2, \ldots, x_n]$ to obtain the first hidden vector or output vector sequence (depending on whether $\overrightarrow{RNN}$ is an Elman or Jordan network respectively): $\overrightarrow{RNN}([x_1, x_2, \ldots, x_n]) = [\overrightarrow{h_1}, \overrightarrow{h_2}, \ldots, \overrightarrow{h_n}]$ (forward encoding).

(ii) Run the second RNN $\overleftarrow{RNN}$ from right to left over $[x_1, x_2, \ldots, x_n]$ to obtain the second hidden vector or output vector sequence: $\overleftarrow{RNN}([x_n, x_{n-1}, \ldots, x_1]) = [\overleftarrow{h_n}, \overleftarrow{h_{n-1}}, \ldots, \overleftarrow{h_1}]$ (backward encoding).

(iii) Obtain the concatenated sequence $h = [h_1, h_2, \ldots, h_n]$ where $h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}]$.

(iv) Decode the sentence with the third RNN R_d (the decoding model) using h as the input vector, i.e, replacing x_i by h_i in the formula (3.2), (3.3), (3.4) and (3.5).

Conceptually, the encoding RNNs $\overrightarrow{RNN}$ and $\overleftarrow{RNN}$ can be different but in this work, for simplicity and consistency, we assume that we only have a single encoding model, i.e, $\overrightarrow{RNN} = \overleftarrow{RNN} = R_e$. Note that R_e and R_d can be any model in {ELMAN, JORDAN, ELMAN_GRU, JORDAN_GRU}.

The observation is, at the time step i, the forward hidden vector $\overrightarrow{h_i}$ represents the encoding for the past word context (from position 1 to i) while the backward hidden vector $\overleftarrow{h_i}$ is the summary for the future word context (from position n to

i). Consequently, the concatenated vector $h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}]$ constitutes a distributed representation that is specific to the word at position i but still encapsulates the context information over the whole sentence at the same time. This effectively provides the networks a much richer representation to decode the sentence. The bidirectional network for R_e = ELMAN and R_d = JORDAN is given on the left of Figure 3.2.

We notice that (Mesnil et al., 2013) also investigate the bidirectional models for the task of slot filling in spoken language understanding. However, compared to the work presented here, (Mesnil et al., 2013) does not use any special transition memory cells (like the GRUs we are employing in this work) to avoid numerical stability issues (Pascanu et al., 2012). Besides, they form the inputs h for the decoding phase from a larger context of the forward and backward encoding outputs, while performing word-wise, independent classification; in contrast, we use only the current output vectors in the forward and backward encodings for h, but perform recursive computations to decode the sentence via the RNN model R_d (demonstrated on the left of Figure 3.2).

3.1.4 Training and Inference

We train the networks locally. In particular, each training example consists of a word x_i and its corresponding label y_i in a sentence $X = [x_1, x_2, \ldots, x_n]$ (denoted by $I = (x_i, y_i, X)$). In the encoding phase, we first compute the necessary inputs according to the specific model of interest. This can be the original input vectors $[x_1, x_2, \ldots, x_n]$ in the four basic models or the concatenated vectors $[h_1, h_2, \ldots, h_n]$

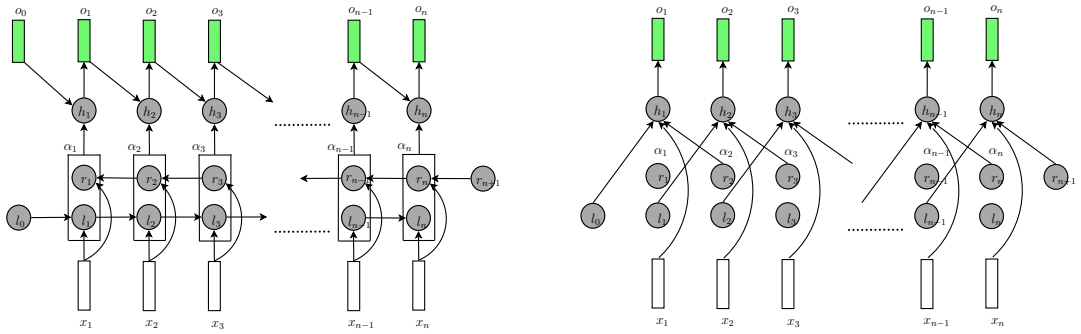

Figure 3.2: The bidirectional models. The model on the right is from(Mesnil et al., 2013) with the forward and backward context size of 1. l_0, r_{n+1} are the zero vectors.

in the bidirectional models. Eventually, in the decoding phase, a sequence of v_d input vectors preceding the current position i is fed into the decoding network R_d to obtain the output vector sequence. The last vector in this output sequence corresponds to the probabilistic label distribution for the current position i, to be used to compute the objective function. For example, in the bidirectional models, the input sequence for the decoding phase is $h_{i-v_d}h_{i-v_d+1}\dots h_i$ while the output sequence is: $R_e([h_{i-v_d}, h_{i-v_d+1}, \dots, h_i]) = [o_{i-v_d}, o_{i-v_d+1}, \dots, o_i]$.

In this work, we employ the stochastic gradient descent algorithm[4] to update the parameters via minimizing the negative log-likelihood objective function:

$$\mathbf{nll}(I) = -\log(o_i[y_i]). \tag{3.6}$$

Finally, besides the weight matrices in the networks, the word embeddings are also optimized during training to obtain the task-specific word embeddings for MD. The gradients are computed via back-propagation and inference is performed by

4. We tried the *AdaDelta* algorithm and the dropout regularization but do not see much difference.

running the networks over the whole sentences and taking argmax over the output sequence:

$$y_i = \text{argmax}(o_i). \tag{3.7}$$

3.2 Word Representation

Following (Collobert et al., 2011), we pre-train word embeddings from a large corpus and employ them to initialize the word representations in the models. One of the state-of-the-art models to train word embeddings has been proposed recently in (Mikolov et al., 2013b) that introduce two log-linear models, i.e the continuous bag-of-words model (CBOW) and the continuous skip-gram model (Skip-gram). The CBOW model attempts to predict the current word based on *the average of the context word vectors* while the Skip-gram model aims to predict the surrounding words in a sentence given the current word.

In this work, besides the CBOW and skip-gram models, we examine a concatenation - based variant of CBOW (C-CBOW) to train word embeddings and compare the three models to gain insights into which kind of model is effective to obtain word representations for the MD task. The objective of C-CBOW is to predict the target word using *the concatenation of the vectors of the words surrounding it*, motivated from our strategy to decide the label for a word based on the concatenated context vectors. Intuitively, the C-CBOW model would perform better than CBOW as the concatenation mechanism helps to assign different weights to different context words, thereby being more flexible than CBOW that

applies a single weight for all the context words. CBOW, Skip-gram and C-CBOW are illustrated in Figure 3.3.

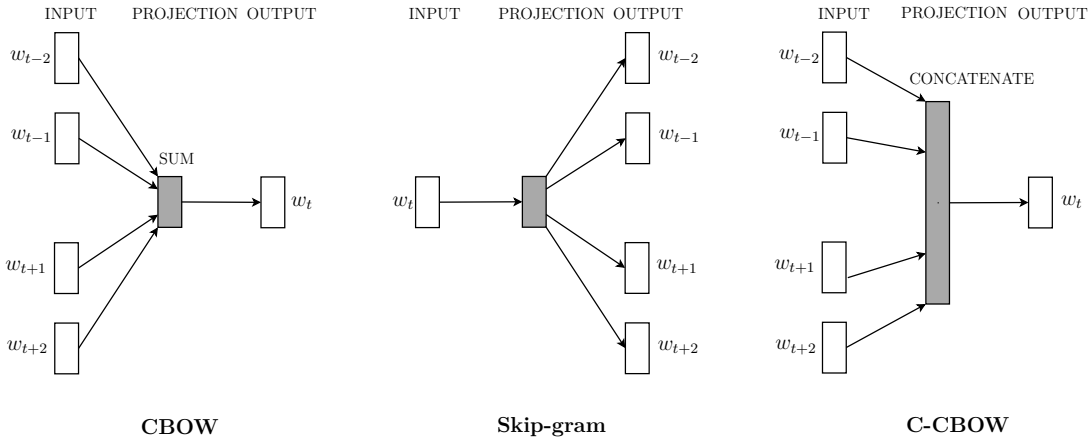

Figure 3.3: Methods to Train Word Embeddings

3.3 Experiments

3.3.1 Dataset

In order to investigate the robustness across domains, following the prior work (Nguyen and Grishman, 2015a; Plank and Moschitti, 2013), we utilize the ACE 2005 dataset which contains 6 domains: broadcast news (`bn`), newswire (`nw`), broadcast conversation (`bc`), telephone conversation (`cts`), weblogs (`wl`), usenet (`un`) and 7 entity types: person, organization, GPE, location, facility, weapon, vehicle. The union of `bn` and `nw` is considered as a single domain, called `news`. We take half of `bc` as the only development data and use the remaining data and domains for evaluation. Some statistics about the domains are given in Table 3.1.

Domain	#Docs	#Sents	#Mentions
news	332	6487	22460
bc	60	3720	9336
cts	39	5900	9924
wl	119	2447	6538
un	49	2746	6507
Total	599	21300	54765

Table 3.1: ACE 2005 Dataset

Regarding the robustness across languages, we further evaluate the RNN models on the CoNLL 2002 dataset for Dutch Named Entity Recognition[5] (Carreras et al., 2002; Sang and Meulder, 2002). The CoNLL dataset comes along with the training data, validation data and test data, annotated for 4 types of entities: person, organization, location and miscellaneous.

3.3.2 Resources and Parameters

In all the experiments with RNNs below, we employ the context window $v_c = 5$, the decoding window $v_d = 9$. We find that the optimal number of hidden units (or the dimension of the hidden vectors) and the learning rate vary according to the dataset. For the ACE 2005 dataset, we use 200 hidden units with learning rate = 0.01 while these numbers are 100 and 0.06 respectively for the Dutch CoNLL dataset. Note that the number of hidden units is kept the same in both the encoding phase and the decoding phase.

For word representation, we train the word embeddings for English from the Gigaword corpus augmented with the newsgroups data from BOLT (Broad Opera-

5. http://www.cnts.ua.ac.be/conll2002/ner

tional Language Technologies) (6 billion tokens) while the entire Dutch Wikipedia pages (310 million tokens) are extracted to train the Dutch word embeddings. We utilize the `word2vec` toolkit[6] (modified to add the C-CBOW model) to learn the word representations. Following (Baroni et al., 2014), we use the context window of 5, subsampling set to $1e$-05 and negative sampling with the number of negative instances set to 10. The dimension of the vectors is set to 300 to make it comparable with the `word2vec` toolkit. Finally, we use the standard BIO2 tagging schema for both ACE 2005 and Dutch CoNLL datasets.

3.3.3 Model Architecture Evaluation

In this section, we evaluate different RNN models by training the models on the **news** domain and report the performance on the development set. As presented in the previous sections, we have 4 basic models $M = \{$ELMAN, JORDAN, ELMAN_GRU, JORDAN_GRU$\}$ and 16 bidirectional models (4 choices for the encoding and decoding models R_e, R_d in M). The performance for the basic models and the bidirectional models are shown in Table 3.2 and Table 3.3 respectively[7].

Model(R_d)	F1
ELMAN	80.70
JORDAN	80.46
ELMAN_GRU	80.85
JORDAN_GRU	**81.06**

Table 3.2: The basic models' performance

There are several important observations from the three tables:

6. `https://code.google.com/p/word2vec`
7. The experiments in this section use C-CBOW to pre-train word embeddings.

R_d \ R_e	ELMAN	ELMAN_GRU
ELMAN	80.99	81.42
JORDAN	81.14	81.68
ELMAN_GRU	80.53	81.16
JORDAN_GRU	80.98	**82.37**
R_d \ R_e	JORDAN	JORDAN_GRU
ELMAN	79.12	79.64
JORDAN	79.21	80.85
ELMAN_GRU	79.80	80.41
JORDAN_GRU	79.76	81.02

Table 3.3: The bidirectional models' performance

-Elman vs Jordan: In the encoding phase, the Elman models consistently outperform the Jordan models when the same decoding model is applied in the bidirectional architecture. In the decoding phase, however, it turns out that the Jordan models are better most of the time over different model architectures (basic or bidirectional).

-With vs Without GRUs: It is clear from the tables that GRUs are very helpful in the encoding part of the bidirectional architecture for MD. However, for the decoding part, we can only see the clear benefit of GRUs in the basic models and the bidirectional architecture when R_e is a Jordan model.

-Regarding different model architectures, in general, the bidirectional models are more effective than the basic models, confirming the effectiveness of bidirectional modeling to achieve a richer representation for MD.

The best basic model (F1 = 81.06%) and the best bidirectional model (F1 = 82.37%) are called BASIC and BIDIRECT respectively. In the following, we only focus on these best models in the experiments.

3.3.4 Comparison to other Bidirectional RNN Work

Mesnil et al., 2013 also present a RNN system with bidirectional modeling for the slot filling task. As described in Section 3.1.3, the major difference between the bidirectional models in this work and (Mesnil et al., 2013)'s is the recurrence in our decoding phase. Table 3.4 compares the performance of the bidirectional model from (Mesnil et al., 2013), called MESNIL, and the BIDIRECT model. In order to verify the effectiveness of recurrence in decoding, the performance of MESNIL incorporated with the JORDAN_GRU model in the decoding phase (MESNIL+JORDAN_GRU) is also reported.

Model	P	R	F1
MESNIL (Mesnil et al., 2013)	81.01	79.67	80.33
MESNIL + JORDAN_GRU	82.17	79.56	80.85
BIDIRECT	82.91	81.83	**82.37**

Table 3.4: Comparison to (Mesnil et al., 2013).

In general, we see that the bidirectional model in this work is much better than the model in (Mesnil et al., 2013) for MD. This is significant with $p < 0.05$ and a large margin (an absolute improvement of 2.04%). More interestingly, MESNIL is further improved when it is augmented with the JORDAN_GRU decoding, verifying the importance of recurrence in decoding for MD.

3.3.5 Word Embedding Evaluation

The section investigates the effectiveness of different techniques to learn word embeddings to initialize the RNNs for MD. Table 3.5 presents the performance

of the BASIC and BIDIRECT models on the development set (trained on `news`) when the CBOW, SKIP-GRAM and C-CBOW techniques are utilized to obtain word embeddings from the same English corpus. We also report the performance of the models when they are initialized with the `word2vec` word embeddings from (Mikolov et al., 2013b) (trained with the Skip-gram model on 100 billion words of Google News) (WORD2VEC). All of these word embeddings are updated during the training of the RNNs to induce the task-specific word embeddings . Finally, for comparison purpose, the performance for the following two scenarios is also included: (i) the word vectors are initialized randomly (not using any pre-trained word embeddings) (RANDOM), and (ii) the word vectors are loaded from the C-CBOW pre-trained word embeddings but fixed during the RNN training (FIXED).

Word Embeddings	Model	
	BASIC	BIDIRECT
RANDOM	79.30	79.76
FIXED	80.36	81.52
WORD2VEC	80.92	81.41
CBOW	78.61	79.74
SKIP-GRAM	81.45	81.96
C-CBOW	81.06	**82.37**

Table 3.5: Comparison of methods for word embeddings.

The first observation is that we need to borrow some pre-trained word embeddings and update them during the training process to improve the MD performance (comparing C-CBOW, RANDOM and FIXED). Second, C-CBOW is much better than CBOW, confirming our intuition in Section 3.2. Third, we do not see much difference in terms of MD performance when we enlarge the corpus to learn word embeddings (comparing SKIP-GRAM and WORD2VEC that is trained with the

skip-gram model on a much larger corpus). Finally, we achieve the best performance when we apply the C-CBOW technique in the BIDIRECT model. From now on, for consistency, we use the C-CBOW word embeddings in all the remaining experiments in this chapter.

3.3.6 Cross-Domain Experiments

This section evaluates the MD systems on the cross-domain settings to gain an insight into their operation when the domain changes. The state-of-the-art systems for MD have been the joint extraction system for entity mentions and relations from (Li and Ji, 2014a), the information networks to unify the outputs of three information extraction tasks: entity mentions, relations and events using structured perceptron from (Li et al., 2014b) and the Maximum Entropy Markov Model (MEMM) system from (Florian et al., 2006). These systems extensively hand-design a large set of features (parsers, gazetteers, word clusters, coreference etc) to capture the useful structures for MD. In this work, we use the MEMM system in (Florian et al., 2006) as the baseline and compare it with the RNN systems. The reason for this choice is twofold: (i) as shown in Section 5.4 of (Li and Ji, 2014a), the performance of the joint systems are comparable to the MEMM system in (Florian et al., 2006), and (ii) similar to our work, the MEMM system in (Florian et al., 2006) only focuses on the MD task while the joint systems in (Li and Ji, 2014a; Li et al., 2014b) involves the predictions for other tasks, making it less comparable to our work, especially on the cross-domain setting for MD. Evaluating the joint models in (Li and Ji, 2014a; Li et al., 2014b) on the cross-domain setting

for MD is another important dimension, however, out of the scope of the current work.

We note that the performance of the MEMM system reported in this work is obtained from the actual system in (Florian et al., 2006) and the feature set of the MEMM[8] system also includes the four features we are using in the RNN models (Section 3.1).

Following the previous work on the cross-domain settings for the ACE 2005 dataset (Nguyen et al., 2015c; Plank and Moschitti, 2013), we treat `news` as the source domain and the other domains: `bc`, `cts`, `wl` and `un` as the target domains. We then examine the systems on two scenarios: (i) the systems are trained and tested on the source domain via 5-fold cross validation (in-domain performance), and (ii) the systems are trained on the source domain but evaluated on the target domains. Besides, in order to understand the effect of the features on the systems, we report the systems' performance both including and excluding the features described in Section 3.1. Table 3.6 presents the results.

System	Without Features					With Features				
	In-Domain	bc	cts	wl	un	In-Domain	bc	cts	wl	un
MEMM	76.90	71.73	78.02	66.89	67.77	**82.55**	78.33	87.17	**76.70**	76.75
BASIC	79.01	**77.06**	85.42	73.00	72.93	81.99	78.75	86.51	76.60	76.94
BIDIRECT	**80.00†**	76.27†	**85.64†**	**73.79†**	**73.88†**	82.52	**79.65†**	**88.43†**	**76.70**	**77.03**

Table 3.6: System's performance on the cross-domain setting. Cells marked with †designate the BIDIRECT models that significantly outperform ($p < 0.05$) the MEMM model on the specified domains.

To summarize, we find that the RNN systems significantly outperform the MEMM system across all the target domains when the features are not applied.

8. We also tried the CRF model with the same feature set as the MEMM system but it is worse in our case.

	MEMM				BIDIRECT				BIDIRECT-MEMM			
	bc	cts	wl	un	bc	cts	wl	un	bc	cts	wl	un
bc	75.20	86.60	70.25	72.38	75.49	87.51	70.75	73.04	**0.29**	**0.91†**	**0.50†**	**0.66†**
cts	66.91	89.76	68.74	69.72	68.23	91.24	68.82	70.27	**1.32†**	**1.48†**	**0.08**	**0.55†**
wl	74.94	86.53	77.07	75.90	74.73	86.79	76.35	75.37	-0.21	**0.26**	-0.72	-0.53
un	72.72	86.75	72.04	73.47	73.53	88.29	73.16	74.00	**0.81†**	**1.45†**	**1.12†**	**0.53†**

Table 3.7: Comparison between MEMM and BIDIRECT. Cells marked with †designate the statistical significance ($p < 0.05$). The columns and rows correspond to the source and target domains respectively. BIDIRECT-MEMM means performance subtraction.

The BIDIRECT system still yields the best performance among systems being investigated (except in domain `bc`). This is also the case when the features from Section 3.1 are included and demonstrates the robustness of the BIDIRECT model in the domain shifts. We further support this result in Table 3.7 where we report the performance of the MEMM and BIDIRECT systems (with features) on different domain assignments for the source and the target domains. Finally, we also see that the features are very useful for both the MEMM and the RNNs.

3.3.7 Named Entity Recognition for Dutch

The previous sections have dealt with mention detection for English. In this section, we want to explore the capacity of the systems to quickly and effectively adapt to a new language. In particular, we evaluate the systems on the named entity recognition task (the simplified version of the MD task) for Dutch using the CoNLL 2002 dataset. The state-of-the-art performance for this dataset in the CoNLL evaluation is due to (Carreras et al., 2002) who utilize the AdaBoost classifier. In (Nothman et al., 2013), the authors leverage data from Wikipedia and

are able improve the state-of-the-art performance for Dutch. Very recently, while we are preparing this work, (Gillick et al., 2015) introduce a multilingual language processing system based on bytes and also report the performance on this dataset. Table 3.8 compares the systems.

System	P	R	F1
State-of-the-art in CoNLL	77.83	76.29	77.05
Nothman et al., 2013	-	-	78.60
Gillick et al., 2015	-	-	78.08
*Gillick et al., 2015**	-	-	*82.84*
MEMM	80.25	77.52	78.86
BASIC	82.98	81.53	82.25
BIDIRECT	84.08	82.82	**83.45**

Table 3.8: Performance on Dutch CoNLL 2002.

We note that the system in (Gillick et al., 2015) is also based on RNNs and the row labeled with * (Gillick et al., 2015) corresponds to the system trained on multiple datasets instead of the single CoNLL dataset for Dutch, so it is not comparable to ours.

The most important conclusion from the table is that the RNN models in this work significantly outperform MEMM as well as the other comparable system by large margins (up to 22% reduction in relative error). This proves that the proposed RNN systems are less subject to the language changes than MEMM and the other systems. Finally, BIDIRECT is also significantly better than BASIC, testifying to its robustness across languages.

3.4 Related Work

Both named entity recognition (Ando and Zhang, 2005; Bikel et al., 1997; Borthwick et al., 1997; Cherry and Guo, 2015; Florian et al., 2003; Lin and Wu, 2009; Miller et al., 2004; Passos et al., 2014; Ratinov and Roth, 2009; Ritter et al., 2011; Sang and Meulder, 2003; Suzuki and Isozaki, 2008; Turian et al., 2010) and mention detection (Florian et al., 2004) have been extensively studied with various evaluation in the last decades: MUC6, MUC7, CoNLL'02, CoNLL'03 and ACE. The previous work on MD has examined the cascade models (Florian et al., 2006), transferred knowledge from rich-resource languages to low-resource ones via machine translation (Zitouni and Florian, 2008) or improved the systems on noisy input (Florian et al., 2010). Besides, some recent work also tries to solve MD jointly with other tasks such as relation or event extraction to benefit from their inter-dependencies (Kate and Mooney, 2010; Li and Ji, 2014a; Li et al., 2014b; Roth and Yih, 2007). However, none of these work investigates RNNs for MD on the cross-domain and language settings as we do in this work.

Regarding neural networks for NER, Collobert et al., 2011 propose a CNN-based framework while Mesnil et al., 2013 and Yao et al., 2013, 2014 investigate the RNNs for the slot filling problem in spoken language understanding. Although our work also examines the RNNs, we consider the mention detection problem with an emphasis on the robustness of the models in the domain shifts and language changes which has never been explored in the literature before.

3.5 Conclusion

We systematically investigate various RNNs to solve the MD problem which suggests that bidirectional modeling is a very helpful mechanism for this task. In particular, the bidirectional model outperforms a very strong baseline of the feature-based exponential models in the cross-domain setting, thus demonstrating its robustness across domains. We also show that the bidirectional model is more portable to new languages as it is significantly better than the best reported systems for NER in Dutch (up to 22% reduction in relative error). In the future, we plan to apply the bidirectional modeling technique to other tasks as well as study the combination of different network architectures and resources to further improve the performance of the systems.

Chapter 4

Deep Learning for Relation Extraction

This chapter presents several deep learning models for relation extraction. There are two major parts in this chapter. The first part introduces a convolutional neural network that does not require feature engineering for relation extraction while the second part aims at the other extreme, exploring the combination of convolutional neural networks, recurrent neural networks and feature engineering to achieve the state-of-the-art RE performance. The works in this chapter have been published in (Nguyen and Grishman, 2015a) and (Nguyen and Grishman, 2016c).

4.1 Convolutional Neural Networks for Relation Extraction

The *relation extraction* (RE) task can be divided into two steps: detecting if a relation mention corresponding to some entity mention pair of interest represents some relation and classifying the detected relation mentions into some predefined classes. If we only need to categorize the given relation mentions that are known to express some expected relation (perfect detection), we are left with the *relation classification* (RC) task. One variation of relation classification is that one might have non-relation examples in the dataset but the number of those is comparable to the number of the other examples. The non-relation examples, therefore, can be treated as a usual relation class. Relation extraction, on the other hand, often comes with a tremendously unbalanced dataset where the number of the non-relation examples far exceeds the others, making relation extraction more challenging but more practical than relation classification. Our present work focuses on the relation extraction task with an unbalanced corpus.

As we can see in Chapter 2, traditional RE systems require extensive feature engineering and rely on existing natural language processing (NLP) modules to analyze relation mentions and extract features, including part of speech taggers, chunkers, name taggers, and parsers. Besides the costly effort for feature engineering, the traditional relation extractors might be subject to the error propagation introduced by the imperfect quality of the supervised existing NLP toolkits. For instance, all the tasks mentioned above are known to suffer from a performance

loss when they are applied to out-of-domain data (Blitzer et al., 2006; Daume, 2007; McClosky et al., 2010), causing the degradation of the RE systems based on them. In this section, we target an independent RE system that both avoids complicated feature engineering and minimizes the reliance on the supervised NLP modules for features, potentially alleviating the error propagation and advancing our performance in this area.

To be concrete, our relation extraction system is provided only with raw sentences marked with the positions of the two entities of interest[1]. The only elements we can derive from this structure are the words, the k-grams and their positions in the sentences, suggesting a paradigm in which relation mentions are represented by features that depend on these elements. Eventually, word embeddings that are capable of capturing latent semantic and syntactic properties of words (Bengio et al., 2003; Collobert and Westion, 2008; Mikolov et al., 2013b; Mnih and Hinton, 2008; Turian et al., 2010), and convolutional neural networks (CNN) that are able to recognize specific classes of k-gram and induce more abstract representations (Kalchbrenner et al., 2014) are a natural combination one should apply to obtain more effective representations for RE in this setting.

Convolutional neural networks (dating back to the 1980s) are a type of feed-forward artificial neural networks whose layers are formed by a convolution operation followed by a pooling operation (Kalchbrenner et al., 2014; LeCun et al., 1988). Recently, with the emerging interests of the community in deep learning, CNNs have been revived and effectively applied in various NLP tasks, including semantic

1. For evaluation purpose, we assume the positions of the two entities of interest in the sentences like most previous studies in this area. These are the only external features we need to achieve an end-to-end relation extractor.

parsing (Yih et al., 2014), search query retrieval (Shen et al., 2014a), sentence modeling and classification (Kalchbrenner et al., 2014; Kim, 2014), name tagging and semantic role labeling (Collobert et al., 2011). For relation classification and extraction, there are two prior works on CNNs for relation classification (Liu et al., 2013)[2] and (Zeng et al., 2014); however, at the time of publishing, there was no work on employing CNNs for relation extraction. This work was the first attempt to fill in that gap and serves as a baseline for future research in this area.

Our convolutional neural network is built upon that of (Kalchbrenner et al., 2014) and (Kim, 2014) which are originally proposed for sentence classification and modeling. We adapt the network for relation extraction by introducing the position embeddings to encode the relative distances of the words in the sentence to the two entities of interest. In contrast to the models in (Liu et al., 2013) and (Zeng et al., 2014) for relation classification that apply a single window size, our model for relation extraction incorporates various window sizes for convolutional filters, allowing the network to capture wider ranges of k-grams to be helpful for relation extraction. In addition, rather than initializing the word embeddings randomly as do (Liu et al., 2013) and fixing the randomly generated position embeddings during training as do (Zeng et al., 2014), we use pre-trained word embeddings for initialization and optimize both word embeddings and position embeddings as model parameters. More importantly, rather than using extrinsic features (either from human annotation or other pre-processing modules) to enrich the representation as do (Liu et al., 2013) and (Zeng et al., 2014), our model (adapted for RC

2. The title of the paper (Liu et al., 2013) on relation extraction is misleading since the authors actually do relation classification, according to the experimental description.

where entity heads are given) avoids the use of manual linguistic resources and supervised NLP toolkits constructed externally, utilizing word embeddings that can be trained automatically in an unsupervised framework as the only external resource for the whole system.

