Neural Methods for Event Extraction

Emanuela Boroş

université PARIS-SACLAY

NNT : 2018SACLS302

Thèse de doctorat

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir

Neural Methods for Event Extraction

Thèse de doctorat de l'Université Paris-Saclay
préparée à l'Université Paris-Sud

Ecole doctorale n°580 Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Orsay, le 27/09/2018, par

EMANUELA BOROŞ

Composition du Jury :

Nicolas Sabouret Professeur, Université Paris-Saclay, LIMSI	Président
Patrice Bellot Professeur, Aix-Marseille Université, LIS	Rapporteur
Philippe Muller Maître de conférences, HDR, Université Paul Sabatier, IRIT	Rapporteur
Nathalie Camelin Maître de conférences, Le Mans Université, LIUM	Examinateur
Brigitte Grau Professeur, ENSIIE, LIMSI	Directrice de thèse
Romaric Besançon Ingénieur chercheur, CEA, LIST	Co-encadrant
Olivier Ferret Ingénieur chercheur, CEA, LIST	Co-encadrant, Invité

Résumé

Du point de vue du traitement automatique des langues (TAL), l'extraction des événements dans les textes est la forme la plus complexe des processus d'extraction d'information, qui recouvrent de façon plus générale l'extraction des entités nommées et des relations qui les lient dans les textes. Le cas des événements est particulièrement ardu car un événement peut être assimilé à une relation n-aire ou à une configuration de relations. Par rapport aux relations ne faisant intervenir que deux entités, s'ajoute donc une dimension nouvelle obligeant à sortir bien souvent du cadre de la phrase, ce qui constitue une difficulté supplémentaire. En pratique, un événement est décrit par un déclencheur (le mot ou l'expression qui évoque l'événement) et un ensemble de participants à cet événement (c'est-à-dire des arguments ou des rôles) dont les valeurs sont des extraits de texte.

Alors que la recherche en extraction d'information a largement bénéficié des jeux de données étiquetés manuellement pour apprendre des modèles permettant l'analyse des textes, la disponibilité de ces ressources reste un problème important. En outre, de nombreuses approches en extraction d'information fondées sur l'apprentissage automatique reposent sur la possibilité d'extraire à partir des textes de larges ensembles de traits définis manuellement grâce à des outils de TAL élaborés. De ce fait, l'adaptation à un nouveau domaine constitue un défi supplémentaire.

Cette thèse présente plusieurs stratégies pour améliorer la performance d'un système d'extraction d'événements en utilisant des approches fondées sur les réseaux de neurones et en exploitant les propriétés morphologiques, syntaxiques et sémantiques des plongements de mots. Ceux-ci ont en effet l'avantage de ne pas nécessiter une modélisation a priori des connaissances du domaine et de générer automatiquement un ensemble de traits beaucoup plus vaste pour apprendre un modèle.

Nous avons proposé plus spécifiquement différents modèles d'apprentissage profond pour les deux sous-tâches liées à l'extraction d'événements : la détection d'événements et la détection d'arguments. La détection d'événements est considérée comme une sous-tâche importante de l'extraction d'événements dans la mesure où la détection d'arguments est très directement dépendante de son résultat. La détection

d'événements consiste plus précisément à identifier des instances d'événements dans les textes et à les classer en types d'événements précis. Classiquement, un même événement peut apparaître sous la forme de différentes expressions et ces expressions peuvent elles-mêmes représenter des événements différents dans des contextes différents, d'où la difficulté de la tâche. La détection des arguments s'appuie sur la détection de l'expression considérée comme déclencheur de l'événement et assure la reconnaissance des participants de l'événement. Parmi les difficultés à prendre en compte, il faut noter qu'un argument peut être commun à plusieurs événements et qu'il ne s'identifie pas nécessairement à une entité nommée facilement reconnaissable.

En préalable à l'introduction de nos nouveaux modèles, nous commençons par présenter en détail le modèle de l'état de l'art qui en constitue la base. Des expériences approfondies sont menées sur l'utilisation de différents types de plongements de mots et sur l'influence des différents hyperparamètres du modèle en nous appuyant sur le cadre d'évaluation ACE 2005, standard d'évaluation pour cette tâche.

Nous proposons ensuite deux nouveaux modèles permettant d'améliorer un système de détection d'événements. L'un permet d'augmenter le contexte pris en compte lors de la prédiction d'une instance d'événement (déclencheur d'événement) en utilisant un contexte phrastique, tandis que l'autre exploite la structure interne des mots en profitant de connaissances morphologiques en apparence moins nécessaires mais dans les faits importantes. Nous proposons enfin de reconsidérer la détection des arguments comme une extraction de relation d'ordre supérieur et nous analysons la dépendance de cette détection vis-à-vis de la détection d'événements.

Acknowledgements

This thesis was an extreme roller-coaster, with many ups and downs. Firstly and most importantly, I am grateful to my three advisors, Brigitte Grau, Romaric Besançon, and Olivier Ferret, for their continuous guidance throughout these years, for their professional expertise, for their way of working together, for their sense of humor, for understanding my stubbornness that slowly, in time, I grew out of, and most of all, for accepting my return to finish the thesis and for putting their trust and patience in me, after a dark period in my life from which I did not think I would ever come back or even be alive.

I am thankful also for my doctoral committee, not only accepting to put valuable time in reading and reviewing my thesis, but also for accepting a small delay for the presentation, Patrice Bellot, Philippe Muller, Nathalie Camelin, and Nicolas Sabouret.

I would like to thank also the colleagues at LIMSI (ILES), even though I wasn't there many times. Special thanks are for Anne Vilnat for the help and for trusting my ability to finish the thesis. Also, I would like to thank the doctoral school Sciences et Technologies de l'Information et de la Communication (EDSTIC) and especially Stéphanie Druetta for helping me with the bureaucracy of the inscriptions.

I thank all my colleagues at CEA, LIST (LVIC), not only for the advices, and work-related interactions but also for the developed friendships. Special thanks are for Dorian Kodelja, with whom I shared many ideas.

When the funding from CEA, LIST finished and, after a while, I decided to return, Balázs Kégl and Akin Kazakci made it possible for me to work at Laboratoire de l'Accélérateur Linéaire (LAL). All the interesting discussions and being around Balázs gave me another view over not only the future of artificial intelligence but also over the world-related problems. Also, I thank my colleagues at LAL, Mehdi, Yetkin, and Victoria for their machine learning related views and, impressively, for their artistic views.

I am extremely grateful to Christopher Kermorvant, with whom I started working

as an entrepreneur, who not only invested in me, but he also took me under his supervision at Teklia, and now I am very proud of what we are developing.

I thank also my roommates and friends for helping me improve considerably my French and helping me better integrate in a foreign country : Odélie, Laurent, Peppino, Guillaume, and Yann.

I would like to thank my parents for worrying about my health and for finally understanding that the thesis was very important to me.

Finally, I would like to warmly thank my boyfriend Themis for his unwavering support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As the amount of unstructured text data that humanity produces overall increases, so does the need to intelligently process it and extract different types of knowledge from data. A constantly encountered problem in processing this amount of data is the fact that most of them are initially unstructured, e.g. written in a human-understandable language. Information has many forms and sources, from broadcast news, blogs to comments and conversations. Thus, it may also consist of different types of discourse, professional or noisy discussions. Content extraction from such unstructured data is the process of identifying only the key terms in a text that are relevant to a specific application. By means of Information Extraction (IE), the terms are extracted, put in relation with existing data structures and stored in a structured way, e.g. databases. Domain specialists and data analysts can spend many hours analyzing over endless streams of text documents for extracting and classifying references to the content of interest. The area of IE has the important task of finding these relevant data in sets of documents and also to store them in an appropriate form for future use. A growing attention to this field led to a deeper text analysis that builds a bridge between the cognitive models of text understanding and the algorithmic approach to Natural Language Processing (NLP). More exactly, IE is the task of automatically extracting entities, relations, and events from unstructured

texts. Therefore, the process of scanning a text for relevant information mainly implies three levels of extraction tasks: named entity recognition (NER), relation extraction (RE) and event extraction (EE). NER represents the detection of target entities in text and RE is the identification of binary relations between entities.

EE involves identifying instances of specified types of events and the corresponding arguments in text, which is an important and challenging IE task. It can be considered as the identification of n-ary relations among entities and therefore, all IE tasks are interdependent. Usually, an EE system makes benefit of the previous tasks (NER, RE), by performing them as separate tasks or in a joint approach. In this thesis, we approach the EE task without relying on NER or RE, which involves identifying event triggers (Event Detection) and the corresponding arguments (Argument Detection and Classification) associated with them and classifying them into specific event types.

IE has many practical applications since by automatically identifying relevant information, we can reduce the human labor and speed up this process. For example, due to the market sensitivity to emerging news, investors on financial markets need to continuously monitor financial events for deciding when buying and selling equities. One might benefit from a personalized news system, with events ranging from news sites to newspapers or a calendar where events are automatically detected and stored from messages or mails. Event extraction can also be useful in risk analysis applications and monitoring systems, where a daily evolution of a situation in a particular domain can be easily and efficiently supervised. For example, the acquisition and the use of information about events and relationships in an organization's external environment - environmental scanning - can permit an organization to improve its position in the future. In the medical domain, an IE system can be used to support and facilitate biological analyses for the management of biological databases. For example, after the detection of mentions of bio-entities e.g. genes, proteins or other molecules, a relation extraction system will identify and encode events, such as descriptions of protein - protein interactions and save them as records in a database.

Having these automatically created structured representations allows the inference of new associations through knowledge discovery.

1.1 Contributions of this thesis

This dissertation investigates models for the two sub-tasks of Event Extraction (EE): Event Detection (ED) and Argument Detection and Classification. ED involves identifying triggers of specified types of events in text and classifying them into event types precisely. The argument detection and classification task is considered as the next step after event detection and finds the event participants.

Classically, events are extracted at sentence level, even if some studies showed that the extraction might also benefit from additional information at a more global view: previous extracted events/relations/entities, document-level. Event detection is considered a crucial and quite challenging sub-task of event extraction, as the argument detection and classification heavily depends on it. It also can be challenging as the same event might appear in the form of different expressions, either single words or multiword units, and these expressions might represent different events in different contexts.

Event extraction often comes with tremendously unbalanced datasets where the number of the non-event examples far exceeds the others, making event and argument detection more challenging. Furthermore, most standard approaches make strong language assumptions and require human-designed features.

The neural network models studied in this thesis address these two shortcomings and provide effective and general representations for sentences without assuming any human feature engineering nor the use of advanced natural language processing tools. They take place in the perspective defined by [Collobert et al., 2011] and the more general trend to design end-to-end models. This trend has several motivations, from a better adaptation of all the components of the model to the final task to

the absence of dependence on external tools and a better ability to implement joint approaches for dealing with the interactions between the components of the model or the replacement of feature engineering by the automatic learning of representations adapted to the task. We chose this kind of approach in order to provide solutions for event extraction that do not rely on domain knowledge, which is generally costly to obtain, and are language independent as possible, as some advanced NLP tools may not exist for some language or be prone to errors. Our hypothesis is that an event extraction system can be implemented being provided an annotated corpus with existing events.

Two main contributions of our thesis are related to the main event detection issues by considering two levels of focus for detecting triggers.

Most NN event detection models are based on a model that aims at classifying a word given a window. Our first contribution seeks to enlarge the context taken into account by the base model we consider. We propose a new global representation of the sentences, computed by an NN model that we add to the base model.

In a complementary way, our second focus addresses the morphological diversity of words, which is a way to tackle the problem of data sparsity. More precisely, we propose in this second contribution both to extend our base model with a character level model and to apply specific data augmentation techniques.

One concern shared by these two contributions is that we propose new features that do not depend on NLP tools and resources (i.e., syntactic parsers, Part-of-Speech taggers, Named Entity Recognizer, etc.) in order to combat the errors propagated by these tools.

We also analyze the chosen baseline model for event detection and we conduct extensive experiments on the usage of word representations (embeddings) which are considered to provide general representations of words that can capture their hidden syntactic and semantic properties. An in-depth analysis of their impact on

the performance of event detection is performed.

Finally, for studying the impact of trigger detection and classification, we approach the argument role prediction as a relation extraction task where we pair the event mention (trigger) and one argument in a relation in order to analyze the effect of the ED on the detection of arguments.

1.2 Structure and outline of the thesis

This thesis is organized as follows: Chapter 2 summarizes the related work on event extraction and introduces the evaluation framework based on ACE corpus and ACE types of events.

In Chapter 3, we describe a baseline approach for event detection based on a convolutional neural network (CNN) applied to a window of text around potential triggers that automatically learns features from the sequence of words (word embeddings) and classifies the middle word as an event type. We evaluate the model in a general setting and perform an analysis of results and hyperparameters.

Chapter 4 proposes two new NN architectures that aim at augmenting the baseline neural-based model for event detection with additional features with no dependence on handcrafted features or language.

Firstly, we add a feature called sentence embeddings that can catch a global information about the trigger word. We create an encoder network (a bidirectional recurrent neural network) used to produce a vector representation of the sentence. We consider that representing the whole sentence will contribute at better distinguishing between event types.

Secondly, we incorporate character embeddings as an extra feature with the motivation that these features can capture morphological and shape information, whereas word embeddings are meant to capture syntactic and semantic information. Also,

we benefit from the possibility of representing misspelled or custom words and the treatment of words that appear infrequently (an advantage over the word embedding models that suffer from lack of enough training opportunity for infrequent words). We also propose a type of data augmentation that we assume helpful in capturing the morphological information of the context of extraction.

Chapter 5 presents argument role prediction as a Relation Extraction (RE) task and, at the same time, evaluates a use case for our best event detection model by analyzing the influence of our model over argument prediction.

Chapter 6 concludes the thesis as a whole, pointing out some possible directions for future work.

Chapter 2

State-of-the-art

The research in information extraction has been driven forward by a long history that started with the MUC (Message Understanding Conferences) [Grishman and Sundheim, 1996] from 1987 through 1998 under the auspices of the US government (ARPA/DARPA) and continued with the Automatic Content Extraction (ACE) program [Doddington et al., 2004], and more recently the Text Analysis Conferences (TAC) [1]. Another known competition initiated in 2004 by the Informatics for Integrating Biology and the Bedside (i2b2) was designed to encourage the development of NLP techniques for the extraction of medication-related information from narrative patient records, in order to accelerate the translation of clinical findings into novel diagnostics and prognostics.

For MUC, participating groups have been given predefined types of information from a particular domain (templates called *messages*), following that the output of each participant's system to produce text snippets that satisfied the specified types of interest. First MUCs involved sanitized forms of military messages about naval sightings and engagements. For the MUC event extraction tasks, MUC-3/4 had data on Latin American terrorist incidents (MUC 1991, MUC 1992), and MUC-6 data regarding executive successions (MUC 1995) [2].

1. https://tac.nist.gov/about/index.html
2. https://cs.nyu.edu/cs/faculty/grishman/muc6.html

The ACE dataset contains datasets in multiple languages for the 2005 Automatic Content Extraction (ACE) evaluation [3] with various types annotated for entities, relations, and events, from various information sources (e.g., broadcast conversations, broadcast news and telephone conversations). The data were created by Linguistic Data Consortium (LDC) with support from the ACE Program. The proposed tasks by ACE are more challenging than their MUC forerunners. In particular, the increased complexity resulted from the inclusion of various information sources (e.g., broadcast and telephone conversations, broadcast news) and the introduction of more fine-grained entity types (e.g., facilities, geopolitical entities, etc.). More recently, the Event track was introduced in the Text Analysis Conference (TAC) evaluation workshops and proposed a set of tasks focusing on the extraction of events from text along with their participants and relations.

MUC, ACE and TAC initiatives are of central importance to the IE field since they provide a set of corpora that are available to the research community for the evaluation and comparison of IE systems and approaches.

An IE pipeline usually involves three levels of extraction tasks: **entity extraction** (the detection of target entities in a text), **relation extraction** (the identification of binary relations between entities) and **event extraction** (the identification of n-ary relations among entities).

Generally, an IE system starts with the detection and classification of relevant entities found in the text, which is usually referred to as Named Entity Recognition (NER). Most commonly, IE systems search for person names, organizations, and geographical locations. The choice of the precise types of entities to be extracted depends greatly on the task and the notion of Named Entity (NE) is extended to include items that do not really denote a name or an entity, e.g. measurements, prices, or temporal expressions or any other kinds of entities as product names or medical entities.

3. https://catalog.ldc.upenn.edu/ldc2006t06

For example, in processing the sentence *Meanwhile **Blair** arrived in **Washington** late **Wednesday** for two days of talks with **Bush** at the **Camp David presidential retreat**.*, an NER system would detect that ***Blair*** and ***Bush*** do refer to two **Person**s, ***Washington*** and ***Camp David presidential retreat*** - two **Location**s and that ***Wednesday*** is a temporal expression.