We explore different model architectures systematically and demonstrate that the best model performance is achieved when multiple window sizes are implemented and the word embeddings, once initialized by some "universal" embeddings, are allowed to adapt during the optimization process to reach an effective state for relation extraction. We evaluate our models on both relation classification and relation extraction tasks. For relation classification, experiments show that our model (without any external features and resources) outperforms the state-of-the-art models regardless of whether the external features are included in these models or not. For relation extraction, our model is significantly better than the baseline models that use the words and the embeddings themselves as the features.

4.1.1 Model

Our convolutional neural network for relation extraction consists of four main layers: (i) the look-up table to encode words in sentences by real-valued vectors, (ii) the convolutional layer to recognize k-grams, (iii) the pooling layer to determine the most relevant features and (iv) a logistic regression layer (a fully connected neural network with a softmax at the end) to perform classification (Collobert et al., 2011; Kalchbrenner et al., 2014; Kim, 2014). Figure 4.1 gives an overview of the network.

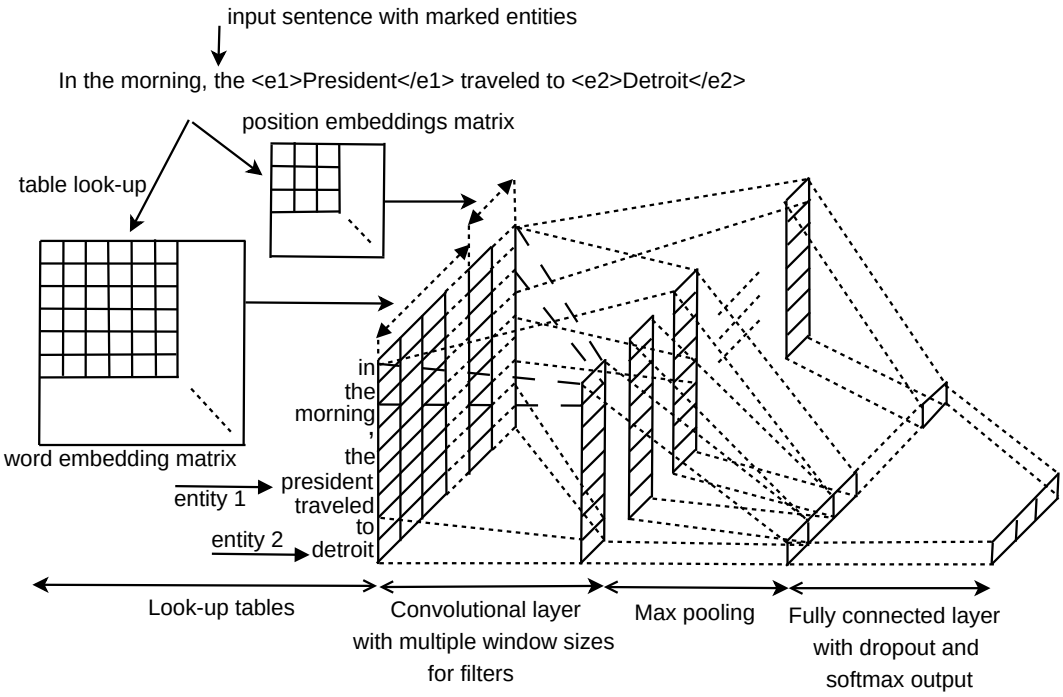

Figure 4.1: Convolutional Neural Network for Relation Extraction.

4.1.1.1 Word Representation

The input to the CNN for relation extraction consists of sentences marked with the two entity mentions of interest. In order to facilitate CNNs with fixed length inputs, we compute the maximal separation between entity mentions linked by a relation and choose an input width greater than this distance. We insure that every input (relation mention) has this length by trimming longer sentences and padding shorter sentences with a special token.

Let n be the length of the relation mentions and $W = [w_1, w_2, \ldots, w_n]$ be some relation mention where w_i is the i-th word in the mention. Also, let w_{i_1} and w_{i_2} be the two heads of the two entity mentions of interest. Before entering the

network, each word w_i is first transformed into a vector emb_i by looking up the word embedding table *EMB* that can be initialized either by a random process or by some pre-trained word embeddings. Besides, in order to embed the positions of the two entity heads as well as the other words in the relation mention into the representation, for each word w_i, its relative distances to the two entity heads $i-i_1$ and $i-i_2$ are also mapped into real-value vectors $dist_{i_1}$ and $dist_{i_2}$ respectively using a position embedding table *DIST* (initialized randomly) (Collobert et al., 2011; Liu et al., 2013; Zeng et al., 2014). Note that the relative distances only range from $-n+1$ to $n-1$ so the position embedding matrix *DIST* has size $(2n-1) \times m_d$ (m_d is a hyperparameter indicating the dimensionality of the position embedding vectors).

Finally, the word embeddings emb_i and the position embeddings $dist_1$ and $dist_2$ are concatenated into a single vector $x_i = [emb_i, dist_{i_1}, dist_{i_2}]^\top$ to represent the word w_i. As a result, the original sentence W can now be viewed as a matrix X of size $(m_e + 2m_d) \times n$ where m_e is the dimensionality of the word embedding vectors.

$$X = [x_1, x_2, \ldots, x_n] \tag{4.1}$$

4.1.1.2 Convolution

In the next step, the matrix X representing the input relation mention is fed into the convolutional layer to extract higher level features. Given a window size k, a filter is seen as a weight matrix $\mathbf{f} = [\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_k]$ ($\mathbf{f}_i$ is a column vector of size $m_e + 2m_d$). The core of this layer is obtained from the application of the

convolutional operator on the two matrices X and $\mathbf{f}$ to produce a score sequence $\mathbf{s} = [s_1, s_2, \ldots, s_{n-k+1}]$:

$$s_i = g(\sum_{j=0}^{k-1} \mathbf{f}_{j+1}^{\top} x_{j+i}^{\top} + b) \tag{4.2}$$

where b is a bias term and g is some non-linear function.

This process can then be replicated for various filters with different window sizes to increase the k-gram coverage of the model.

For relation extraction, we call the k-grams accompanied with relative positions of its words the augmented k-grams. It is instructive to think about the filter $\mathbf{f}$ as representing some hidden class of the augmented k-grams and the scores s_i as measuring the possibility the augmented k-gram at position i belongs to the corresponding hidden class (although these scores are not probabilities at all). The trained weights of the filter $\mathbf{f}$ would then amount to a feature detector that learns to recognize the hidden class of the augmented k-grams (Kalchbrenner et al., 2014).

4.1.1.3 Pooling

The rationale of the pooling layer is to further abstract the features generated from the convolutional layer by aggregating the scores for each filter to introduce the invariance to the absolute positions but preserve the relative positions of the k-grams between themselves and the entity heads at the same time. The popular aggregating function is max as it bears responsibility for identifying the most important or relevant features from the score sequence. Concretely, for each filter $\mathbf{f}$, its score sequence $\mathbf{s}$ is passed through the max function to produce a single number:

$p_{\mathbf{f}}^k = \max\{\mathbf{s}\} = \max\{s_1, s_2, \ldots s_{n-w+1}\}$ which can be interpreted as estimating the possibility some augmented k-gram of the hidden class of $\mathbf{f}$ appears in the context.

4.1.1.4 Regularization and Classification

In the final step, the pooling scores for every filter are concatenated into a single feature vector $\mathbf{z} = [p_1, p_2, \ldots, p_m]$ to represent the relation mention. Here, m is the number of filters in the model and p_i is the pooling score of the i-th filter. Before actually applying this feature vector, following (Kim, 2014; Srivastava et al., 2014), we execute a dropout for regularization by randomly setting to zero a proportion ρ of the elements of the feature vector[3] $\mathbf{z}$ to produce the vector $\mathbf{z}_d$. The dropout vector $\mathbf{z}_d$ is then fed into a fully connected layer of standard neural networks followed by a softmax layer in the end to perform classification. The fully connected layer induces a weight matrix *FULL* as model parameters.

At test time, the unseen relation mentions are scored using the feature vectors that are not dropped out. We also rescale the weights whose Frobenius norms exceed a hyperparameter as (Kim, 2014).

Overall, the parameters for the presented CNN are: the word embedding matrix *EMB*, the position embedding matrix *DIST*, the m filter matrices, and the weight matrix *FULL* for the fully connected layer. The gradients are computed using back-propagation while training is done via stochastic gradient descent with shuffled mini-batches and the AdaDelta update rule (Kim, 2014; Zeiler, 2012).

3. Following the Bernoulli distribution.

4.1.2 Experiments

4.1.2.1 Hyperparameters and Resources

For all the experiments below, we use: tanh for the non-linear function, 150 filters for each window size in the model and position embedding vectors with dimensionality of $m_d = 50$[4]. Regarding the other parameters, we use the same values as do (Kim, 2014), i.e, the dropout rate $\rho = 0.5$, the mini-batch size of 50, the hyperparameter for the Frobenius norms of 3.

Finally, we utilize the pre-trained word embeddings `word2vec` from (Mikolov et al., 2013b) which have dimensionality of $m_e = 300$ and are trained on 100 billion words of Google News using the continuous bag-of-words architecture. These embeddings are publicly available here[5]. Vectors for the words not included in the pre-trained embeddings are initialized randomly. Besides the word embeddings `word2vec`, the model does not use any other NLP toolkits or resources.

4.1.2.2 Datasets

We evaluate our models on two datasets: the SemEval-2010 Task 8 dataset (Hendrickx et al., 2010) for relation classification and the ACE 2005 dataset for relation extraction.

The SemEval dataset can be downloaded here[6] and contains 10,717 annotated examples, including 8,000 examples for training and 2,717 examples for testing. Each example is a sentence annotated for a pair of entities of interest and the

4. These values produce the best performance during our experimental process.
5. `https://code.google.com/p/word2vec`
6. `http://docs.google.com/View?id=dfvxd49s36c28v9pmw`

corresponding relation class for this entity pair. There are 9 ordered relationships (with two directions) and an undirected *Other* class, resulting in 19 classes. A pair is counted as correct if the order of the entities in the relationship is correct. For the ACE 2005 dataset, documents are annotated for 6 major relation classes and 7 entity types. In order to generate the non-relation examples or the examples for the *Other* class, we collect every pair of entity mentions within a single sentence and not included in the annotated relation set. To reduce the noise, we truncate the generated dataset by removing all the examples whose distances between the two entity heads are greater than 15. This results in a considerably unbalanced dataset of 8,365 positive examples of the 6 annotated relation classes and 79,147 negative examples of the class *Other*. The distributions of the relation classes on the two datasets are shown in Table 4.1. As we can see, the ACE dataset is much more biased toward the *Other* class than the SemEval dataset and thus more appropriate for relation extraction experiments.

4.1.2.3 Evaluation of Model Architectures

We investigate the effectiveness of different window sizes of filters by running the proposed CNN model on window sizes of 2, 3, 4 and 5. To understand the behavior of the model on multiple window sizes, we further test it on the following window size combinations: (4,5), (3,4,5) and (2,3,4,5). In each of these window size configurations, we evaluate the system on three different scenarios: (i) the word embeddings and the position embeddings are randomly initialized and optimized

ACE 2005 (87,512)		**SemEval 2010 (10,717)**	
Relation	%	Relation	%
ORG-AFF	2.8	Cause-Effect	12.4
PER-SOC	1.2	Component-Whole	11.7
ART	1.0	Entity-Destination	10.6
PART-WHOLE	1.4	Entity-Origin	9.1
GEN-AFF	1.1	Product-Producer	8.8
PHYS	2.1	Member-Collection	8.6
Other	**90.4**	Message-Topic	8.4
		Content-Container	6.8
		Instrument-Agency	6.2
		Other	**17.4**

Table 4.1: ACE 2005 and SemEval 2010 relation class distributions.

during the training process (denoted by *nonstatic.rand*), (ii) the word embeddings are initialized by the pre-trained word embeddings; the position embeddings are initialized randomly and the two embeddings are kept unchanged during the training (denoted by *static.word2vec*), (iii) the two embeddings are initialized as in case (ii) but they are optimized as model parameters when the model is trained (denoted by *nonstatic.word2vec*). These experiments are carried out for relation extraction on the ACE 2005 dataset via 5-fold cross validation. Table 4.2 presents the system performance on Precision (P), Recall (R) and F1 score (F).

The key observations from the table are[7]:

(i) From rows 1, 2, 3, 4, we see that evaluating window sizes individually is quite intricate. It is unclear which window size is the best size for CNNs on relation extraction. For instance, on the *nonstatic.rand* mode, the window size 4 seems to outperform the others while on the other modes, the window sizes 3 and 5 turn out

7. The statements at points (ii) and (iii) are significant at confidence levels $\geq 95\%$.

		nonstatic.rand			static.word2vec			nonstatic.word2vec		
#	window sizes	P	R	F	P	R	F	P	R	F
1	2	69.56	41.64	52.04	74.66	41.03	52.90	72.74	49.49	58.87
2	3	68.47	42.73	52.57	74.19	42.16	53.73	72.50	50.75	59.66
3	4	68.17	43.39	52.94	73.60	41.90	53.35	72.56	49.81	58.97
4	5	66.83	43.46	52.55	73.52	42.60	53.89	71.70	51.08	59.57
5	4-5	66.18	46.12	54.25	72.69	45.23	55.71	71.88	52.36	60.50
6	3-4-5	67.54	45.73	54.43	71.99	46.85	56.73	71.21	53.24	60.86
7	2-3-4-5	66.42	47.20	55.12	72.60	46.77	56.85	71.25	53.91	**61.32**

Table 4.2: System performance on various window size combinations and architectures.

to be better. Besides, the performance gaps between the window sizes are small, making it hard to draw a conclusive judgement. In any case, the window size 2 seems to be the worst, suggesting that the 2-grams might be less informative than the others on representing relation mentions for CNNs on this dataset.

(ii) While the results on evaluating single window sizes are hard to analyze, the results for multiple window sizes are quite clear and conclusive. Moving from single window sizes of 2, 3, 4 or 5 (rows 1, 2, 3 and 4 respectively) to the configuration with two window sizes 4 and 5 (row 5) gives us consistent improvements on all the model architectures. The performance is then consistently enhanced when more window sizes are included, resulting in the best performance when all the window sizes 2, 3, 4 and 5 are employed. This demonstrates the advantages of the models with multiple window sizes over the single window size models in (Liu et al., 2013) and (Zeng et al., 2014).

(iii) Regarding different model architectures, the picture is even clearer. No matter which window size configuration is applied, we consistently see the *non-static.word2vec* architecture performs most effectively, followed by the *static.word2vect*

setting which is in turn followed by the *nonstatic.rand* model. This suggests the undeniable benefits of initializing the word embeddings by some "universal" pre-trained values and updating the embeddings to reflect RE specific embeddings when training the models (Collobert et al., 2011; Kim, 2014). For the next experiments, we always use all the window sizes 2, 3, 4 and 5 with the *nonstatic.word2vec* architecture.

4.1.2.4 Relation Extraction Experiment

We compare our system with the traditional feature-based relation extraction systems when these systems are only allowed to use the same information and resources as our systems, i.e, the words in the relation mentions, the positions of the two entity heads and the word embeddings.

Given the sentences and the positions of the two entity heads, the features extracted by the state-of-the-art feature-based systems include: the heads of the two entity mentions; the words in the context before mention 1; after mention 2 and between two mentions; the bigrams, the word sequences between two entities, the order of two mentions, the number of words between two mentions (Jiang and Zhai, 2007a; Sun et al., 2011; Zhou et al., 2005). The feature-based system using this feature set is called *Words*. Armed with the word embeddings, one can further introduce these embeddings into the head words or the words in the context as additional features (Nguyen and Grishman, 2014a). We call the system *Words* augmented with the embeddings for the two heads *Words-HM-Wed* and *Words* augmented with the embeddings for words in the contexts *Words-WC-Wed*. We apply

the MaxEnt framework with L2 regularization in the Mallet toolkit[8] to train these feature-based models (as (Jiang and Zhai, 2007a; Nguyen and Grishman, 2014a; Sun et al., 2011)). Table 4.3 shows the performance of the three baseline systems and our proposed CNN via 5-fold cross validation on the ACE 2005 dataset.

System	P	R	F
Words	54.95	43.73	48.69
Words-WC-Wed	50.10	44.47	47.11
Words-HM-Wed	57.01	55.74	56.36
Our CNN	71.25	53.91	**61.32**

Table 4.3: Performance of relation extraction systems.

The first observation is that adding the word embeddings to the words in the context hurt the performance of the feature-based systems while augmenting the heads of the entities with word embeddings significantly improves the feature-based systems. This is consistent with the results reported by (Nguyen and Grishman, 2014a) and demonstrates that the ability to wisely pick the words for embeddings and avoid embeddings on specific locations is crucial to the feature-based systems. More importantly, our proposed CNN significantly outperforms all the baseline models at the confidence levels $\geq 95\%$, an improvement of 4.96% over the best feature-based system *Words-HM-Wed* (Nguyen and Grishman, 2014a). This result indicates that CNNs are a better way to employ word embeddings for relation extraction.

Remember that although the traditional systems can achieve a performance greater than 72% on the ACE dataset (Qian et al., 2008; Sun et al., 2011), they

8. `http://mallet.cs.umass.edu`

come at the expense of elaborate feature engineering as well as much more expensive feature extraction. In particular, the feature extractors of these feature-based systems require: (i) the perfect entity and mention type information hand-labeled laboriously by human annotators; (ii) the extensive usage of the existing supervised NLP toolkits and resources (constituent and dependency parsers, dictionaries, gazetteers etc) which might be unavailable for various domains in reality. The absence of the perfect (hand-annotated) entity and mention type information (i.e, point (i) above) greatly impairs these feature-based systems' performance. For instance, both (Plank and Moschitti, 2013) and (Nguyen and Grishman, 2014a) report a performance less than 60% on the ACE 2005 dataset when the perfect entity type and mention type features are not employed although the other features with extensive feature engineering (i.e point (ii) above) are still included. As a result, in a more realistic setting where hand-annotated features are prohibitive, the proposed CNN requires much less feature engineering and resources but still performs better than the traditional feature-based systems.

4.1.2.5 Relation Classification Experiment

In order to further verify the effectiveness of the system, we test the system on the relation classification task with the SemEval 2010 dataset and compare the results with the state-of-the-art systems in this area. Table 4.9 describes the performance of various traditional systems that are based on classifiers such as MaxEnt and SVM with series of supervised and manual features[9] (Hendrickx et al., 2010)

9. i.e, the features extracted from supervised pre-processing NLP modules and manual resources.

as well as the more recent systems based on convolutional neural networks (Zeng et al., 2014) (O-CNN), recursive neural networks (RNN), matrix-vector recursive neural networks (MVRNN) (Socher et al., 2012b) or log-quadratic factor-based compositional embedding model (FCM) (Yu et al., 2014)[10].

As we can see, among the systems *not using any supervised and manual features* (i.e, POS, WordNet, name tagging, dependency parse, patterns etc), our system significantly outperforms the state-of-the-art system FCM (80.6%) (Yu et al., 2014) with an improvement of 2.2%. More interestingly, even without supervised and manual features, our system can still work comparably to the other systems utilizing these features as the vital components. For instance, the supervised features (dependency parse and name tagging) are crucial to FCM (Yu et al., 2014) to significantly improve its performance. We attribute our performance advantage over the closely-related system O-CNN (Zeng et al., 2014) to the multiple window sizes, the optimization of the position embeddings during training and possibly the superiority of the embeddings `word2vec` we use.

4.1.2.6 Impact of Unbalanced Dataset

Shifting from relation classification to relation extraction with an unbalanced corpus, we witness a large performance gap as described above. In this section, we study the impact of the unbalanced corpus on the performance of relation extractors for both convolutional neural networks and traditional feature-based approaches (*Words* and *Words-HM-Wed*). In particular, we vary the ratio of positive (true relations) and negative (the class *Other*) examples in the ACE 2005 dataset

10. These are the macro-averaged F1-scores, computed by the officially provided scorer.

Classifier	Feature Sets	F
SVM	POS, WordNet, morphological features, thesauri, Google k-grams	77.6
MaxEnt	POS, WordNet, morphological features, noun compound system, thesauri, Google k-grams	77.6
SVM	POS, WordNet, prefixes and other morphological features, dependency parse, Levin classes, PropBank, FrameNet, NomLex-Plus, Google k-grams, paraphrases, TextRunner	82.2
RNN	-	74.8
RNN	POS, name tagging, WordNet	77.6
MVRNN	-	79.1
MVRNN	POS, name tagging, WordNet	82.4
O-CNN	-	78.9
O-CNN	WordNet	82.7
FCM	-	80.6
FCM	dependency parse, name tagging	83.0
Our CNN	-	**82.8**

Table 4.4: Performance of relation classification systems.

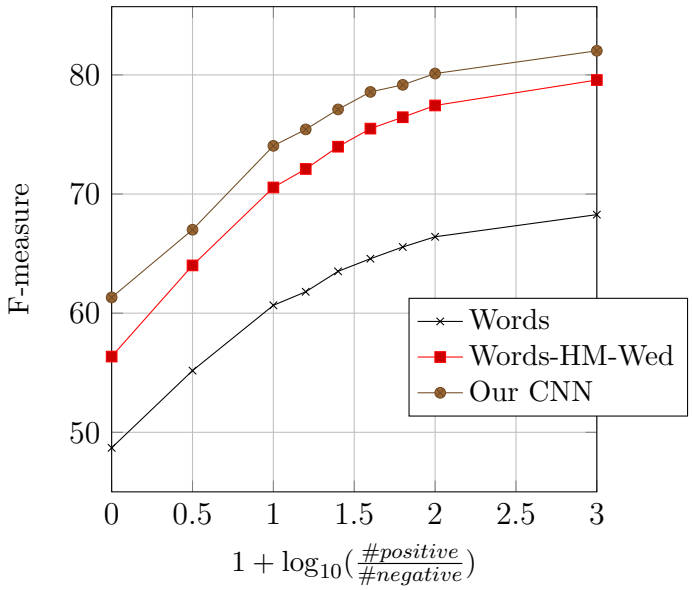

Figure 4.2: F measures vs positive/negative ratios

and see how the system performance responds to this variation. Figure 4.2 shows the curves. This is a 5-fold cross validation experiment and all the comparisons are significant at confidence levels $\geq 95\%$.

From the figure, we see that all the models improve consistently with the increase of the ratio of the positive and negative examples. The performance peaks with an improvement of about 20% for all the models when the number of examples of the class *Other* is small relative to the others. In other words, the systems attain their best performance when relation extraction is reduced to the relation classification problem, suggesting that relation extraction is much more challenging than relation classification. Finally, for all the ratio values, we consistently see that the convolutional neural network is superior to the others, once again

confirming its advantages.

4.2 Combining Neural Networks and Log-linear Models to Improve Relation Extraction

As we have discussed in Chapter 2, the feature-based method for RE extensively leveraged linguistic analysis and knowledge resources to construct the feature representations, involving the combination of *discrete* properties such as lexicon, syntax, and gazetteers. This approach is able to exploit the symbolic (discrete) structures within relation mentions; however they suffer from the difficulty to generalize over the unseen words (Gormley et al., 2015), motivating some recent work on employing the *continuous* representations of words (word embeddings) to do RE (Nguyen and Grishman, 2014a; Nguyen et al., 2015c). The most popular method involves neural networks (NNs) that effectively learn hidden and continuous structures of relation mentions from such word embeddings, thus achieving the top performance for RE (Nguyen and Grishman, 2015a; Santos et al., 2015a; Xu et al., 2015; Zeng et al., 2014).

The NN research for relation extraction and classification so far has centered around two main network architectures: convolutional neural networks (CNN)[11] (Nguyen and Grishman, 2015a; Santos et al., 2015a; Zeng et al., 2015) and recursive/recurrent neural networks (Socher et al., 2012b; Xu et al., 2015). The distinction between convolutional neural networks and recurrent neural networks

11. as we can see in the previous section.

(RNN) for RE is that the former aim to generalize the short and consecutive context (i.e, the k-grams) of the relation mentions (Lei et al., 2015; Nguyen and Grishman, 2015a) while the latter adaptively accumulate the context information in the whole sentence via the memory units, thereby encoding the long and possibly non-consecutive patterns for RE (Hochreiter and Schmidhuber, 1997). Consequently, the traditional feature-based method (i.e, the *log-linear* or MaxEnt model with hand-crafted and discrete features), the CNNs and the RNNs tend to focus on different angles for RE. Guided by this intuition, in this work, we propose to combine the three models to further improve the performance of RE. Note that this is in contrast to the work in Section 4.1 that avoids the use of feature engineering with discrete features completely.

While the architecture design of CNNs for RE is quite established due to the extensive studies in the last couple of years, the application of RNNs to RE is only very recent and the optimal designs of RNNs for RE are still a subject of ongoing research. In this work, we first perform a systematic exploration of various network architectures to seek the best RNN model for RE. In the next step, we extensively study different methods to assemble the log-linear model, CNNs and RNNs for RE, leading to the combined models that yield the state-of-the-art performance on the ACE 2005 and SemEval datasets. To the best of our knowledge, this is the first work to systematically examine the RNN architectures as well as combine them with CNNs and the traditional feature-based approach for RE.

4.2.1 Models

Relation mentions consist of sentences marked with two entity mentions of interest. In this section, we examine two different representations for the sentences in RE: (i) the standard representation, called SEQ that takes all the words in the sentences into account and (ii) the dependency representation, called DEP that only considers the words along the dependency paths between the two entity mention heads of the sentences. In the following, unless indicated specifically, all the statements about the sentences hold for both representations SEQ and DEP.

Following Section 4.1, we assume that the input sentences of the relation mentions have the same fixed length n. We also denote $W = [w_1, w_2, \ldots, w_n]$ as the input sentence of some relation mention, where w_i is the i-th word in the sentence, and w_{i_1} and w_{i_2} are the two heads of the two entity mentions of interest. In order to prepare the relation mention for neural networks in this section, we also transform each word w_i into a real-valued vector x_i. However, this section uses the concatenation of the following seven vectors, motivated by the previous research on feature analysis for RE (Gormley et al., 2015; Sun et al., 2011; Zeng et al., 2014; Zhou et al., 2005):

- The real-valued word embedding vector emb_i of w_i, obtained by looking up the word embedding table *EMB*.

- The real-valued distance embedding vectors $dist_{i_1}$, $dist_{i_2}$ to encode the relative distances $i - i_1$ and $i - i_2$ of w_i to the two entity heads of interest w_{i_1} and w_{i_2}: $dist_{i_1} = DIST[i - i_1]$, $dist_{i_2} = DIST[i - i_2]$ where $DIST$ is the position embedding table (initialized randomly).

- The real-valued embedding vectors for entity types ent_i and chunks $chunk_i$ to embed the entity type and chunking information for w_i. These vectors are generated by looking up the entity type and chunk embedding tables (also initialized randomly) (i.e, *ENT* and *CHUNK* respectively) for the entity type and chunking label of w_i: $ent_i = ENT$[entity type label of w_i], $chunk_i = CHUNK$[chunking label of w_i].

- The binary vector dep_i with one dimension to indicate whether the word w_i is on the dependency path between w_{i_1} and w_{i_2} or not.

- The binary vector rel_i whose dimensions correspond to the possible relations between words in the dependency trees. The value at a dimension of rel_i is only set to 1 if there exists one edge of the corresponding relation connected to w_i in the dependency tree.

The transformation from the word w_i to the vector $x_i = [emb_i, dist_{i_1}, dist_{i_2}, ent_i, chunk_i, dep_i, rel_i]$ essentially converts the relation mention with the input sentence W into a real-valued matrix $X = [x_1, x_2, \ldots, x_n]$, to be used by the neural networks presented below.

4.2.1.1 The Separate Models

We review two typical NN architectures for RE underlying the combined models in this work.

The Convolutional Neural Networks

In CNNs (Kalchbrenner et al., 2014; Nguyen and Grishman, 2015a)[12], given a window size of k, we have a set of feature maps (filters). Each feature map

12. The CNN model in this section is very similar to that of Section 4.1. The only difference is in the representations of the words in the sentences. We review it here to introduce necessary notations for this section.

$\mathbf{f}$ is a weight matrix $\mathbf{f} = [\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_k]$ where $\mathbf{f}_i$ is a vector to be learned during training as the model parameters. The core of CNNs is the application of the convolutional operator on the input matrix X and the filter matrix $\mathbf{f}$ to produce a score sequence (also called the hidden vector) $\mathbf{s} = [s_1, s_2, \ldots, s_{n-k+1}]$, interpreted as a more abstract representation of the input matrix X:

$$s_i = g(\sum_{j=0}^{k-1} \mathbf{f}_{j+1} x_{j+i} + b) \tag{4.3}$$

where b is a bias term and g is the tanh function.

In the next step, we further abstract the scores in $\mathbf{s}$ by aggregating it via the max function to obtain the max-pooling score. We then repeat this process for all the feature maps with different window sizes k to generate a vector of the max-pooling scores. In the final step, we pass this vector into some standard multilayer neural network, followed by a softmax layer to produce the probabilistic distribution $P_{\mathrm{C}}(y|X)$ over the possible relation classes y in the prediction task.

The Recurrent Neural Networks

In RNNs, we consider the input matrix $X = [x_1, x_2, \ldots, x_n]$ as a sequence of column vectors indexed from 1 to n. At each step i, we compute the hidden vector α_i from the current input vector x_i and the previous hidden vector α_{i-1} using the non-linear transformation function ϕ: $\alpha_i = \Phi(x_i, \alpha_{i-1})$.

This recurrent computation can be done via three different directional mechanisms: (i) the forward mechanism that recurs from 1 to n and generates the forward hidden vector sequence: $\overrightarrow{RNN}([x_1, x_2, \ldots, x_n]) = [\overrightarrow{h_1}, \overrightarrow{h_2}, \ldots, \overrightarrow{h_n}]$, (ii) the backward mechanism that runs RNNs from n to 1 and results in the backward hid-

den vector sequence $\overleftarrow{RNN}([x_n, x_{n-1}, \ldots, x_1]) = [\overleftarrow{h_n}, \overleftarrow{h_{n-1}}, \ldots, \overleftarrow{h_1}]$[13], and (iii) the bidirectional mechanism that performs RNNs in both directions to produce the forward and backward hidden vector sequences, and then concatenate them at each position to generate the new hidden vector sequence $[h_1, h_2, \ldots, h_n]$: $h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}]$.

Given the hidden vector sequence $[h_1, h_2, \ldots, h_n]$ obtained from one of the three mechanisms above, we study the following two strategies to generate the representation vector v^R for the initial relation mention. Note that this representation vector can be again fed into some standard multilayer neural network with a softmax layer in the end, resulting in the distribution $P_{\mathrm{R}}(y|X)$ for the RNN models:

- The *HEAD* strategy: In this strategy, v^R is the concatenation of the hidden vectors at the positions of the two entity mention heads of interest: $v^R = [h_{i_1}, h_{i_2}]$. This is motivated by the importance of the two mention heads in RE (Nguyen and Grishman, 2014a; Sun et al., 2011).

- The *MAX* strategy: This strategy is similar to our max-pooling mechanism in CNNs. In particular, v^R is obtained by taking the maximum along each dimension of the hidden vectors $h_1, h_2, \ldots, h_n$. The idea is to further abstract the hidden vectors by retaining only the most important feature in each dimension.

Regarding the non-linear function, the simplest form of Φ in the literature considers it as a one-layer feed-forward neural network, called *FF*: $h_i = FF(x_i, h_{i-1}) = \Phi(Ux_i + Vh_{i-1})$ where ϕ is the *sigmoid* function, U and V are weight matrices. Unfortunately, the application of *FF* is prone to the "*vanishing gradient*" problem (Bengio et al., 1994), making it challenging to train RNNs properly. In this work,

13. The initial hidden vectors are set to the zero vector.

we also use *Gated Recurrent Units* (Cho et al., 2014a) (*GRU*) to alleviate this problem (as in Section 3.1.2).

4.2.1.2 The Combined Models

We first present three different methods to assemble CNNs and RNNs: ensembling, stacking and voting, to be investigated in this work. The combination of the neural networks with the log-linear model would be discussed in the next section.

1. Ensembling

In this method, we first run some CNN and RNN in Section 4.2.1.1 over the input matrix X to gather the corresponding distributions $P_C(y|X)$ and $P_R(y|X)$. We then combine the CNN and RNN by multiplying their distributions (element-wise): $P_{\text{ensemble}}(y|X) = \frac{1}{Z}P_{\text{C}}(y|X)P_{\text{R}}(y|X)$ (Z is a normalization constant).

2. Stacking

The overall architecture of the stacking method is to use one of the two network architectures (i.e, CNNs and RNNs) to generalize the hidden vectors of the other architecture. The expectation is that we can learn more effective features for RE via such a deeper architecture by alternating between the local and global representations provided by CNNs and RNNs.

We examine two variants for this method. The first variant, called *RNN-CNN*, applies the CNN model in Section 4.2.1.1 on the hidden vector sequence generated by some RNN in Section 4.2.1.1 to perform RE. The second variant, called *CNN-RNN*, on the other hand, utilizes the CNN model to acquire the hidden vector sequence, that is, in turn, fed as the input into some RNN for RE. For the second

variant, as the length of the hidden vector $\mathbf{s} = [s_1, s_2, \ldots, s_{n-k+1}]$ in the CNN model depends on the specified window size k for the corresponding feature map $\mathbf{f}$, we need to pad the input matrix X with $\lfloor \frac{k}{2} \rfloor$ zero column vectors on both sides to ensure the same fixed length n for all the hidden vectors: $\mathbf{s} = [s_1, s_2, \ldots, s_n]$. Besides, we need to re-arrange the scores in the hidden vectors from different feature maps of the CNN so they are grouped according to the positions in the sentence, thus being compatible with the input requirement of RNNs.

3. Voting

Instead of integrating CNNs and RNNs at the model level as the two previous methods, the voting method makes decisions for a relation mention X by voting the individual decisions of the different models. While there are several voting schemes in the literature, for this work, we employ the simplest scheme of majority voting. If there is more than one relation class receiving the highest number of votes, the relation class returned by a model and having the highest probability would be chosen.

4.2.2 The Hybrid Models

In order to further improve the RE performance of models above, we investigate the integration of these neural network models with the traditional log-linear model that relies on various linguistic features from the past research on RE (Gormley et al., 2015; Sun et al., 2011; Zhou et al., 2005). Specifically, in such integration models (called *the hybrid models*), the relation class distribution is obtained from the element-wise multiplication between the distributions of the neural network

models and the log-linear model. Let us take the ensembling model in Section 4.2.1.2 as an example. The corresponding hybrid model in this case would be: $P_{\text{hybrid}}(y|X) = \frac{1}{Z} P_{\text{C}}(y|X) P_{\text{R}}(y|X) P_{\text{login}}(y|X)$, assuming $P_{\text{login}}(y|X)$ is the distribution of the log-linear model and Z is the normalization constant. The parameters of the log-linear model are learnt jointly with the parameters of the neural networks.

Hypothesis: Let S be the set of relation mentions correctly predicted by some neural network model in some dataset (the coverage set). The introduction of the log-linear model into this neural network model essentially changes the coverage set of the network, resulting in the new coverage set S' that might or might not subsume the original set S. In this work, we hypothesize that although S and S' overlap, there are still some relation mentions that only belong to either set. Consequently, we propose to implement a majority voting system (called the *hybrid-voting system*) on the outputs of the network and its corresponding hybrid model to enhance both models.

Note that the voting models in Section 4.2.1.2 involve the voting on two models (i.e, CNN and RNN). In order to integrate the log-linear model into such voting models, we first augment the separate CNN and RNN models with the log-linear model before we perform the voting procedure on the resulting models. Finally, the corresponding *hybrid-voting* systems would involve the voting on four models (CNN, hybrid CNN, RNN and hybrid RNN).

4.2.2.1 Training

We train the models by minimizing the negative log-likelihood function using the stochastic gradient descent algorithm with shuffled mini-batches and the AdaDelta update rule (Zeiler, 2012). The gradients are computed via back-propagation while regularization is executed by dropout on the hidden vectors before the multi-layer neural networks (Srivastava et al., 2014). During training, besides the weight matrices, we also optimize the embedding tables $EMB, DIST, ENTandCHUNK$ to achieve the optimal state. Finally, we rescale the weights whose Frobenius norms exceed a hyperparameter (Nguyen and Grishman, 2015a).

4.2.3 Experiments

4.2.3.1 Resources and Parameters

For all the experiments below, we utilize the pre-trained word embeddings `word2vec` with 300 dimensions from (Mikolov et al., 2013b) to initialize the word embedding table EMB. The parameters for CNNs and training the networks are inherited from the previous studies, i.e, the window size set for feature maps = $\{2, 3, 4, 5\}$, 150 feature maps for each window size, 50 dimensions for all the embedding tables (except the word embedding table EMB), the dropout rate = 0.5, the mini-batch size = 50, the hyperparameter for the Frobenius norms = 3 (Nguyen and Grishman, 2015a). Regarding RNNs, we employ 300 units in the hidden layers.

4.2.3.2 Dataset

We evaluate our models on two datasets: the ACE 2005 dataset for relation extraction and the SemEval-2010 Task 8 dataset (Hendrickx et al., 2010) for relation classification.

The ACE 2005 corpus comes with 6 different domains: broadcast conversation (`bc`), broadcast news (`bn`), telephone conversation (`cts`), newswire (`nw`), usenet (`un`) and weblogs (`wl`). Similar to the previous chapters, following the common practice of domain adaptation research on this dataset (Gormley et al., 2015; Nguyen and Grishman, 2014a; Nguyen et al., 2015c; Plank and Moschitti, 2013), we use `news` (the union of `bn` and `nw`) as the training data, half of `bc` as the development set and the remainder (`cts`, `wl` and the other half of `bc`) as the test data. Note that we are using the data prepared by (Gormley et al., 2015), thus utilizing the same data split on `bc` as well as the same data processing and NLP toolkits. The total number of relations in the training set is 43,497[14]. We employ the BIO annotation scheme to capture the chunking information for words in the sentences and only mark the entity types of the two entity mention heads (obtained from human annotation) for this dataset.

The SemEval dataset concerns the relation classification task that aims to determine the relation type (or no relation) between two entities in sentences. In order to make it compatible with the previous research (Gormley et al., 2015; Socher et al., 2012b), for this dataset, besides the word embeddings and the distance embeddings, we apply the name tagging, part of speech tagging and WordNet features

14. It was an error in (Gormley et al., 2015) that reported 43,518 total relations in the training set. The authors acknowledged this error.

(inherited from (Socher et al., 2012b) and encoded by the real-valued vectors for each word). The other settings are also adopted from the past studies (Socher et al., 2012b; Xu et al., 2015).

4.2.3.3 RNN Architectures

This section evaluates the performance of various RNN architectures for RE on the ACE 2005 development set. In particular, we compare different design combinations of the following four factors: (i) sentence representations (i.e, SEQ or DEP), (ii) transformation functions Φ (i.e, FF or GRU), (iii) the strategies to employ the hidden vector sequence for RE (i.e, HEAD or MAX), and (iv) the directions to run RNNs (i.e, forward ($\rightarrow$), backward ($\leftarrow$) or bidirectional ($\rightleftharpoons$)). Table 4.5 presents the results.

Systems			DEP	SEQ
FF	HEAD	$\rightleftharpoons$	60.78	63.22
		$\rightarrow$	55.55	60.05
		$\leftarrow$	57.69	58.54
	MAX	$\rightleftharpoons$	50.00	51.22
		$\rightarrow$	52.08	53.96
		$\leftarrow$	45.07	33.50
GRU	HEAD	$\rightleftharpoons$	63.32	63.23
		$\rightarrow$	63.69	62.77
		$\leftarrow$	61.57	62.55
	MAX	$\rightleftharpoons$	60.96	**64.24**
		$\rightarrow$	61.97	**64.59**
		$\leftarrow$	61.56	**64.30**

Table 4.5: Performance (F1 scores) of RNNs on the dev set.

The main conclusions include:

(i) Assuming the same choices for the other three corresponding factors, GRU is more effective than FF, SEQ is better than DEP most of the time, and HEAD outperforms MAX (except in the case where SEQ and GRU are applied) for RE with RNNs. Note that the improvement of SEQ over DEP can be partly explained by the domination of the relation mentions with short distances between the two entity mentions in the ACE 2005 dataset.

(ii) Regarding the direction mechanisms, the bidirectional mechanism achieves the best performance for the HEAD strategy while the forward direction is the best mechanism for the MAX strategy. This can be partly explained by the lack of past or future context information in the HEAD strategy when we follow the backward or forward direction respectively.

The best performance corresponds to the application of the SEQ representation, the GRU function and the MAX strategy that would be used in all the RNN models below. We call such RNN models with the forward, backward and bidirectional mechanism **FORWARD**, **BACKWARD** and **BIDIRECT** respectively. We also apply the SEQ representation for the CNN model (called CNN) in the following experiments for consistency.

4.2.3.4 Evaluating the Combined Models

We evaluate the combination methods for CNNs and RNNs presented in Section 4.2.1.2. In particular, for each method, we examine three models that are combined from one of the three RNN models FORWARD, BACKWARD, BIDIRECT and the CNN model. For instance, in the stacking method, the three combined mod-

els corresponding to the *RNN-CNN* variant are FORWARD-CNN, BACKWARD-CNN, BIDIRECT-CNN while the three combined models corresponding to the *CNN-RNN* variant are CNN-FORWARD, CNN-BACKWARD, CNN-BIDIRECT. The notations for the other methods are self-explanatory. The model performance on the development set is given in Table 4.6 that also includes the performance of the separate models (i.e, CNN, FORWARD, BACKWARD, BIDIRECT) for convenient comparison.

Model	P	R	F1
BIDIRECT	69.16	59.97	64.24
FORWARD	69.33	60.45	64.59
BACKWARD	65.60	63.05	64.30
CNN	68.35	59.16	63.42
Ensembling			
CNN-BIDIRECT	71.22	54.13	61.51
CNN-FORWARD	66.19	59.64	62.75
CNN-BACKWARD	65.09	60.13	62.51
Stacking			
CNN-BIDIRECT	66.55	59.97	63.09
CNN-FORWARD	69.46	63.05	**66.10**
CNN-BACKWARD	72.58	58.35	64.69
BIDIRECT-CNN	65.63	61.59	63.55
FORWARD-CNN	73.13	58.67	65.11
BACKWARD-CNN	67.60	58.51	62.73
Voting			
CNN-BIDIRECT	71.08	60.94	**65.62**
CNN-FORWARD	70.38	59.32	64.38
CNN-BACKWARD	69.78	61.75	**65.52**

Table 4.6: Performance of the combination methods.

The first observation is that the ensembling method is not an effective way to combine CNNs and RNNs as its performance is worse than the separate models. Second, regarding the stacking method, the best way to combine CNNs and RNNs

in this framework is to assemble the CNN model and the FORWARD model. In fact, the combination of the CNN and FORWARD models helps improve the performance of the separate models in both variants of this method (referring to the models CNN-FORWARD and FORWARD-CNN). Finally, the voting method is also helpful as it outperforms the separate models with the CNN-BIDIRECT and CNN-BACKWARD combinations.

For the following experiments, we would only focus on the three best combined models in this section, i.e, the CNN-FORWARD model in the stacking method (called STACK-FORWARD) and the CNN-BIDIRECT and CNN-BACKWARD models in the voting methods (called VOTE-BIDIRECT and VOTE-BACKWARD respectively).

4.2.3.5 Evaluating the Hybrid Models

This section investigates the hybrid and *hybrid-voting* models (Section 4.2.2) to see if they can further improve the performance of the neural network models. In particular, we evaluate the separate models: CNN, BIDIRECT, FORWARD, BACKWARD, and the combined models: STACK-FORWARD, VOTE-BIDIRECT and VOTE-BACKWARD when they are augmented with the traditional log-linear model (the hybrid models). Besides, in order to verify the hypothesis in Section 4.2.2, we also test the corresponding *hybrid-voting* models. The experimental results are shown in Table 4.7.

There are three main conclusions:

(i) For all the models in columns "*Neural Networks*", "*Hybrid Models*" and

Model	Neural Networks			Hybrid Models			Hybrid-Voting Models		
	P	R	F1	P	R	F1	P	R	F1
CNN	68.35	59.16	63.42	66.44	64.51	65.46	69.07	63.70	66.27
BIDIRECT	69.16	59.97	64.24	68.04	59.00	63.19	71.13	60.29	65.26
FORWARD	69.33	60.45	64.59	66.11	63.86	64.96	72.69	61.26	66.49
BACKWARD	65.60	63.05	64.30	66.03	62.07	63.99	71.56	63.21	67.13
Combined Models									
VOTE-BIDIRECT	71.08	60.94	65.62	69.24	62.40	65.64	71.30	62.40	**66.55**
STACK-FORWARD	69.46	63.05	66.10	65.93	68.07	66.99	69.32	66.29	**67.77**
VOTE-BACKWARD	69.78	61.75	65.52	67.30	63.05	65.10	70.79	64.02	**67.23**

Table 4.7: Performance of the hybrid models on the ACE 2005 development set.

"*Hybrid-Voting Models*", we see that the combined models outperform their corresponding separate models (only except the hybrid model of VOTE-BACKWARD), thereby further confirming the benefits of the combined models.

(ii) Comparing columns "*Neural Networks*" and "*Hybrid Models*", we find that the traditional log-linear model significantly helps the CNN model. The effects on the other models are not clear.

(iii) More interestingly, for all the neural networks being examined (either separate or combined), the corresponding *hybrid-voting* systems substantially improve both the neural network models as well as the corresponding hybrid models, testifying to the hypothesis about the *hybrid-voting* approach in Section 4.2.2. Note that the simpler voting systems on three models: the log-linear model, the CNN model and some RNN model (i.e, either BIDIRECT, FORWARD or BACKWARD) produce worse performance than the *hybrid-voting* methods (the respective performance is 66.13%, 65.27%, and 65.96%).

4.2.3.6 Comparing to the State of the art

The state-of-the-art system on the ACE 2005 for the unseen domains has been the feature-rich compositional embedding model (FCM) and the hybrid FCM model from (Gormley et al., 2015). In this section, we compare the proposed *hybrid-voting* systems with these state-of-the-art systems on the test domains `bc`, `cts`, `wl`. Table 4.8 reports the results. For completeness, we also include the performance of the log-linear model and the separate models CNN, BIDIRECT, FORWARD, BACKWARD, serving as the other baselines for this work.

System	bc			cts			wl			Ave
	P	R	F1	P	R	F1	P	R	F1	
The State-of-the-art Systems										
FCM	66.56	57.86	61.90	65.62	44.35	52.93	57.80	44.62	50.36	55.06
Hybrid FCM	74.39	55.35	63.48	74.53	45.01	56.12	65.63	47.59	55.17	58.26
Separate Systems										
Log-Linear	68.44	50.07	57.83	73.62	41.57	53.14	60.40	47.31	53.06	54.68
CNN	65.62	61.06	63.26	65.92	48.12	55.63	54.14	53.68	53.91	57.60
BIDIRECT	65.23	61.06	63.07	66.15	49.26	56.47	55.91	51.56	53.65	57.73
FORWARD	63.64	59.39	61.44	60.12	50.57	54.93	55.54	54.67	55.10	57.16
BACKWARD	60.44	61.2	60.82	58.20	54.01	56.03	51.03	52.55	51.78	56.21
Hybrid-Voting Systems										
VOTE-BIDIRECT	70.40	63.84	**66.96†**	66.74	49.92	57.12†	59.24	54.96	57.02†	60.37
STACK-FORWARD	65.75	66.48	66.11†	63.58	51.72	57.04†	56.35	57.22	56.78†	59.98
VOTE-BACKWARD	69.57	63.28	66.28†	65.91	52.21	**58.26†**	58.81	55.81	**57.27†**	**60.60**

Table 4.8: Comparison to the state of the art on the ACE 2005 dataset. The cells marked with †designates the models that are significantly better than the other neural network models ($\rho < 0.05$) on the corresponding domains.

From the table, we see that although the separate neural networks outperform the FCM model across domains, they are still worse than the hybrid FCM model due to the introduction of the log-linear model into FCM. However, when the networks are combined and integrated with the log-linear model, they (the *hybrid-voting* systems) become significantly better than the FCM models across all the

domains (up to 2% improvement on the average absolute F score), *yielding the state-of-the-art performance for the unseen domains in this dataset*.

4.2.3.7 Relation Classification Experiments

We further evaluate the proposed systems for the relation classification task on the SemEval dataset. Table 4.9 presents the performance of the seprate models, the proposed systems as well as the other representative systems on this task. The most important observation is that the *hybrid-voting* systems VOTE-BIDIRECT and VOTE-BACKWARD achieve the state-of-the-art performance for this dataset, further highlighting their benefit for relation classification. The *hybrid-voting* STACK-FORWARD system performs less effectively in this case, possibly due to the small size of the SemEval dataset that is not sufficient to training such a deep model.

4.2.4 Analysis

In order to better understand why the combination of CNNs and RNNs outperforms the individual networks, we evaluate the performance breakdown per relation for the CNN and BIDIRECT models. The results on the development set of the ACE 2005 dataset are provided in Table 4.10.

One of the main insights is that although CNN and BIDIRECT have comparable overall performance, their recalls on individual relations are very divergent. In particular, BIDIRECT has much better recall for the PHYS relation while the recalls of CNN are significantly better for the ART, ORG-AFF and GEN-AFF

Classifier	F
SVM (Hendrickx et al., 2010)	82.2
RNN (Socher et al., 2012b)	77.6
MVRNN (Socher et al., 2012b)	82.4
CNN (Zeng et al., 2014)	82.7
CR-CNN (Santos et al., 2015a)	**84.1†**
FCM (Gormley et al., 2015)	83.0
Hybrid FCM (Gormley et al., 2015)	83.4
DepNN (Liu et al., 2015)	83.6
SDP-LSTM (Xu et al., 2015)	83.7
Systems in this work	
CNN	83.5
BIDIRECT	81.8
FORWARD	81.9
BACKWARD	82.4
VOTE-BIDIRECT	**84.1**
STACK-FORWARD	83.4
VOTE-BACKWARD	**84.1**

Table 4.9: Performance of relation classification systems. The "†" refers to special treatment of the `Other` class.

Relation Class	CNN			BIDIRECT		
	P	R	F1	P	R	F1
PHYS	66.7	34.7	45.7	57.4	50.9	54.0
PART-WHOLE	68.6	67.8	68.2	74.4	70.1	72.2
ART	64.2	51.2	57.0	68.6	41.7	51.9
ORG-AFF	70.2	83.0	76.0	79.3	76.1	77.7
PER-SOC	71.1	59.3	64.6	69.6	59.3	64.0
GEN-AFF	65.9	55.1	60.0	59.0	46.9	52.3
all	68.4	59.2	63.4	69.2	60.0	64.2

Table 4.10: The performance breakdown per relation for CNN and BIDIRECT on the development set.

relations. A closer investigation reveals two facts: (i) the PHYS relation mentions that are only correctly predicted by BIDIRECT involve long distances between two entity mentions, such as the PHYS relation between "*Some*" (a person entity) and "*desert*" (a location entity) in the following sentence: "*Some of the 40,000 British troops are kicking up a lot of dust in the Iraqi desert making sure that nothing is left behind them that could hurt them.*", and (ii) the ART, ORG-AFF, GEN-AFF relation mentions only correctly predicted by CNN contain patterns between the two entity mentions that are short but meaningful enough to decide the relation classes, such as "*The Iraqi unit in possession of those guns*" (the ART relation between "*unit*" and "*guns*"), or "*the al Qaeda chief operations officer*" (the ORG-AFF relation between "*al Qaeda*" and "*officer*"). The failure of CNN on the PHYS relation mentions with long distances originates from its mechanism to model short and consecutive k-grams (up to length 5 in our case), causing difficulty in capturing long and/or non-consecutive patterns. BIDIRECT, on the other hand, fails to predict the short (but expressive enough) patterns for ART, ORG-AFF, GEN-AFF because it involves the hidden vectors that only model the context words outside the short patterns, potentially introducing unnecessary and noisy information into the max-pooling scores for prediction. Eventually, the combination of RNNs and CNNs helps to compensate for the drawbacks of each model.

4.3 Related Work

For relation extraction/classification, most work on neural networks has focused on the relation classification task. In particular, (Socher et al., 2012b) study the

recursive NNs that recur over the tree structures while (Xu et al., 2015) investigate recurrent NNs. Regarding CNNs, (Zeng et al., 2014) examine CNNs via the sequential representation of sentences, (Santos et al., 2015a) explore a ranking loss function with data cleaning while (Zeng et al., 2015) propose dynamic pooling and multi-instance learning. For RE, (Yu et al., 2015) and (Gormley et al., 2015) work on the feature-rich compositional embedding models. Finally, the only work that combines NN architectures is due to (Liu et al., 2015) but it only focuses on the stacking of the recursive NNs and CNNs for relation classification.

4.4 Conclusion

We present a CNN for relation extraction that emphasizes an unbalanced corpus and minimizes usage of external supervised NLP toolkits for features. The network uses multiple window sizes for filters, position embeddings for encoding relative distances and pre-trained word embeddings for initialization in a non-static architecture. The experimental results demonstrate the effectiveness of the proposed CNN on both RC and RE.

In addition, we investigate different methods to combine CNNs, RNNs as well as the hybrid models to integrate the log-linear model into the NNs. The experimental results demonstrate that the stacking and majority voting between CNNs, RNNs and their corresponding hybrid models are the best combination methods. We achieve the state-of-the-art performance for both relation extraction and relation classification.

Chapter 5

Deep Learning for Event Detection

This chapter focuses on the problem of event detection (ED) or trigger prediction, i.e, identifying instances of specified types of events in text. Associated with each event mention is a phrase, the event trigger (most often a single verb or nominalization), which evokes that event. Our task, more precisely stated, involves identifying event triggers and classifying them into specific types. For instance, according to the ACE 2005 annotation guideline[1], in the sentence "*A police* ***officer*** *was* ***killed*** *in* ***New Jersey today***", an event detection system should be able to recognize the word "*killed*" as a trigger for the event "*Die*". This task is quite challenging, as the same event might appear in the form of various trigger expressions and an expression might represent different events in different contexts. ED is a

1. https://www.ldc.upenn.edu/sites/www.ldc.upenn.edu/files/english-events-guidelines-v5.4.3.pdf

crucial component in the overall task of event extraction, which also involves event argument discovery.

In order to develop deep learning models for event detection, we first show that a customization of the CNN architecture in Section 4.1 can produce a highly effective model with little need for feature engineering. We then extend this CNN model to allow non-consecutive convolutions, thus further improving the performance for ED. The works in this chapter have been published in (Nguyen and Grishman, 2015b) and (Nguyen and Grishman, 2016e).

5.1 Convolutional Neural Networks for Event Detection

Recent systems for event extraction have employed either a pipeline architecture with separate classifiers for trigger and argument labeling (Gupta and Ji, 2009; Huang and Riloff, 2012; Ji and Grishman, 2008; Li et al., 2013a; Liao and Grishman, 2011; McClosky et al., 2011; Patwardhan and Rilof, 2009) or a joint inference architecture that performs the two subtasks at the same time to benefit from their inter-dependencies (Li et al., 2013b; Riedel and McCallum, 2011a, 2011b; Venugopal et al., 2014). Both approaches have coped with the ED task by elaborately hand-designing a large set of features (*feature engineering*) and utilizing the existing supervised natural language processing (NLP) toolkits and resources (i.e name tagger, parsers, gazetteers etc) to extract these features for statistical classifiers. In this section, we approach ED via a different perspective that relies on convo-

lutional neural networks (CNN) to automatically learn features from sentences, and to minimize the dependence on supervised toolkits and resources for features. The CNN models in this section are very similar to those of Section 4.1, but we need to customize them to capture the special structures for ED. To the best of our knowledge, this is the first work on event detection via CNNs at the time of publishing this work.

First, we evaluate CNNs for ED in the general setting and show that CNNs, though not requiring complicated feature engineering, can still outperform the state-of-the-art feature-based methods extensively relying on the other supervised modules and manual resources for features. Second, we investigate CNNs in a domain adaptation (DA) setting for ED. We demonstrate that CNNs significantly outperform the traditional feature-based methods with respect to generalization performance across domains due to: (i) their capacity to mitigate the error propagation from the pre-processing modules for features, and (ii) the use of word embeddings to induce a more general representation for event trigger candidates. We believe that this is also the first research on domain adaptation using CNNs.

5.1.1 Model

We formalize the event detection problem as a multi-class classification problem. Given a sentence, for every token in that sentence, we want to predict if the current token is an event trigger: i.e, does it express some event in the pre-defined event set or not (Li et al., 2013b)? The current token along with its context in the sentence constitute an event trigger candidate or an example in multi-class

classification terms. In order to prepare for CNNs, we limit the context to a fixed window size by trimming longer sentences and padding shorter sentences with a special token when necessary. Let $2n + 1$ be the fixed window size, and $W = [w_0, w_1, \ldots, w_n, \ldots, w_{2n-1}, w_{2n}]$ be some trigger candidate where the current token is positioned in the middle of the window (token w_n). Before entering the CNNs, each token w_i is transformed into a real-valued vector by looking up the following embedding tables to capture different characteristics of the token:

- Word Embedding Table (initialized by some pre-trained word embeddings): to capture the hidden semantic and syntactic properties of the tokens (Collobert and Westion, 2008; Turian et al., 2010).