Usually, the next step in IE is the identification of binary relations between such entities, referred as the arguments of the relations. Relation Extraction (RE) is practically the transformation of unstructured data into structured information by identifying the links between named entities and deciding which ones are relevant for the task at hand.

For example, in the sentence in the medical domain: *A culture taken from the lumbar drain showed **Staphylococcus aureus** resistant to the **Nafcillin**.*, that is an example of a hospital discharge summary [Ryan, 2011], after the entities **Staphylococcus aureus** and **Nafcillin** are identified, a relation that reveals that the treatment has not been successful can be extracted.

Following the example for NER, in the sentence *Meanwhile **Blair** arrived in **Washington** late **Wednesday** for two days of talks with **Bush** at the **Camp David presidential retreat**.*, three binary relations can be extracted between the entities found by NER: (***Blair***, ***Washington***), (***Blair, Camp David presidential retreat***) and (***Bush, Camp David presidential retreat***), all three being relations of physical change of location.

The next step is the detection of n-ary relations between entities, a step that can be named *template filling* or Event Extraction (EE). This task has the purpose of identifying events and their descriptive entities in a large number of documents, which is an important but challenging Information Extraction (IE) task because it usually implies the usage of the previous IE tasks (NER, RE). Associated with each event mention is a phrase, the event trigger (most often a single verb but also phrasal verbs, nouns, phrasal nouns, pronouns and adverbs), which evokes best that event.

More precisely, the task involves identifying event triggers (subtask called Event Detection, ED) associated with corresponding arguments (subtask usually known as Argument Detection and Classification) and classifying them into specific event types or in a *template* with an event trigger and a number of associated arguments.

Following the example for NER and RE, in the sentence *Meanwhile* ***Blair*** *arrived in* ***Washington*** *late* ***Wednesday*** *for two days of* ***talks*** *with* ***Bush*** *at the* ***Camp David presidential retreat.***, ***talks*** is an event trigger denoting an event of type **Meet** between two **Person**s: ***Blair*** and ***Bush***, at a **Location**: ***Washington*** and at a **Time**: ***Wednesday***. This event can be considered as an n-ary relation between the entities (***Blair, Washington, Bush, Camp David presidential retreat***), and thus the EE can be considered as an n-ary or multi-way RE.

Despite the usefulness and wide prospective applicability of IE, several issues and challenges have to be overcome until an IE system is widely adopted as an effective tool in practice.

— **Annotation cost**: practical issues related to the high cost of manual annotation of texts (e.g. human resources) have to be faced. We have to minimize the human effort while keeping the quality of an IE system. Data annotation needs human expertise and this causes labor-intensive work for data interpretation at two levels. Firstly, an IE system may use NLP resources and tools, created using lots of annotated documents and secondly, an IE system needs a higher-level of annotation of relations or events.

— **Feature engineering**: the complex choice of features from NLP tools and resources (i.e., parsers, part of speech taggers etc.) and the difficult decision making in combining them represent an important issue too since the errors from these sources propagate to the downstream tasks. For example, an NER system may mistakenly detect the wrong entity needed by an RE system, which downgrades the accuracy of the RE.

— **Context of extraction**: the extraction of the needed information can be

approached at a local level, as in the case of the detection and extraction of entities, relations or events that are fully expressed within a single sentence. However, in a significant number of cases, almost 40% in the MUC evaluations [Stevenson, 2006], several sentences or paragraphs of a text have to be examined to identify a relation or an event. Thus, IE includes complex tasks requiring to aggregate information that may be spread across a document. In EE, for example, an entire document may have to be examined to identify an event, where, for attaching all the entities to this event, coreference resolution may be a task at hand for finding all expressions that refer to the same entity in a set of phrases.

— **Type of approach**: for characterizing the different types of approaches, we can differentiate two dimensions. The first one refers to their degree of supervision, i.e. the degree of specification of the information need. For instance, extracting all earthquake events compared to extracting all events in general. In practice, this first dimension makes a clear distinction between supervised, unsupervised and semi-supervised learning approaches. In supervised and semi-supervised, the information to extract is fully specified and there is a need for expert data annotation since usually, there is a small amount of annotated data and a large amount of unlabeled data available. Supervised methods can perform quite well with enough training data, but annotating sufficient data is time and resource-consuming [Mooney, 1999]. Semi-supervised methods aim to reduce the annotated data required, ideally to a small set of labeled data [Patwardhan and Riloff, 2007, Yangarber et al., 2000, Lin et al., 2003, Surdeanu et al., 2006]. The unsupervised methods make use of clustering or topic detection to extract similar relations or events. For example, the OpenIE (Open Information Extraction) paradigm was created to operate in a totally domain-independent manner and at Web scale by making a single pass over a corpus to get the extraction templates of interest and extracting a diverse set of relational tuples without requiring any

relation-specific human input [Banko et al., 2009, Banko et al., 2008, Etzioni et al., 2005, Talukdar et al., 2008, Moschitti et al., 2003]. These extractors may produce extractions that are unspecific or ambiguous when taken out of context and also, with potentially thousands of templates of interest, these systems could not provide a convincing solution to the knowledge acquisition bottleneck. The second dimension we distinguish for characterizing existing approaches refers to the way the extraction model is defined: either automatically from a set of annotations, which corresponds to data-driven approaches, or by the means of a human work, which corresponds to expertise-driven approaches. Hybrid approaches combining these two types of approaches also exist. The data-driven approaches aim at converting data to features through the use of statistics, data mining, and machine learning and thus, they require a lot of data for getting statistically significant features. The expertise-driven methods exploit existing expert knowledge about how to extract information from texts, usually through pattern-based approaches. These systems are extremely dependent on the coverage of this knowledge and thus, they may alleviate the work to do or extend its coverage by using techniques such as bootstrapping or optimizing their knowledge-based algorithms by means of machine learning, or vice-versa [Huang and Riloff, 2012a, Yangarber et al., 2000]. An example of a hybrid system that tries to combine both data- and expertise-driven techniques, is the *Rapier* system [Mooney, 1999] that learns pattern-matching rules to extract fillers for the slots in an extraction template. The human interaction of the system consisted of providing lots of documents with filled-in templates.

— **Architecture**: as previously mentioned, all IE tasks are interdependent. For instance, entities detected by an NER can further help Relation Extraction (RE), taking these as input. A pipeline approach is more modular based on generic components simpler to implement, faster to run and debug, but errors in upstream components are often compounded and propagated to

the downstream stages. A joint approach [Li et al., 2013b, Nguyen et al., 2016a, Feng et al., 2017] is less prone to error propagation but more complex in implementation.

These aspects could eventually impair the performance of IE systems. In light of the high annotation cost and domain specialists required, we are following the current state-of-the-art systems for EE that involve neural network models to improve the event extraction systems and tend to some extent to overcome these fundamental limitations mentioned above.

Consequently, our research goal in this thesis is to develop neural-based learning models that can automatically induce representations of human language, in particular its structure and meaning, in order to solve the event extraction task by overcoming its main challenges.

2.1 Event Extraction

The Event Extraction (EE) constitutes a challenging task with the purpose of quickly identifying events and their entities in a large number of documents. An event is described by a set of participants (*i.e.* attributes or roles) whose values are text excerpts.

The EE implies identifying instances of specified types of events in text and the arguments associated to them. Each event is represented by a phrase, a sentence or a span of text, the *event trigger* (most often single verbs or phrasal verbs, but also nouns, phrasal nouns, pronouns and adverbs), which evokes that event. After the detection and classification of the triggers, the arguments of the event must be found. Event arguments are entity mentions or temporal expressions that are involved in an event (as participants).

These two main sub-tasks, trigger and argument role prediction are highly interde-

pendent: trigger identification is usually treated as an independent task and argument finding and attribute assignment are each dependent on the results of trigger identification. Finally, the event extraction task can be treated as an n-ary relation extraction task, where the components of a relation can be the triggers that represent the central component and the arguments that are related to the trigger.

Let's take, for instance, this sentence: *There was the free press in Qatar, Al Jazeera, but its'* ***offices in Kabul and Baghdad*** *were* ***bombed*** *by* ***Americans****.*

Typically, an event in a text is expressed by the following components:

— **Event mention**: an occurrence of an event with a particular type. These are usually sentences or phrases that describe an event. The sentence above is an **Attack** event mention.

— **Event trigger**: the word that most clearly expresses the event mention. The **Attack** from the sentence is revealed by the event trigger word **bombed**.

— **Event argument**: an entity mention, temporal expression or value (e.g. *Sentence*, *Crime*, *Job-Title*) that serves as a participant or attribute with a specific role in an event mention.

— **Argument role**: the relationship between an argument and the event in which it participates. The argument roles that should be extracted in this case are: *Americans* that has the role of an **Attacker**, the **Place**s where the event produced are *Kabul* and *Baghdad* and the **Target** of the bombing is comprised by the *offices in Kabul and Baghdad*. **Attacker** and **Target** are roles of arguments that are specific for **Conflict.Attack** event type.

For example, Table 2.1 gives some examples of ACE events. Each event takes one or more arguments. Most arguments are entities, which are references in the text to people, organizations, locations, facilities, weapons, or vehicles.

At sentence-level extraction, one may consider that EE has some links with Semantic Role Labeling (SRL). Indeed, in the case of semantic text representations, *events*

Table 2.1 – ACE *Life* event subtypes and entities

***Life* event subtypes**	**Arguments**
Be Born	Person, Time, Place
Marry	Person, Time, Place
Injury	Agent, Victim, Instrument, Time, Place
Divorce	Person, Time, Place
Die	Agent, Victim, Instrument, Time, Place

(called *frames*) include a predicate, which is the main determinant of what type of event it represents, and arguments. Some resources containing more or less specific *frames* have been developed manually, e.g. FrameNet [Baker et al., 1998, Ruppenhofer et al., 2016], PropBank [Kingsbury and Palmer, 2002], VerbNet [Schuler, 2005] and NomBank [Meyers et al., 2004] [4]. The *events*, in this case, are considered as *frames* and "should not be confused with events as defined in IE, which correspond more closely to the everyday notion of an event, such as a political, financial" or biomedical event [Abend and Rappoport, 2017].

One difference between *frames* and *events* in IE is not only that an SRL system assumes the existence of semantic roles in every sentence, but also, the predicate is not always representative of an event (each predicate does not necessarily refer to an event and conversely, not all event triggers are predicates). For example, let's take this paragraph: *Barghouti was one of the major leaders of the First Intifada in **1987**, leading Palestinians in a mass **uprising** against Israeli occupation of **the West Bank**. During the **uprising**, he was arrested by Israel and deported to Jordan, where he stayed for seven years until he was permitted to return under the terms of the Oslo Accords in 1994.* Accordingly to the annotators of the ACE 2005 Dataset [5], only one event is relevant for a human evaluator, a **Conflict.Demonstrate** event that is represented in the first sentence by the trigger word **uprising**. Its annotated arguments are *the West Bank*: **Place** and *1987*: temporal expression (**Time-**

4. These resources are available mainly for English but some of them have been developed for other languages, such as the French FrameNet [Djemaa et al., 2016] or Verb∋Net [Pradet et al., 2014], the adaptation of VerbNet for French.

5. https://catalog.ldc.upenn.edu/ldc2006t06

Within). The same word also appears in the second sentence but is not considered as an event trigger in that case since the sentence contains no argument for the corresponding event. However, **uprising** is a predicate in both cases. Conversely, an event trigger does not always correspond to a predicate. For instance, **fatal** in the phrase *The fatal accident ...* can be a trigger for a **Die** event. The abundance of roles detected by an SRL system could also represent another challenge to the task of EE, the selection of the relevant entities from the detected arguments with semantic roles. Another inconvenience is the reliance of an SRL to a prior complete morphological and syntactical analysis that can make an IE system prone to error propagation. Regardless of these issues, SRL has been successfully used in NER as features [Wattarujeekrit et al., 2005], or to generate predicate-argument structures from the output of parsers to improve EE (more exactly, event detection) [Surdeanu et al., 2003, Grishman et al., 2005, Meyers et al., 2001]. Our conclusion is that even if EE and SRL are different, we can imagine that SRL can help EE. But in that situation, we have to face the *pipeline effect*: errors in SRL results, which are still at a high level in current tools, are likely to impair EE, which certainly explains why SRL tools are not used widely for EE.

The main approaches developed for the task of EE are presented in the next section.

2.1.1 Event Extraction approaches

In order to better generalize the systems developed for the Event Extraction task, one can divide the prior work in: pattern-based systems [Krupka et al., 1991, Hobbs et al., 1992, Riloff, 1996a, Riloff, 1996b, Yangarber et al., 2000], machine learning systems based on engineered features (i.e. feature-based) [Freitag, 1998, Chieu et al., 2003, Surdeanu et al., 2006, Ji et al., 2008, Patwardhan and Riloff, 2009, Liao and Grishman, 2010, Huang and Riloff, 2011, Hong et al., 2011, Li et al., 2013b, Bronstein et al., 2015] and neural-based approaches [Chen et al., 2015b, Nguyen and Grishman, 2015a, Nguyen et al., 2016a, Feng et al., 2016].

2.1.2 Pattern-based approaches

In light of the high annotation cost of expert manual annotations, several pattern-based (rule-based) systems have been proposed, to speed up the annotation process. The pattern-based approaches first acquire a set of patterns, where the patterns consist of a predicate, an event trigger, and constraints on its local syntactic context. They also include a rich set of ad-hoc lexical features (e.g. compound words, lemma, synonyms, Part-of-Speech (POS) tags), syntactic features (e.g. grammar-level features, dependency paths) and semantic features (e.g., features from a multitude of sources, WordNet [6], gazetteers) to identify role fillers. Earlier pattern-based extraction systems were developed for the MUC conferences [Krupka et al., 1991, Hobbs et al., 1992, Riloff, 1996a, Yangarber et al., 2000].

For instance, the *AutoSlog* system [Riloff, 1996b] automatically created extraction patterns that could be used to construct dictionaries of important elements for a particular domain and a text where the elements of interest were manually tagged only for the training stage. Later, [Riloff, 1996a] makes the observation that patterns occurring with substantially higher frequency in relevant documents than in irrelevant documents are likely to be good extraction patterns. They propose the separation between relevant and irrelevant syntactic patterns and a re-ranking of the patterns. The system named *AutoSlog-TS* attempted to overcome the necessity of having a hand-labeled input requiring only pre-classified texts and a set of generic syntactic patterns. The main drawback of this system is the requirement of manual inspection of the patterns, which can be costly.

Many proposed approaches targeted the minimization of human supervision with a bootstrapping technique for event extraction. [Huang and Riloff, 2012a] proposed a bootstrapping method to extract event arguments using only a small amount of annotated data. After the manual inspection of the patterns, another effort was made for performing manual filtering of resulting irrelevant patterns.

6. https://wordnet.princeton.edu/

[Yangarber et al., 2000] developed another bootstrapping approach, starting with some seed patterns, using these patterns to identify some relevant documents, using these documents to identify additional patterns, etc. The authors in [Sudo et al., 2003] also proposed to sort relevant from irrelevant documents using a topic description and information retrieval engine. This approach was further refined in [Surdeanu et al., 2006], which explored alternative pattern ranking strategies. [Stevenson and Greenwood, 2005] used a lexical database for the English language *WordNet*-based similarity to expand an initial set of event patterns. The systems in [Patwardhan and Riloff, 2007, Patwardhan, 2010] are built upon a sentence classifier that distinguishes between relevant and irrelevant regions and learns domain-relevant extraction patterns using a semantic affinity measure. Later, [Bronstein et al., 2015] takes the example trigger terms mentioned in the guidelines as seeds, and then applies an event-independent similarity-based classifier for trigger labeling. Thus, a great amount of effort has been put in to overcome the manual annotation of data.

2.1.3 Feature-based approaches

Most recent event extraction frameworks are feature-based approaches applied at the sentence-level or to a larger context (document-level). These approaches usually require effort to develop rich sets of features.