- Position Embedding Table: to embed the relative distance $i - n$ of the token w_i to the current token w_n. In practice, we initialize this table randomly.

- Entity Type Embedding Table: If we further know the entity mentions and their entity types[2] in the sentence, we can also capture this information for each token by looking up the entity type embedding table (initialized randomly) using the entity type associated with each token. We employ the BIO annotation scheme to assign entity type labels to each token in the trigger candidate using the heads of the entity mentions.

For each token w_i, the vectors obtained from the three look-ups above are concatenated into a single vector x_i to represent the token. As a result, the original event trigger W is transformed into a matrix:

2. For convenience, when referring to entities in this work, we always include ACE timex and values.

$$X = [x_0, x_1, \ldots, x_n, \ldots, x_{2n-1}, x_{2n}] \tag{5.1}$$

of size $d \times (2n + 1)$ (d is the dimensionality of the concatenated vectors of the tokens).

The matrix representation x is then passed through a convolution layer, a max pooling layer and a softmax at the end to perform classification (as in Section 4.1). In the convolution layer, we have a set of feature maps (filters) $\{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_m\}$ for the convolution operation. Each feature map $\mathbf{f}_i$ corresponds to some window size k and can be essentially seen as a weight matrix of size $d \times k$. Figure 5.1 illustrates the proposed CNN.

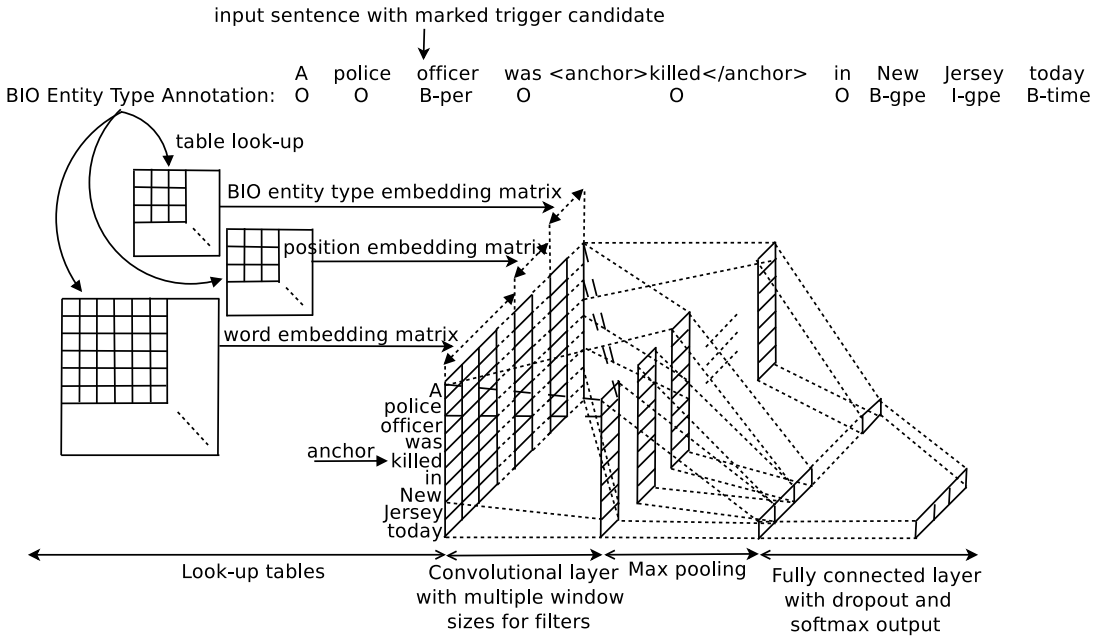

Figure 5.1: Convolutional Neural Network for Event Detection.

Again, similar to Section 4.1, the gradients are computed using back-propagation;

regularization is implemented by dropout (Srivastava et al., 2014); and training is done via stochastic gradient descent with shuffled mini-batches and the AdaDelta update rule (Zeiler, 2012). During the training, we also optimize the weights of the three embedding tables at the same time to reach an effective state.

5.1.2 Experiments

5.1.2.1 Dataset, Hyperparameters and Resources

As the benefit of multiple window sizes in the convolution layer has been demonstrated in the previous work (Kalchbrenner et al., 2014; Kim, 2014; Nguyen and Grishman, 2015a), in the experiments below, we use window sizes in the set {2, 3, 4, 5} to generate feature maps. We utilize 150 feature maps for each window size in this set. The window size for triggers is set to 31 while the dimensionality of the position embeddings and entity type embeddings is 50[3]. We inherit the values for the other parameters from (Kim, 2014), i.e, the dropout rate $\rho = 0.5$, the mini-batch size = 50, the hyperparameter for the Frobenius norms = 3. Finally, we employ the pre-trained word embeddings `word2vec` with 300 dimensions from (Mikolov et al., 2013b) for initialization.

We evaluate the presented CNN over the ACE 2005 corpus for event detection. For comparison purposes, we utilize the same test set with 40 newswire articles (672 sentences), the same development set with 30 other documents (836 sentences) and the same training set with the remaning 529 documents (14,849 sentences) as the previous studies on this dataset (Ji and Grishman, 2008; Li et al., 2013b; Liao and

3. These values are chosen for their best performance on the development data.

Grishman, 2010b). The ACE 2005 corpus has 33 event subtypes that, along with one class "*None*" for the non-trigger tokens, constitutes a 34-class classification problem.

In order to evaluate the effectiveness of the position embeddings and the entity type embeddings, Table 5.1 reports the performance of the proposed CNN on the development set when these embeddings are either included or excluded from the systems. With the large margin of performance, it is very clear from the table that the position embeddings are crucial while the entity embeddings are also very useful for CNNs on ED.

Systems		P	R	F
-Entity Types	-Position	16.8	12.0	14.0
	+Position	75.0	63.0	**68.5**
+Entity Types	-Position	17.0	15.0	15.9
	+Position	75.6	66.4	**70.7**

Table 5.1: Performance on the development set.

For the experiments below, we examine the CNNs in two scenarios: excluding the entity type embeddings (CNN1) and including the entity type embeddings (CNN2). We always use position embeddings in these two scenarios.

5.1.2.2 Performance Comparison

The state-of-the-art systems for event detection on the ACE 2005 dataset have followed the traditional feature-based approach with rich hand-designed feature sets, and statistical classifiers such as MaxEnt and perceptron for structured prediction in a joint architecture (Hong et al., 2011; Li et al., 2013b). In this section,

we compare the proposed CNNs with these state-of-the-art systems on the blind test set. Table 5.2 presents the overall performance of the systems with gold-standard entity mention and type information[4].

Methods	P	R	F
Sentence-level in (Hong et al., 2011)	67.6	53.5	59.7
MaxEnt with local features in (Li et al., 2013b)	74.5	59.1	65.9
Joint beam search with local features in (Li et al., 2013b)	73.7	59.3	65.7
Joint beam search with local and global features in (Li et al., 2013b)	73.7	62.3	67.5
Cross-entity in (Hong et al., 2011) †	72.9	64.3	68.3
CNN1: CNN without any external features	71.9	63.8	67.6
CNN2: CNN augmented with entity types	71.8	66.4	**69.0**

Table 5.2: Performance with gold-standard entity mentions and types. † beyond sentence level.

As we can see from the table, considering the systems that only use sentence-level information, CNN1 significantly outperforms the MaxEnt classifier as well as the joint beam search with local features from (Li et al., 2013b) (an improvement of 1.6% in F1 score), and performs comparably with the joint beam search approach using both local and global features (Li et al., 2013b). This is remarkable since CNN1 does not require any external features[5], in contrast to the other feature-based systems that extensively rely on such external features to perform well. More interestingly, when the entity type information is incorporated into CNN1, we obtain CNN2 that still only needs sentence level information but achieves the

4. Entity mentions and types are used to introduce more features into the systems.

5. External features are the features generated from the supervised NLP modules and manual resources such as parsers, name tagger, entity mention extractors (either automatic or manual), gazetteers, etc.

state-of-the-art performance for this task (an improvement of 1.5% over the best system with only sentence level information (Li et al., 2013b)).

Except for the CNN1, all the systems reported in Table 5.2 employ the gold-standard (perfect) entities mentions and types from manual annotation which might not be available in practice. Table 5.3 compares the performance of CNN1 and the feature-based systems in a more realistic setting, where entity mentions and types are acquired from an automatic high-performing name tagger and information extraction system (Li et al., 2013b). Note that CNN1 is eligible for this comparison as it does not utilize any external features, thus avoiding usage of the name tagger and the information extraction system to identify entity mentions and types.

Methods	F
Sentence level in (Ji and Grishman, 2008)	59.7
MaxEnt with local features in (Li et al., 2013b)	64.7
Joint beam search with local features in (Li et al., 2013b)	63.7
Joint beam search with local and global features in (Li et al., 2013b)	65.6
CNN1: CNN without any external features	**67.6**

Table 5.3: Performance with predicted entity mentions and types.

5.1.2.3 Domain Adaptation Experiment

We evaluate the robustness across domains of the CNN models in this section. In particular, we compare the proposed CNNs with the feature-based systems under the domain adaptation setting for event detection (as in Chapters 2 and 3). In such a setting, we take training data in some *source domain* to learn models

that can work well on *target domains*. The target domains are supposed to be so dissimilar from the source domain that the learning techniques would suffer from a significant performance loss when trained on the source domain and applied to the target domains. We refer the reader to Chapters 1 and 2 for a description of the domain adaptation setting. To make it clear, we address the unsupervised DA problem in this section, i.e, no training data in the target domains (Blitzer et al., 2006; Nguyen et al., 2015c; Plank and Moschitti, 2013).

We also run the experiments in this part over the ACE 2005 dataset but focus more on the difference between domains. Following the common practice of domain adaptation research on this dataset (Nguyen and Grishman, 2014a; Nguyen et al., 2015c; Plank and Moschitti, 2013), we use `news` (the union of `bn` and `nw`) as the source domain and `bc`, `cts`, `wl` as three different target domains. We take half of `bc` as the development set and use the remaining data for testing.

Table 5.4 presents the performance of five systems: the MaxEnt classifier with the local features from (Li et al., 2013b) (called *MaxEnt*); the state-of-the-art joint beam search systems with: (i) only local features (called *Joint+Local*); and (ii) both local and global features (called *Joint+Local+Global*) in (Li et al., 2013b) (the baseline systems); CNN1 and CNN2 via 5-fold cross validation. For each system, we train a model on the training set of the source domain `news` and report the performance of this model on the test set of the source domain (in-domain performance) as well as the performance of the model on the three target domains `bc`, `cts` and `wl` (out-of-domain performance)[6].

6. The performance of the feature-based systems *MaxEnt*, *Joint+Local* and *Joint+Local+Global* are obtained from the actual systems in (Li et al., 2013b).

System	In-domain(bn+nw)			bc			cts			wl		
	P	R	F	P	R	F	P	R	F	P	R	F
MaxEnt	74.5	59.4	66.0	70.1	54.5	61.3	66.4	49.9	56.9	59.4	34.9	43.9
Joint beam search in (Li et al., 2013b)												
Joint+Local	73.5	62.7	67.7	70.3	57.2	63.1	64.9	50.8	57.0	59.5	38.4	46.7
Joint+Local+Global	72.9	63.2	67.7	68.8	57.5	62.6	64.5	52.3	57.7	56.4	38.5	45.7
CNN1	70.9	64.0	67.3	71.0	61.9	66.1†	64.0	55.0	59.1	53.2	38.4	44.6
CNN2	69.2	67.0	**68.0**	70.2	65.2	**67.6†**	68.3	58.2	**62.8†**	54.8	42.0	**47.5**

Table 5.4: In-domain (first column) and Out-of-domain performance for event detection (columns two to four). Cells marked with †designate CNN models that significantly outperform ($p < 0.05$) all the reported feature-based methods on the specified domain.

The main conclusions from the table include:

(i) The baseline systems *MaxEnt*, *Joint+Local*, *Joint+Local+Global* achieve high performance on the source domain, but degrade dramatically on the target domains due to the domain shifts.

(ii) Comparing CNN1 and the baseline systems, we see that CNN1 performs comparably with the baseline systems on the source domain (in-domain performance) (as expected), substantially outperforms the baseline systems on two of the three target domains (i.e, bc and cts), and is only less effective than the joint beam search approach on the wl domain; (iii) Finally and most importantly, we consistently achieve the best adaptation performance across all the target domains with CNN2 by only introducing entity type information into CNN1. In fact, CNN2 significantly outperforms the feature-based systems with $p < 0.05$ and large margins of about 5.0% on the domains bc and cts, clearly testifying to the benefits of CNNs on DA for ED.

5.2 Non-consecutive Convolutional Neural Networks for Event Detection

The prior CNN models for ED are characterized by the temporal convolution operators that linearly map the vectors for the k-grams in the sentences into the feature space. Such k-gram vectors are obtained by concatenating the vectors of the k consecutive words in the sentences (Chen et al., 2015; Nguyen and Grishman, 2015b). In other words, the previous CNN models for ED only focus on modeling the consecutive k-grams. Unfortunately, such consecutive mechanism is unable to capture the long-range and non-consecutive dependencies that are necessary to the prediction of trigger words. For instance, consider the following sentence with the trigger word "*leave*" from the ACE 2005 corpus:

*The mystery is that she took the job in the first place or didn't **leave** earlier.*

The correct event type for the trigger word "*leave*" in this case is "*End-Org*" (ending a position). However, the previous CNN models might not be able to detect "*leave*" as an event trigger or incorrectly predict its type as "*Movement*". This is caused by their reliance on the consecutive local k-grams such as "*leave earlier*". Consequently, we need to resort to the non-consecutive pattern "*job leave*" to correctly determine the event type of "*leave*" in this case.

Guided by this intuition, we propose to improve the previous CNN models for ED by operating the convolution on all possible non-consecutive k-grams in the sentences. We aggregate the resulting convolution scores via the *max-pooling* function to unveil the most important non-consecutive k-grams for ED. The aggre-

gation over all the possible non-consecutive k-grams is made efficient with dynamic programming.

Note that our work is related to (Lei et al., 2015) who employ the non-consecutive convolution for the sentence and news classification problems. Our work is different from Lei et al., 2015 in that we model the relative distances of words to the trigger candidates in the sentences via position embeddings, while (Lei et al., 2015) use the absolute distances between words in the k-grams to compute the decay weights for aggregation. To the best of our knowledge, this is the first work on non-consecutive CNN for ED.

We systematically evaluate the proposed model in the general setting as well as the domain adaptation setting. The experiment results demonstrate that our model significantly outperforms the current state-of-the-art models in such settings.

5.2.1 Model

We also formalize ED as a multi-class classification problem as in the previous section. In order to make it compatible with the previous work, we follow the same preprocessing steps as the previous section (Section 5.1), i.e, limiting the context of the event trigger candidates to some fixed window size, transforming every token in the context into a vector using its word embeddings, position embeddings and entity type embeddings. The result of this process is the matrix $X = [x_0, x_1, \ldots, x_n, \ldots, x_{2n-1}, x_{2n}]$ to represent the input trigger candidate W as we can see in Equation 5.1. This matrix can be seen as a sequence of real-valued

vectors $X = (x_0, x_1, \ldots, x_{2n})$ that will be used as input in the following CNN models.

5.2.1.1 The Traditional CNN

Given the window size k, the traditional CNN models (as in the previous section 5.1) for ED consider the following set of $2n + 1$ consecutive k-gram vectors:

$$C = \{u_i : 0 \leq i \leq 2n\} \tag{5.2}$$

Vector u_i is the concatenation of the k consecutive vectors preceding position i in the sequence X:

$$u_i = [x_{i-k+1}, x_{i-k+2}, \ldots, x_i] \in \mathbb{R}^{dk} \tag{5.3}$$

where the out-of-index vectors are simply set to all zeros.

The core of the CNN models is the convolution operation, specified by the filter vector $\mathbf{f} \in \mathbb{R}^{dk}$. In CNNs, $\mathbf{f}$ can be seen as a feature extractor for the k-grams that operates via the dot product with each element in C. This produces the following convolution score set:

$$S(C) = \{\mathbf{f}^T u_i : 0 \leq i \leq 2n\} \tag{5.4}$$

In the next step, we aggregate the features in S with the max function, resulting in the aggregation score:

$$p_{\mathbf{f}}^k = \max S(C) = \max\{s_i : 0 \leq i \leq 2n\} \tag{5.5}$$

Afterward, $p_{\mathbf{f}}^k$ is often transformed by a non-linear function g[7] to generate the transformed score $g(p_{\mathbf{f}}^k)$, functioning as the extracted feature for the initial trigger candidate W.

We can then repeat this process for different window sizes k and filters $\mathbf{f}$, generating multiple features $g(p_{\mathbf{f}}^k)$ to capture various aspects of the trigger candidate W. Finally, such features are concatenated into a single representation vector for W, to be fed into a feed-forward neural network with a softmax layer in the end to perform classification.

5.2.1.2 The Non-consecutive CNN

As mentioned in the introduction, the limitation of the previous CNN models for ED is the inability to encode the non-consecutive k-grams that might be crucial to the trigger prediction. This limitation originates from Equation 5.2 in which only the consecutive k-gram vectors are considered. In order to overcome such limitation, we propose to model all possible non-consecutive k-grams in the trigger candidate, leading to the following set of non-consecutive k-gram vectors:

$$N = \{v_{i_1 i_2 \ldots i_k} : 0 \leq i_1 < i_2 < \ldots < i_k \leq 2n\}$$

where: $v_{i_1 i_2 \ldots i_k} = [x_{i_1}, x_{i_2}, \ldots, x_{i_k}] \in \mathbb{R}^{dk}$ and the number of elements in N is $|N| = \binom{2n+1}{k}$.

The non-consecutive CNN model then follows the procedure of the traditional CNN model in Section 5.2.1.1 to compute the representation vector for classification. The only difference is that the computation is done on the input set

7. The `tanh` function in this work.

N instead of C. In particular, the convolution score set in this case would be $S(N) = \{\mathbf{f}^T v : v \in N\}$, while the aggregating score would be:

$$p_{\mathbf{f}}^k = \max S(N) = \max\{s : s \in S(N)\} \tag{5.6}$$

5.2.1.3 Implementation

Note that the maximum operation in Equation 5.5 only requires $O(n)$ operations while the naive implementation of Equation 5.6 would need $O(|N|) = O(n^k)$ operations. In this work, we employ the dynamic programming (DP) procedure below to reduce the computation time for Equation 5.6.

Assuming the filter vector $\mathbf{f}$ is the concatenation of the k vectors $\mathbf{f}_1, \ldots, \mathbf{f}_k \in \mathbb{R}^d$: $\mathbf{f} = [\mathbf{f}_1, \ldots, \mathbf{f}_k]$, Equation 5.6 can be re-written by:

$$\begin{aligned} p_{\mathbf{f}}^k = &\max\{\mathbf{f}_1^T x_{i_1} + \ldots + \mathbf{f}_k^T x_{i_k} \\ &: 0 \leq i_1 < i_2 < \ldots < i_k \leq 2n\} \end{aligned} \tag{5.7}$$

Let D_t^j be the dynamic programming table representing the maximum convolution score for the sub-filter $[\mathbf{f}_1, \ldots, \mathbf{f}_j]$ over all possible non-consecutive j-gram vectors in the subsequence $(x_0, x_1, \ldots, x_t)$ of X:

$$\begin{aligned} D_t^j = &\max\{\mathbf{f}_1^T x_{i_1} + \ldots + \mathbf{f}_j^T x_{i_j} \\ &: 0 \leq i_1 < i_2 < \ldots < i_j \leq t\} \end{aligned} \tag{5.8}$$

where $1 \leq j \leq k$, $j - 1 \leq t \leq 2n$.

Note that $p_{\mathbf{f}}^k = D_{2n}^k$.

We can solve this DP problem by the following recursive formulas[8]:

$$D_t^j = \max\{D_{t-1}^j, D_{t-1}^{j-1} + \mathbf{f}_j^T x_t\} \tag{5.9}$$

The computation time for this procedure is $O(kn)$ and remains linear in the sequence length.

5.2.1.4 Training

We train the networks using stochastic gradient descent with shuffled mini-batches, the AdaDelta update rule, back-propagation and dropout. During the training, we also optimize the embedding tables (i.e, word, position and entity type embeddings) to achieve the optimal states. Finally, we rescale the weights whose Frobenius norms exceed a predefined threshold (Nguyen and Grishman, 2015a).

5.2.2 Experiments

We apply *the same parameters, resources and the ACE 2005 corpus* as Section 5.1.2.1 (Nguyen and Grishman, 2015b) to ensure the compatible comparison.

5.2.2.1 The General Setting

We compare the non-consecutive CNN model (*NC-CNN*) with the state-of-the-art systems on the ACE 2005 dataset in Table 5.5. These systems include:

1) The feature-based systems with rich hand-designed feature sets as in Section 5.1, including: the MaxEnt model with local features in (Li et al., 2013b)

8. We ignore the base cases as they are trivial.

(*MaxEnt*); the structured perceptron model for joint beam search with local features (*Joint+Local*), and with both local and global features in (Li et al., 2013b) (*Joint+Local+Global*); and the sentence-level and cross-entity models in (Hong et al., 2011).

2) The neural network models, i.e, the CNN model in Section 5.1 (Nguyen and Grishman, 2015b) (*CNN*), the dynamic multi-pooling CNN model (*DM-CNN*) in (Chen et al., 2015) and the bidirectional recurrent neural networks (*B-RNN*) in (Nguyen et al., 2016a).

3) The probabilistic soft logic based model to capture the event-event correlation in (Liu et al., 2016).

Methods	F
Sentence-level in (Hong et al., 2011)	59.7
MaxEnt (Li et al., 2013b)	65.9
Joint+Local (Li et al., 2013b)	65.7
Joint+Local+Global (Li et al., 2013b)	67.5
Cross-entity in (Hong et al., 2011) †	68.3
Probabilistic soft logic (Liu et al., 2016) †	69.4
CNN (Nguyen and Grishman, 2015b)	69.0
DM-CNN (Chen et al., 2015)	69.1
B-RNN (Nguyen et al., 2016a)	69.3
NC-CNN	**71.3**

Table 5.5: Performance with gold-standard entity mentions and types. † beyond sentence level.

The most important observation from the table is that the non-consecutive CNN model significantly outperforms all the compared models with a large margin. In particular, *NC-CNN* is 2% better than B-RNN (Nguyen et al., 2016a),

the state-of-the-art system that only relies on the context information within the sentences of the trigger candidates. In addition, although *NC-CNN* only employs the sentence-level information, it is still better than the other models that further exploit the document-level information for prediction (an improvement of 1.9% over the probabilistic soft logic based model in (Liu et al., 2016)). Finally, comparing *NC-CNN* and the CNN model in (Nguyen and Grishman, 2015b), we see that the non-consecutive mechanism significantly improves the performance of the traditional CNN model for ED (up to 2.3% in absolute F-measures with $p < 0.05$).

5.2.2.2 The Domain Adaptation Experiments

This section evaluates the robustness of *NC-CNN* for ED in the domain adaptation setting. The setting is exactly the same as those in Section 5.1.2.3. The only exception is with respect to the domain `un` of the ACE 2005 dataset that is also used as one of the target domains (besides `bc`, `cts` and `wl`) in this section. We report the performance on `un` to further demonstrate the benefit of *NC-CNN*. Table 5.6 shows the performance of the systems in the domain adaptation setting (i.e, similar to Table 5.4).

System	In-domain(bn+nw)			bc			cts			wl			un		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
MaxEnt	74.5	59.4	66.0	70.1	54.5	61.3	66.4	49.9	56.9	59.4	34.9	43.9	-	-	-
Joint+Local	73.5	62.7	67.7	70.3	57.2	63.1	64.9	50.8	57.0	59.5	38.4	46.7	-	-	-
Joint+Local+Global	72.9	63.2	67.7	68.8	57.5	62.6	64.5	52.3	57.7	56.4	38.5	45.7	-	-	-
B-RNN	71.4	63.5	67.1	70.7	62.1	66.1	70.0	54.4	61.0	52.7	38.3	44.2	66.2	46.0	54.1
DM-CNN	75.9	62.7	68.7	75.3	59.3	66.4	74.8	52.3	61.5	59.2	37.4	45.8	72.2	44.5	55.0
CNN	69.2	67.0	68.0	70.2	65.2	67.6	68.3	58.2	62.8	54.8	42.0	**47.5**	64.6	49.9	56.2
NC-CNN	74.9	66.5	**70.4**†	73.6	64.7	**68.8**†	71.7	57.3	**63.6**	57.8	40.3	47.4	71.7	49.0	**58.1**†

Table 5.6: Performance on the source domain and on the target domains. Cells marked with †designates that NC-CNN significantly outperforms ($p < 0.05$) all the compared methods on the specified domain.

We notice that the performance of the systems *MaxEnt*, *Joint+Local*, *B-RNN*, *CNN* and *Joint+Local+Global* is obtained from the actual systems in the original work (Li et al., 2013b; Nguyen and Grishman, 2015b; Nguyen et al., 2016a). The performance of *DM-CNN*, on the other hand, is from our re-implementation of the system in (Chen et al., 2015) using the same hyper-parameters and resources as *CNN* and *NC-CNN* for a fair comparison.

From the table, we see that *NC-CNN* is significantly better than the other models on the source domain. This is consistent with the conclusions in Section 5.2.2.1 and further confirms the effectiveness of *NC-CNN*. More importantly, *NC-CNN* outperforms *CNN* and the other models on the target domains `bc`, `cts` and `un`, and performs comparably with *CNN* on `wl`. The performance improvement is significant on `bc` and `un` ($p < 0.05$), thereby verifying the robustness of *NC-CNN* for ED across domains.

5.3 Related Work

There have been three major approaches to event detection in the literature. First, the pattern-based approach explores the application of patterns to identify the instances of events, in which the patterns are formed by predicates, event triggers and constraints on the syntactic context (Cao et al., 2015a, 2015b; Grishman et al., 2005).

Second, the feature-based approach relies on linguistic intuition to design effective feature sets for statistical models for ED, ranging from the local sentence-level representations (Ahn, 2006; Li et al., 2013a), to the higher level structures such

as the cross-sentence or cross-event information (Gupta and Ji, 2009; Hong et al., 2011; Ji and Grishman, 2008; Li et al., 2015; Liao and Grishman, 2010a, 2010b, 2011; McClosky et al., 2011; Patwardhan and Rilof, 2009). Some recent work on the feature-based approach has also investigated event trigger detection in the joint inference with event argument prediction (Li et al., 2013b; Poon and Vanderwende, 2010; Riedel et al., 2009; Riedel and McCallum, 2011a, 2011b; Venugopal et al., 2014) to benefit from their inter-dependencies.

Finally, neural networks have been introduced into ED very recently with the early work on convolutional neural networks (Chen et al., 2015; Nguyen and Grishman, 2015b). The other work includes: (Nguyen et al., 2016a) that employs bidirectional recurrent neural networks to perform event trigger and argument labeling jointly, (Jagannatha and Yu, 2016) that extracts event instances from health records with recurrent neural networks, (Nguyen et al., 2016b) that proposes a two-stage training algorithm for event extension with neural networks and (Nguyen et al., 2016g) that applies non-consecutive CNNs for event nugget tasks.

5.4 Conclusion

We present a CNN for event detection that automatically learns effective feature representations from pre-trained word embeddings, position embeddings as well as entity type embeddings, and reduces the error propagation. We conducted experiments to compare the proposed CNN with the state-of-the-art feature-based systems in both the general setting and the domain adaptation setting. The experimental results demonstrate the effectiveness as well as the robustness across

domains of the CNN. In addition, we extend such CNN architecture to employ non-consecutive convolutions, yielding the state-of-the-art performance for ED on both the general setting and the domain adaptation setting.

Chapter 6

Memory-augmented Networks for Joint Inference in Information Extraction

We can view the tasks in the information extraction (IE) pipeline in Figure 1.2 as sequences of prediction tasks. For instance, for trigger prediction or event detection, a document can be seen as a sequence of words in which a prediction is performed for every word in the sequence (i.e, predicting whether a word is a trigger word of some event types or not). For relation extraction, the sequences of prediction tasks corresponds to the sequences of entity mention pairs appearing in the same sentences of the documents. Consequently, the whole information extraction pipeline translates into a very long sequence of prediction tasks by concatenating the sequences of the individual tasks.