The feature-based approaches rely mainly on designing large effective feature sets for statistical models, ranging from *local features* [Grishman et al., 2005, Ahn, 2006, Li et al., 2013a], to the higher level structures such as cross-document, cross-sentence and cross-event information e.g. *global features* [Gupta and Sarawagi, 2009, Hong et al., 2011, Ji et al., 2008, Li et al., 2015, Liao and Grishman, 2010, Patwardhan and Riloff, 2009]. The discrete local features include: lexical features (e.g. unigrams/bigrams of text context, lemma, synonyms, Part-of-Speech (POS) tags, Brown clusters [Brown et al., 1992]), syntactic features (e.g. dependency paths)

and semantic features (e.g., features from a multitude of sources, WordNet [Miller et al., 1990], gazetteers). Using Natural Language Processing (NLP) toolkits for extracting this type of features may lead to severe error propagation, has a cost in terms of computational efficiency and second, limits the application of the models to languages for which such NLP tools are available. The cross-* features are usually inferred from known instances to predict the attributes of unknown instances. As an example, given an **Attack** event, the cross-event inference can predict its type by using the related events (**Die**) co-occurring with it within the same document or same sentence. An **Attack** event often co-occurs with a **Die** event in the same document or sentence, but rarely co-occurs with a **Marry** event.

Local features were considered to be insufficient to reliably and accurately perform event extraction in complex domains [Grishman et al., 2005, Ahn, 2006]. Because of this, [Finkel et al., 2005, Ji et al., 2008, Patwardhan and Riloff, 2007, Hong et al., 2011] tried to use a larger context besides sentences as in discourse, document, cross-document or cross-entity information to improve the traditional sentence-level event extraction systems.

For instance, [Ji et al., 2008] used global information from related documents (cross-sentence), and [Gupta and Sarawagi, 2009] extracted implicit time information. [Patwardhan and Riloff, 2009] used a sentential context by doing an a priori classification of sentences to decide if they contain or not an event mention. In the same manner, [Liao and Grishman, 2010] leveraged document-level cross-event information and topic-based features. More exactly, the authors use information about other types of events to make predictions or resolve ambiguities regarding a given event.

[Huang and Riloff, 2012b] sustained that there is a need for a larger view over event extraction, regarding the relevancy of an entity for an event and thus, other discourse features were developed. With a bottom-up approach to event extraction, the candidate role fillers for a specific event template slot is identified independently

and then, along with discourse properties, the textual cohesion is modeled.

[Li et al., 2013b] implements a joint model via structured prediction with cross-event features. Concretely, they use structured perceptron with beam search to jointly extract triggers and arguments that co-occur in the same sentence. One drawback of this approach is the usage of a large set of features dependable of NLP toolkits and resources.

Inspired by the work of [Li et al., 2013b], [Yang and Mitchell, 2016] uses a similar large set of features provided by NLP toolkits and resources. Their contribution is the addition of a document context for extracting events creating a unified model that jointly extracts events from sentences across the whole document.

This type of approach suffers from a great labor put in feature engineering. A large set of elaborated hand-designed features can bring a visible improvement in accuracy, but the same additions can make a system slow and intractable in large-scale applications [Ji et al., 2008, Patwardhan and Riloff, 2009, Liao and Grishman, 2010, Huang et al., 2012, Li et al., 2013b, Li et al., 2013a, Li et al., 2014].

2.1.4 Neural-based approaches

The current state-of-the-art systems for event extraction involve neural network models to improve event extraction. Neural networks have been introduced into event extraction very recently with the purpose of overcoming two fundamental limitations of the traditional feature-based approaches: complicated feature engineering for rich feature sets and error propagation from the preceding stages which generate these features. All these models use word embeddings, a general word representation that is produced by training a deep learning model on a large unlabeled dataset. Consequently, word embeddings replace the hard matches of words in the feature-based approaches with the soft matches of continuous word vectors.

Neural-based approaches for event extraction have used two types of models: Con-

volutional Neural Networks (CNNs), following work in the field of image processing, and Recurrent Neural Networks (RNNs), which are more particularly adapted to the sequential aspect of texts. These two kinds of approaches have also been associated.

[Nguyen and Grishman, 2015a] and [Chen et al., 2015b] deal with the event detection problem with a model based on CNNs. [Nguyen and Grishman, 2016] improve the previous CNN models [Nguyen and Grishman, 2015a] for event detection by taking into account the possibility to have non-consecutive n-grams as basic features instead of continuous n-grams. Both models use word embeddings for representing windows of text that are trained like the other parameters of the neural network. As another extension of the basic CNN models, [Nguyen and Grishman, 2018] proposes to apply Graph Convolution Networks for exploiting syntactic dependency relations.

Concerning RNNs, [Jagannatha and Yu, 2016] extracts event instances from health records with Bidirectional Recurrent Neural Networks (Bi-RNNs) while [Nguyen et al., 2016a] proposes a joint framework with the same type of neural networks for predicting at the same time event triggers and their arguments. This last work is benefiting from the advantages of the two models as well as addressing issues inherent in the existing approaches. The authors also systematically investigate the usage of memory vectors/matrices to store the prediction information during the course of labeling sentences features. Additionally, [Nguyen et al., 2016a] augments their system with discrete local features inherited from [Li et al., 2013b]. [Duan et al., 2017] and [Zhao et al., 2018] explore another extension of RNNs by integrating a larger context through a document representation while [Hong et al., 2018] exploits a generative adversarial network for discarding spurious detections.

Further, [Feng et al., 2016] develops a hybrid neural network (a CNN and an RNN) to capture both sequence and chunk information from sentences, and use them to train an event detector that does not depend on any handcrafted features.

We will focus on a deeper analysis of these systems in later chapters.

2.2 Event extraction evaluation

As mentioned at the beginning of this chapter, the event extraction task has been developed through several evaluations, mainly MUC, ACE, and TAC. Each evaluation defined specific metrics and rules but the work coming after ACE generally adopted those used in [Ji et al., 2008], which are globally close to those of TAC. More precisely, the evaluation in this context is based on event mentions and event arguments. An event mention is a phrase or a sentence within which an event is described, including trigger and arguments. The triggers are words that most clearly express the event and can have an arbitrary number of arguments.

The evaluation is based on the standard metrics: Precision (P), Recall (R), and F-measure (F1), defined by the following equations:

$P = \frac{TruePositives}{TruePositives+FalsePositives}$

$R = \frac{TruePositives}{TruePositives+FalseNegatives}$

$F1 = \frac{2 \cdot P \cdot R}{P+R}$

True positives are the samples classified as belonging correctly to a class. False negatives are classified as not belonging to a class, incorrectly. False positives are the samples classified as belonging to a class, incorrectly. Thus, precision is the fraction of relevant samples among the retrieved samples, while recall is the fraction of relevant samples that have been retrieved over the total amount of relevant samples. The F1 is the harmonic mean between these two. Because the data in EE tasks usually suffer from class imbalance, we compute the micro-averages of these metrics for aggregating the contributions of all classes.

Following previous work [Li et al., 2013b, Chen et al., 2015b, Huang et al., 2012, Nguyen et al., 2016a, Nguyen and Grishman, 2015a], the following criteria are used to determine the correctness of a predicted event mention:

— A **trigger** is correct if its event subtype and offsets match those of a reference trigger.

— An **argument** is correctly identified if its event type and offsets match those of any of the reference argument mentions.

2.3 In-depth analysis of the state-of-the-art approaches

So far, we discussed the previous systems developed for the Event Extraction (EE) task in relation to the main challenges, the high cost of manual annotation of data, the complex feature engineering.

In order to position our work, we continue with an in-depth analysis of the systems on the ACE 2005 Dataset in Table 2.2.

Table 2.2 – Performance of the state-of-the-art systems for event detection on the blind test data. [1] refers to approaches beyond sentence level

Number	**Approaches**	**Event Detection (P/R/F1) %**	**Argument Classification (P/R/F1) %**
1	MaxEnt with local features in [Li et al., 2013b]	74.5 / 59.1 / 65.9	65.4 / 33.1 / 43.9
2	Cross-event in [Liao and Grishman, 2010]	68.7 / 68.9 / 68.8	45.1 / 44.1 / 44.6
3	[1]Cross-entity in [Hong et al., 2011]	72.9 / 64.3 / 68.3	51.6 / 45.5 / 48.3
4	Joint at document level in [Yang and Mitchell, 2016]	75.1 / 63.3 / 68.7	70.6 / 36.9 / 48.4
5	Joint beam search with local and global features in [Li et al., 2013b]	73.7 / 62.3 / 67.5	64.7 / 44.4 / 52.7
6	Dynamic multi-pooling CNN in [Chen et al., 2015b]	75.6 / 63.6 / 69.1	62.2 / 46.9 / 53.5
7	Joint Recurrent Neural Networks in [Nguyen et al., 2016a]	66.0 / 73.0 / **69.3**	54.2 / 56.7/ **55.4**

These systems can be divided according to the type of architecture:

— pipeline, where the subtask of argument role prediction follows the event detection one: [Liao and Grishman, 2010, Hong et al., 2011] (2, 3), MaxEnt

with local features in [Li et al., 2013b] and the dynamic multi-pooling CNN in [Chen et al., 2015b] (1, 5)

— joint inference, where the prediction of triggers and arguments are performed at the same time: joint beam search with local and global features in [Li et al., 2013b] and joint Recurrent Neural Networks (RNNs) in [Nguyen et al., 2016a] (4, 6)

Also, they are differentiated by the choice of features:

— discrete local features or sentence-level features: [Liao and Grishman, 2010, Hong et al., 2011, Yang and Mitchell, 2016, Li et al., 2013b] (1, 2, 3, 4, 5)

— global features such as cross-document, cross-sentence and cross-event information: [Liao and Grishman, 2010, Hong et al., 2011, Yang and Mitchell, 2016] and the joint beam search with local and global features in [Li et al., 2013b] (1, 3, 4, 5)

— learned feature representations (feature embeddings): [Chen et al., 2015b, Nguyen et al., 2016a] (6, 7)

This use of both local and global features proved to be useful in the case of the joint architecture in [Li et al., 2013b] (5). One global feature of this system is the trigger global feature that captures the dependencies between two triggers within the same sentence. For example, an **Attack** event often co-occurs with a **Die** event in the same sentence, but rarely co-occur with a **Marry** event. Via these global features, the system benefits from the inter-dependencies among event triggers or arguments. These dependencies are also taken into consideration by [Yang and Mitchell, 2016], which seems to improve the performance for triggers classification, but not for argument classification.

Despite this advantage, this system devoted a great effort in engineering large feature sets with local features (lemma and synonyms, Brown clusters, unigrams/bigrams of Part-of-Speech (POS) tags, unigrams/bigrams of the current and context words, dependency types associated the current token etc.) extracted by existing Natural Language Processing (NLP) tools and resources (i.e., parsers, part of speech tag-

gers etc.), which require themselves a significant work on manual annotation and a considerable amount of expert domain knowledge and expertise. Using these sets of features in systems as in [Liao and Grishman, 2010, Hong et al., 2011] (2, 3) may lead to severe error propagation and can make these methods difficult to be applied across different domains or other languages.

Also, [Li et al., 2013b]'s system (5) suffers from the inability to extract meaningful structure for the event extraction task that happens mainly due to the hand-crafted local feature sets. The pipelined system in [Chen et al., 2015b] (6) overcomes these problems by having no reliance on discrete local features or sentence-level features. Instead of lexical features, they introduce the use of word embeddings and instead of syntactic features (dependency paths), they propose a dynamic multi-pooling layer for CNN, which returns the maximum value in each part of the sentence according to event triggers and arguments. Word embeddings have been shown to be able to capture the meaningful semantic and syntactic regularities of words [Bengio et al., 2003, Mikolov et al., 2013a].

The state-of-the-art results are obtained by [Nguyen et al., 2016a] (7), a joint approach via recurrent neural networks (RNNs) with memory vectors/matrices to store the prediction information during the course of labeling the sentences in order to capture the inter-dependencies between triggers and arguments. This model also benefits from the usage of word representations (embeddings) which can capture hidden syntactic and semantic properties revealing the underlying feature representations from data.

Thus, the joint prediction and the usage of memory vectors/matrices that work as global features can improve the gracefulness of error recovery. Moreover, the system also includes discrete local features inherited from [Li et al., 2013b]. The binary vector whose dimensions correspond to the possible relations between words in the dependency tree is added to the model. This approach basically inherits all the benefits from both feature-based and neural-based systems as well as overcoming

their inherent issues.

2.4 Conclusions

From the work presented in this chapter, we can observe that neural-based methods [Chen et al., 2015b, Nguyen et al., 2016a] (6, 7) proved that state-of-the-art results can be obtained by renouncing complicated feature engineering of rich feature sets, with the exception of [Nguyen et al., 2016a] that relies on the discrete features proposed by [Li et al., 2013b]. One can notice the fact that the initial trend was to avoid engineered features but such features were in fact quickly reintroduced (generally under the form of feature embeddings that can take into account the similarity between features).

As it is generally done in the field of IE, we approach the EE as two subtasks: Event Detection (ED) and event argument role prediction, starting from a baseline model proposed by [Nguyen and Grishman, 2015a] based on a CNN and a minimum required feature set: feature representations (embeddings) of words and trigger and arguments positions. We aim at overcoming the complicated feature engineering that depends on NLP tools and sources and thus, avoiding the error propagation from these tools. As triggers are often ambiguous words that can refer to different events, we will study how to improve the modeling of the context at the sentence level for better representing the event itself. Another point concerns the improvement of the word embeddings themselves. Neural network models require a lot of training data, and current datasets for event extraction do not contain all the variants for trigger terms, either semantic variants with synonyms or morphological variants. Embeddings are expected to represent these variations. Thus, we will make an in-depth study of different word embedding models given as input of the neural-based model, and we will propose a new architecture in order to account for morphological variants, that are under-represented in our dataset.

We chose to analyze the dependence of the argument detection and classification subtask on the event detection one by developing a pipeline approach. Our motivation is that it is empirically common to build independent predictors in a setting where joint prediction can apply because it is simpler to implement and faster to run and debug.

The next chapter introduces [Nguyen and Grishman, 2015a]'s model for Event Detection (ED) as a chosen baseline and an in-depth analysis of the usage of word embeddings and other hyperparameters.

Chapter 3

Event Detection

This chapter focuses on the problem of Event Detection (ED) or trigger prediction, first subtask of Event Extraction (EE). This step implies the identification of instances of specified types of events in text. Each event mention is characterized by a sentence that contains an event trigger (the word or the phrase that evokes that event).

The main challenge relies on the choice of features. In the past, proposed approaches have coped with the event detection task by using a large set of hand-designed features and using the existing supervised Natural Language Processing (NLP) toolkits and resources (i.e. name tagger, parsers, gazetteers etc.) to extract these features to be fed into statistical classifiers [Ji et al., 2008, Patwardhan and Riloff, 2009, Liao and Grishman, 2010, Huang et al., 2012, Li et al., 2013b, Li et al., 2013a, Li et al., 2014]. It is often challenging to adapt these prior methods to multiple languages since they require extensive linguistic knowledge for feature engineering. Using Natural Language Processing (NLP) toolkits for extracting the features may also lead to severe error propagation and can make these methods difficult to be applied across different domains or other languagess.

By contrast, current approaches [Nguyen et al., 2016b, Nguyen et al., 2016c, Nguyen and Grishman, 2016, Nguyen and Grishman, 2015a, Chen et al., 2015b]) indicate

that deep learning is a compelling solution to avoid the aforementioned problems by automatically extracting meaningful features from raw text without relying entirely on existing NLP toolkits, thus minimizing the dependence on these toolkits and resources for features and alleviating the error propagation.

This chapter is dedicated to an in-depth study of a state-of-the-art model we chose as a baseline with a specific focus of its input word embeddings and a first attempt to improve its performance through the use of Retrofitting [Faruqui et al., 2015]. Firstly, we remind the main theoretical key points that are needed for understanding the model. We then present in more details the task of Event Detection (ED) and the challenges that it implies. We describe the chosen baseline system of [Nguyen and Grishman, 2015a] based on a Convolutional Neural Network (CNN) and evaluate the presented CNN over the ACE 2005 corpus. We present an analysis of its results in relation to its hyperparameters, with a specific emphasis on the impact of different types of word embeddings. Finally, we examine to which extent a method such as Retrofitting can be used for biasing a priori these embeddings for the ED task. The next section presents the theoretical basis of NN.

3.1 Background theory

3.1.1 Multi-Layer Perceptrons (MLPs)

Multi-Layer Perceptrons (MLPs) constitute the basis of neural networks. An MLP contains neurons organized in layers. Several neurons are connected to the same inputs $x_1, \ldots, x_n$, with a different set of weights W. The outputs of all these neurons are inputs for a new layer of neurons. An MLP with a single hidden layer can be represented graphically as in Figure 3.1.

A single-layered MLP is a function $f : R^D \rightarrow R^L$, where D is the size of input vector x and L is the size of the output vector $f(x)$, such that: $f(x) = \psi(b^{(2)} +$

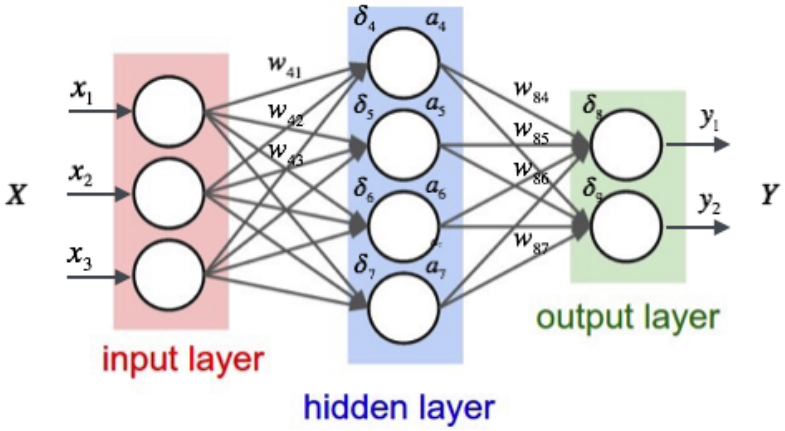


Figure 3.1 – An MLP with a single hidden layer. Source: https://medium.com/@curiousily/tensorflow-for-hackers-part-iv-neural-network-from-scratch-1a4f504dfa8

$W^{(2)}(\sigma(b^{(1)} + W^{(1)}x)))$, with bias vectors $b^{(1)}, b^{(2)}$, weight matrices $W^{(1)}, W^{(2)}$ and the activation functions ψ and σ.

The vector $h(x) = \Phi(x) = s(b^{(1)} + W^{(1)}x)$ constitutes the hidden layer. $W^{(1)} \in R^{D \times D_h}$ is the weight matrix connecting the input vector to the hidden layer. Each column $W^{(1)}_{\cdot i}$ represents the weights from the input units to the i-th hidden unit. Typical choices for the activation functions σ include the hyperbolic tangent function $tanh$, $tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$, the logistic sigmoid function, $sigmoid(a) = \frac{1}{1+e^{-a}}$, or rectified linear function $ReLu$, $ReLu(a) = \max(0, a)$.

The output vector is then obtained as: $o(x) = \psi(b^{(2)} + W^{(2)}h(x))$. In the case of multi-class classification, the probabilities can be obtained by choosing the *softmax* function as $\psi\ y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$.

3.1.2 Training neural networks

Training a neural network consists in adjusting its parameters, the connection weights, so that the model is able to perform the task at hand. The goal is to optimize a criterion that reflects the quality of the network. Given such a criterion, one may apply numerical optimization methods such as gradient descent. The backpropagation algorithm [Rumelhart et al., 1988] takes advantage of the structure of the networks to

apply these methods. Algorithms were also designed to handle the temporal aspect of recurrent neural networks, such as the Backpropagation Through Time (BPTT) algorithm [Williams and Zipser, 1995].

3.1.3 Optimization problem

A neural network performs a series of simple computations on the input to return an output, corresponding to a classification. To assess the quality of the neural network, one can compare the output $y(x)$ to the desired one z, and calculate a measure of error. For example, it may be the squared error $(y(x) - z)^2$ for binary classification. For multi-class classification, the output is a vector $y(x)$. The desired output may be encoded as a vector z, with all components set to 0 except for the one corresponding to the true class, and the squared error is: $E_{MSE}(x, z) = \|y(x) - z\|^2$. The components of y are posterior probabilities. With *softmax*, $y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$, the outputs are positive and sum up to one, so one can use the Cross-Entropy (CE) criterion: $E_{CE}(x, z) = -\sum_{i=1}^{C} z_i \log y_i(x)$. Given a dataset $S = (x^{(i)}, z^{(i)})$, where $i = 1, 2 \ldots, N$ of examples labeled with the expected outputs, one can compute a global loss function: $E(S) = \frac{1}{N} \sum_{i=1}^{N} E(x^{(i)}, z^{(i)})$.

The optimal parameters $\theta = \theta_1, \ldots, \theta_N$ are obtained by minimizing the error function. Numerical optimization methods, such as gradient descent, are applied to reach this minimum.

The gradient descent proceeds as follows:

— for each training example $(x^{(i)}, z^{(i)})$, compute $y(x^{(i)})$ and $E(x^{(i)}, z^{(i)})$

— compute the error $E(S)$, and its derivative with respect to the parameters $\frac{\partial E}{\partial \theta_i}$

— update the parameters in the direction of the gradient $\theta_i \leftarrow \theta_i \eta \frac{\partial E}{\partial \theta_i}$, where η is the *learning rate*

In order to compute the error, and therefore take one step in adjusting the param-

eters, one has to go through the whole dataset, which can lead to a slow convergence. In Stochastic Gradient Descent (SGD), the parameters are updated after every training example, or every few ones, and allows to better explore and exploit the parameter space. The momentum technique [Rumelhart et al., 1985] consists in adding a fraction of the previous update in the current one $\Delta\theta(t) = \eta\frac{\partial E}{\partial \theta_i} + \rho\Delta\theta(t-1)$.

The choice of *learning rate* may be crucial. Instead of keeping it fixed, one may change it during training. One can pre-define a schedule for learning rates, or decrease it when the objective function does not improve, or with rules such as in [Bottou, 2010] where its decrease is inversely proportional to the number of epochs, or the *AdaGrad* technique [Duchi et al., 2011], *Adadelta* [Zeiler, 2012], *RMSProp* or *Adam* [Kingma and Ba, 2014], where the *learning rate* is adaptive.

Finally, while the error on the training set is minimized, the system has to perform well on unseen examples and avoid *overfitting* the training data. If the number of parameters in the model is large enough, the network can memorize the training examples, and be excellent at predicting the correct targets for examples of the training set but perform poorly on unseen example. This problem is called *overfitting*. *Overfitting* is prevented by regularization, explained in the next section.

3.1.4 Regularization

The criterion optimized in the training of neural networks is a measure of error on the training set. The *early stopping* method consists in stopping the training procedure when the error on these validation data increases, while the training error is decreasing. We present two regularization methods which help in reducing the *overfitting* phenomenon: **weight decay** and **dropout**.

Weight Decay

The weight decay technique consists in adding a penalty to the cost function, which depends on the weights of the network.

A common formulation is the following: $E_{reg}(S) = E(S) + \lambda \sum_{i,k,j} (w_{ij}^k)^2$ where the regularization term is the sum of squares of all the weights w_{ij} of all layers k, $E(S)$ is the original cost function and λ controls the relative importance given to the original cost compared to the regularization part.

The practical effect of weight decay is that the training procedure will promote solutions with small weights. It is generally observed that neural networks overfit less with those constraints, which might be explained by the fact that with small weights, the network is less sensitive to small changes of the input.

Dropout

The dropout technique was proposed by [Hinton et al., 2012] to reduce *overfitting*. It consists in randomly ignoring some of the units of the network during training. When dropout is applied to a hidden layer, a sample of units is dropped for each training example, with some probability. The forward pass computes the output of the network without those dropped units and corresponding connections. The backpropagation procedure is performed in this network with missing nodes.

Using dropout is equivalent to train simultaneously 2^N network architectures which share weights, where N is a large number. One architecture is randomly selected at each training step. At test time, no unit is dropped, and the whole network is used. Because a layer following dropout had fewer inputs during the training procedure, the weights are multiplied by $(1 - p)$, where p is the dropping probability, when the whole network is used. For an MLP with a single hidden layer and a *softmax* output, this is equivalent to computing the geometric mean of the outputs of all 2^N possible architectures [Hinton et al., 2012].

One of the motivations of dropout is to prevent hidden units to rely on the output of others and make them useful for classification by themselves. The underlying goal is to reduce *overfitting*, hence making it a form of regularization. This technique has been successfully applied to MLPs [Dahl et al., 2013], Convolutional Neural Networks (CNNs) [Krizhevsky et al., 2012], Recurrent Neural Networks (RNNs) [Zaremba et al., 2014].

3.1.5 Gradient-based learning

After its introduction in the 1970s, the backpropagation algorithm was not fully appreciated until [Rumelhart et al., 1988]. This work defined several neural network models that were based on this algorithm for learning and were able to solve previously insoluble problems. Today, the backpropagation algorithm is widely used. Gradient descent consists in calculating the gradient of the error with respect to the parameters of one layer at a time, starting from the output layer and going sequentially to the input layer. The algorithm can be applied when the connections between layers form a directed acyclic graph, and the error is propagated from the outputs to the inputs, hence the term backpropagation, and the parameter updates are computed on the way. The only requirement is to be able to compute the derivatives of the outputs of a layer with respect to its inputs.

Stochastic Gradient Descent (SGD) works according to the same principles as ordinary gradient descent but proceeds more quickly by estimating the gradient from just a few examples at a time instead of the entire training set. In its purest form, the gradient is estimated from a single example at a time. Minibatch SGD works identically to SGD, except that more than one training example are used to make each estimate of the gradient.

3.1.6 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are MLPs with a special structure. CNNs are a type of feed-forward neural networks whose layers are formed by a convolution operation followed by a pooling operation [LeCun et al., 1998a, Kalchbrenner et al., 2014]. This means that the first layers do not use all input features at the same time but rather features that are connected. This type of neural network is a well-known deep learning architecture inspired by the natural visual perception mechanism of the living creatures. CNNs became the mainstream method for solving various computer vision problems, such as image classification [Russakovsky et al., 2015], object detection [Russakovsky et al., 2015, Everingham et al., 2010], semantic segmentation [Dai et al., 2016], image retrieval [Tolias et al., 2015], tracking [Nam and Han, 2016], text detection [Jaderberg et al., 2014] and many others. At first, in 1990, [LeCun et al., 1990] published the seminal paper establishing the modern framework of CNN, and later improved it in [LeCun et al., 1998a]. They developed a multi-layer neural network called *LeNet-5* which could classify handwritten digits. This is a highly influential paper that promoted deep convolutional neural networks for image processing. Two factors made this possible: firstly, the availability of large enough datasets and secondly, the development of powerful enough GPUs to efficiently train large networks.

Besides two classic papers on training neural networks [LeCun et al., 1998b, Bengio, 2012], which are still highly relevant, there is very little guidance on the plethora of design choices and hyperparameter settings of CNNs with the consequence that researchers proceed by trial-and-error experimentation and architecture copying, sticking to established neural network types. With good results in *ImageNet* competition, the *AlexNet* [Krizhevsky et al., 2012], *VGGNet* [Ergun and Sert, 2016], *GoogLeNet (Inception)* [Szegedy et al., 2015] and *ResNet* [He et al., 2016] became the standard. Improvements of many components of the CNN architecture like the nonlinearity type (activation function), pooling, structure and learning have been

recently proposed.

CNNs use layers with convolution filters that are applied to local features [LeCun et al., 1998a]. They are useful for classification tasks in which we expect to find local clues regarding class membership, but these clues are spread all over the input sequence of features. Pooling layers allow the model to capture these local clues, regardless of their position.

Recently, with the emerging interest in deep learning, CNNs have been revived and effectively applied in various Natural Language Processing (NLP) tasks, including semantic parsing and question answering [Bordes et al., 2014, Yih et al., 2014, Yin et al., 2016], search query retrieval [Shen et al., 2014b], sentence modeling and classification [Kalchbrenner et al., 2014, Kim, 2014], name tagging and semantic role labeling [Collobert, 2011, Fonseca and Rosa, 2013], relation classification and extraction [Liu et al., 2013, Zeng et al., 2014, Nguyen and Grishman, 2015b, Nguyen and Grishman, 2016], short-text categorization, sentiment classification [Kim, 2014], paraphrase identification [Yin and Schütze, 2015], event detection [Nguyen and Grishman, 2015a, Nguyen et al., 2016a]. In NLP, they process a sequence of data by extracting the most informative parts for the sequence of words and only considers their resulting activations.

Convolutions

One common application of convolutional neural networks is image processing. In an image, it is assumed that pixels that are spatially close together cooperate on forming a particular feature of interest much more than ones on opposite corners of the image. Also, if there is a feature relevant to the classification of an image, it will remain important, regardless of its location in an image.

In the same manner, in NLP, we can think of a sequence x with n words, as a matrix where each entry in x is represented by a d-dimensional dense vector (word

embedding), thus the input x is represented as a feature map of dimensionality $n \times d$. This can be interpreted as a 1D *image*.

In Computer Vision, filters slide over local patches of an image, but in NLP, these filters slide over rows of the matrix (a list of words), with the intuition that there is a compositional aspect, as in how an adjective is modifying a noun. The convolution layer is used for representation learning from sliding w-grams. For an input sequence with n entries: $\{x_0, \ldots, x_n\}$ with vector $c_i \in R^{wd}$ be the concatenated embeddings of w entries $\{x_{i-w+1}, \ldots, x_i\}$ where w is the filter width and $0 < i < w$. Embeddings for x_i, with $i < 1$ or $i > n$ are padded with a special token (typically a zero-vector). The representation $p_i \in R^d$ for the w-gram $\{x_{i-w+1}, \ldots, x_i\}$ is generated using the convolution weights $W \in R^{d \times wd}$, and then activation function ψ is applied, $p_i = \psi(W \times c_i + b)$, where bias $b \in R_d$ and ψ is typically one of the $tanh$, $sigmoid$ or $ReLu$ activation functions.

A big argument for CNNs is that they are fast and they are also efficient in terms of representation. Thus, convolutional filters give the possibility of learning good representations, without needing to represent explicitly w-grams (since the sequences produced by the sliding filters are typically compared with w-grams).

Pooling layers

A key aspect of CNNs are pooling layers, typically applied after the convolutional layers. A max-pooling layer takes the maximum of features over small blocks of a previous layer. The output tells us if a feature was present in a region of the previous layer, but not the precise position. Thus, pooling layers subsample their input and keep the most salient information, keep the local information captured by the filters but lose the position of the feature and the global information about locality. All representations produced by the convolutional filters $p_i \in R^d$ are used to generate the representation of an input sequence x by *max-pooling*: $x_j = max\Big(p_{1,j}, p_{2,j}, \ldots\Big)$, where $j = \{1, \ldots, d\}$ and x is a d-dimensional dense vector.

3.2 Word embeddings

Before discussing the neural-based model for the Event Detection (ED) proposed by [Nguyen and Grishman, 2015a] in more depth, it is important to pay attention to how features are represented. The concept of feature embeddings (vector representations of each core feature) is the main attraction in Natural Language Processing (NLP) approaches because the field has seen success in switching from linear models over sparse inputs to non-linear neural-network models over dense inputs.

This representation is preferred because:

— feature vectors are now a parameter of the model and can be trained

— feature interdependence can be captured, conforming to the distributional hypothesis that states words in similar contexts have similar meanings

These embeddings can encode features such as words, Part-of-Speech (POS) tags or any other linguistic information. The distributed representations of words remain a standard practice in today's NLP systems, both in shallow and deep architectures [Goldberg, 2016].

Most word representations fall into one of two categories: discrete or continuous. Discrete representations or one-hot encoding (or 1-of-K encoding, with K the vocabulary size) consist in representing a word as a binary vector that is all 0 values except the index of the word in the vocabulary, which is marked with a 1. They can also consist of memberships in a hard clustering of words, e.g., via k-means or the [Brown et al., 1992] algorithm. This representation of a word suffers from data sparsity, namely, for words that are rare in the training data, their corresponding model parameters will be poorly estimated. Moreover, another problem is the so-called *curse of dimensionality*. Because an index vector over a large vocabulary is very sparse, models can easily overfit to the training data. These limitations of these discrete representations have prompted researchers to investigate unsupervised methods for inducing word representations over large unlabeled corpora.

Continuous representations (or distributed representations or embeddings) [Deerwester et al., 1990, Levy et al., 2015] consist of low-dimensional, real-valued vectors for each word, typically induced via neural language models [Bengio et al., 2003, Mnih and Hinton, 2007]. That is, each word is embedded into a d-dimensional space, and represented as a vector in that space. This vector can capture similarities between words and make models more robust. Word embeddings can be learned in an unsupervised way to capture distributional similarities and be fine-tuned in a supervised fashion. In a neural-based model, these vectors can then be fine-tuned like the other parameters of the neural network.

By encoding word information in real-valued vectors, a diverse set of NLP tasks have been shown to benefit, as in syntactic parsing [Bansal et al., 2014, Lazaridou et al., 2013], named entity recognition [Wu et al., 2015, Ma et al., 2016, Peng and Dredze, 2016], sentiment analysis [Maas et al., 2011, Socher et al., 2013], document retrieval [Turian et al., 2010, Collobert and Weston, 2008, Turney and Pantel, 2010]. Additionally, due to the fact that they can be induced directly from unannotated corpora, they are available in different languages and domains. Intrinsic evaluations on various tasks have been performed to help refine vector learning methods to discover representations that capture syntactic and semantic information about the tasks [Turney and Pantel, 2010, Ghannay et al., 2016].