Given this view, the previous chapters and work on deep learning for information extraction have only concerned solving the predictions in the sequences for IE separately or independently. An illustration is given in Figure 6.1.

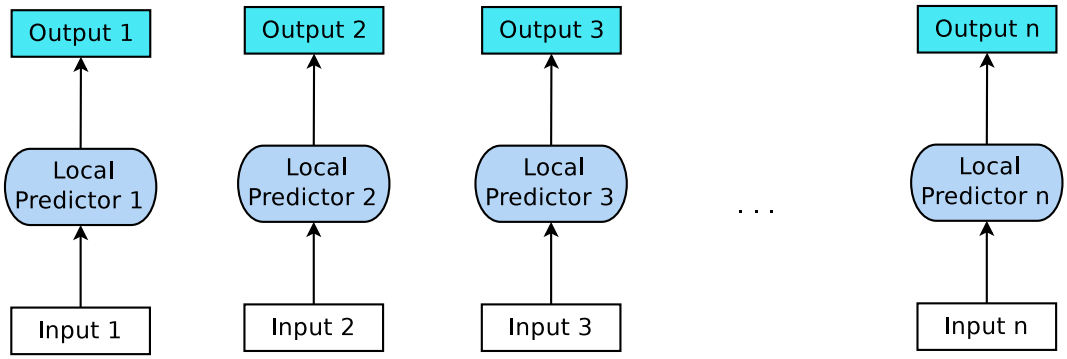

Figure 6.1: A sequence of prediction tasks in information extraction.

As we can see from the figure, there are n prediction tasks in the sequence indexed from 1 to n. Each prediction task has its corresponding input and output. The current deep learning work for IE has essentially modeled every task in this sequence by neural networks that run separately and carry no information from one step (task) to the other steps. Among several problems, such independent modeling is unable to capture long range dependencies or interdependencies among the outputs of the prediction tasks in the sequence that might be important to the prediction. For example, consider the following sentence for the task of event detection:

At least 10 people were killed when US drones fired missiles at three vehicles, an senior security official told AFP.

There are two event trigger words in this sentence: "*killed*" for the event class

(type) "*Die*" and "*fired*" for the event type "*Attack*". It is often simple to recognize the event type "*Die*" for "*killed*" based on the word itself, but it is more challenging to identify the event type for "*fired*" as this word is more ambiguous (i.e, having multiple possible meanings depending on context such as "*Attack*", "*End Position*" etc). However, if we notice that a "*Die*" event is very likely followed by an "*Attack*" event appearing in the same sentence, we can infer that the event type for "*fired*" should be "*Attack*" more easily based on the event type of "*killed*". Note that such dependencies can occur between event types of words that are very far from each other in the sentences (i.e, long-range dependencies). As a result, if we solve the prediction tasks for words in the sentences independently, we will not be able to capture those interdependencies.

6.1 General Framework

In this dissertation, we propose a general memory-augmented neural network to allow the inclusion of the long-range dependencies into the modeling of the prediction sequences in IE. An overview of this framework is shown in Fiture 6.2.

The main proposal from this figure includes:

1. Instead of solving the prediction tasks in the sequence separately, we will simultaneously perform such prediction tasks from left to right (joint modeling or joint inference).

2. During the modeling process from 1 to n, we will maintain a memory at every step. The memory of the current step i is computed from the memory of the previous step $i-1$ and the output of the current step. In general, the memory at

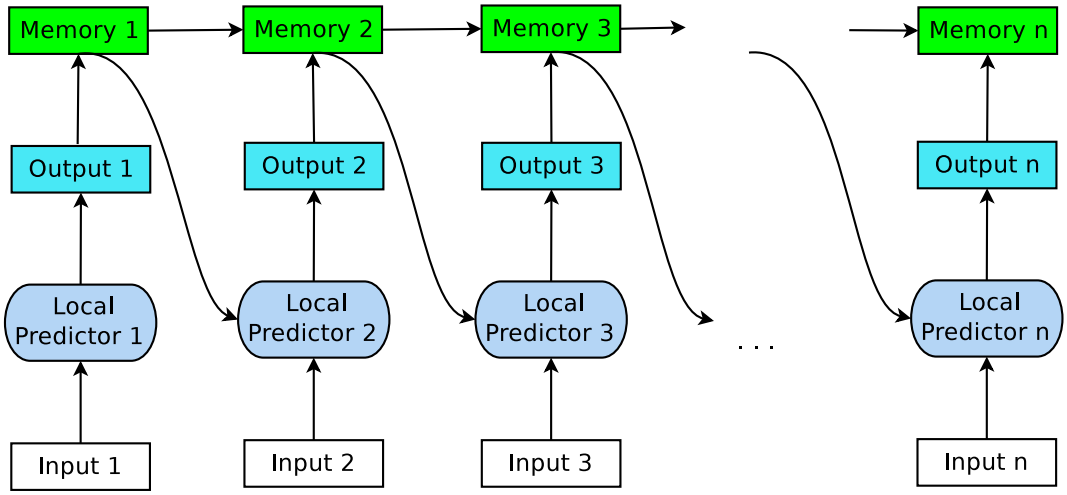

Figure 6.2: Memory-augmented neural networks.

step i is expected to memorize or summarize the outputs that we have made so far (i.e, from step 1 to i).

3. When we make a prediction for the step $i + 1$, we will use the memory from the previous step (i.e, step i) in addition to the current input as the evidence. This mechanism helps to incorporate the previous decisions (including the previous decisions or labels that are very far from the current step) into the prediction of the current step, thus being able to exploit the long-range dependencies among outputs of the prediction tasks.

In order to demonstrate the effectiveness of such memory-augmented neural networks, we seek their applications on two important problems of information extraction, i.e, event extraction (EE) and entity linking (EL) in the following sections. The works in this chapter have been published in (Nguyen et al., 2016a) and (Nguyen et al., 2016f).

6.2 Event Extraction with Memory-augmented Neural Networks

6.2.1 Event Extraction Task

We focus on the EE task of the Automatic Context Extraction (ACE) evaluation[1]. ACE defines an event as something that happens or leads to some change of state. We employ the following terminology:

- **Event mention**: a phrase or sentence in which an event occurs, including one trigger and an arbitrary number of arguments.
- **Event trigger**: the main word that most clearly expresses an event occurrence.
- **Event argument**: an entity mention (including temporal expression and value (e.g. *Job-Title*)) that servers as a participant or attribute with a specific role in an event mention.

ACE annotates 33 types[2] (e.g., "*Attack*, "*Die*", "*Start-Position*") for event mentions that also correspond to the types of the event triggers. Each event type has its own set of roles to be filled by the event arguments. For instance, the roles for the "*Die*" event include "*Place*", "*Victim*" and "*Time*". The total number of roles for all the event types is 36.

1. `http://projects.ldc.upenn.edu/ace`
2. These are actually 33 subtypes of the 8 event types annotated in the ACE 2005 dataset, but we call them event types here for convenience.

Given an English text document, an event extraction system needs to recognize event triggers with specific types and their corresponding arguments with the roles for each sentence. Following the previous work (Chen et al., 2015; Li et al., 2013b; Liao and Grishman, 2011), we assume that the argument candidates (i.e, the entity mentions, temporal expressions and values) are provided (by the ACE annotation) to the event extraction systems.

For instance, we are supposed to extract two event mentions in the following sentence:

In Baghdad, a cameraman died when an American tank fired on the street.

The first event mention is associated with the trigger word "*died*" of type "*Die*", and has "*cameraman*" as the "*Victim*" role and "*Baghdad*" as the "*Place*" role. The second event mention, on the other hand, corresponds to the trigger word "*fired*" of type "*Attack*", and involves "*cameraman*" for the "*Target*" role and "*Baghdad*" for the "*Place*" role.

6.2.2 Prior Deep Learning Work for Event Extraction

This section describes the previous deep learning work for event extraction that motivates our current work.

Assume that we have a sentence with 5 words $[W_1, W_2, W_3, W_4, W_5]$ in which words W_2 and W_5 are the heads of two entity mentions appearing in this sentence. Figure 6.3 shows the prediction tasks in event extraction we need to solve for this sentence.

The first column of this figure represents the trigger prediction tasks (rectangles)

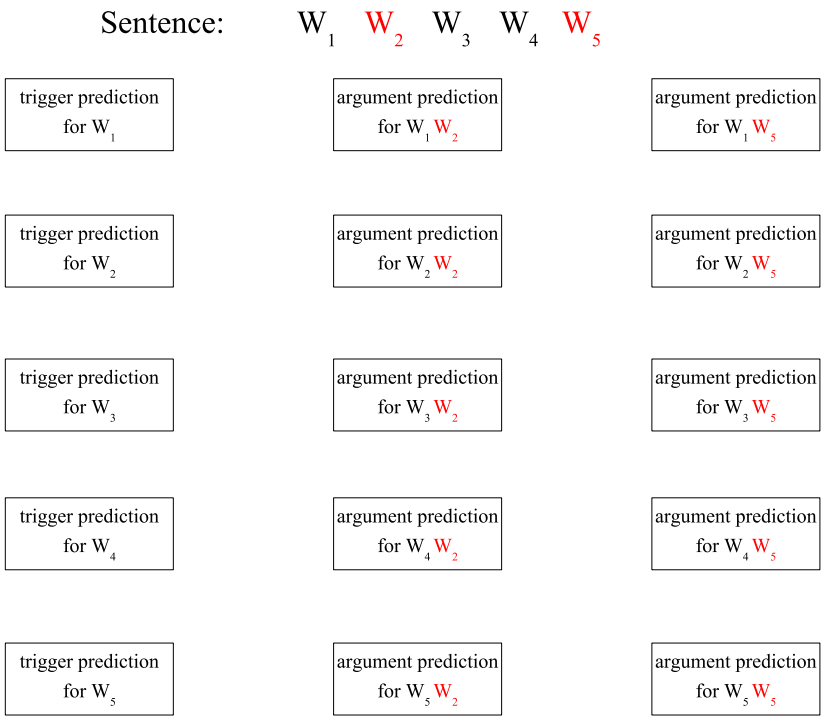

Figure 6.3: Prediction tasks for event extraction.

for every word (i.e, from word W_1 to word W_5) in this sentence while the rectangles of the second and the third column correspond to the argument prediction tasks for the entity mentions W_2 and W_5 respectively. For instance, the argument prediction task for W_3 and W_2 (the rectangle in the middle of the figure) aims at predicting the role of W_2 with respect to the event associated with trigger word W_3 (assuming W_3 is a trigger word).

Under this view, the prior deep learning work for EE has only modeled the tasks in the rectangles independently via some neural networks (Chen et al., 2015), ig-

noring long-range dependencies among the tasks. One such interdependence is the correlations among the event types within the same sentences that have been demonstrated at the beginning of this chapter. Another important interdependence in EE is regarding to the argument roles and the event types. For example, reconsider the example sentence with two event mentions we have in Section 6.2.1:

In Baghdad, a cameraman died when an American tank fired on the street.

It is often simple for the argument classifiers of the previous deep learning models for EE to realize that "*cameraman*" is the "*Target*" argument of the "*Die*" event due to the proximity between "*cameraman*" and "*died*" in this sentence. However, as "*cameraman*" is far away from "*fired*", the argument classifiers in such models might fail to recognize "*cameraman*" as the "*Target*" argument for the event "*Attack*" with their local features. Fortunately, we can overcome this issue by relying on the global features to encode the fact that a "*Victim*" argument for the "*Die*" event is often the "*Target*" argument for the "*Attack*" event in the same sentence. Again, exploiting such interdependencies is impossible in the prior deep learning models.

In order to address such limitations, we can apply the proposed memory-augmented neural networks to perform EE. In particular, taking the example with 5 words in the sentence above as an example, we simultaneously (jointly) solve the tasks (rectangles) in Figure 6.3 using the order specified in Figure 6.4 (joint modeling). At every step along that order, we organize a memory to store the prediction outputs in the previous steps, and use such memories as additional evidence in the next predictions to capture the interdependencies. A detailed joint

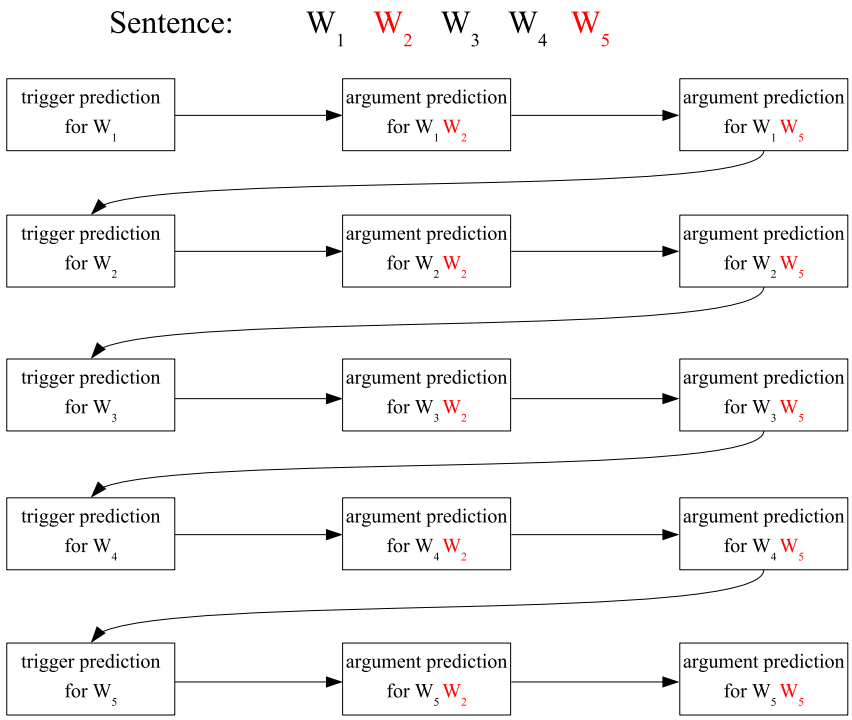

Figure 6.4: Memory-augmented neural networks for event extraction.

model for EE will be presented in the next section. Note that we often call the memory-augmented networks for EE as the joint models in the following for convenience.

6.2.3 Model

We formalize the EE task as follow. Let $W = [w_1, w_2, \ldots, w_n]$ be a sentence where n is the sentence length and w_i is the i-th token. Also, let $E = [e_1, e_2, \ldots, e_k]$

be the entity mentions[3] in this sentence (k is the number of the entity mentions and can be zero). Each entity mention comes with the offsets of the head and the entity type. We further assume that $i_1, i_2, \ldots, i_k$ be the indexes of the last words of the mention heads for $e_1, e_2, \ldots, e_k$, respectively.

In EE, for every token w_i in the sentence, we need to predict the event type (if any) for it. If w_i is a trigger word for some event of interest, we then need to predict the roles (if any) that each entity mention e_j plays in such event.

The joint model for event extraction in this work consists of two phases: (i) the *encoding phase* that applies recurrent neural networks to induce a more abstract representation of the sentence, and (ii) the *prediction phase* that uses the new representation to perform event trigger and argument role identification simultaneously for W. Figure 6.5 shows an overview of the model.

6.2.3.1 Encoding

Sentence Encoding

In the encoding phase, we first transform each token w_i into a real-valued vector x_i using the concatenation of the following three vectors:

1. The word embedding vector of w_i: This is obtained by looking up a pre-trained word embedding table *EMB* (Collobert and Westion, 2008; Mikolov et al., 2013b; Turian et al., 2010).

2. The real-valued embedding vector for the entity type of w_i: This vector is motivated from the prior work (Nguyen and Grishman, 2015b) and generated by looking up the entity type embedding table (initialized randomly) for the entity

3. i.e, including ACE entity mentions, times and values.

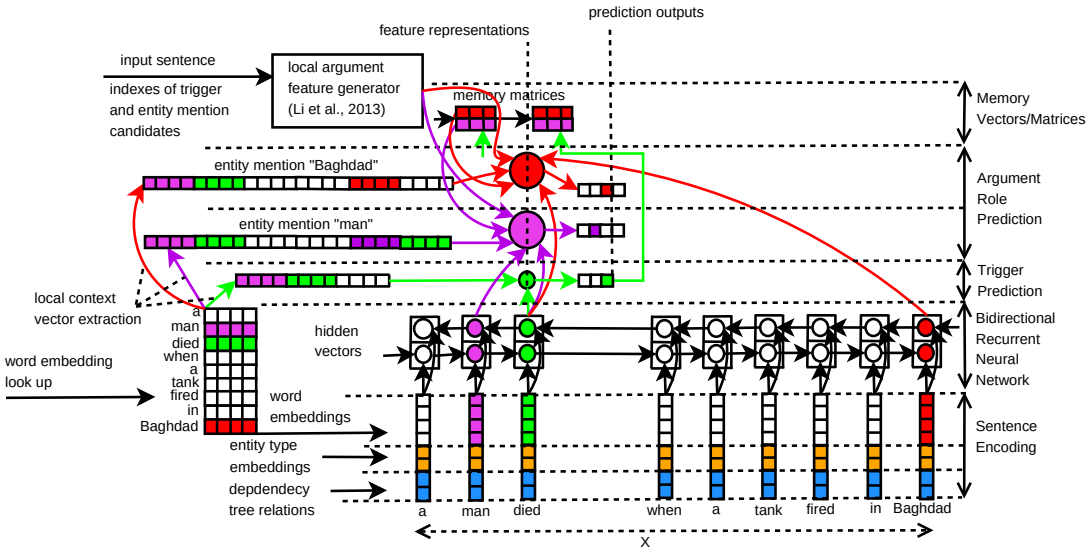

Figure 6.5: The joint EE model for the input sentence "*a man died when a tank fired in Baghdad*" with local context window $d = 1$. We only demonstrate the memory matrices $G_i^{\text{arg/trg}}$ in this figure. Green corresponds to the trigger candidate "*died*" at the current step while violet and red are for the entity mentions "*man*" and "*Baghdad*" respectively.

type of w_i. Note that we also employ the BIO annotation schema to assign entity type labels to each token in the sentences using the heads of the entity mentions as do (Nguyen and Grishman, 2015b).

3. The binary vector whose dimensions correspond to the possible relations between words in the dependency trees. The value at each dimension of this vector is set to 1 only if there exists one edge of the corresponding relation connected to w_i in the dependency tree of W. This vector represents the dependency features that are shown to be helpful in the previous research (Li et al., 2013b).

Note that we do not use the relative position features, unlike the prior work on neural networks for EE (Chen et al., 2015; Nguyen and Grishman, 2015b) because

we predict the whole sentence for triggers and argument roles jointly, thus having no fixed positions for anchoring in the sentences.

The transformation from the token w_i to the vector x_i essentially converts the input sentence W into a sequence of real-valued vectors $X = [x_1, x_2, \ldots, x_n]$, to be used by recurrent neural networks to learn a more effective representation.

Recurrent Neural Networks

Consider the input sequence $X = [x_1, x_2, \ldots, x_n]$. At each step i, we compute the hidden vector $\overrightarrow{h_i}$ based on the current input vector x_i and the previous hidden vector $\overrightarrow{h_{i-1}}$, using the non-linear transformation function Φ: $\overrightarrow{h_i} = \Phi(x_i, \overrightarrow{h_{i-1}})$. This recurrent computation is done over X to generate the hidden vector sequence $[\overrightarrow{h_1}, \overrightarrow{h_2}, \ldots, \overrightarrow{h_n}]$, denoted by $\overrightarrow{\text{RNN}}([x_1, x_2, \ldots, x_n]) = [\overrightarrow{h_1}, \overrightarrow{h_2}, \ldots, \overrightarrow{h_n}]$.

An important characteristics of the recurrent mechanism is that it adaptively accumulates the context information from position 1 to i into the hidden vector $\overrightarrow{h_i}$, making $\overrightarrow{h_i}$ a rich representation. However, $\overrightarrow{h_i}$ is not sufficient for the event trigger and argument predictions at position i as such predictions might need to rely on the context information in the future (i.e, from position i to n). In order to address this issue, we run a second RNN in the reverse direction from x_n to x_1 to generate the second hidden vector sequence: $\overleftarrow{\text{RNN}}[(x_n, x_{n-1}, \ldots, x_1)] = [\overleftarrow{h_n}, \overleftarrow{h_{n-1}}, \ldots, \overleftarrow{h_1}]$ in which $\overleftarrow{h_i}$ summarizes the context information from position n to i. Eventually, we obtain the new representation $[h_1, h_2, \ldots, h_n]$ for X by concatenating the hidden vectors in $[\overrightarrow{h_1}, \overrightarrow{h_2}, \ldots, \overrightarrow{h_n}]$ and $[\overleftarrow{h_n}, \overleftarrow{h_{n-1}}, \ldots, \overleftarrow{h_1}]$. Note that h_i essentially encapsulates the context information over the whole sentence (from 1 to n) with a greater focus on position i. Finally, we use *Gated Recurrent*

Units for the non-linear function Φ to mitigate the vanishing gradient problem (Bengio et al., 1994; Cho et al., 2014a; Chung et al., 2014).

6.2.3.2 Prediction

In order to jointly predict triggers and argument roles (or simultaneously perform the prediction tasks in EE) for W, we maintain a binary memory vector G_i^{trg} for triggers, and binary memory matrices G_i^{arg} and $G_i^{\text{arg/trg}}$ for arguments (at each time i). These vector/matrices are set to zeros initially ($i = 0$) and updated during the prediction process for W.

Given the bidirectional representation $h_1, h_2, \ldots, h_n$ in the encoding phase and the initialized memory vector/matrices, the joint prediction procedure loops over n tokens in the sentence (from 1 to n). At each time step i, we perform the following three stages in order:

(i) *trigger prediction* for w_i.

(ii) *argument role prediction* for all the entity mentions $e_1, e_2, \ldots, e_k$ with respect to the current token w_i.

(iii) *compute* G_i^{trg}, G_i^{arg} and $G_i^{\text{arg/trg}}$ for the current step using the previous memory vector/matrices G_{i-1}^{trg}, G_{i-1}^{arg} and $G_{i-1}^{\text{arg/trg}}$, and the prediction output in the earlier stages.

The output of this process would be the predicted trigger type t_i for w_i, the predicted argument roles $a_{i1}, a_{i2}, \ldots, a_{ik}$ and the memory vector/matrices G_i^{trg}, G_i^{arg} and $G_i^{\text{arg/trg}}$ for the current step. Note that t_i should be the event type if w_i

is a trigger word for some event of interest, or "*Other*" in the other cases. a_{ij}, in constrast, should be the argument role of the entity mention e_j with respect to w_i if w_i is a trigger word and e_j is an argument of the corresponding event, otherwise a_{ij} is set to "*Other*" ($j = 1$ to k).

Trigger Prediction

In the trigger prediction stage for the current token w_i, we first compute the feature representation vector R_i^{trg} for w_i using the concatenation of the following three vectors:

- h_i: the hidden vector to encapsulate the global context of the input sentence.
- L_i^{trg}: the local context vector for w_i. L_i^{trg} is generated by concatenating the vectors of the words in a context window d of w_i:

  $L_i^{\text{trg}} = [EMB[w_{i-d}], \ldots, EMB[w_i], \ldots, EMB[w_{i+d}]]$.
- G_{i-1}^{trg}: the memory vector from the previous step.

The representation vector $R_i^{\text{trg}} = [h_i, L_i^{\text{trg}}, G_{i-1}^{\text{trg}}]$ is then fed into a feed-forward neural network F^{trg} with a softmax layer in the end to compute the probability distribution $P_{i;t}^{\text{trg}}$ over the possible trigger types: $P_{i;t}^{\text{trg}} = P_i^{\text{trg}}(l = t) = F_t^{\text{trg}}(R_i^{\text{trg}})$ where t is a trigger type. Finally, we compute the predicted type t_i for w_i by: $t_i = \text{argmax}_t(P_{i;t}^{\text{trg}})$.

Argument Role Prediction

In the argument role prediction stage, we first check if the predicted trigger type t_i in the previous stage is "*Other*" or not. If yes, we can simply set a_{ij} to "`Other`" for all $j = 1$ to k and go to the next stage immediately. Otherwise, we

loop over the entity mentions $e_1, e_2, \ldots, e_k$. For each entity mention e_j with the head index of i_j, we predict the argument role a_{ij} with respect to the trigger word w_i using the following procedure.

First, we generate the feature representation vector R_{ij}^{arg} for e_j and w_i by concatenating the following vectors:

- h_i and h_{i_j}: the hidden vectors to capture the global context of the input sentence for w_i and e_j, respectively.
- L_{ij}^{arg}: the local context vector for w_i and e_j. L_{ij}^{arg} is the concatenation of the vectors of the words in the context windows of size d for w_i and w_{i_j}:

  $L_{ij}^{\text{arg}} = [EMB[w_{i-d}], \ldots, EMB[w_i], \ldots, EMB[w_{i+d}],$
  $EMB[w_{i_j-d}], \ldots, EMB[w_{i_j}], \ldots, EMB[w_{i_j+d}]]$.
- BIN_{ij}: the hidden vector for the binary feature vector VEC_{ij}. VEC_{ij} is based on the local argument features between the tokens i and i_j from (Li et al., 2013b). BIN_{ij} is then computed by feeding VEC_{ij} into a feed-forward neural network F^{binary} for further abstraction: $BIN_{ij} = F^{\text{binary}}(VEC_{ij})$.
- $G_{i-1}^{\text{arg}}[j]$ and $G_{i-1}^{\text{arg/trg}}[j]$: the memory vectors for e_j that are extracted out of the memory matrices G_{i-1}^{arg} and $G_{i-1}^{\text{arg/trg}}$ from the previous step.

In the next step, we again use a feed-forward neural network F^{arg} with a softmax layer in the end to transform $R_{ij}^{\text{arg}} = [h_i, h_{i_j}, L_{ij}^{\text{arg}}, BIN_{ij}, G_{i-1}^{\text{arg}}[j], G_{i-1}^{\text{arg/trg}}[j]]$ into the probability distribution $P_{ij;a}^{\text{trg}}$ over the possible argument roles: $P_{ij;a}^{\text{arg}} = P_{ij}^{\text{arg}}(l = a) = F_a^{\text{arg}}(R_{ij}^{\text{arg}})$ where a is an argument role. Eventually, the predicted argument role for w_i and e_j is $a_{ij} = \text{argmax}_a(P_{ij;a}^{\text{arg}})$.

Note that the binary vector VEC_{ij} enriches the feature representation R_{ij}^{arg} for argument labeling with the discrete structures discovered in the prior work on feature analysis for EE (Li et al., 2013b). These features include the shortest dependency paths, the entity types, subtypes, etc.

The Memory Vector/Matrices

An important characteristic of EE is the existence of the dependencies between trigger labels and argument roles within the same sentences (Li et al., 2013b). In this work, we encode these dependencies into the memory vectors/matrices G_i^{trg}, G_i^{arg} and $G_i^{\text{arg/trg}}$ ($i = 0$ to n) and use them as features in the trigger and argument prediction explicitly (as shown in the representation vectors R_i^{trg} and R_{ij}^{arg} above). We classify the dependencies into the following three categories:

1. The **dependencies among trigger types**: are captured by the memory vectors G_i^{trg} ($G_i^{\text{trg}} \in \{0,1\}^{n_T}$ for $i = 0, \ldots, n$, and n_T is the number of the possible trigger types). At time i, G_i^{trg} indicates which event types have been recognized before i. We obtain G_i^{trg} from G_{i-1}^{trg} and the trigger prediction output t_i at time i: $G_i^{\text{trg}}[t] = 1$ if $t = t_i$ and $G_{i-1}^{\text{trg}}[t]$ otherwise.

A motivation for such dependencies is that if a *Die* event appears somewhere in the sentences, the possibility for the later occurrence of an *Attack* event would be likely as we have shown in the previous sections.

2. The **dependencies among argument roles**: are encoded by the memory matrix G_i^{arg} ($G_i^{\text{arg}} \in \{0,1\}^{k \times n_A}$ for $i = 0, \ldots, n$, and n_A is the number of the possible argument roles). At time i, G_i^{arg} summarizes the argument roles that the entity mentions has played with some event in the past. In particular, $G_i^{\text{arg}}[j][a] =$

1 if and only if e_j has the role of a with some event before time i. G_i^{arg} is computed from G_{i-1}^{arg}, and the prediction outputs t_i and $a_{i1}, \ldots, a_{ik}$ at time i: $G_i^{\text{arg}}[j][a] = 1$ if $t_i \neq$ "*Other*" and $a = a_{ij}$, and $G_{i-1}^{\text{arg}}[j][a]$ otherwise (for $j = 1$ to k).