Many methods of deriving word embeddings were explored in the NLP community. We will describe types of models to learn such word vectors in as in [Collobert et al., 2011, Huang et al., 2012, Mikolov et al., 2013b, Luong et al., 2013, Pennington et al., 2014]. These models form a good basis for understanding neural-based approaches and can be used for other simple word classification tasks.

Word embeddings are created through feature learning technique that maps the words to real-valued vectors in a low-dimensional space. By leveraging large corpora of unlabeled text, such continuous space representations can be computed for capturing syntactic and semantic information about words. Basically, a neural net-

work that takes as input words from a vocabulary yields word embeddings as the weights of the network.

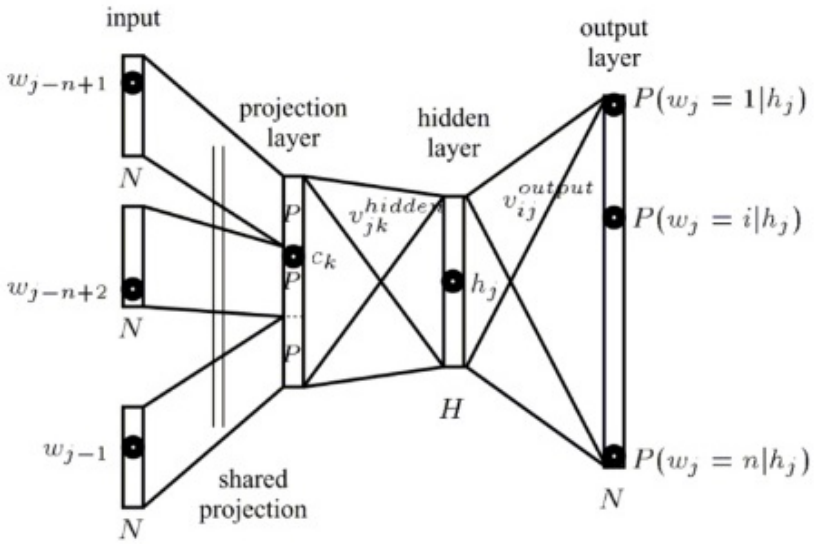


Figure 3.2 – A neural language model [Bengio et al., 2003]

In comparing models for creating word embeddings, we will assume the following: a training corpus containing a sequence of N training words $\{w_1, \ldots, w_N\}$ that belong to a vocabulary V whose size is $|V|$. A context of n words is assumed with every word associated with an input embedding v_w with d dimensions and an output embedding. An objective function J_θ is optimized with regard to model parameters θ and the output consists of a score $f_\theta(x)$ for every input x.

3.2.1 Neural language model

The classic neural language model proposed by [Bengio et al., 2003] consists of a one-hidden-layer feed-forward neural network that predicts the next word in a sequence as in Figure 3.2. Their model maximizes the following objective (with the regularization term omitted):

$J_\theta = \frac{1}{N} \sum_{t=1}^{N} \log f_\theta(w_t, w_{t-1}, \cdots, w_{t-n+1})$, where
$f_\theta(w_t, w_{t-1}, \cdots, w_{t-n+1}) = p(w_t|w_{t-1}, \cdots, w_{t-n+1})$ is the output of the model (i.e. the probability computed by the *softmax* function), where n is the number of pre-

vious words fed into the model.

[Bengio et al., 2003] was among the first to introduce word embeddings learned by neural networks, real-valued feature vectors in a space. The model has three main layers:

- an embedding layer that generates word embeddings by multiplying an index vector with a word embedding matrix.
- one or more layers that produce an intermediate representation of the input, e.g. a fully-connected layer that applies a non-linear function (activation function, e.g. $tanh$, $sigmoid$, $ReLu$) to the concatenation of word embeddings of n previous words.
- a final layer (*softmax*) that produces a probability distribution over words in a vocabulary V. *softmax* is proportional to the number of words in V, thus discovering methods that alleviate the computational cost related to computing the *softmax* over a large vocabulary [Chen et al., 2015a] became one of the main challenges in both neural language and word embedding models.

Starting from this notion of neural language model, different approaches have been proposed for creating word embeddings through neural networks [Collobert and Weston, 2008, Collobert et al., 2011, Mikolov et al., 2013a, Pennington et al., 2014, Joulin et al., 2016a, Joulin et al., 2017, Bojanowski et al., 2017, Levy and Goldberg, 2014]. These approaches differ in the type of the architecture and the data used to train the model. We will detail a subset of them in the next subsections.

3.2.2 Collobert&Weston (C&W)

The model proposed by [Collobert and Weston, 2008, Collobert et al., 2011] is a direct extension of the model of [Bengio et al., 2003]. It targets the alleviation of the computational cost related to computing the *softmax* over a large vocabulary

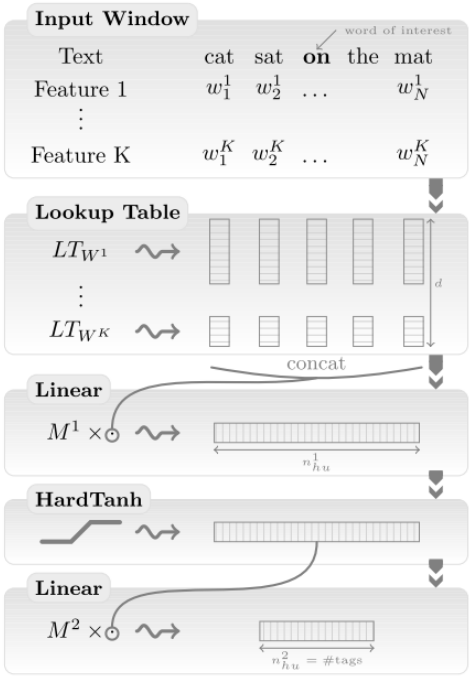


Figure 3.3 – The C&W model [Collobert et al., 2011]

by proposing a pairwise ranking criterion: $J_\theta = \sum_{x \in X} \sum_{w \in V} \max\{0, 1 f_\theta(x) + f_\theta(x^w)\}$

This objective function trains a network to output a higher score f_θ for a correct word sequence than for an incorrect one. For every sequence of words x containing n words, they generate an incorrect one x^w by replacing the middle word with a random chosen word from the vocabulary V. Their objective maximizes the distance between the scores for the correct and the incorrect window with a margin of 1.

Their model follows the architecture proposed by [Bengio et al., 2003] where the intermediate layers are fully-connected followed by the *hardtanh* non-linear layer, where $hardtanh(x) = -1$ if $x < -1; x$ if $-1 \leq x \leq 1; 1$ if $x > 1$.

3.2.3 Word2vec

In the last few years, the *Word2vec* [1] approach has become the standard approach for word embeddings. *Word2vec* is not technically a component of deep learning, with

1. Code and pre-trained embeddings available at `https://code.google.com/archive/p/word2vec/`

the reasoning being that its architecture is neither deep nor uses non-linearities. [Mikolov et al., 2013a] proposes two architectures for learning word embeddings: *Continuous Bag-of-Words (CBOW)* and *Skip-gram*, that are efficient to train and computationally less expensive.

One important goal that has been overcome is the fact that the proposed techniques are able to learn high-quality word embeddings from huge datasets. This is due to the use of hierarchical *softmax* where the vocabulary is represented as a Huffman binary tree [Huffman, 1952]. Huffman trees assign short binary codes to frequent words, and this further reduces the number of output units that need to be evaluated.

The two methods for learning word embeddings with *Word2vec* are presented next.

Word2vec: Continuous Bag-of-Words (CBOW) model Unlike in the classic neural language model proposed by [Bengio et al., 2003], where a prediction of a new word is based on a past word, [Mikolov et al., 2013a] uses a context of the targeted word, both the n words before and after the target word w_t to predict current word w_t, as shown in Figure 5.1. This is known as a *Continuous Bag-of-Words (CBOW)*, due to the fact that it uses continuous representations where the order has no importance.

The purpose of *CBOW* is only marginally different than that of the language model one:

$$J_\theta = \frac{1}{N} \sum_{t=1}^{N} \log p(w_t \mid w_{t-n}, \cdots, w_{t-1}, w_{t+1}, \cdots, w_{t+n})$$

Rather than feeding n previous words into the model, the model receives a window of n words around the target word w_t at each time step t.

Word2vec: Continuous Skip-gram model In the *Skip-gram* model, rather than using the surrounding words to predict the center word as with *CBOW*, it predicts the surrounding context words given a center word as in Figure 3.5.

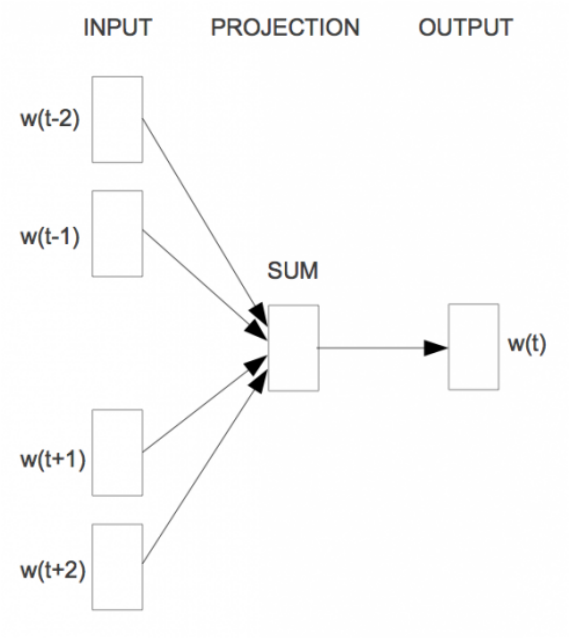


Figure 3.4 – Continuous bag-of-words (CBOW) model. Source: [Mikolov et al., 2013a]

The *Skip-gram* objective thus sums the log probabilities of the surrounding n words to the left and to the right of the target word w_t to produce the following objective:

$$J_\theta = \frac{1}{N} \sum_{t=1}^{N} \sum_{-n \leq j \leq n, \neq 0} \log p(w_{t+j} \mid w_t)$$

3.2.4 FastText

fastText [Bojanowski et al., 2017] is another popular word embedding model, which can be seen as a variant of *Word2vec* including character sequences in the learning of embeddings. The difference between the embeddings generated by [Collobert et al., 2011, Mikolov et al., 2013a] and *fastText* embeddings [Joulin et al., 2016a, Joulin et al., 2017, Bojanowski et al., 2017] is that *fastText*[2] takes into consideration the internal structure of words, which can be of a great impact when working with morphologically rich languages (as in Finnish, Turkish or French). [Joulin et al., 2016a, Joulin et al., 2017, Bojanowski et al., 2017] try to improve word embeddings by using character-level information, proposing to represent words as the sum of

2. Code and pre-trained embeddings available at `https://github.com/facebookresearch/fastText`

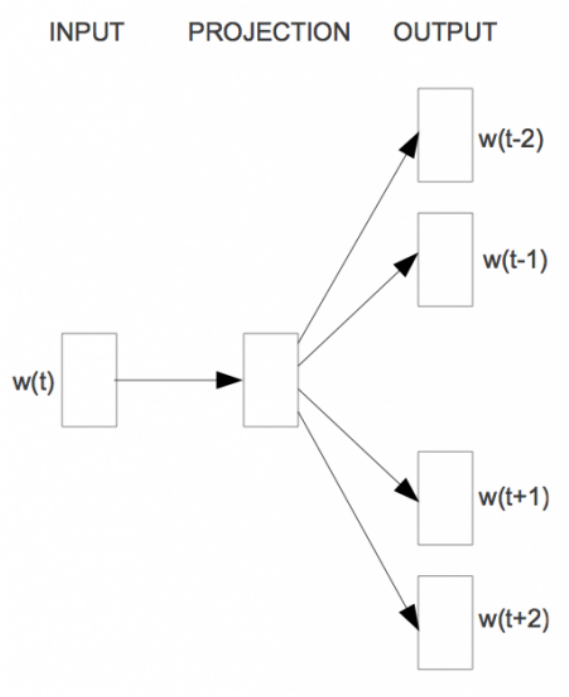


Figure 3.5 – Skip-gram model. Source: [Mikolov et al., 2013a]

learned representations for character n-grams, using an extension of the *Skip-gram* model [Mikolov et al., 2013a] presented in Section 3.2.3.

Each word w_t is represented as a bag of character n-grams and the word itself. If we consider a scoring function which maps pairs of words (word, context) $s(w, c)$ to real-valued scores, the problem of predicting context words becomes a binary classification task, where we predict the presence or the absence of those context words. For every word w_t at position t, we consider all context words as positive examples (semantically related with the word at focus) and random words from the dictionary as negative examples. The objective function is given by the following equation and computed with negative sampling:

$J_\theta = \frac{1}{N}\sum_{t=1}^{N}\Big[\sum_{c\in c_t} l(s(w_t, w_c)) + \sum_{n\in N_{t,c}} l(-s(w_t, n))\Big]$, where $N_{t,c}$ is a set of negative examples sampled from the vocabulary and l the logistic loss function $l \mapsto \log(1 + e^{-x})$.

In contrast with *Skip-gram*, where the score between the vectors u_{w_t} and v_{w_c} is the scalar product between them $s(w_t, c_t) = t_{w_t}^T v_{w_c}$, the *fastText* model considers a word

as a set of character n-grams. If we denote the n-gram vector as z and v as output vector representation of word w (context word), the scoring function is now:

$s(w, c) = \sum_{g \in G_w} z_g^T v_c$, where G_w is the set of n-grams appearing in w.

3.2.5 Dependency-based

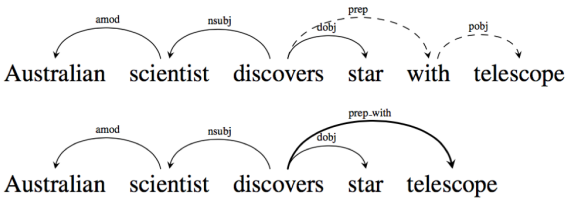
Dependency-based [3] word embeddings proposed by [Levy and Goldberg, 2014] generalize the *Word2vec* Skip-gram model and move from linear bag-of-words contexts to arbitrary word contexts. The objective function is the same as in *Skip-gram* model. In particular, the dependency-based contexts of a word include the syntactic context derived from automatically produced dependency parse-trees. While *Skip-gram* (paragraph 3.2.3) embeddings yield broad topical similarities, these embeddings can yield more functional similarities of cohyponym nature (a word or phrase that shares the same hypernym as another word or phrase: e.g. *fruit* is a hypernym of *apple*, and thus *apple* is, therefore, a cohyponym of *pear*).

The authors of [Levy and Goldberg, 2014] prove their choice of using arbitrary word contexts using this sentence as an example:
Australian scientist discovers star with telescope.

The *Skip-gram* model considers the following window contexts of size 2 for the word *discovers*: *Australian, scientist* and *star, with.* In this case, these context windows will miss some important clues, as in *telescope*, and include some irrelevant ones, like *Australian*, which may result in *stars* and *scientist* as neighbors in the embedded space. They propose the dependency-based context based on the syntactic relations the word participates in.

They motivate the usage of small arbitrary syntactic context windows by declaring that a larger window will, indeed, capture topical content, but not more focused

3. Code and pre-trained embeddings available at `https://levyomer.wordpress.com/2014/04/25/dependency-based-word-embeddings/`



WORD	CONTEXTS
australian	scientist/amod^{-1}
scientist	australian/amod, discovers/nsubj^{-1}
discovers	scientist/nsubj, star/dobj, telescope/prep_with
star	discovers/dobj^{-1}
telescope	discovers/prep_with^{-1}

Figure 3.6 – Dependency-based context extraction example. **Top**: the dependency parse tree. The preposition relations are collapsed into single arcs, making *telescope* a direct modifier of *discovers*. **Bottom**: the contexts extracted for each word in the sentence. Source: [Levy and Goldberg, 2014]

information about the target word. Using the syntactic context derived from automatically produced dependency parse-trees thus as shown in Figure 3.6, *telescope* will have more chances to be a neighbor of *discovers*.

3.2.6 GloVe

In contrast to *Word2vec*, *GloVe*[4] [Pennington et al., 2014] (*Global Vectors*) seeks to illustrate that the ratio of the co-occurrence probabilities of two words (rather than their co-occurrence probabilities themselves) contains information and they encode this information as vector differences. For this to be accomplished, the authors of [Pennington et al., 2014] propose a weighted least squares objective J that directly aims at reducing the difference between the dot product of the vectors of two words and the logarithm of their number of co-occurrences:

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(w_i^N \tilde{w}_j + b_i + \tilde{b}_j \log X_{ij} \right)^2$$

where w_i and b_i are the word vector and bias respectively of word i, $\tilde{w}_j$ and $\tilde{b}_j$ are the context word vector and bias respectively of word j, X_{ij} is the number of times word

4. Code and pre-trained embeddings available at `http://nlp.stanford.edu/projects/glove/`

i occurs in the context of word j, and f is a weighting function that assigns lower weights to rare and frequent co-occurrences. In this way, this objective associates the logarithm of ratios of co-occurrence probabilities X_{ij} with vector differences in the word vector space.

As co-occurrence counts can be directly encoded in a word context co-occurrence matrix, *GloVe* takes such a matrix rather than the entire corpus as input (but it requires one upfront pass through the dataset).

3.2.7 Conclusions

The quality of word vectors is crucial for any application. To measure this quality, intrinsic and extrinsic evaluations have been developed. Intrinsic evaluations measure the quality of word embeddings on word similarity or word analogy tasks based on human judgments that imply semantic and syntactic word relationships. Extrinsic evaluations use embeddings as features in models for other tasks, such as named entity recognition (NER) [Santos and Guimaraes, 2015, Lample et al., 2016] semantic role labeling (SRL) [Shen et al., 2014a] or part-of-speech (POS) tagging [Collobert et al., 2011], relation extraction (RE) [Zeng et al., 2014, Nguyen and Grishman, 2015b] and event extraction (Section 2.1.4). A more detailed presentation of intrinsic and extrinsic evaluations of word embeddings can be found in [Schnabel et al., 2015].

In intrinsic evaluations, when comparing the two models of *Word2vec* (*CBOW* and *Skip-gram*) on word analogy tasks, the authors of [Mikolov et al., 2013a, Mikolov et al., 2013b] observed that the *CBOW* architecture works better on the syntactic tasks while the *Skip-gram* works slightly worse on syntactic tasks than the *CBOW* model but better on semantic tasks. In the case of *fastText*, which incorporated character n-grams into the *Skip-gram* model, the authors of [Bojanowski et al., 2017] concluded that morphological information significantly improves the syntax and in contrast, it does not help for semantic tasks and even degrades the performance.

The *GloVe* model [Pennington et al., 2014] handles semantic regularities explicitly by using a global log-bilinear model regression which combines the global matrix factorization and the local context vectors when training word embeddings. The authors of [Pennington et al., 2014] state that *GloVe* outperforms *CBOW* and *C&W* on word analogy and word similarity, while using a corpus less than half the size. In extrinsic evaluations, more exactly on a common named entity recognition (NER) benchmark, the CoNLL-2003 shared task for NER [Tjong Kim Sang and De Meulder, 2003], *GloVe* vectors also outperform both *CBOW* and *C&W* vectors.

In the next section, we present the Event Detection subtask and describe the chosen baseline system based on a Convolutional Neural Network (CNN) proposed by [Nguyen and Grishman, 2015a]. We evaluate the presented CNN over the ACE 2005 corpus, with an in-depth analysis of the usage of pre-trained embeddings obtained with the previously presented methods.

3.3 Event Detection

Event Detection (ED) is considered a crucial and quite challenging subtask of event extraction and involves identifying instances of specified types of events in text and classifying them into event types precisely. Associated with each event mention is a phrase, the event trigger (most often a single verb, noun, phrasal verbs, and nouns, but also pronouns and adverbs), which evokes that event. More precisely, this subtask involves identifying event triggers and classifying them into a specific type. It can be challenging because the same event might appear in the form of multi-word expressions and these expressions might represent different events in different contexts.

The baseline neural-based model that we chose is proposed by [Nguyen and Grishman, 2015a] for the ED task. We rely on this model because it avoids using complicated feature engineering of rich feature sets that depend on NLP tools and

sources and thus, it avoids the error propagation from these tools. The authors approach the task by applying a Convolutional Neural Network (CNN) to a window of text around potential triggers that automatically learns features from the sequence of words and aims at classifying the targeted word as an event type. This simple model got correct results and will easily allow us to combine it with other models in a more complex architecture for integrating different kinds of information.

Firstly, we focus on the meaning of a trigger word and the challenges met by this task while performing a corpus analysis based on the ACE 2005 dataset. Then, we evaluate our implementation of the model in a general setting, in comparison with the state-of-the-art systems, followed by a deeper analysis of the hyperparameters of the model (e.g. word embeddings).

3.3.1 What is an event trigger?

An event trigger is a word or multi-word that depicts the occurrence of an event in a text. The problem can be thought of in the following manner: given a trigger or word, there are two possibilities: the trigger can represent an event of interest, or it can represent another event or something else in which we have no interest.

The same event might appear in the form of various trigger expressions and an expression might represent different event types in different contexts. For example, *transfer* could refer to transferring ownership of an item, transferring money, or transferring personnel from one location to another. Each sense of the word is linked with an event type. In the same manner, *fired* can correspond to an **attack** type of event as in *an American tank fired on the street* or it can express the dismissal of an employee from a job as in *Hillary Clinton was fired from the House Judiciary Committee's Watergate investigation.* A trigger can also appear in the form of a multi-word expression. For example, in the sentence *We finally ended up at the station*, the trigger word *ended up* is a phrasal verb. Usually, in this case, only *ended* is taken into consideration. At the same time, trigger words are not restricted by

the Part-of-Speech (POS) and can also be nouns, phrasal nouns, but also adverbs and pronouns.

3.3.2 Corpus analysis

We used for our experiments, as most EE systems, the annotated data ACE 2005 corpus provided by the ACE evaluation [5]. Thus, we will describe it in this section. ACE events are restricted to a range of types, each with a set of subtypes. Thus, only the events of an appropriate type are annotated in a document.

The corpus has 8 types of events, with 33 subtypes. These are the types of events:

- **Business**: Start-Org, Merge-Org, Declare-Bankruptcy, End-Org
- **Conflict**: Attack, Demonstrate
- **Contact**: Meet, Phone-Write
- **Life**: Be-Born, Marry, Divorce, Injure, Die
- **Movement**: Transport
- **Justice**: Arrest-Jail, Release-Parole, Trial-Hearing, Charge-Indict, Sue, Convict, Sentence, Fine, Execute, Extradite, Acquit, Appeal, Pardon
- **Transaction**: Transfer-Ownership, Transfer-Money
- **Personnel**: Start-Position, End-Position, Nominate, Elect

Events are distinguished from their mentions in text. An event mention or a trigger is a span of text (an extent, usually a sentence) with a distinguished trigger word and zero or more arguments, which are entity mentions, timestamps, or values in the extent. Since there is nothing inherent in the task that requires the two levels of type and subtype, we will refer to the combination of event type and subtype (e.g., **Life.Die**) as the event type. If we consider, for instance, this sentence, *There was the free press in Qatar, Al Jazeera but its' offices in Kabul and Baghdad were bombed by Americans.*, an event extractor should detect a *Conflict.Attack* event mention,

5. https://catalog.ldc.upenn.edu/ldc2006t06

with the trigger word *bombed*.

As mentioned in [Li, 2015], trigger words can be of many different POS tags besides verb (usually, an EE system considers that a trigger is most often a single verb, an assumption made in order to ease the task of Event Detection). Figure 3.7 illustrates the distribution of event triggers with respect to POS tags. In fact, the majority of triggers are verbs (47.92%) and nouns (42.42%), but there exist some exceptions such as adjectives (ADJ) with 3.56%, proper nouns (PROPN), 1.66%, pronouns (PRON), and adverbs (ADV). We remind here that this analysis is dependent on the NLP tools used (in our case, we relied on the SpaCy library [Honnibal and Johnson, 2015] [6]).

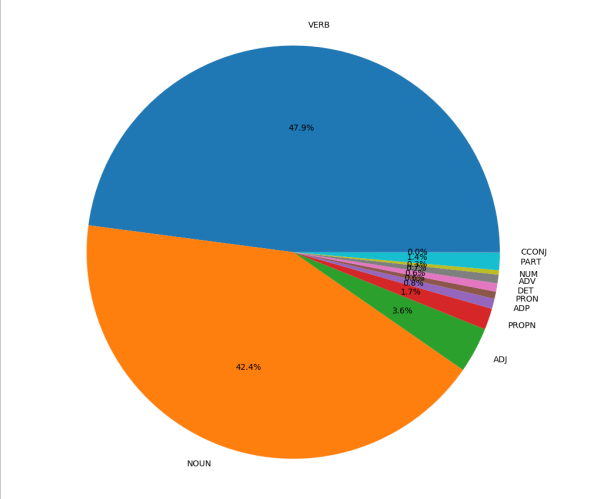


Figure 3.7 – Distribution of POS of all the true triggers

One of the main challenges is that some trigger words are ambiguous indicators of particular types of events. For example:

— *fire* can be either a *Conflict.Attack* event (*fire a weapon*) or *Personnel.End-Position* event (*fire a person*), with the cases distinguishable by the semantic type of the direct object. *discharge* has the same ambiguity and the same disambiguation rule.

— *leave* can be either a *Movement.Transport* event (*he left the building*) or a *Personnel.End-Position* event (*he left the administration*), again generally

6. More information available at `https://spacy.io/`

distinguishable by the type of the direct object.

Another challenge for this dataset is the number of triggers per sentence.

For example, in the sentence: *Frenetic merger and acquisition activity doesn't mean much in the early days of a business, it's just as likely that ridiculously overfunded startups are buying out their bankrupt competitors to create larger and less stable conglomerates.*, there are five trigger words, *merger, acquisition, buying, bankrupt, create* that depict the three different types of events, *Business.Merge-Org* mentioned by *merger* and *create*, **Transaction.Transfer-Ownership** by *buying* and *acquisition*, *Business.Declare-Bankruptcy* by *bankrupt*, with *startups* as the *Buyer* argument, *competitors* and *business* as *Organization* argument.

Table 3.1 – Distribution of some words as event triggers in the data splits (train / test / valid)

Trigger candidates	advance %			leave %			fire %		
	train/test/valid 8/1/4			train/test/valid 35/9/9			train/test/valid 93/1/13		
Movement.Transport	25.0	0.0	75.0	17.1	22.2	11.1			
Conflict.Attack							34.4		23.0
Personnel.End-Position				2.8	0.0	33.3	2.1		23.0
Other	75.0	100.0	25.0	80.0	77.7	28.8	63.4	100.0	53.8

Event detection often comes with a tremendously unbalanced dataset where the number of the non-event examples far exceeds the others, making trigger detection more challenging. The train set of the ACE 2005 corpus has 79.98% negative instances (sentences that do not contain events of interest) and 98.13% of all the words are not triggers of events of interest.

Moreover, a true trigger like *fire*, for example, refers to an *Attack* or *End-Position* event for 36.5% of its occurrences in the train dataset, and in the other 63.4% of the cases, it represents another type of event or something else in which we have no interest. At the same time, none of the *fire* instances in the test set represents an event of interest. For illustrating such imbalances, Table 3.1 presents statistics about the distribution of some event triggers, more precisely *advance, fire, leave*, among the

train/test/valid datasets. For example, it shows that the candidate trigger *advance* is an event of no interest in 100% of the cases in the test set but refers to a *Transport* event in 75% of the cases in the validation set, which can be clearly a problem for finding good values for hyperparameters.

Next, we present the Convolutional Neural Network (CNN) model for the chosen baseline [Nguyen and Grishman, 2015a].

3.3.3 Event Detection CNN

We chose as a baseline model, a Convolutional Neural Network (CNN), proposed by [Nguyen and Grishman, 2015a] where the Event Detection (ED) task is modeled as a word classification task. Although an event trigger may in principle be more than one word, more than 95% of the triggers in the data consist of a single word. The others are in the form of multi-word expressions (e.g. phrasal verbs, nouns). Furthermore, in the training data, triggers are not restricted by their Part-of-Speech (POS) (to nouns, verbs, adjectives, adverbs, pronouns). Thus, trigger identification for a document is reduced to the task of classifying each word in the document into one of 34 classes (the 33 event types plus an *Other* class for words that are not an event anchor). In the case of [Nguyen and Grishman, 2015a], the authors consider only the single word trigger candidates, meanwhile, in our case, we also consider the multi-word candidates. For example, for a phrasal verb, we consider as trigger candidates, both the verb and the adverb preposition. At the prediction time, the two consecutive candidate triggers are concatenated and evaluated in the proper form.

So, considering a sentence, we want to predict for each word of the sentence, if the current token is a trigger and decide what event type it represents. The current token $x^{(i)}$ is surrounded by a context that constitutes the main entry for the CNN. In order to consider a limited sized context, longer sentences are trimmed and shorter ones are padded with a special token. Let $x = [x^{(0)}, x^{(1)}, ..., x^{(N)}]$ be a sentence with

words from 0 to N. Given a document, we first generate a set of trigger candidates $\mathcal{T}$. For each trigger candidate $x^{(i)} \in \mathcal{T}$, we associate it a context window. We consider $2 \times n + 1$ the size of the context window, thus a trigger candidate $x^{(0)}$ is represented as $x = [x^{(-n)}, x^{(-n+1)}, \ldots, x^{(0)}, \ldots, x^{(n-1)}, x^{(n)}]$. Each context token $x^{(i)}$ has as features the word itself and the relative position of the token to the trigger candidate $x^{(0)}$. This linear distance in between two words in a sentence may serve as an informative feature, as the position of the trigger is a strong signal for this prediction task.

That is, each core feature is embedded into a d-dimensional space, and represented as a vector in that space. The feature embeddings (the real values of the vector entries for each feature, in this case, words and distances) are treated as model parameters that need to be trained together with the other components of the network. Thus, each feature is mapped to a vector retrieved from the following embedding tables:

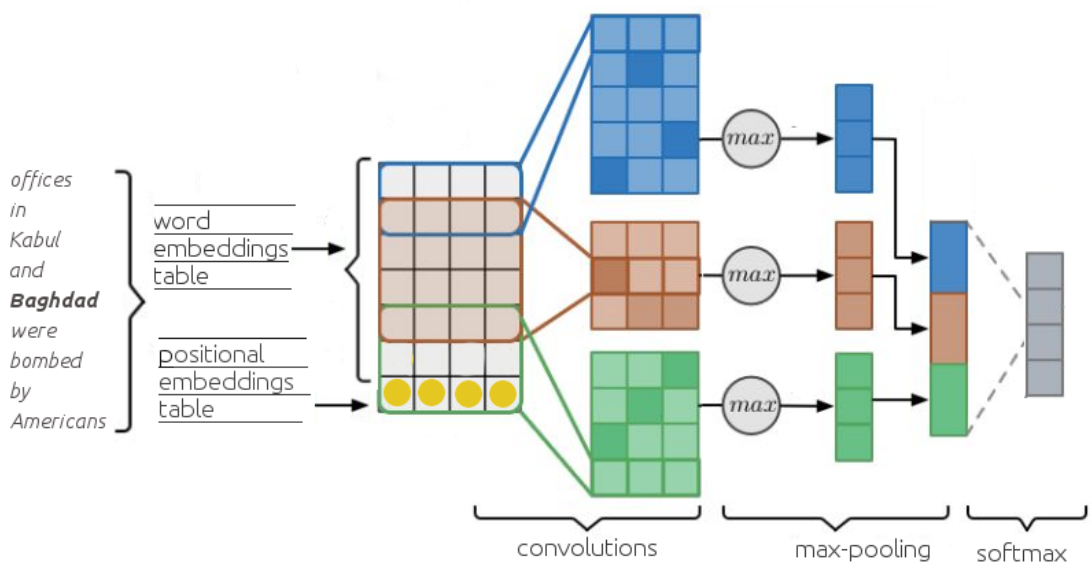


Figure 3.8 – CNN model for ED, where *Baghdad* is the current trigger candidate in a context window of $2 \times 4 + 1$ words

— Word Embeddings Table (initialized or not by some pre-trained word embeddings): to capture the hidden semantic and syntactic properties of the tokens

— Positional Embeddings Table: to embed the relative distance i of the token

$x^{(i)}$ to the current token $x^{(0)}$. Each distance value is associated with a d-dimensional vector (in practice, the table is initialized randomly), and these distance embedding vectors are then trained as regular parameters in the network [Zeng et al., 2014, Dos Santos and Gatti, 2014, Zhu et al., 2015, Nguyen and Grishman, 2015a].

For each token $x^{(i)}$, the vectors $X_{n\times 2+1,mt}$ obtained from the table look-ups above are concatenated into a single vector to represent the token. As a result, the original event trigger x is transformed into a matrix X, where d is the distance embedding, w is the word embedding and mt is the size of the concatenated embeddings.