3. The **dependencies between argument roles and trigger types**: are encoded by the memory matrix $G_i^{\text{arg/trg}}$ ($G_i^{\text{arg/trg}} \in \{0,1\}^{k \times n_T}$ for $i = 0$ to n). At time i, $G_i^{\text{arg/trg}}$ specifies which entity mentions have been identified as arguments for which event types before. In particular, $G_i^{\text{arg/trg}}[j][t] = 1$ if and only if e_j has been detected as an argument for some event of type t before i. $G_i^{\text{arg/trg}}$ is computed from $G_{i-1}^{\text{arg/trg}}$ and the trigger prediction output t_i at time i: $G_i^{\text{arg/trg}}[j][t] = 1$ if $t_i \neq$ "*Other*" and $t = t_i$, and $G_{i-1}^{\text{arg/trg}}[j][t]$ otherwise (for all $j = 1$ to k).

6.2.3.3 Training

Denote the given trigger types and argument roles for W at training time as $T = t_1^*, t_2^*, \ldots, t_n^*$ and $A = (a_{ij}^*)_{i=1,n}^{j=1,k}$. We train the network by minimizing the joint negative log-likelihood function C for triggers and argument roles:

$$\begin{aligned} C(T, A, X, E) &= -\log P(T, A|X, E) \\ &= -\log P(T|X, E) - \log P(A|T, X, E) \\ &= -\sum_{i=1}^{n} \log P_{i;t_i^*}^{\text{trg}} \\ &\quad -\sum_{i=1}^{n} I(t_i^* \neq \text{“}Other\text{“}) \sum_{j=1}^{k} \log P_{ij;a_{ij}^*}^{\text{arg}} \end{aligned} \tag{6.1}$$

where I is the indicator function.

We apply the stochastic gradient descent algorithm with mini-batches and the AdaDelta update rule (Zeiler, 2012). The gradients are computed using back-propagation. During training, besides the weight matrices, we also optimize the word and entity type embedding tables to achieve the optimal states. Finally, we rescale the weights whose Frobenius norms exceed a hyperparameter (Kim, 2014; Nguyen and Grishman, 2015a).

6.2.4 Word Representation

Following the prior work (Chen et al., 2015; Nguyen and Grishman, 2015b), we pre-train word embeddings from a large corpus and employ them to initialize the word embedding table. In this work, following Chapter 3, besides the CBOW and SKIP-GRAM models in (Mikolov et al., 2013a, 2013b), we examine a concatenation-based variant of CBOW (C-CBOW) to train word embeddings and compare the three models to understand their effectiveness for EE. The objective of C-CBOW is to predict the target word using *the concatenation of the vectors of the words surrounding it*.

6.3 Experiments

6.3.1 Resources, Parameters and Dataset

For all the experiments below, in the encoding phase, we use 50 dimensions for the entity type embeddings, 300 dimensions for the word embeddings and 300 units in the hidden layers for the RNNs.

Regarding the prediction phase, we employ the context window of 2 for the local features, and the feed-forward neural networks with one hidden layer for F^{trg}, F^{arg} and F^{binary} (the size of the hidden layers are 600, 600 and 300 respectively).

Finally, for training, we use the mini-batch size = 50 and the parameter for the Frobenius norms = 3.

These parameter values are either inherited from the prior research (Chen et al., 2015; Nguyen and Grishman, 2015b) or selected according to the validation data.

We pre-train the word embeddings from the English Gigaword corpus utilizing the `word2vec` toolkit[4] (modified to add the C-CBOW model). Following (Baroni et al., 2014), we employ the context window of 5, the subsampling of the frequent words set to $1e$-05 and 10 negative samples.

We also evaluate the model with the ACE 2005 corpus for event extraction. For the purpose of comparison, we use the same data split as the previous work (Chen et al., 2015; Ji and Grishman, 2008; Li et al., 2013b; Liao and Grishman, 2010b; Nguyen and Grishman, 2015b). This data split includes 40 newswire articles (672 sentences) for the test set, 30 other documents (836 sentences) for the development set and 529 remaining documents (14,849 sentences) for the training set. Also, we follow the criteria of the previous work (Chen et al., 2015; Ji and Grishman, 2008; Li et al., 2013b; Liao and Grishman, 2010b) to judge the correctness of the predicted event mentions.

4. `https://code.google.com/p/word2vec`

6.3.1.1 Memory Vector/Matrices

This section evaluates the effectiveness of the memory vector and matrices presented in Section 6.2.3.2. In particular, we test the joint model on different cases where the memory vector for triggers G^{trg} and the memory matrices for arguments $G^{arg/trg}$ and G^{arg} are included or excluded from the model. As there are 4 different ways to combine $G^{arg/trg}$ and G^{arg} for argument labeling and two options to employ G^{trg} or not for trigger labeling, we have 8 systems for comparison in total. Table 6.1 reports the identification and classification performance (F1 scores) for triggers and argument roles on the development set. Note that we are using the word embeddings trained with the C-CBOW technique in this section.

System		*No*	$G^{arg/trg}$	G^{arg}	$G^{arg/trg}$+G^{arg}
No	Trigger	67.9	**68.0**	64.6	64.2
	Argument	55.6	**58.1**	55.2	53.1
G^{trg}	Trigger	63.8	61.0	61.3	66.8
	Argument	55.2	56.6	54.7	53.6

Table 6.1: Performance of the memory vector/matrices on the development set. *No* means not using the memory vector/matrices.

We observe that the memory vector G^{trg} is not helpful for the joint model as it worsens both trigger and argument role performance (considering the same choice of the memory matrices $G^{arg/trg}$ and G^{arg} (i.e, the same column in the table) except in the column with $G^{arg/trg} + G^{arg}$).

The clearest trend is that $G^{arg/trg}$ is very effective in improving the performance of argument labeling. This is true with both the inclusion and exclusion of G^{trg}. G^{arg} and its combination with $G^{arg/trg}$, on the other hand, have negative effect on

this task. Finally, $G^{\text{arg/trg}}$ and G^{arg} do not contribute much to the trigger labeling performance in general (except in the case where G^t, $G^{\text{arg/trg}}$ and G^{arg} are all applied).

These observations suggest that the dependencies *among trigger types* and *among argument roles* are not strong enough to be helpful for the joint model in this dataset. This is in contrast to the dependencies *between argument roles and trigger types* that improve the joint model significantly.

The best system corresponds to the application of the memory matrix $G^{\text{arg/trg}}$ and will be used in all the experiments below.

6.3.1.2 Word Embedding Evaluation

We investigate different techniques to obtain the pre-trained word embeddings for initialization in the joint model of EE. Table 6.2 presents the performance (for both triggers and argument roles) on the development set when the CBOW, SKIP-GRAM and C-CBOW techniques are utilized to obtain word embeddings from the same corpus. We also report the performance of the joint model when it is initialized with the `word2vec` word embeddings from (Mikolov et al., 2013a, 2013b) (trained with the Skip-gram model on Google News) (WORD2VEC). Finally, for comparison, the performance of the random word embeddings (RANDOM) is also included. All of these word embeddings are updated during the training of the model.

The first observation from the table is that RANDOM is not good enough to initialize the word embeddings for joint EE and we need to borrow some pre-

Word Embeddings	Trigger	Argument
RANDOM	59.9	50.1
SKIP-GRAM	66.7	57.1
CBOW	66.5	53.8
WORD2VEC	66.9	56.4
C-CBOW	**68.0**	**58.1**

Table 6.2: Performance of the word embedding techniques.

trained word embeddings for this purpose. Second, SKIP-GRAM, WORD2VEC and CBOW have comparable performance on trigger labeling while the argument labeling performance of SKIP-GRAM and WORD2VEC is much better than that of CBOW for the joint EE model. Third and most importantly, among the compared word embeddings, it is clear that C-CBOW significantly outperforms all the others. We believe that the better performance of C-CBOW stems from its concatenation of the multiple context word vectors, thus providing more information to learn better word embeddings than SKIP-GRAM and WORD2VEC. In addition, the concatenation mechanism essentially helps to assign different weights to different context words, thereby being more flexible than CBOW that applies a single weight for all the context words.

From now on, for consistency, C-CBOW would be utilized in all the following experiments.

6.3.1.3 Comparison to the State of the Art

The state-of-the-art systems for EE on the ACE 2005 dataset have been the pipelined system with dynamic multi-pooling convolutional neural networks by (Chen et al., 2015) (**DMCNN**) and the joint system with structured prediction

and various discrete local and global features by (Li et al., 2013b) (**Li's structure**).

Note that the pipelined system in (Chen et al., 2015) is also the best-reported system based on neural networks for EE. Table 6.3 compares these state-of-the-art systems with the joint RNN-based model in this work (denoted by **JRNN**). For completeness, we also report the performance of the following representative systems:

1) **Li's baseline**: This is the pipelined system with local features by (Li et al., 2013b).

2) **Liao's cross event**: is the system by (Liao and Grishman, 2010b) with the document-level information.

3) **Hong's cross-entity** (Hong et al., 2011): This system exploits the cross-entity inference, and is also the best-reported pipelined system with discrete features in the literature.

Model	Trigger Identification (%)			Trigger Identification + Classification (%)			Argument Identification (%)			Argument Role (%)		
	P	R	F	P	R	F	P	R	F	P	R	F
Li's basline	76.2	60.5	67.4	74.5	59.1	65.9	74.1	37.4	49.7	65.4	33.1	43.9
Liao's cross-event†	N/A			68.7	68.9	68.8	50.9	49.7	50.3	45.1	44.1	44.6
Hong's cross-entity†	N/A			72.9	64.3	68.3	53.4	52.9	53.1	51.6	45.5	48.3
Li's structure	76.9	65.0	70.4	73.7	62.3	67.5	69.8	47.9	56.8	64.7	44.4	52.7
DMCNN	80.4	67.7	**73.5**	75.6	63.6	69.1	68.8	51.9	59.1	62.2	46.9	53.5
JRNN	68.5	75.7	71.9	66.0	73.0	**69.3**	61.4	64.2	**62.8**	54.2	56.7	**55.4**

Table 6.3: Overall performance on the blind test data. "†" designates the systems that employ evidence beyond sentence level.

From the table, we see that JRNN achieves the best F1 scores (for both trigger and argument labeling) among all of the compared models. This is significant with the argument role labeling performance (an improvement of 1.9% over the best-reported model DMCNN in (Chen et al., 2015)), and demonstrates the benefit of

the joint model with RNNs and memory features in this work. In addition, as JRNN significantly outperforms the joint model with discrete features in (Li et al., 2013b) (an improvement of 1.8% and 2.7% for trigger and argument role labeling respectively), we can confirm the effectiveness of RNNs to learn effective feature representations for EE.

6.3.2 Sentences with Multiple Events

In order to further prove the effectiveness of JRNN, especially for those sentences with multiple events, we divide the test data into two parts according to the number of events in the sentences (i.e, single event and multiple events) and evaluate the performance separately, following (Chen et al., 2015). Table 6.4 shows the performance (F1 scores) of JRNN, DMCNN and two other baseline systems, named **Embeddings+T** and **CNN** in (Chen et al., 2015). Embeddings+T uses word embeddings and the traditional sentence-level features in (Li et al., 2013b) while CNN is similar to DMCNN, except that it applies the standard pooling mechanism instead of the dynamic multi-pooling method (Chen et al., 2015).

The most important observation from the table is that JRNN significantly outperforms all the other methods with a large margin when the input sentences contain more than one event (i.e, the row labeled with **1/N** in the table). In particular, JRNN is 13.9% better than DMCNN on trigger labeling while the corresponding improvement for argument role labeling is 6.5%, thereby further suggesting the benefit of JRNN with the memory features. Regarding the performance on the single event sentences, JRNN is still the best system on trigger labeling although

Stage	Model	1/1	1/N	all
Trigger	Embedding+T	68.1	25.5	59.8
	CNN	72.5	43.1	66.3
	DMCNN	74.3	50.9	69.1
	JRNN	**75.6**	**64.8**	**69.3**
Argument	Embedding+T	37.4	15.5	32.6
	CNN	51.6	36.6	48.9
	DMCNN	**54.6**	48.7	53.5
	JRNN	50.0	**55.2**	**55.4**

Table 6.4: System performance on single event sentences (**1/1**) and multiple event sentences (**1/N**).

it is worse than DMCNN on argument role labeling. This can be partly explained by the fact that DMCNN includes the position embedding features for arguments and the memory matrix $G^{\text{arg/trg}}$ in JRNN is not functioning in this single event case.

6.4 Entity Linking with Memory-augmented Neural Networks

The previous section has highlighted the benefits of the memory-augmented neural networks for event extraction. This section applies the memory-augmented networks on the task of entity linking (EL) to further demonstrate their advantages.

6.4.1 Entity Linking

Entity linking (EL) is the task to map entity mentions in documents to their correct entries (called target entities) in some existing knowledge bases (KB). As

we use the typical knowledge base of Wikipedia in this work, the problem becomes linking entity mentions in documents to their corresponding pages in Wikipedia. For instance, consider the following text:

Chelsea have long-standing rivalries with North London clubs Arsenal and Tottenham Hotspur. A strong rivalry with Leeds United dates back to several heated and controversial matches in the 1960s and 1970s.

In this text, an entity linking system should be able to link the entity mentions "*Chelsea*", "*North London*", "*Arsenal*", "*Tottenham Hotspur*", and "*Leeds United*" to their corresponding Wikipedia pages about football clubs, rather than the cities in London. EL is a challenging problem of natural language processing, as the same entity might be presented in various names, and the same entity mention string might refer to different entities in different contexts (ambiguity).

In order to tackle the ambiguity in EL, the general framework is to first generate a set of target entities in the knowledge bases as the referent candidates for each entity mention in the documents (target candidates), and then solve a ranking problem to disambiguate the entity mention. The key challenge in this paradigm is the ranking model that computes the relevance of each target entity candidate to the corresponding entity mention using the available context information in both the documents and the knowledge bases.

6.4.1.1 Prior Deep Learning Work for Entity Linking

The sequence of prediction tasks in entity linking corresponds to the sequence of entity mentions in documents where we need to perform a prediction for each

entity mention (disambiguation). The previous deep learning models have only solved entity linking by independently disambiguating entity mentions in documents (Francis-Landau et al., 2016) (the local approach for EL). Figure 6.6 demonstrates the prediction tasks that the previous deep learning models have done for our example text above.

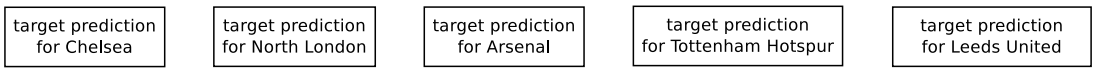

Figure 6.6: Prediction tasks for entity linking.

In this figure, each rectangle is associated with a prediction task for an entity mention in the text that would be modeled by a neural network (Francis-Landau et al., 2016). As we can see, there are no connections between the rectangles, leading to the independent disambiguation of the entity mentions.

Unfortunately, this independent mechanism in the local approach overlooks the topical coherence among target entities referred by entity mentions within the same document. The central idea is that the referent entities of some mentions in a document might in turn introduce useful information to link other mentions in that document due to the semantic relatedness among them. For instance, consider the entity mention "*Liverpool*" in the following sentence that appears after the aforementioned example text in the same document:

More recently a rivalry with Liverpool has grown following repeated clashes in cup competitions.

If we just rely on "*Liverpool*" and its context in this sentence, we might not be able to recognize the correct realistic entity for it as the context is not sufficient to

suggest an unique entity (i.e, "*Liverpool*" can be any sport clubs in this sentence based on the phrase "*cup competitions*"). However, if we refer the the previous entity mentions (i.e, '*Chelsea*", "*North London*", "*Arsenal*", "*Tottenham Hotspur*", and "*Leeds United*") that are already known as football clubs, we can use the semantic relatedness to infer that '*Liverpool*" is also a football club in this case. Such coherence has been shown to be effective for EL in the previous feature-based work (Alhelbawy and Gaizauskas, 2014; Han et al., 2011; He et al., 2013b; Hoffart et al., 2011; Pershina et al., 2015; Ratinov et al., 2011; Sil et al., 2012), but it is not exploited in the current EL models based on deep learning.

In this work, we overcome this limitation by employing the memory-augmented neural networks for EL. Specifically, given a document, we *simultaneously* perform linking for every entity mention from the beginning to the end of the document, as demonstrated in Figure 6.7. For each entity mention, we utilize convolutional neural networks (CNN) to obtain the distributed representations for the entity mention as well as its target candidates. These distributed representations are then used for two purposes: (i) computing the local similarities for the entity mention and target candidates, and (ii) functioning as the input for the recurrent neural networks (RNN) that runs over the entity mentions in the documents. The role of the RNNs is to accumulate information about the previous entity mentions and target entities, and provide them as the global constraints for the linking process of the current entity mention (i.e, the memory in our general memory-augmented neural networks). We systematically evaluate the proposed model on multiple datasets in both the general setting and the domain adaptation setting.

The experiment results show that the proposed model outperforms the current state-of-the-art models on the evaluated datasets. To our knowledge, this is also the first work investigating the EL problem in the domain adaptation setting.

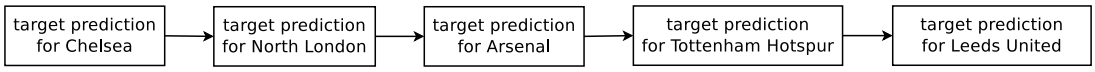

Figure 6.7: Memory-augmented neural networks for entity linking.

6.4.2 Model

The entity linking problem in this work can be formalized as follows. Let DOC be the input document and $E = \{ent_1, ent_2, \ldots, ent_k\}$ be the entity mentions in DOC. The goal is to map each entity mention ent_i to its corresponding Wikipedia page (entity) or return "*NIL*" if ent_i is not present in Wikipedia. For each entity mention $ent_i \in DOC$, let $P_i = \{page_{i1}, page_{i2}, \ldots page_{in_i}\}$ be its set of Wikipedia candidate pages (entities)[5] where n_i is the number of page candidates for ent_i. Also, let $page_i^* \in P_i$ be the correct target entity for ent_i.

Following (Francis-Landau et al., 2016), we represent each entity mention ent_i by the triple $ent_i = (suf_i, ctx_i, doc_i)$, where suf_i is the *surface string* of ent_i, ctx_i is the *immediate context* (within some predefined window) of ent_i and doc_i is the *entire document* containing ent_i. Essentially, suf_i, ctx_i and doc_i are the sequences

5. For a fair comparison, we use the target candidates provided by (Francis-Landau et al., 2016). Essentially, a query generation is executed for each entity mention, whose outputs are combined with link counts to retrieve the potential entities (including "*NIL*"). The query generation itself involves removing stop words, plural suffixes, punctuation, and leading or trailing words.

of words to capture the contexts or topics of ent_i at multiple granularities. For the target candidate pages $page_{ij}$, we use the *title* til_{ij} and *body content* bdy_{ij} to represent them ($page_{ij} = (til_{ij}, bdy_{ij})$). For convenience, we also denote $page_i^* = (til_i^*, bdy_i^*)$ for the correct entity pages. Again, til_{ij}, bdy_{ij}, til_i^* and bdy_i^* are also sequences of words.

In order to link the entity mentions, the strategy is to assign a relevance score $\phi(ent_i, page_{ij})$ for each target candidate $page_{ij}$ of ent_i, and then use these scores to rank the candidates for each mention. In this work, we decompose $\phi(ent_i, page_{ij})$ as the sum of the two following factors:

$$\phi(ent_i, page_{ij}) = \phi_{local}(ent_i, page_{ij}) + \phi_{global}(ent_1, ent_2, \ldots, ent_i, P_1, P_2, \ldots, P_i) \tag{6.2}$$

In this formula, $\phi_{local}(ent_i, page_{ij})$ represents the local similarities between ent_i and p_{ij}, i.e, only using the information related to ent_i and $page_{ij}$. $\phi_{global}(ent_1, ent_2, \ldots, ent_i, P_1, P_2, \ldots, P_i)$, on the other hand, additionally considers the other mentions and candidates in the document, attempting to model the interdependence among these objects. The denotation $\phi_{global}(ent_1, ent_2, \ldots, ent_i, P_1, P_2, \ldots, P_i)$ implies that we are computing the ranking scores for all the target candidates of all the entity mentions in each document simultaneously, preserving the order of the entity mentions from the beginning to the end of the document.

The model in this work consists of three main components: (i) the encoding component that applies convolutional neural networks to induce the distributed representations for the input sequences suf_i, ctx_i, doc_i, til_{ij}, and bdy_{ij}, (ii) the local component that computes the local similarities $\phi_{local}(ent_i, page_{ij})$ for each

entity mention ent_i, and (iii) the global component that runs recurrent neural networks on the entity mentions $\{ent_1, ent_2, \ldots, ent_k\}$ to generate the global features $\phi_{global}(ent_1, ent_2, \ldots, ent_i, P_1, P_2, \ldots, P_i)$.

6.4.2.1 Encoding

Let W be some context word sequence of the entity mentions or target candidates (i.e, $x \in \{suf_i, ctx_i, doc_i\}_i \cup \{til_{ij}, page_{ij}\}_{i,j} \cup \{til_i^*, bdy_i^*\}_i$). In order to obtain the distributed representation for W, we first transform each word $w_i \in W$ into a real-valued, m_e-dimensional vector x_i using the word embedding table EMB (Mikolov et al., 2013b): $x_i = EMB[w_i]$. This essentially converts the word sequence W into a sequence of vectors that is padded with zero vectors to form a fixed-length sequence of vectors $X = [x_1, x_2, \ldots, x_n]$ of length n.

In the next step, we apply the convolution operation over X to generate the hidden vector sequence, that is then transformed by a non-linear function g and pooled by the *sum* function (Francis-Landau et al., 2016). Following the previous work on CNN (Nguyen and Grishman, 2015a, 2015b), we utilize the set L of multiple window sizes to parameterize the convolution operation. Each window size $l \in L$ corresponds to a convolution matrix $M_l \in \mathbb{R}^{v \times lm_e}$ of dimensionality v. Eventually, the concatenation vector $\bar{W}$ of the resulting vectors for each window size in L would be used as the distributed representation for W:

$$\bar{W} = \bigoplus_{l \in L} \sum_{i=1}^{n-l+1} g(M_l x_{i:(i+l-1)}) \tag{6.3}$$

where $\bigoplus$ is the concatenation operation over the window set L and $x_{i:(i+l-1)}$ is the

concatenation vector of the given word vectors.

For convenience, let $\bar{suf}_i$, $\bar{ctx}_i$, $\bar{doc}_i$, $\bar{til}_{ij}$, $\bar{bdy}_{ij}$, $\bar{til}^*_i$ and $\bar{bdy}^*_i$ be the distributed representations of suf_i, ctx_i, doc_i, til_{ij}, bdy_{ij}, til^*_i and bdy^*_i obtained by the convolution procedure above, respectively. Note that we apply the same set of convolution parameters for each type of text granularity in the source document DOC as well as in the target entity side. The vector representations of the context would then be fed into the next components to compute the features for EL.

6.4.2.2 Local Similarities

We employ the local similarities $\phi_{local}(ent_i, page_{ij})$ from Francis-Landau et al., 2016, the state-of-the-art neural network model for EL. In particular:

$$\begin{aligned}\phi_{local}(ent_i, page_{ij}) &= \phi_{sparse}(ent_i, page_{ij}) + \phi_{CNN}(ent_i, page_{ij}) \\ &= W_{sparse}F_{sparse}(ent_i, page_{ij}) + W_{CNN}F_{CNN}(ent_i, page_{ij})\end{aligned} \tag{6.4}$$

In this formula, W_{sparse} and W_{CNN} are the weights for the feature vectors F_{sparse} and F_{CNN} respectively. $F_{sparse}(ent_i, page_{ij})$ is the sparse feature vector obtained from (Durrett and Klein, 2014). This vector captures various linguistic properties and statistics that have been discovered in the previous studies for EL. The representative features include the anchor text counts from Wikipedia, the string match indications with the title of the Wikipedia candidate pages, or the information about the shape of the queries for candidate generations (Francis-Landau et al., 2016).

$F_{CNN}(ent_i, page_{ij})$, on the other hand, involves the cosine similarities between

the representation vectors at multiple granularities of ent_i and $page_{ij}$. In particular:

$$\begin{aligned} &F_{CNN}(ent_i, page_{ij}) \\ &\quad = [cos(\bar{suf}_i, \bar{til}_{ij}), cos(\bar{ctx}_i, \bar{til}_{ij}), cos(\bar{doc}_i, \bar{til}_{ij}), \\ &\quad cos(\bar{suf}_i, \bar{bdy}_{ij}), cos(\bar{ctx}_i, \bar{bdy}_{ij}), cos(\bar{doc}_i, \bar{bdy}_{ij})] \end{aligned} \tag{6.5}$$

The intuition for this computation is that the similarities at different levels of contexts might help to enforce the potential topic compatibility between the contexts of the entity mentions and target candidates for EL (Francis-Landau et al., 2016).

6.4.2.3 Global Similarities

In order to encapsulate the coherence among the entity mentions and their target entities, we run recurrent neural networks over the sequences of the representation vectors for the entity mentions (i.e, the vector sequences for the surface strings $(\bar{suf}_1, \bar{suf}_2, \ldots, \bar{suf}_k)$ and for the immediate contexts $(\bar{ctx}_1, \bar{ctx}_2, \ldots, \bar{ctx}_k)$) and the target entities (i.e, the vector sequences for the page titles $(\bar{til}^*_1, \bar{til}^*_2, \ldots, \bar{til}^*_k)$ and for the body contents $(\bar{bdy}^*_1, \bar{bdy}^*_2, \ldots, \bar{bdy}^*_k)$)[6].

Let us take the representation vector sequence of the body contents of the target pages $(\bar{bdy}^*_1, \bar{bdy}^*_2, \ldots, \bar{bdy}^*_k)$[7] as an example. The recurrent neural network with the recurrent function Φ for this sequence will generate the hidden vector sequence $(h^b_1, h^b_2, \ldots, h^b_k)$ where: $h^b_i = \Phi(h^b_{i-1}, \bar{bdy}^*_i)$.

6. Note that we have different recurrent neural networks for different context vector sequences.

7. In the training process, $(\bar{bdy}^*_1, \bar{bdy}^*_2, \ldots, \bar{bdy}^*_k)$ are obtained from the golden target entities while at test time, they are retrieved from the predicted target entities.

Each vector h_i^b in this sequence encodes or summarizes the information about the content of the previous target entities (i.e, before i) in the document due to the property of RNN.

Given the hidden vector sequence, when predicting the target entity for the entity mention ent_i, we ensure that the target entity is consistent with the global information stored in h_{i-1}^b. This is achieved by using the cosine similarities between h_{i-1}^b and the representation vectors of each target candidate $page_{ij}$ of ent_i, (i.e, $cos(h_{i-1}^b, \bar{title}_{ij})$ and $cos(h_{i-1}^b, \bar{body}_{ij})$) as the global features for the ranking score.

We can repeat this process for the other representation vector sequences in both the entity mention side and the target entity side. The resulting global features would then be grouped into a single feature vector to compute the global similarity score $\phi_{global}(ent_1, ent_2, \ldots, ent_i, P_1, P_2, \ldots, P_i)$ as in the local similarity section. An overview of the whole model is presented in Figure 6.8.

Regarding the recurrent function Φ, we also employ the *Gated Recurrent Units* (Cho et al., 2014a) to alleviate the "*vanishing gradient problem*" of RNN. Finally, for training, we jointly optimize the parameters for the CNNs, RNNs and weight vectors by maximizing the log-likelihood of a labeled training corpus. We utilize the stochastic gradient descent algorithm and the AdaDelta update rule Zeiler, 2012. The gradients are computed via back-propagation. Following (Francis-Landau et al., 2016), we do not update the word embedding table during training.