The matrix representation X is then passed through a convolutional layer, as seen in Figure 3.8, for the candidate trigger *Baghdad*. The word and positional embeddings of the words in the context window are concatenated and passed through the convolutional layer, where we have a set of feature maps (filters) $\{f_0, f_1, ..., f_n\}$ for the convolution operation. Each feature map f_i corresponds to some filter size k and can be essentially seen as a weight matrix of size $mt \times x$. A max-over-time pooling operation [Collobert et al., 2011] is applied over every feature map and the maximum value is taken as the feature corresponding to this particular filter [Kim, 2014, Kalchbrenner et al., 2014]. These features form the penultimate layer and are passed to a fully connected *softmax* layer whose output is the probability distribution over labels.

3.3.4 Evaluation framework

For comparison purposes, we use the same test set with 40 newswire articles (672 sentences), the same development set with 30 other documents (863 sentences) and the same training set with the remaining 529 documents (14,849 sentences) as in previous studies of this dataset [Ji et al., 2008, Liao and Grishman, 2010, Li et al., 2013b, Nguyen and Grishman, 2015a, Nguyen et al., 2016a]. Following previous work [Ji et al., 2008, Liao and Grishman, 2010, Hong et al., 2011, Nguyen and Grishman,

2015a, Chen et al., 2015b], a trigger is correct if its event subtype and offsets match those of a reference trigger. We use Precision (P), Recall (R) and F-measure (F1) to evaluate the overall performance.

3.3.5 Results and comparison with existing systems

We present an analysis of the state-of-the-art systems on the ACE 2005 dataset in Table 3.2. The baseline model for event detection will be referred as *CNN without any external features* [Nguyen and Grishman, 2015a] and our implementation of this model as *CNN without any external features, our implementation.*

Hyperparameters

In [Nguyen and Grishman, 2015a], the set of parameters is as it follows. The filter sizes for the convolutional layer with $ReLu$ activation are from the set $\{2, 3, 4, 5\}$ to generate feature maps, each of them with size 150. The window size for triggers is set to 31 while the dimensionality of the position embeddings is 50. Some of these values are inherited from [Kim, 2014], as in the dropout rate of 0.5 applied after the max-over-time pool operation. The mini-batch size is 50 and the embeddings for initialization are the pre-trained word embeddings *Word2vec* for Google News [Mikolov et al., 2013a].

However, the source code for [Nguyen and Grishman, 2015a] has not been published and the reproducibility of their system has proven difficult due to different reasons: the choice of the NLP tools for pre-processing (sentence splitting, tokenization), which can influence considerably such a system, the hyperparameters are vaguely presented and some assumptions and choices are not stated. Finally, the reporting of the results does not take into account the intrinsic variability of nondeterministic approaches.

In order to replicate their results, we tuned the model parameters on the development

data and obtained a different configuration of parameters. The parameters used for our implementation of the CNN model for event detection are depicted as follows. The filter sizes used in the experiments are in the same set but we use 300 feature maps for each filter size in this set. We use the same dropout rate of 0.5 after the max-over-time pool operation and we also apply a dropout of 0.3 to the embeddings. The window size and the dimensionality of the position embeddings remain the same. The size of the mini-batch is set to 128 and we employed also the same pre-trained word embeddings.

The gradients are computed using back-propagation, regularization is implemented by early stopping [Prechelt, 1998], with a patience of 2 epochs, consisting in stopping the training as soon as the error on the validation set is higher than it was in the previous epoch. Training is done via stochastic gradient descent with shuffled mini-batches and the *Adam* update rule, learning rate of 0.001, different from [Kim, 2014, Nguyen and Grishman, 2015a] who used *Adadelta*. During the training, the word and positional embeddings tables are optimized at the same time to reach an effective state [Kim, 2014].

Results

The systems in Table 3.2 include both feature-based and neural-based methods, either with a pipeline or a joint inference approach:

— The feature-based systems with local features (rich hand-designed feature sets) and global features such as cross-document, cross-sentence, and cross-event information: the pipelined architecture in [Hong et al., 2011, Ji et al., 2008] (1, 2), both models proposed by [Li et al., 2013b] (3, 4), the pipelined *MaxEnt*s and the joint architecture (the structured perceptron model for joint beam search), cross-* features based systems [Ji et al., 2008, Hong et al., 2011, Liao and Grishman, 2010, Yang and Mitchell, 2016] (5, 9, 10, 11). We also consider here the *Seed-based* method in [Bronstein et al., 2015] (16) that

Table 3.2 – Performance of the state-of-the-art systems for event detection on the blind test data; [1]: beyond sentence level; [2]: with gold-standard entity mentions and types. The systems are given by ascending F1. Systems are referred by their number in the text.

Number	Approaches	Precision	Recall	F1
1	Sentence-level in [Hong et al., 2011]	67.5	53.5	59.7
2	Sentence-level in [Ji et al., 2008]	67.6	53.5	59.7
3	Joint beam search with local features in [Li et al., 2013b]	73.7	59.3	65.7
4	[2]MaxEnt with local features in [Li et al., 2013b]	74.5	59.1	65.9
5	Cross-sentence + Cross-doc inference in [Ji et al., 2008]	60.2	76.4	67.3
6	Joint beam search with local and global features in [Li et al., 2013b]	73.7	62.3	67.5
7	CNN without any external features, our implementation	72.9	62.9	**67.5**
8	CNN without any external features [Nguyen and Grishman, 2015a]	71.9	63.8	**67.6**
9	[1]Cross-entity in [Hong et al., 2011]	72.9	64.3	68.3
10	Joint at document level in [Yang and Mitchell, 2016]	75.1	63.3	68.7
11	Cross-event in [Liao and Grishman, 2010]	68.7	68.9	68.8
12	CNN augmented with entity types [Nguyen and Grishman, 2015a]	71.8	66.4	69.0
13	Dynamic multi-pooling CNN in [Chen et al., 2015b]	75.6	63.6	69.1
14	Joint RNN in [Nguyen et al., 2016a]	66.0	73.0	69.3
15	Non-Consecutive CNN in [Nguyen et al., 2016b]	–	–	71.3
16	Seed-based in [Bronstein et al., 2015]	80.6	67.1	73.2
17	Hybrid neural network in [Feng et al., 2016]	84.6	64.9	73.4

takes the example triggers in the training set as seeds, and then applies an event-independent similarity-based classifier for event detection.

— The neural-based models: the CNN model without any external features in [Nguyen and Grishman, 2015a] (12), the dynamic multi-pooling CNN model [Chen et al., 2015b] (13), the bidirectional joint Recurrent Neural Networks (RNNs) [Nguyen et al., 2016a] (14) and the non-consecutive CNN in [Nguyen et al., 2016b] (15). Also, we consider the model that holds the state-of-the-art event detection results, the hybrid model proposed by [Feng et al., 2016]

(17) that benefits from the features combination obtained from a CNN and a Long-Short-Term Memory Units (LSTM).

As it can be seen in the presented results, the neural-based models outperform all the feature-based models, sentence-level (e.g. local: lemma, synonyms, Part-of-Speech (POS) tags, features from a multitude of sources, WordNet [Leacock and Chodorow, 1998]) or beyond sentence level (e.g. global: cross-document, cross-sentence and cross-event information). It is remarkable since these models, with the exception of [Nguyen et al., 2016a] (13) that uses the dependency paths as semantic features, do not require any external features, in contrast to the other feature-based systems [Yang and Mitchell, 2016, Liao and Grishman, 2010, Hong et al., 2011, Ji et al., 2008, Li et al., 2013b] (10, 11, 1, 4, 6) that extensively rely on such external features to perform well.

Considering the systems that only use sentence level information, *CNN without any external* [Nguyen and Grishman, 2015a] (7, 8) significantly outperforms the *MaxEnt* classifier [Li et al., 2013b] (4) as well as the joint beam search with local features from [Li et al., 2013b] (6). *CNN without any external features, our implementation* (7) [Nguyen and Grishman, 2015a] has a drop of 0.1 in *F1*.

We believe that this model is gaining potential over the other systems, regarding the need of almost no preprocessing of data, no reliance on Natural Language Processing (NLP) toolkits for feature extraction and no cross-sentence nor cross-document inference.

Next experiments are based on this baseline model. We will present also how different parameters influence the performance of such a model.

3.3.6 Word embeddings analysis

In this section, we investigate different methods to obtain the pre-trained word embeddings used for initializing the baseline Convolutional Neural Network (CNN)

model. Several factors influence the quality of word vectors, from their size and training algorithm to the amount and quality of their training data. The ability to perform well on word similarity and analogy tasks, which implies different semantic-syntactic word relationships, is a way to characterize their quality but is not necessarily linked in a straightforward way to their performance in downstream applications.

Table 3.3 presents the performance for trigger classification on the blind test set for the list of pre-trained embeddings presented in Section 3.2: pre-trained *Word2vec*[7] embeddings on Google News corpora [Mikolov et al., 2013a], *GloVe* [Pennington et al., 2014] trained on English Wikipedia 2014[8] and Gigaword 5[9], Collobert&Weston (C&W) [Collobert et al., 2011] and dependency-based [Levy and Goldberg, 2014] trained on English Wikipedia (2007 and respectively 2014) and *fastText*[10] embeddings trained on English Wikipedia 2010 [Shaoul, 2010].

We also trained word embeddings on the Gigaword 5 corpus (North American News) using the *Word2vec* toolkit and we include them in the analysis. These *Gigaword* embeddings were trained with the *Skip-gram* model presented in Section 3.2.3 (use of the center word to predict the context), a context of 5 words and an embedding size equal to 400.

We take into consideration that the embeddings differ in size and that the dataset on which they are trained on has distinct data sources. We analyze the results accordingly.

Finally, for comparison, the performance of random word embeddings drawn from a uniform distribution is also included. All word embeddings are updated during the training of the model.

7. Code and pre-trained embeddings available at `https://code.google.com/archive/p/word2vec/`
8. Data available at `https://dumps.wikimedia.org/`
9. Data available at `https://catalog.ldc.upenn.edu/LDC2011T07`
10. Data available at `http://www.psych.ualberta.ca/~westburylab/downloads/westburylab.wikicorp.download.html`

Table 3.3 – Performance of the baseline CNN with different word embeddings (algorithm and data sources). Values written in italic are assumed.

Embeddings Type	**Size**	**Corpora**	**No. Tokens**	**Vocab. Size**	**P**	**R**	**F1**
Random, baseline	50				65.7	45.2	53.6
Random, baseline	300				62.3	54.2	58.0
Random, baseline	400				69.3	51.1	58.8
C&W [Collobert et al., 2011]	50	Wikipedia 2007	*450M*	103k	50.1	56.6	53.1
Dependency-based [Levy and Goldberg, 2014]	300	English Wikipedia 2014	*1.2B*	175k	67.7	55.8	61.2
GloVe [Pennington et al., 2014]	300	English Wikipedia 2014 + Gigaword 5	6B	400K	63.5	62.9	63.2
Gigaword trained	400	Gigaword 5	4B		68.1	60.6	64.1
fastText [Joulin et al., 2016b]	300	English Wikipedia 2010	900M	2M	65.8	64.1	64.9
Word2vec [Mikolov et al., 2013a]	300	Google News	**100B**	3M	72.9	62.9	**67.5**

First, we can globally observe from Table 3.3 that using pre-trained word embeddings as features results in performance lifts when compared to the baselines. It should be noted that we have considered baselines with three different sizes for a fair comparison in terms of size.

We can more precisely observe that all pre-trained word embeddings of size 300 perform better than the random ones, suggesting that they may contain relevant information for the task. Among the pre-trained embeddings of size 300, it is clear that *Word2vec* embeddings outperform all the others.

The size of the data on which the embeddings have been trained on has unsurprisingly a significant impact on the performance of the task. *Dependency-based* [Levy and Goldberg, 2014] embeddings have been trained on around 1 billion words, *Glove* [Pennington et al., 2014] on 6 billion words, *fastText* [Bojanowski et al., 2017] on 900

million words and *Word2vec* [Mikolov et al., 2013a] on 100 billion words. Moreover, the *Word2vec* embeddings also have the largest coverage in terms of vocabulary, with 3 million words. Thus, the better performance of *Word2vec* embeddings can be explained not only from the dataset that has been used for learning, Google News, in comparison with the other two that use mainly English Wikipedia but also from the greater coverage of words.

For studying more specifically the influence of the size of the embeddings, we performed experiments with randomly generated embeddings. Note that in Figure 3.9, the most efficient embedding size is 600 on this type of task but with enough dimensions, between 300 and 600, the performances of the embeddings are quite comparable, with F1 values between 58% and 59%.

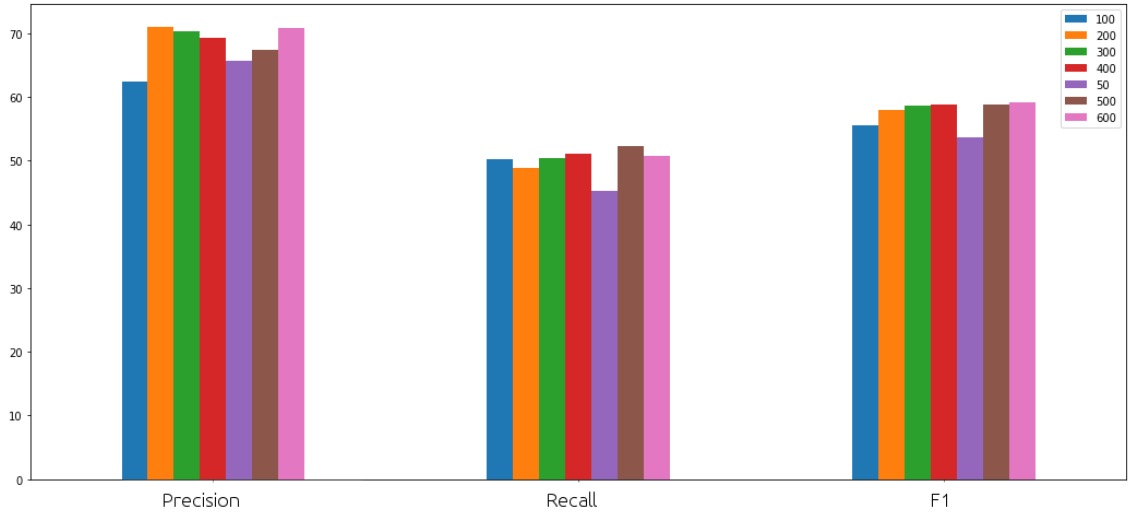


Figure 3.9 – Different sizes for random embeddings

From the point of view of the training algorithm, most of our tested embeddings, *Word2vec*, *Gigaword trained*, *fastText* and *dependency-based* embeddings, were trained using *Skip-gram* or a variation of *Skip-gram*, which does not allow conclusions to be drawn. *Gigaword trained*, *Word2vec*, *fastText* outperform on the ED task the *dependency-based* embeddings. *Dependency-based* embeddings [Levy and Goldberg, 2014] are known to provide structural information, are less topical and exhibit more functional similarity than the original *Word2vec* embeddings. The evaluation of syn-

tax based word embeddings and dependency context embeddings on different tasks done by the authors of [Melamud et al., 2016] proves the usefulness of *dependency-based* embeddings on parsing tasks. While Table 3.3 is not sufficient for drawing strong conclusions about this point, it seems that for ED, the use of *dependency-based* embeddings is not necessarily the best option.

The embeddings trained with *fastText* performed similarly to the embeddings trained on *Gigaword* with *Word2vec* while the corpus used for training *fastText* had only 900 million words, compared to the 4 billion words of the *Gigaword* corpus. This suggests that the character model used by *fastText* is particularly interesting in this context.

Finally, we assume that the performance of *C&W*, that obtained a score close to the random embeddings, is due not only to the size of the embeddings, 50, but also from the size of the corpus used, a small version of English Wikipedia.

Globally, we consider that the high performance obtained by *Word2vec* embeddings comes from the dataset that has been used for learning, namely large news datasets. Meanwhile, we consider that *fastText* embeddings bring improvements in performance because of their additional morphological information (character-level information) in learning word vectors using an extension of *Skip-gram*.

To understand the behavior of word relevance for the Event Detection (ED) task, we can look-up a set of neighbors for the difficult trigger candidates mentioned in Section 3.3.1, respectively *fire, advance* and *leave*, in Table 3.4. In this case, the nearest neighbors of a word are the words with the highest cosine similarity between their respective vectors. The cosine similarity is calculated by the dot product of two numeric vectors and is normalized by the product of the vector lengths, so that output values close to 1 indicate high similarity $cos(x, y) = \frac{x \cdot y}{\|x\| \cdot \|y\|}$.