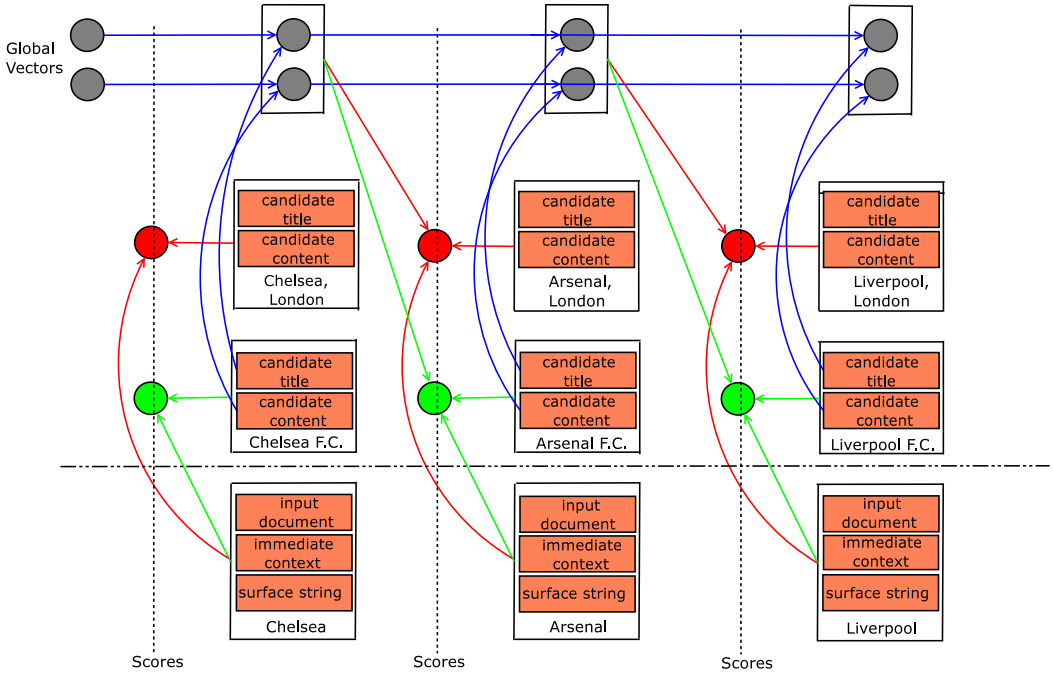

Figure 6.8: Joint model for learning local and global features for a document with 3 entity mentions: "*Chelsea*", "*Arsenal*" and "*Liverpool*". Each of the entity mentions has two entity candidate pages (either a football club or a city).The orange rectangles denote the CNN-induced representation vectors $\bar{suf}_i$, $\bar{ctx}_i$, $\bar{doc}_i$, $\bar{til}_{ij}$ and $\bar{bdy}_{ij}$. The circles in red and green are the ranking scores for the target candidates, in which the green circles correspond to the correct target entities. Finally, the circles in grey are the hidden vectors (i.e, the global vectors) of the RNNs running over the entity mentions. We only show the global entity vectors in this figure to improve the visualization.

6.4.3 Experiments

6.4.3.1 Datasets

Following (Francis-Landau et al., 2016), we evaluate the models on 4 different entity linking datasets:

i) ACE (Bentivogli et al., 2010): This corpus is from the 2005 evaluation of

NIST. It is also used in (Fahrni and Strube, 2014 and Durrett and Klein, 2014).

ii) CoNLL-YAGO (Hoffart et al., 2011): This corpus is originally from the CoNLL 2003 shared task of named entity recognition for English.

iii) WP (Heath and Bizer, 2011): This dataset consists of short snippets from Wikipedia.

iv) WIKI (Ratinov et al., 2011): This dataset contains 10,000 randomly sampled Wikipedia articles. The task is to disambiguate the links in each article[8].

For all the datasets, we use the standard data splits (for training data, test data and development data) as the previous works for comparable comparison (Francis-Landau et al., 2016).

6.4.3.2 Parameters and Resources

For all the experiments below, in the CNN models to learn the distributed representations for the inputs, we use window sizes in the set $L = \{2, 3, 4, 5\}$ for the convolution operation with the dimensionality $v = 200$ for each window size[9]. The non-linear function for transformation is $g = \tanh$.

We employ the English Wikipedia dump from June 2016 as our reference knowledge base.

8. As noted by (Francis-Landau et al., 2016) and (Nguyen et al., 2014c), the original Wikipedia dump in (Ratinov et al., 2011) is no longer accessible, so we cannot duplicate the results or conduct comparable experiments with (Ratinov et al., 2011). We instead compare our performance with (Francis-Landau et al., 2016) that provides the access to their Wikipedia dump.

9. As we need to compute the cosine similarities between the hidden vectors of the RNN models and the representation vectors of the target candidates, the number of hidden units for the RNN is set to $200|L| = 800$ naturally.

Regarding the input contexts for the entity mentions and the target candidates, we utilize the window size of 10 for the immediate context ctx_i, and only extract the first 100 words in the documents for doc_i and bdy_{ij}.

Finally, we pre-train the word embedings on the whole English Wikipedia dump using the `word2vec` toolkit (Mikolov et al., 2013b). The training parameters are set to the default values in this toolkit. The dimensionality of the word embeddings is 300.

Note that every parameter and resource in this work is either taken from the previous work (Francis-Landau et al., 2016; Nguyen and Grishman, 2016c) or selected by the development data.

6.4.3.3 Evaluating the Global Features

In this section, we evaluate the effectiveness of the global features for EL. In particular, we differentiate two types of global features based on the side of information we expect to enforce the coherence. The first type of global features (***global-mention***) concerns the entity mention side and involves applying the global RNN models on the CNN-induced representation vectors of the entity mentions (i.e, the surface vectors $(\bar{suf}_1, \bar{suf}_2, \ldots, \bar{suf}_k)$ and the immediate context vectors $(\bar{ctx}_1, \bar{ctx}_2, \ldots, \bar{ctx}_k)$). The second type of global features (***global-entity***), on the other hand, focuses on the target entity side and models the coherence with the representation vectors of the target entities (i.e, the page title vectors $(\bar{til}^*_1, \bar{til}^*_2, \ldots, \bar{til}^*_k)$ and the body content vectors $(\bar{bdy}^*_1, \bar{bdy}^*_2, \ldots, \bar{bdy}^*_k)$). Table 6.5 reports the development performance (F1 scores) of the proposed model on dif-

ferent cases where the *global-mention* and *global-entity* features are included or excluded from the model.

Global Features	Dataset		
	ACE	CoNLL	WP
No	86.1	89.3	84.0
global-mention	86.8	90.2	84.2
global-entity	**86.9**	**90.7**	**84.2**
global-mention + global-entity	86.2	90.6	84.0

Table 6.5: Performance of the global features on the development set. *No* means not using the global features.

The most important observation from the table is that the global features, in general, help to improve the performance of the model on different datasets. This is substantial on the ACE and CoNLL datasets when only one type of the global features (either *global-mention* or *global-entity*) is integrated into the model. The combination of *global-mention* and *global-entity* is not very effective as it is actually worse than the performance of the individual global feature types. This suggests that *global-mention* and *global-entity* might cover overlapping information and their combination would inject redundancy into the model. The best performance is achieved by the *global-entity* features that would be used in all the evaluations below.

6.4.4 Comparing to the Previous Work

This section compares the proposed system (called *Global-RNN*) with the state-of-the-art models on our four datasets. These systems include the neural network model in (Francis-Landau et al., 2016), the joint model for entity analysis

in (Durrett and Klein, 2014) and the AIDA-light system with two-stage mapping in (Nguyen et al., 2014c)[10]. Table 6.6 shows the performance of the systems on the test sets with the reference knowledge base of the June 2016 Wikipedia dump. We also include the performance of the systems on the December 2014 Wikipedia dump that was used and provided by (Francis-Landau et al., 2016) for further and compatible comparison.

Systems	Wikipedia 2014				Wikipedia 2016			
	ACE	CoNLL	WP	WIKI	ACE	CoNLL	WP	WIKI
DK2014 (Durrett and Klein, 2014)	79.6	-	-	-	-	-	-	-
AIDA-LIGHT (Nguyen et al., 2014b)	-	84.8	-	-	-	-	-	-
Local CNN (Francis-Landau et al., 2016)	**89.9**	85.5	90.7	82.2	86.1	84.5	90.4	81.4
Global-RNN	89.7	**87.2†**	**91.2†**	**83.7†**	**87.8†**	**86.5†**	**91.2†**	**81.7**

Table 6.6: Performance of the systems. Cells marked with †designate the *Global-RNN* models that significantly outperform the *Local CNN* model ($\rho < 0.05$).

First, we see that the performance of the systems significantly drop when we switch from Wikipedia 2014 to Wikipedia 2016 (especially for the datasets ACE and CoNLL). This is can be partly explained by the inclusion of new entities (pages) into Wikipedia from 2014 to 2016 that has made the entity mentions in the datasets more ambiguous[11]. Second and more importantly, *Global-RNN* significantly outperforms the all the compared models (except for the ACE dataset on Wikipedia 2014 and the WIKI dataset on Wikipedia 2016), thereby demonstrating the benefits of the joint modeling for local and global features via neural networks for EL in this work.

10. We note that (Alhelbawy and Gaizauskas, 2014) and (Pershina et al., 2015) also use the CoNLL-YAGO dataset for their experiments. However, since they evaluate the models on the whole dataset rather than the test set as the other works do, they are not comparable to the performance we report in this work.

11. The number of Wikipedia pages in 2014 is about 4.5 million while this number is 5 million in June 2016.

6.4.5 Domain Adaptation Experiments

The purpose of this section is to further evaluate the models in the domain adaptation setting to investigate their cross-domain robustness for EL. We refer the reader to Chapters 1 and 2 to learn more about the domain adaptation setting.

One of the key strategies of the domain adaptation techniques is the search for the domain-independent features that are discriminative across different domains (Blitzer et al., 2006; Jiang and Zhai, 2007b; Nguyen and Grishman, 2014a; Plank and Moschitti, 2013). These invariants serve as the connectors between different domains and help to transfer the knowledge from one domain to the others. For EL, we hypothesize that the global coherence is an effective domain-independent feature that would help to improve the cross-domain performance of the models. The intuition is that the entities mentioned in a document of any domains should be related to each other. Eventually, we expect that the proposed model with global coherence features would be more robust to domain shifts than the local approach (Francis-Landau et al., 2016).

Dataset

We use the ACE dataset to evaluate the cross-domain performance of the models as it involves documents in 6 different domains: broadcast conversation (`bc`), broadcast news (`bn`), telephone conversation (`cts`), newswire (`nw`), usenet (`un`) and webblogs (`wl`) that facilitate our experiments. Following the common practice of domain adaptation research on this dataset, we use `news` (the union of `bn` and `nw`) as the source domain and `bc`, `cts`, `wl`, `un` as four different target domains. We take half of `bc` as the development set and use the remaining data for testing.

Evaluation

Table 6.7 compares *Global-RNN* with the neural network EL model in (Francis-Landau et al., 2016), the best reported model on the ACE dataset in the literature[12]. In this table, the models are trained on the source domain `news`, and evaluated on `news` itself (in-domain performance) (via 5-fold cross validation) as well as on the 4 target domains `bc`, `cts`, `wl`, `un` (out-of-domain performance). The experiments in this section are done with the 2016 Wikipedia dump.

Models	Domain				
	in-domain	bc	cts	wl	un
Local CNN Francis-Landau et al., 2016	90.6	87.8	88.7	80.2	82.1
Global-RNN	**91.0**	**88.7†**	**88.9**	**81.3†**	**83.1†**

Table 6.7: Cross-domain performance. Cells marked with †designate the *Glob-RNN* models that significantly outperform the *Local CNN* model ($\rho < 0.05$).

The first observation from the table is that the performance of all the compared systems on the target domains is much worse than the corresponding in-domain performance. In particular, the performance gap between the in-domain performance and the the worst out-of-domain performance (on the domain `wl`) is up to 10%, thus indicating the mismatches between the source and the target domains for EL. Second and most importantly, *Global-RNN* is consistently better than the model with only local features in (Francis-Landau et al., 2016) over all the target domains (although it is less pronounced in the `cts` domain). This demonstrates the cross-domain robustness of the proposed model and confirms our hypothesis about the domain-independence of the global coherence features for EL.

12. The performance of the model from (Francis-Landau et al., 2016) reported in this work is obtained by running their actual released system.

Analysis

In order to better understand the performance gap in the domain adaptation experiments for EL, we visualize the representation vectors of the entity mentions in different domains. In particular, after *Global-RNN* is trained, we retrieve the representation vectors $\bar{c}_i$ for the immediate contexts of the entity mentions in the source and target domains, project them into the 2-dimension space via the t-SNE algorithm and plot them. Figure 6.9 shows the plot.

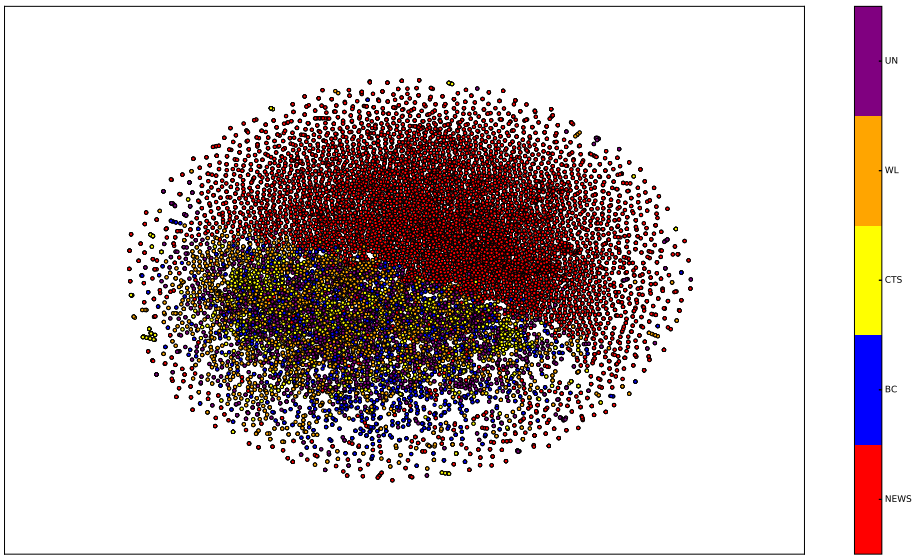

Figure 6.9: t-SNE visualization on the representation vectors ctx_i of different domains.

As we can see from the figure, the entity mentions in the target domains `bc`, `cts`, `wl` and `un` are quite separated from those of the source domain `news`, thereby explaining the performance loss in the domain adaption experiments.

It is not clear in Figure 6.9 why the models perform much worse on the target domains `wl` and `un` than the other domains (i.e, `bc` and `cts`). We further investigate this problem by computing the similarities between the target domains and the source domain. While there are several methods to estimate domain similarities (Plank and Noord, 2011), in this work, we employ the mean of the cosine similarities of every mention pair in the two domains of interest. Specifically, let DOM_1 and DOM_2 be the two domains of interest, and $DOM_1 = \{dom_1^1, dom_1^2, \ldots, dom_1^{l_1}\}$ and $DOM_2 = \{dom_2^1, dom_2^2, \ldots, dom_2^{l_2}\}$ be the sets of the representation vectors for the entity mentions in DOM_1 and DOM_2 respectively ($l_1 = |DOM_1|, l_2 = |DOM_2|$). The similarity between DOM_1 and DOM_2 is then given by:

$$Sim(DOM_1, DOM_2) = 100 \times \frac{\sum_{i=1}^{l_1} \sum_{j=1}^{l_2} cos(dom_1^i, dom_2^j)}{l_1 l_2} \tag{6.6}$$

Table 6.8 shows the similarities between the source domain `news` and each target domains `bc`, `cts`, `wl` and `un` with respect to the representation vectors of the immediate context $\bar{ctx}_i$ (*context*) and the target entity title $\bar{til}_i^*$ (*title*) for the entity mentions ent_i. We also include the similarities in which the representation vectors are the local feature vectors $F_{CNN}(ent_i, page_i^*)$ in Equation 6.5 (*interaction*). The goal of the local feature similarities is to characterize how the entity mentions in different domains interact with their target entities.

It is clear from the table that `wl` is the most dissimilar domain from the source domain. This is followed by `un` and partly explains the performance in Table 6.7.

Domain	*context*	*title*	*interaction*
bc	10.7	2.0	34.4
cts	11.4	2.0	32.6
wl	9.2	0.8	30.3
un	9.5	1.4	31.1

Table 6.8: Similarities to the source domain `news`.

6.5 Related Work

As the memory-augmented neural networks for information extraction are new, we only focus on the related work for event extraction and entity linking. The related work for event extraction can be found in Chapter 5. We only review the related work for entity linking in this section.

Entity linking or disambiguation has been studied extensively in NLP research, falling broadly into two major approaches: local and global disambiguation. Both approaches share the goal of measuring the similarities between the entity mentions and the target candidates in the reference KB. The local paradigm focuses on the internal structures of each separate mention-entity pair, covering the name string comparisons between the surfaces of the entity mentions and target candidates, entity popularity or entity type and so on (Bunescu and Pasca, 2006; Cassidy et al., 2011; Ji and Grishman, 2011; Mendes et al., 2011; Milne and Witten, 2008; Shen et al., 2014b; Sil et al., 2012; Zheng et al., 2010). In contrast, the global approach jointly maps all the entity mentions within documents to model the topical coherence. Various techniques have been exploited for capturing such semantic consistency, including Wikipedia category agreement (Cucerzan, 2007), Wikipedia link-based measures (Hoffart et al., 2011; Kulkarni et al., 2009; Shen et al., 2012),

Point-wise Mutual Information measures (Ratinov et al., 2011), integer linear programming (Cheng and Roth, 2013), PageRank (Alhelbawy and Gaizauskas, 2014; Pershina et al., 2015), stacked generalization (He et al., 2013a), to name a few. The entity linking techniques and systems have been actively evaluated at the NIST-organized Text Analysis Conference (Ji et al., 2014).

Neural networks are applied to entity linking very recently. He et al., 2013b learn enttiy representation via Stacked Denoising Auto-encoders. Sun et al., 2015 employ convolutional neural networks and neural tensor networks to model mentions, entities and contexts while Francis-Landau et al., 2016 combine CNN-based representations with sparse features to improve the performance. However, none of this work utilizes recurrent neural networks to capture the coherence features as we do in this work.

6.6 Conclusion

We propose a novel neural network framework based on memory to address the joint inference problem for information extraction. In such a framework, the prediction tasks for IE are solved simultaneously; and memories are maintained during the process to capture the interdependencies among the outputs of the tasks. We demonstrate the advantages of the proposed framework on two important tasks of IE, i.e, event extraction and entity linking. In particular, the memory-augmented neural networks can well exploit the interdependencies between event types and argument roles for EE and the topical coherence among the entities of documents for EL. The extensive experiments show that we achieve the state-of-the-art per-

formance for both tasks on the benchmark datasets.

Chapter 7

Conclusion and Future Work

This dissertation departs from the traditional feature-based methods and develops new deep learning models for information extraction tasks. The major motivations for such deep learning models are the ability to generalize over vocabulary (i.e, mitigate the unseen word/feature problems) and the automatic learning of effective feature representations. The dissertation involves three main parts. The first part in Chapter 2 introduces the use of word embeddings to generate robust features that can work well on different domains (domain adaptation) for relation extraction. The second part involves Chapters 3, 4 and 5 which explore various deep learning models for entity mention detection, relation extraction and event detection respectively. We show that such deep learning models help minimize the effort of feature engineering while achieving the best reported performance for such tasks. Finally, the third part in Chapter 6 proposes memory-augmented neural networks to simultaneously solve multiple prediction tasks for information extraction (joint inference). The memory-augmented neural networks maintain

memories to capture long range dependencies in information extraction tasks. We apply this general framework to event extraction and entity linking, resulting in the systems with state-of-the-art performance for both of the tasks.

We envision much future research on deep learning for information extraction following this dissertation. In this section, we describe two promising directions to explore in the future.

The first direction concerns the unsupervised deep learning models for information extraction. One of the major drawback of the current work is the requirement of large training datasets (labeled data) to obtain good performance (supervised learning). Such large training datasets are often very expensive to obtain in practice as they require proper creation of annotation guidelines, effective training for annotators, good domain knowledge, strict quality control etc. This hinders the portability of the systems to new domains where training data is not available or limited. In contrast to labeled data, unlabeled data and/or weakly labeled data are often abundant in application domains. How can we make use of unlabeled data and weakly labeled data to automatically build schemas and extract high-quality information from text with deep learning? The first step could be investigating unsupervised learning models in deep learning that have been applied successfully in other problems such as autoencoder models (Vincent et al., 2010), generative adversarial networks (GAN) (Goodfellow et al., 2014) etc. The insights from such research could suggest us to develop better unsupervised deep learning models for information extraction.

The second direction considers the multitask deep learning frameworks for in-

formation extraction. The main idea is to explore deep learning models that can simultaneously solve several information extraction tasks to improve the performance of the individual tasks (joint learning). Deep learning facilitates such multitask learning objectives as it allows the shared representations among different tasks. This encourages the communication among multiple tasks so they can fix the errors from each other and produce better representations for the tasks. We have seen some positive evidence for this direction in Chapter 6.

Bibliography

Agichtein, Eugene and Gravano, Luis (2000). "Snowball: Extracting Relations from Large Plain-Text Collections". In: *Proceedings of the Fifth ACM Conference on Digital Libraries*.

Ahn, David (2006). "The Stages of Event Extraction". In: *Proceedings of the Workshop on Annotating and Reasoning about Time and Events*.

Alhelbawy, Ayman and Gaizauskas, Robert (2014). "Graph Ranking for Collective Named Entity Disambiguation". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Ando, Rie and Zhang, Tong (2005). "A High-Performance Semi-Supervised Learning Method for Text Chunking". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *Proceedings of the International Conference on Learning Representations (ICLR)*.

Baroni, Macro, Bernardini, Silvia, Ferraresi, Adriano, and Zanchetta, Eros (2009). "The WaCky Wide Web: a Collection of Very Large Linguistically Processed Web-crawled Corpora". In: *Proceedings of Language Resources and Evaluation*.

BIBLIOGRAPHY

Baroni, Marco and Zamparelli, Roberto (2010). "Nouns are Vectors, Adjectives are Matrices: Representing Adjective-Noun Constructions in Semantic Space". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Baroni, Marco, Dinu, Georgiana, and Kruszewski, Germán (2014). "Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Becker, M., Hachey, B., Alex, B., and Grove, L. (2005). "Optimising Selective Sampling for Bootstrapping Named Entity Recognition". In: *Proceedings of the ICML-2005 Workshop on Learning with Multiple Views*.

Bengio, Yoshua, Ducharme, Réjean, Vincent, Pascal, and Jauvin, Christian (2003). "A Neural Probabilistic Language Model". In: *Journal of Machine Learning Research 3*.

Bengio, Yoshua, Simard, Patrice, and Frasconi, Paolo (1994). "Learning Long-term Dependencies with Gradient Descent is Difficult". In: *Journal of Machine Learning Research 3*.

Bentivogli, Luisa, Forner, Pamela, Giuliano, Claudio, Marchetti, Alessandro, Pianta, Emanuele, and Tymoshenko, Kateryna (2010). "Extending English ACE 2005 Corpus Annotation with Ground-truth Links to Wikipedia". In: *Proceedings of the 2nd Workshop on The People's Web Meets NLP: Collaboratively Constructed Semantic Resources*.

Bikel, D. M., Miller, S., Schwartz, R., and Weischedel, R. (1997). "Nymble: a High-Performance Learning Name-finder". In: *ANLP*.

BIBLIOGRAPHY

Blacoe, W. and Lapata, M. (2012). "A Comparison of Vector-based Representations for Semantic Composition". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Blitzer, John, McDonald, Ryan, and Pereira, Fernando (2006). "Domain Adaptation with Structural Correspondence Learning". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Blitzer, John, Dredze, Mark, and Pereira, Fernando (2007). "Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Blitzer, John, Foster, Dean, and Kakade, Sham (2011). "Domain Adaptation with Coupled Subspaces". In: *Proceedings of the Conference on Artificial Intelligence and Statistics*.

Bloehdorn, Stephan and Moschitti, Alessandro (2007). "Exploiting Structure and Semantics for Expressive Text Kernels". In: *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*.

Borthwick, Andrew, Sterling, John, Agichtein, Eugene, and Grishman, Ralph (1997). "Exploiting Diverse Knowledge Sources via Maximum Entropy in Named Entity Recognition". In: *Sixth Workshop on Very Large Corpora*.

Boschee, Elizabeth, Weischedel, Ralph, and Zamanian, Alex (2005). "Automatic Information Extraction". In: *Proceedings of the International Conference on Intelligence Analysis*.

Brin, Sergey (1998). "Extracting Patterns and Relations from the World Wide Web". In: *Proceedings of the International Workshop on the World Wide Web and Databases*.

BIBLIOGRAPHY

Brown, P. F., deSouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. (1992). "Class-Based n-gram Models of Natural Language". In: *Computational Linguistics*.

Bunescu, Razvan and Mooney, Raymond (2005a). "A Shortest Path Dependency Kernel for Relation Extraction". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Bunescu, Razvan and Mooney, Raymond (2005b). "Subsequence Kernels for Relation Extraction". In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*.

Bunescu, Razvan and Pasca, Marius (2006). "Using Encyclopedic Knowledge for Named Entity Disambiguation". In: *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*.

Cao, Kai, Li, Xiang, and Grishman, Ralph (2015a). "Improving Event Detection with Dependency Regularization". In: *Proceedings of the Recent Advances in Natural Language Processing (RANLP)*.

Cao, Kai, Li, Xiang, Fan, Miao, and Grishman, Ralph (2015b). "Improving Event Detection with Active Learning". In: *Proceedings of the Recent Advances in Natural Language Processing (RANLP)*.

Carreras, Xavier, Màrques, Lluís, and Padró, Lluís (2002). "Named Entity Extraction using AdaBoost". In: *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*.

Cassidy, Taylor, Chen, Zheng, Artiles, Javier, Ji, Heng, Deng, Hongbo, Ratinov, Lev-Arie, Zheng, Jing, Han, Jiawei, and Roth, Dan (2011). "CUNY-UIUC-SRI TAC-KBP2011 Entity Linking System Description". In: *Proceedings of Text Analysis Conference (TAC)*.

BIBLIOGRAPHY

Chan, Yee Seng and Roth, Dan (2010). "Exploiting Background Knowledge for Relation Extraction". In: *Proceedings of the International Conference on Computational Linguistics (COLING)*.

Chen, Yubo, Xu, Liheng, Liu, Kang, Zeng, Daojian, and Zhao, Jun (2015). "Event Extraction via Dynamic Multi-Pooling Convolutional Neural Networks". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Cheng, Xiao and Roth, Dan (2013). "Relational Inference for Wikification". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Cherry, Colin and Guo, Hongyu (2015). "The Unreasonable Effectiveness of Word Representations for Twitter Named Entity Recognition". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference (HLT-NAACL)*.

Cho, Kyunghyun, Merrienboer, Bart van, Gulcehre, Caglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua (2014a). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Cho, Kyunghyun (2014b). "Quick Introduction to Natural Language Processing with Neural Networks". In: *Lecture at the Ecole Polytechnique de Montreal*.

Chung, Junyoung, Gulcehre, Caglar, Cho, KyungHyun, and Bengio, Yoshua (2014). "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *arXiv preprint arXiv:1412.3555*.

BIBLIOGRAPHY

Clark, Kevin and Manning, Christopher D. (2016). "Improving Coreference Resolution by Learning Entity-Level Distributed Representations". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Collins, Michael and Singer, Yoram (1999). "Unsupervised Models for Named Entity Classification". In: *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Collobert, Ronan and Westion, Jason (2008). "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning". In: *Proceedings of the International Conference on Machine Learning (ICML)*.

Collobert, Ronan, Weston, Jason, Bottou, Leon, Karlen, Michael, Kavukcuoglu, Koray, and Kuksa, Pavel (2011). "Natural Language Processing (almost) from Scratch". In: *Journal of Machine Learning Research*.

Craven, Mark and Kumlien, Johan (1999). "Constructing Biological Knowledge Bases by Extracting Information from Text Sources". In: *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*.

Cristianini, Nello and Taylor, John Shawe (2000). "An Introduction to Support Vector Machines and Other Kernel-based Learning Methods". In: *Cambridge University Press*.

Cucerzan, Silviu (2007). "Large-Scale Named Entity Disambiguation Based on Wikipedia Data". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Daume, Hal (2007). "Frustratingly Easy Domain Adaptation". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

BIBLIOGRAPHY

Daume, Hal, Kumar, A., and Saha, A. (2010). "Co-regularization Based Semi-supervised Domain Adaptation". In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*.

Duchi, John, Hazan, Elad, and Singer, Yoram (2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research 12 2121-2159*.

Durrett, Greg and Klein, Dan (2013). "Easy Victories and Uphill Battles in Coreference Resolution". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Durrett, Greg and Klein, Dan (2014). "A Joint Model for Entity Analysis: Coreference, Typing, and Linking". In: *Transactions of the Association for Computational Linguistics (TACL)*.

Fahrni, Angela and Strube, Michael (2014). "A Latent Variable Model for Discourse-aware Concept and Entity Disambiguation". In: *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*.

Florian, Radu, Ittycheriah, Abe, Jing, Hongyan, and Zhang, Tong (2003). "Named Entity Recognition through Classifier Combination". In: *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*.

Florian, R., Hassan, H., Ittycheriah, A., Jing, H., Kambhatla, N., Luo, X., Nicolov, N., and Roukos, S. (2004). "A Statistical Model for Multilingual Entity Detection and Tracking". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference (HLT-NAACL)*.

Florian, Radu, Jing, Hongyan, Kambhatla, Nanda, and Zitouni, Imed (2006). "Factorizing Complex Models: A Case Study in Mention Detection". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

BIBLIOGRAPHY

Florian, Radu, Pitrelli, John, Roukos, Salum, and Zitouni, Imed (2010). "Improving Mention Detection Robustness to Noisy Input". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Francis-Landau, Matthew, Durrett, Greg, and Klein, Dan (2016). "Capturing Semantic Similarity for Entity Linking with Convolutional Neural Networks". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference (HLT-NAACL)*.

Fu, Lisheng and Grishman, Ralph (2013). "An Efficient Active Learning Framework for New Relation Types". In: *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*.

Gillick, Dan, Brunk, Cliff, Vinyals, Oriol, and Subramanya, Amarnag (2015). "Multilingual Language Processing From Bytes". In: *arXiv preprint arXiv:1512.00103*.

Giuliano, Claudio, Lavelli, Alberto, and Romano, Lorenza (2007). "Relation Extraction and the Influence of Automatic Named-entity Recognition". In: *ACM Transactions on Speech and Language Processing (TSLP), Volume 5, Issue 1*.

Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua (2014). "Generative Adversarial Networks". In: *arXiv preprint arXiv:1406.2661*.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron (2016). "Deep Learning". In: *MIT Press*.

Gormley, Matthew R., Yu, Mo, and Dredze, Mark (2015). "Improved Relation Extraction with Feature-Rich Compositional Embedding Models". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

BIBLIOGRAPHY

Grishman, Ralph, Westbrook, David, and Meyers, Adam (2005). "NYU's English ACE 2005 System Description". In: *The ACE 2005 Evaluation Workshop*.

Grishman, R. (2012). "Information Extraction: Capabilities and Challenges". In: *International Winter School in Language and Speech Technologies, Rovira i Virgili University, Spain*.

Gupta, Prashant and Ji, Heng (2009). "Predicting Unknown Time Arguments Based on Cross-Event Propagation". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Han, Xianpei, Sun, Le, and Zhao, Jun (2011). "Collective Entity Linking in Web Text: A Graph-based Method". In: *Proceedings of the Special Interest Group on Information Retrieval (SIGIR)*.

Hasegawa, T., Sekine, S., and Grishman, R. (2004). "Discovering Relations among Named Entities from Large Corpora". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

He, Zhengyan, Liu, Shujie, Song, Yang, Li, Mu, Zhou, Ming, and Wang, Houfeng (2013a). "Efficient Collective Entity Linking with Stacking". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

He, Zhengyan, Liu, Shujie, Li, Mu, Zhou, Ming, Zhang, Longkai, and Wang, Houfeng (2013b). "Learning Entity Representation for Entity Disambiguation". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Heath, Tom and Bizer, Christian (2011). "Linked Data: Evolving the Web into a Global Data Space". In: *Morgan and Claypool, 1st edition*.

Hendrickx, Iris, Kim, Su Nam, Kozareva, Zornitsa, Nakov, Preslav, Séaghdha, Diarmuid Ó, Padó, Sebastian, Pennacchiotti, Marco, Romano, Lorenza, and

Szpakowicz, Stan (2010). "SemEval-2010 Task 8: Multi-Way Classification of Semantic Relations Between Pairs of Nominals". In: *Proceedings of the International Workshop on Semantic Evaluation (SemEval)*.

Hochreiter, Sepp and Schmidhuber, Jurgen (1997). "Long Short-Term Memory". In: *Neural Computation*.

Hoffart, Johannes, Yosef, Mohamed Amir, Bordino, Ilaria, Fürstenau, Hagen, Pinkal, Manfred, Spaniol, Marc, Thater, Stefan, and Weikum, Gerhard (2011). "Robust Disambiguation of Named Entities in Text". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Hoffmann, Raphael, Zhang, Congle, Ling, Xiao, Zettlemoyer, Luke, and Weld, Daniel (2011). "Knowledge-based Weak Supervision for Information Extraction of Overlapping Relations". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Hong, Yu, Zhang, Jianfeng, Ma, Bin, Yao, Jian-Min, Zhou, Guodong, and Zhu, Qiaoming (2011). "Using Cross-entity Inference to Improve Event Extraction". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Huang, Fei and Yates, Alexander (2010). "Exploring Representation-Learning Approaches to Domain Adaptation". In: *The ACL Workshop on Domain Adaptation for Natural Language Processing (DANLP)*.

Huang, Ruihong and Riloff, Ellen (2012). "Modeling Textual Cohesion for Event Extraction". In: *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)*.

Jagannatha, Abhyuday N and Yu, Hong (2016). "Bidirectional RNN for Medical Event Detection in Electronic Health Records". In: *Proceedings of the North*

American Chapter of the Association for Computational Linguistics Conference (HLT-NAACL).

Ji, Heng and Grishman, Ralph (2005). "Improving Name Tagging by Reference Resolution and Relation Detection". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL).*

Ji, Heng, Westbrook, David, and Grishman, Ralph (2005). "Using Semantic Relations to Refine Coreference Decisions". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Ji, Heng and Grishman, Ralph (2008). "Refining Event Extraction through Cross-Document Inference". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL).*

Ji, Heng and Grishman, Ralph (2011). "Knowledge Base Population: Successful Approaches and Challenges". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL).*

Ji, Heng, Nothman, Joel, and Hachey, Ben (2014). "Overview of TAC-KBP2014 Entity Discovery and Linking Tasks". In: *Proceedings of Text Analysis Conference (TAC).*

Jiang, Jing and Zhai, ChengXiang (2007a). "A Systematic Exploration of the Feature Space for Relation Extraction". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference (HLT-NAACL).*

Jiang, Jing and Zhai, ChengXiang (2007b). "Instance Weighting for Domain Adaptation in NLP". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL).*

BIBLIOGRAPHY

Jiang, Jing and Zhai, ChengXiang (2007c). "A Two-stage Approach to Domain Adaptation for Statistical Classifiers". In: *Proceedings of the Conference on Information and Knowledge Management (CIKM)*.

Józefowicz, Rafal, Zaremba, Wojciech, and Sutskever, Ilya (2015). "An Empirical Exploration of Recurrent Network Architectures". In: *Proceedings of the International Conference on Machine Learning (ICML)*.

Kalchbrenner, Nal, Grefenstette, Edward, and Blunsom, Phil (2014). "A Convolutional Neural Network for Modelling Sentences". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Kambhatla, Nanda (2004). "Combining Lexical, Syntactic, and Semantic Features with Maximum Entropy Models for Information Extraction". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Kate, J. Rohit and Mooney, Raymond (2010). "Joint Entity and Relation Extraction Using Card-Pyramid Parsing". In: *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*.

Kim, Yoon (2014). "Convolutional Neural Networks for Sentence Classification". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Kingma, Diederik and Ba, Jimmy (2014). "Adam: A Method for Stochastic Optimization". In: *arXiv preprint arXiv:1412.6980*.

Kuksa, Pavel, Qi, Yanjun, Bai, Bing, Collobert, Ronan, Weston, Jason, Pavlovic, Vladimir, and Ning, Xia (2010). "Semi-supervised Abstraction-augmented String Kernel for Multi-level Bio-relation Extraction". In: *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*.

BIBLIOGRAPHY

Kulkarni, Sayali, Singh, Amit, Ramakrishnan, Ganesh, and Chakrabarti, Soumen (2009). "Collective Annotation of Wikipedia Entities in Web Text". In: *Proceedings of the Association for Computing Machinery's (ACM) Special Interest Group (SIG) on Knowledge Discovery and Data Mining (SIGKDD).*

Lafferty, John, McCallum, Andrew, and Pereira, Fernando (2001). "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Proceedings of the International Conference on Machine Learning (ICML).*

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick (1988). "Gradient-based Learning Applied to Document Recognition". In: *Proceedings of the IEEE, 86(11).*

Lei, Tao, Barzilay, Regina, and Jaakkola, Tommi (2015). "Molding CNNs for Text: Non-linear, Non-consecutive Convolutions". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Li, Peifeng, Zhu, Qiaoming, and Zhou, Guodong (2013a). "Argument Inference from Relevant Event Mentions in Chinese Argument Extraction". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL).*

Li, Qi, Ji, Heng, and Huang, Liang (2013b). "Joint Event Extraction via Structured Prediction with Global Features". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL).*

Li, Qi and Ji, Heng (2014a). "Incremental Joint Extraction of Entity Mentions and Relations". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL).*

Li, Qi, Ji, Heng, Hong, Yu, and Li, Sujian (2014b). "Constructing Information Networks Using One Single Model". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Li, Xiang, Nguyen, Thien Huu, Cao, Kai, and Grishman, Ralph (2015). "Improving Event Detection with Abstract Meaning Representation". In: *Proceedings of ACL-IJCNLP Workshop on Computing News Storylines (CNewS)*.

Liao, Shasha and Grishman, Ralph (2010a). "Filtered Ranking for Bootstrapping in Event Extraction". In: *Proceedings of the 23rd International Conference on Computational Linguistics (COLING)*.

Liao, Shasha and Grishman, Ralph (2010b). "Using Document Level Cross-event Inference to Improve Event Extraction". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Liao, Shasha and Grishman, Ralph (2011). "Acquiring Topic Features to Improve Event Extraction: in Preselected and Balanced Collections". In: *Proceedings of the Recent Advances in Natural Language Processing (RANLP)*.

Lin, Dekang and Wu, Xiaoyun (2009). "Phrase Clustering for Discriminative Learning". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Liu, ChunYang, Sun, WenBo, Chao, WenHan, and Che, WanXiang (2013). "Convolution Neural Network for Relation Extraction". In: *Proceedings of 9th International Conference on Advanced Data Mining and Applications, Part II (ADMA 2013)*.

Liu, Yang, Wei, Furu, Li, Sujian, Ji, Heng, Zhou, Ming, and WANG, Houfeng (2015). "A Dependency-Based Neural Network for Relation Classification". In:

Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL).

Liu, Shulin, Liu, Kang, He, Shizhu, and Zhao, Jun (2016). "A Probabilistic Soft Logic Based Approach to Exploiting Latent and Global Information in Event Classification". In: *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI).*

McClosky, David, Surdeanu, Mihai, and Manning, Chris (2011). "Event Extraction as Dependency Parsing". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL).*

McClosky, David, Charniak, Eugene, and Johnson, Mark (2010). "Automatic Domain Adaptation for Parsing". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference (HLT-NAACL).*

Mendes, Pablo N., Jakob, Max, García-Silva, Andrés, and Bizer, Christian (2011). "DBpedia Spotlight: Shedding Light on the Web of Documents". In: *Proceedings of the 7th International Conference on Semantic Systems.*

Mesnil, Gregoire, He, Xiaodong, Deng, Li, and Bengio, Yoshua (2013). "Investigation of Recurrent Neural Network Architectures and Learning Methods for Spoken Language Understanding". In: *Interspeech.*

Mikolov, Tomas, Chen, Kai, Corrado, Greg, and Dean, Jeffrey (2013a). "Efficient Estimation of Word Representations in Vector Space". In: *Proceedings of the International Conference on Learning Representations (ICLR).*

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg, and Dean, Jeffrey (2013b). "Distributed Representations of Words and Phrases and their Compositionality". In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS).*

BIBLIOGRAPHY

Miller, Scott, Guinness, Jethran, and Zamanian, Alex (2004). "Name Tagging with Word Clusters and Discriminative Training". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference (HLT-NAACL)*.

Milne, David and Witten, Ian H. (2008). "Learning to Link with Wikipedia". In: *Proceedings of the Conference on Information and Knowledge Management (CIKM)*.

Min, Bonan, Shi, Shuming, Grishman, Ralph, and Lin, Chin-Yew (2012). "Ensemble Semantics for Large-scale Unsupervised Relation Extraction". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Mintz, M., Bills, S., Snow, R., and Jurafsky, D. (2009). "Distant Supervision for Relation Extraction without Labeled Data". In: *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Miwa, Makoto, Thompson, Paul, Korkontzelos, Ioannis, and Ananiadou, Sophia (2014). "Comparable Study of Event Extraction in Newswire and Biomedical Domains". In: *Proceedings of the International Conference on Computational Linguistics (COLING)*.

Mnih, Andriy and Hinton, Geoffrey (2007). "Three new Graphical Models for Statistical Language Modelling". In: *Proceedings of the International Conference on Machine Learning (ICML)*.

Mnih, Andriy and Hinton, Geoffrey (2008). "A Scalable Hierarchical Distributed Language Model". In: *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*.

BIBLIOGRAPHY

Moschitti, Alessandro (2006). "Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees". In: *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*.

Moschitti, Alessandro (2008). "Kernel Methods, Syntax and Semantics for Relational Text Categorization". In: *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*.

Nguyen, T. Truc-Vien, Moschitti, Alessandro, and Riccardi, Giuseppe (2009). "Convolution Kernels on Constituent, Dependency and Sequential Structures for Relation Extraction". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Nguyen, Thien Huu and Grishman, Ralph (2014a). "Employing Word Representations and Regularization for Domain Adaptation of Relation Extraction". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Nguyen, Thien Huu, He, Yifan, Pershina, Maria, Li, Xiang, and Grishman, Ralph (2014b). "New York University 2014 Knowledge Base Population Systems". In: *Proceedings of Text Analysis Conference (TAC)*.

Nguyen, Dat Ba, Hoffart, Johannes, Theobald, Martin, and Weikum, Gerhard (2014c). "AIDA-light: High-Throughput Named-Entity Disambiguation". In: *Proceedings of the International World Wide Web Conference (WWW)*.

Nguyen, Luan Minh, Tsang, W. Ivor, Chai, A. Kian Ming, and Chieu, Leong Hai (2014d). "Robust Domain Adaptation for Relation Extraction via Clustering Consistency". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

BIBLIOGRAPHY

Nguyen, Thien Huu and Grishman, Ralph (2015a). "Relation Extraction: Perspective from Convolutional Neural Networks". In: *The NAACL Workshop on Vector Space Modeling for NLP (VSM)*.

Nguyen, Thien Huu and Grishman, Ralph (2015b). "Event Detection and Domain Adaptation with Convolutional Neural Networks". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Nguyen, Thien Huu, Plank, Barbara, and Grishman, Ralph (2015c). "Semantic Representations for Domain Adaptation: A Case Study on the Tree Kernel-based Method for Relation Extraction". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Nguyen, Thien Huu, Cho, Kyunghyun, and Grishman, Ralph (2016a). "Joint Event Extraction via Recurrent Neural Networks". In: *NAACL*.

Nguyen, Thien Huu, Fu, Lisheng, Cho, Kyunghyun, and Grishman, Ralph (2016b). "A Two-stage Approach for Extending Event Detection to New Types via Neural Networks". In: *Proceedings of the 1st ACL Workshop on Representation Learning for NLP (RepL4NLP)*.

Nguyen, Thien Huu and Grishman, Ralph (2016c). "Combining Neural Networks and Log-linear Models to Improve Relation Extraction". In: *Proceedings of IJCAI Workshop on Deep Learning for Artificial Intelligence (DLAI)*.

Nguyen, Thien Huu, Sil, Avirup, Dinu, Georgiana, and Florian, Radu (2016d). "Toward Mention Detection Robustness with Recurrent Neural Networks". In: *Proceedings of IJCAI Workshop on Deep Learning for Artificial Intelligence (DLAI)*.

Nguyen, Thien Huu and Grishman, Ralph (2016e). "Modeling Skip-Grams for Event Detection with Convolutional Neural Networks". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Nguyen, Thien Huu, Fauceglia, Nicolas, Muro, Mariano Rodriguez, Hassanzadeh, Oktie, Gliozzo, Alfio Massimiliano, and Sadoghi, Mohammad (2016f). "Joint Learning of Local and Global Features for Entity Linking via Neural Networks". In: *Proceedings of the International Conference on Computational Linguistics (COLING)*.

Nguyen, Thien Huu, Meyers, Adam, and Grishman, Ralph (2016g). "New York University 2016 System for KBP Event Nugget: A Deep Learning Approach". In: *Proceedings of Text Analysis Conference (TAC)*.

Nothman, Joel, Ringland, Nicky, Radford, Will, Murphy, Tara, and Curran, James R (2013). "Learning Multilingual Named Entity Recognition from Wikipedia". In: *Artificial Intelligence*.

Paperno, Denis, Pham, The Nghia, and Baroni, Marco (2014). "A Practical and Linguistically-motivated Approach to Compositional Distributional Semantics". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua (2012). "On the Difficulty of Training Recurrent Neural Networks". In: *arXiv preprint arXiv:1211.5063*.

Passos, Alexandre, Kumar, Vineet, and McCallum, Andrew (2014). "Lexicon Infused Phrase Embeddings for Named Entity Resolution". In: *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*.

Patwardhan, Siddharth and Rilof, Ellen (2009). "A Unified Model of Phrasal and Sentential Evidence for Information Extraction". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Pedersen, Ted (2008). "Empiricism is Not a Matter of Faith". In: *Computational Linguistics 3*.

Pershina, Maria, He, Yifan, and Grishman, Ralph (2015). "Personalized Page Rank for Named Entity Disambiguation". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference (HLT-NAACL)*.

Plank, Barbara and Noord, Gertjan van (2011). "Effective Measures of Domain Similarity for Parsing". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Plank, Barbara and Moschitti, Alessandro (2013). "Embedding Semantic Similarity in Tree Kernels for Domain Adaptation of Relation Extraction". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Poon, Hoifung and Vanderwende, Lucy (2010). "Joint Inference for Knowledge Extraction from Biomedical Literature". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference (HLT-NAACL)*.

Qian, Longhua, Zhou, Guodong, Kong, Fang, Zhu, Qiaoming, and Qian, Peide (2008). "Exploiting Constituent Dependencies for Tree Kernel-Based Semantic Relation Extraction". In: *Proceedings of the International Conference on Computational Linguistics (COLING)*.

BIBLIOGRAPHY

Raghunathan, Karthik, Lee, Heeyoung, Rangarajan, Sudarshan, Chambers, Nate, Surdeanu, Mihai, Jurafsky, Dan, and Manning, Christopher (2010). "A Multi-Pass Sieve for Coreference Resolution". In: *EMNLP*.

Ratinov, Lev and Roth, Dan (2009). "Design Challenges and Misconceptions in Named Entity Recognition". In: *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*.

Ratinov, Lev, Roth, Dan, Downey, Doug, and Anderson, Mike (2011). "Local and Global Algorithms for Disambiguation to Wikipedia". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Riedel, Sebastian, Chun, Hong-Woo, Takagi, Toshihisa, and Tsujii, Junichi (2009). "A Markov Logic Approach to Bio-Molecular Event Extraction". In: *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*.

Riedel, Sebastian and McCallum, Andrew (2011a). "Fast and Robust Joint Models for Biomedical Event Extraction". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Riedel, Sebastian and McCallum, Andrew (2011b). "Robust Biomedical Event Extraction with Dual Decomposition and Minimal Domain Adaptation". In: *Proceedings of the BioNLP Shared Task 2011 Workshop*.

Riloff, E. (1996). "Automatically Generating Extraction Patterns from Untagged Text". In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*.

Ritter, Alan, Clark, Sam, Mausam, and Etzioni, Oren (2011). "Named Entity Recognition in Tweets: An Experimental Study". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Roth, Dan and Yih, Wen-tau (2004). "A Linear Programming Formulation for Global Inference in Natural Language Tasks". In: *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*.

Roth, D. and Yih, W. (2007). "Global Inference for Entity and Relation Identification via a Linear Programming Formulation". In: *Introduction to Statistical Relational Learning*.

Sam, Rathany Chan, Le, Huong Thanh, Nguyen, Thuy Thanh, and Nguyen, Thien Huu (2011). "Combining Proper Name-Coreference with Conditional Random Fields for Semi-supervised Named Entity Recognition in Vietnamese Text". In: *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*.

Sang, Erik F. Tjong Kim and Meulder, Fien De (2002). "Introduction to the CoNLL-2002 Shared Task: Language-Independent Named Entity Recognition". In: *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*.

Sang, Erik F. Tjong Kim and Meulder, Fien De (2003). "Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition". In: *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*.

Santos, Cicero dos, Xiang, Bing, and Zhou, Bowen (2015a). "Classifying Relations by Ranking with Convolutional Neural Networks". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Schnabel, Tobias and Schütze, Hinrich (2014). "FLORS: Fast and Simple Domain Adaptation for Part-of-Speech Tagging". In: *Transactions of the Association of Computational Linguistics (TACL)*.

Shen, Wei, Wang, Jianyong, Luo, Ping, and Wang, Min (2012). "LINDEN: Linking Named Entities with Knowledge Base via Semantic Knowledge". In: *Proceedings of the International World Wide Web Conference (WWW)*.

Shen, Yelong, He, Xiaodong, Gao, Jianfeng, Deng, Li, and Mesnil, Gregoire (2014a). "Learning Semantic Representations Using Convolutional Neural Networks for Web Search". In: *Proceedings of the International World Wide Web Conference (WWW)*.

Shen, Wei, Wang, Jianyong, and Han, Jiawei (2014b). "Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions". In: *TKDE*.

Shinyama, Y. and Sekine, S. (2006). "Preemptive Information Extraction Using Unrestricted Relation Discovery". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference (HLT-NAACL)*.

Sil, Avirup, Cronin, Ernest, Nie, Penghai, Yang, Yinfei, Popescu, Ana-Maria, and Yates, Alexander (2012). "Linking Named Entities to Any Database". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Singh, Sameer, Riedel, Sebastian, Martin, Brian, Zheng, Jiaping, and McCallum, Andrew (2013). "Joint Inference of Entities, Relations, and Coreference". In: *CIKM Workshop on Automated Knowledge Base Construction (AKBC)*.

Socher, Richard, Perelygin, Alex, Wu, Jean Y., Chuang, Jason, Manning, Christopher D., Ng, Andrew Y., and Potts, Christopher (2012a). "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Socher, Richard, Huval, Brody, Manning, Christopher D., and Ng, Andrew Y. (2012b). "Semantic Compositionality Through Recursive Matrix-Vector Spaces". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Srivastava, N., Hinton, G., Krizhevsky, A., and Salakhutdinov, R. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research 15(1):1929-1958*.

Sterckx, Lucas, Demeester, Thomas, Deleu, Johannes, and Develder, Chris (2014). "Using Active Learning and Semantic Clustering for Noise Reduction in Distant Supervision". In: *Proceedings of the Workshop on Automated Knowledge Base Construction*.

Stevenson, M. and Greenwood, M. A. (2005). "A Semantic Approach to IE Pattern Induction". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Sun, Ang, Grishman, Ralph, and Sekine, Satoshi (2011). "Semi-supervised Relation Extraction with Large-scale Word Clustering". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Sun, Ang and Grishman, Ralph (2012). "Active Learning for Relation Type Extension with Local and Global Data Views". In: *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*.

Sun, Yaming, Lin, Lei, Tang, Duyu, Yang, Nan, Ji, Zhenzhou, and Wang, Xiaolong (2015). "Modeling Mention, Context and Entity with Neural Networks for Entity Disambiguation". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Surdeanu, M., Tibshirani, J., Nallapati, R., and Manning, C. D. (2012). "Multi-instance Multi-label Learning for Relation Extraction". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Suzuki, Jun and Isozaki, Hideki (2008). "Semi-Supervised Sequential Labeling and Segmentation Using Giga-Word Scale Unlabeled Data". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Turian, Joseph, Ratinov, Lev-Arie, and Bengio, Yoshua (2010). "Word Representations: A Simple and General Method for Semi-Supervised Learning". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Venugopal, Deepak, Chen, Chen, Gogate, Vibhav, and Ng, Vincent (2014). "Relieving the Computational Bottleneck: Joint Inference for Event Extraction with High-Dimensional Features". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Vincent, Pascal, Larochelle, Hugo, Lajoie, Isabelle, Bengio, Yoshua, and Manzagol, Pierre-Antoine (2010). "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion". In: *Journal of Machine Learning Research 11*.

Wang, Mengqiu (2008). "A Re-examination of Dependency Path Kernels for Relation Extraction". In: *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*.

Werning, Markus, Machery, Edouard, and Schurz, Gerhard (2006). "Compositionality of Meaning and Content: Foundational Issues (Linguistics & Philosophy)". In: *Linguistics & Philosophy*.

Wiseman, Sam, Rush, Alexander M, Shieber, Stuart M, and Weston, Jason (2015). "Learning Anaphoricity and Antecedent Ranking Features for Coreference Resolution". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Xiao, Min and Guo, Yuhong (2013). "Domain Adaptation for Sequence Labeling Tasks with a Probabilistic Language Adaptation Model". In: *Proceedings of the International Conference on Machine Learning (ICML)*.

Xu, Yan, Mou, Lili, Li, Ge, Chen, Yunchuan, Peng, Hao, and Jin, Zhi (2015). "Classifying Relations via Long Short Term Memory Networks along Shortest Dependency Paths". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yangarber, R., Grishman, R., Tapanainen, P., and Huttunen, S. (2000). "Automatic Acquisition of Domain Knowledge for Information Extraction". In: *Proceedings of the International Conference on Computational Linguistics (COLING)*.

Yangarber, R., Lin, W., and Grishman, R. (2002). "Unsupervised Learning of Generalized Names". In: *Proceedings of the 19th International Conference on Computational Linguistics (COLING)*.

Yangarber, R. (2003). "Counter-Training in Discovery of Semantic Patterns". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Yao, L., Haghighi, A., Riedel, S., and McCallum, A. (2011). "Structured Relation Discovery Using Generative Models". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yao, Kaisheng, Zweig, Geoffrey, Hwang, Mei-Yuh, Shi, Yangyang, and Yu, Dong (2013). "Recurrent Neural Networks for Language Understanding". In: *Interspeech*. Interspeech.

Yao, Kaisheng, Peng, Baolin, Zhang, Yu, Yu, Dong, Zweig, Geoffrey, and Shi, Yangyang (2014). "Spoken Language Understanding Using Long Short-Term Memory Neural Networks". In: *Proceedings of the IEEE Workshop on Spoken Lanuage Technology*.

Yates, A. and Etzioni, O. (2007). "Unsupervised Resolution of Objects and Relations on the Web". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference (HLT-NAACL)*.

Yih, Wen-tau, He, Xiaodong, and Meek, Christopher (2014). "Semantic Parsing for Single-Relation Question Answering". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Yu, Mo, Gormley, Matthew, and Dredze, Mark (2014). "Factor-based Compositional Embedding Models". In: *The NIPS workshop on Learning Semantics*.

Yu, Mo, Gormley, Matthew, and Dredze, Mark (2015). "Combining Word Embeddings and Feature Embeddings for Fine-grained Relation Extraction". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference (HLT-NAACL)*.

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). "Recurrent Neural Network Regularization". In: *arXiv preprint arXiv:1409.2329*.

Zeiler, M. D. (2012). "ADADELTA: An Adaptive Learning Rate Method". In: *arXiv preprint arXiv:1212.5701*.

Zelenko, Dmitry, Aone, Chinatsu, and Richardella, Anthony (2003). "Kernel Methods for Relation Extraction". In: *Journal of Machine Learning Research 3*.

BIBLIOGRAPHY

Zeng, Daojian, Liu, Kang, Lai, Siwei, Zhou, Guangyou, and Zhao, Jun (2014). "Relation Classification via Convolutional Deep Neural Network". In: *Proceedings of the International Conference on Computational Linguistics (COLING)*.

Zeng, Daojian, Liu, Kang, Chen, Yubo, and Zhao, Jun (2015). "Distant Supervision for Relation Extraction via Piecewise Convolutional Neural Networks". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Zhang, Min, Zhang, Jie, Su, Jian, and Zhou, Guodong (2006). "A Composite Kernel to Extract Relations between Entities with both Flat and Structured Features". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Zheng, Zhicheng, Li, Fangtao, Huang, Minlie, and Zhu, Xiaoyan (2010). "Learning to Link Entities with Knowledge Base". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference (HLT-NAACL)*.

Zhou, GuoDong, Su, Jian, Zhang, Jie, and Zhang, Min (2005). "Exploring Various Knowledge in Relation Extraction". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Zhou, Jie and Xu, Wei (2015). "End-to-end Learning of Semantic Role Labeling using Recurrent Neural Networks". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Zitouni, Imed and Florian, Radu (2008). "Mention Detection Crossing the Language Barrier". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.