Table 3.4 reveals that for *Word2vec* and *fastText*, the found neighbors are more relevant than the ones found for *Collobert&Weston*. The neighbors of *fire* represent

Table 3.4 – The neighbors for three difficult trigger candidates

Keywords	Collobert& Weston	Dependency-based	Word2vec	fastText
fire	water, land, ship, mine, fuel, boat, bomb, storm, smoke, sea	fires, gunfire, bushfire, shellfire, wildfire, barrage, conflagration, musketry, fusillade	blaze, fires, Fire, flames, carelessly discarded cigarette	fires, extinguisher, extinguishers, firefights, firefighters, extinguishing
advance	invasion, progress, escape, transfer, surrender, defense, burma, appeal, abort, invitation	progress, advancing, advances, advanced, counterattack, push, spearhead	advancing, advances, Advance, advanced, Trafigura_Vena, Teras_vision	advancing, advancer, advancers, #advance
leave	save, enter, visit, notice, wait, hold, marry, reach, break, abandon	leaving, reenter, abscond, depart, abandon, redecorate, rehire, vacate, remarry, forsake	leaving, stay, depart, Leaving, left, leaves, return, vacate, quit, rejoin	stay, leaving, wait, rejoin, return, join, depart, #leave

mainly in the first case the concept of dismissal from an office or position, as in the event subtype **Job-Title**, while for *Collobert&Weston*, they refer more to a state or an instance of combustion. In the case of *fastText*, as it was expected, the neighbors are mostly morphological derivations and inflections of the target word.

In the next section, due to the fact that *fastText* performed very well in the ED task, we attempt at improving the embeddings by retrofitting the word vectors with a semantic lexicon dedicated to the target types of events.

3.3.7 Retrofitting

Based on the underlying assumption that trigger words can be clustered semantically and that a similarity might exist between trigger words, we include an experiment

based on retrofitted word embeddings.

Arguably, one of the reasons behind the popularity of word embeddings is that they are general purpose: they can be used in a variety of tasks without modification. This can be observed from the word embeddings analysis in the previous section. Although this behavior is sometimes desirable, it may in other cases be detrimental to specific tasks that use word embeddings. For example, when classifying trigger words by event type, we are particularly interested in related words rather than similar ones: knowing that *move*, which represents a *Movement.Transport* event type, is associated with *trip* is much more informative of the topic than knowing that it is a synonym of *walk*.

[Faruqui et al., 2015] introduced *retrofitting* as a graph-based learning technique for using lexical relational resources to obtain higher quality semantic vectors. Retrofitting is applied as a post-processing step by running belief propagation on a graph constructed from lexicon-derived relational information to update word vectors. This allows *retrofitting* to be used on pre-trained word vectors. We constructed the lex-

Table 3.5 – Performance of the state-of-the-art CNN with different retrofitted word embeddings

***Retrofitted* Embeddings Type**	**Precision**	**Recall**	**F1**	**F1** no *retrofitting*
Collobert&Weston [Collobert et al., 2011]	42.3	64.8	51.2↓	53.1
Dependency-based [Levy and Goldberg, 2014]	63.8	57.5	60.5↓	61.2
Word2vec	64.7	62.0	63.3↓	67.5
Gigaword trained	63.1	64.3	63.7↓	64.1
GloVe [Pennington et al., 2014]	60.4	68.8	64.3↑	63.2
fastText [Bojanowski et al., 2017]	64.5	63.2	64.3↓	64.9

icon with all the trigger words from the train partition of the ACE 2015 Corpus. This structured collection indicates that entries are related to each other by the type of event. For example, *go, come, went, take, going, moving, trip, coming* are

associated with *Movement.Transport*.

We apply *retrofitting* for these words, for all the pre-trained embeddings used in the previous experiments and compare with the results obtained when no *retrofitting* was applied.

We can notice from Table 3.5 that *retrofitting* does not globally provide better results for the event detection task. The arrow in every column ↓ reveals that *retrofitting* almost any type of embedding to better represent the types of events decreases the performance for the ED task. *Retrofitting* brings a slight improvement only in the case of *GloVe* embeddings. We assume that these results are due to the main challenge of the task, the fact that some trigger words are ambiguous indicators of particular types of events. For those triggers, the amplification brought by *Retrofitting* is likely to have a negative impact.

3.4 Conclusions

After we presented the baseline model for event detection and performed different experiments for the choice of parameters, the results can be explained by the following reasons:

— firstly, compared with the feature-based methods that benefit from manual engineered feature sets, all neural-based methods perform better by avoiding error propagation from different NLP tools (parsers etc.) and by better representing the semantics of the words;

— CNNs are a good choice for event detection since they capture global representations of text and extract the most informative parts for the sequence of words and only considers their resulting activations;

— experiments show also that the choice of pre-trained embeddings has an important impact on the performance of the ED task. The data used for training the embeddings, their size, and the training algorithm influence the perfor-

mance. We concluded that the higher the amount of data, the better the word vectors and 300 is a good choice for their size. Also, *Skip-gram*-based embeddings (*Word2vec*, *fastText*) have obtained the best word representations for the task;

— regarding *Retrofitting*, the process applied as a post-processing step to improve vector quality does not bring improvements in performance for the ED task. We assume that placing similar words from the chosen semantic lexicon in the same neighborhood is affecting the ability of the model to distinguish between triggers that are common to several event types. For example, the word *fire* remains ambiguous, representing a *Conflict.Attack*, but also a **Personnel.End-Position** event. It is also possible that the *Retrofitting* process creates too much distortion in the space, when applied, to capture some semantic relatedness.

Chapter 4

Deep neural network architectures for Event Detection

We presented in the previous chapter the baseline model proposed by [Nguyen and Grishman, 2015a], a Convolutional Neural Network (CNN) with word and position embeddings.

We introduce in this chapter two deep neural network architectures for event detection. We are going beyond word embeddings in two directions, trying to capture more global information and more local information.

The first contribution consists in using representations at the sentence-level, through sentence embeddings. We assume that the context taken into account by the baseline model is too limited. For long sentences, this limit results from the fixed size of the window in which the convolutions are performed. But more globally, CNNs intrinsically focus on continuous and short range n-grams. An event is defined not only by the trigger word but also by important words that can be found in its context. However, these words are not always grouped in the span of an n-gram. So, our hypothesis is that representing the whole sentence in a way that it can predict the existence of a trigger will further help the model distinguish between event types. We introduce a deep neural network architecture, which combines Bidirectional RNNs

(Bi-RNNs) for adding sentence-level information for every trigger candidate.

The second contribution aims at exploiting two techniques for incorporating the inner information of words as new features. Firstly, we consider creating character-level features that can capture morphological and shape information of words, whereas word embeddings are effective to capture word-level syntactic and semantic information. For example, if we take a new word not present in the training data, *torturing*, but present in the validation data, given its root and suffix (i.e. *tortur-ing*), it is natural to guess that it is a variant of *torture* and the latter probably represents the same type of event, that being *Life.Injure*. At the same time, we target one of the main concerns when working with word embeddings, that is to say, the treatment of the words that occur infrequently since word embedding models suffer from lack of enough training opportunity for infrequent words. Another advantage of adding character-level representations is the possibility of treatment of misspelled or custom words. The second technique is a type of data augmentation that relies on morphological derivation and inflection generation, which practically adds variants of known triggers in train, targeting the same impact as the character-level features. We introduce a neural network architecture where we enrich every trigger candidate context with character-level features and we perform experiments to observe the effect of augmenting the data with new variants of triggers.

Before presenting our models, we will present the theory that underlies our propositions.

4.1 Background theory

4.1.1 Recurrent Neural Networks (RNNs)

One drawback of convolutional and pooling architectures is while they allow capturing salient features, they are sacrificing the structural information. Recurrent

and recursive architectures, on the other hand, work with sequences and trees while preserving the structural information. Thus, two choices are available: either to capture regularities in such structures or to model similarities between such structures. In many cases, this means encoding the structure as a fixed-sized vector, which can then be passed to another statistical model for further processing.

Two types of networks can handle variable sized input vectors: recurrent and recursive neural networks. Recurrent networks [Elman, 1990] are designed to model sequences, while recursive networks [Goller and Kuchler, 1996] are generalizations of recurrent networks that can handle trees. The Recurrent Neural Network (RNN) is a type of deep neural networks that is deep in temporal dimension and has been used extensively in time sequence modeling (speech recognition, text summarization) [Mikolov et al., 2010, Graves, 2012, Zeng et al., 2016].

Since we consider sentences as sequences, we chose RNNs for representing these. RNNs take as their input not just the current input example they see, but also what they have perceived previously in time. In its simplest form, a RNN is a Multi-Layer Perceptron (MLP), but with recurrent layers. Basically, at the time step $t-1$, a prediction is performed, which affects the prediction performed at time step t. Thus a recurrent layer not only receive information from the previous layers, but also from itself, as seen in Figure 4.1.

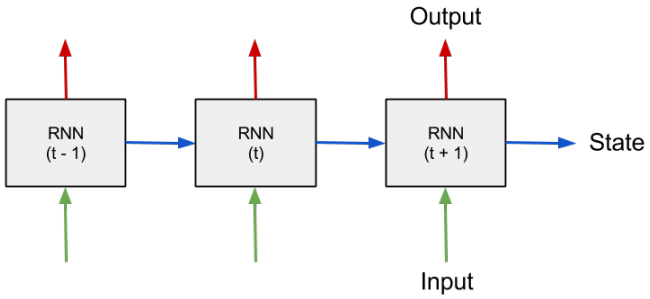


Figure 4.1 – An RNN processing sequential data over time. Source: https://medium.com/@erikhallstrm/hello-world-rnn-83cd7105b767

For example, in Natural Language Processing (NLP), an RNN can map vectors of sentences of variable length to a fixed-length vector by recursively transforming

current sentence vector x_t with the output vector of the previous step h_{t-1}. The transition function is typically a linear layer followed by σ, a pointwise non-linearity layer (activation function). The state of such a layer evolves through time with the following recurrence:

$$h_t = \sigma\Big(W_f\Big[h_{t-1}, x_t\Big] + b_f\Big)$$

where $W_f \in R_{l_h \times (l_t \times l_s)}$, $b_f \in R_{l_h}$, l_h and l_s are dimensions of hidden vector and sentence vector, respectively. x_t represents the input, W_f the corresponding weights, h_{t-1} the layer's outputs at the previous time step and b_f the bias units.

Unfortunately, standard RNNs suffer from the problem of gradient vanishing or exploding [Bengio et al., 1993, Hochreiter and Schmidhuber, 1997], where gradients may grow or decay exponentially over long sequences. This makes it difficult to model long-distance correlations in a sequence. To address this problem, a gating mechanism has been proposed and a type of recurrent neural network that solves this problem. We will present in the next section this type of network, Long-Short-Term Memory Units (LSTM).

4.1.2 Long Short-Term Memory Units (LSTMs)

[Hochreiter and Schmidhuber, 1997] proposed a variant of a recurrent network with so-called Long Short-Term Memory units, or LSTMs, as a solution to the vanishing gradient problem. This type of recurrent network introduces a gating system to control the flow of information, by deciding what information can be *forgotten*, hence the word *memory* in LSTM. The gating system includes: an **input gate**, an **output gate** and a **forget gate**. These gates compute an activation often using the *sigmoid* function. An LSTM cell is shown in Figure 4.2.

The following equations define the behavior of the LSTM unit:

The **forget gate** controls whether the previous state is integrated into the cell state,

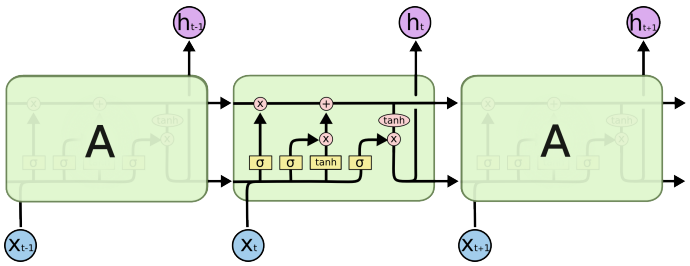


Figure 4.2 – An LSTM processing sequential data over time. Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

or if it will be *forgotten*. The activation function σ for this gate is usually the sigmoid function. The *sigmoid* layer outputs numbers between zero and one, describing how much of each component should be let through, where 0 means totally forget.

$$f_t = \sigma\Big(W_f \cdot \Big[h_{t-1}, x_t\Big] + b_f\Big)$$

The **input gate** controls whether the input of the cell is integrated into the cell state.

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

The cell state is the sum of the previous state C_{t-1}, scaled by the forget gate f_t, and of the cell input i_t, scaled by the input gate. ψ can be the sigmoid or hyperbolic tangent function.

$$\tilde{C}_t = \psi\Big(W_C \cdot \Big[h_{t-1}, x_t\Big] + b_C\Big)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

The **output gate** controls whether the LSTM unit emits the activation:

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

The cell output is computed by applying the activation function C_t to the cell state, scaled by the output gate.

$$h_t = o_t * \psi(C_t)$$

Gated Recurrent Units (GRUs) A gated recurrent unit (GRU) is basically a LSTM without an output gate. A GRU has two gates, a reset gate r, and an update gate z, depicted by the following equations:

$$z_t = \sigma\Big(W_z \cdot \Big[h_{t-1}, x_t\Big]\Big)$$
$$r_t = \sigma\Big(W_r \cdot \Big[h_{t-1}, x_t\Big]\Big)$$
$$\tilde{h_t} = \psi(W \cdot \Big[r_t * h_{t-1}, x_t\Big]\Big)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h_t}$$

The **input** and **forget gates** from LSTMs are combined by the update gate z and the reset gate r is applied directly to the previous hidden state h_{t-1}. ψ can be the sigmoid or hyperbolic tangent function. The forget and input gates from LSTM are combined in the **update gate**. GRUs are quite recent and have been growing increasingly popular due to the decreased number of parameters (in comparison with LSTMs) and thus decreased training time.

4.1.3 Bidirectional RNNs

A bidirectional neural network has two networks, one accessing information in a forward direction and another, in the reverse direction.

These networks have access to the past as well as the future information and hence the output is generated from both the past and future context, as shown in Figure 4.3.

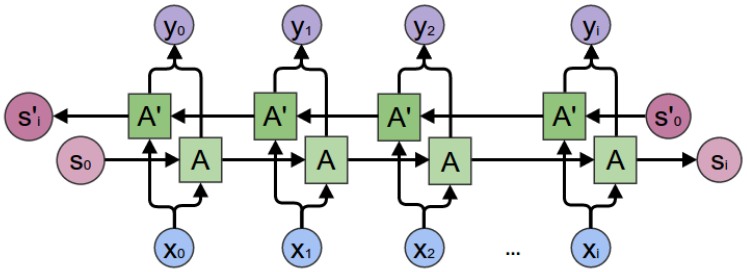


Figure 4.3 – A bidirectional RNN. Source http://colah.github.io/posts/2015-08-Understanding-LSTMs/

4.1.4 Training RNNs

In RNNs, the inputs of recurrent layers include their outputs at the previous time step. Thus, a sequential aspect is added to the layered structure of the network. The Backpropagation Through Time (BPTT) algorithm [Werbos, 1990] consists in propagating the error both from the output to the input layer and to the previous time steps. When the network is unrolled in time, one obtains a directed acyclic graph, on which backpropagation methods can apply. There are input and output layers at every time step t, so the error for each of them should be incorporated in the gradient computations.

4.2 Exploiting sentential context in Event Detection

One important feature used for satisfying the event extraction task is the context information for event detection. Our hypothesis is that, if larger context information is included, the performance could be increased. This can be achieved with the addition of sentence embeddings. In sentence embeddings, sentences, which are variable-length sequences of discrete symbols, are encoded into fixed length continuous vectors that are then used for further prediction tasks.

Since we combine two different neural networks, we propose a deep neural network model where we add an additional feature called sentence embeddings that can catch a global information about the trigger candidate that is being classified. In order to encode the sentence, we create an encoder network (a bidirectional LSTM) used to produce a vector representation of the sentence. We obtain the sentence representation from the encoder and we feed it as an additional feature into the CNN model presented in Chapter 3, before prediction.

4.2.1 Sentence embeddings

The representation of sentences, used in the context of neural networks, falls into two categories: a universal sentence embedding usually trained by unsupervised learning [Hill et al., 2016], such as recursive auto-encoders [Socher et al., 2011], [Socher et al., 2013], *SkipThought* vectors [Kiros et al., 2015], ParagraphVector [Le and Mikolov, 2014], Sequential Denoising Autoencoders (*SDAE*) or *FastSent* [Hill et al., 2016] and models trained specifically for a certain task trained in a supervised fashion, for instance the semantic sentence representations for the textual entailment task [Conneau et al., 2017].

A common approach in creating a sentence representation is by using the final hidden state of an RNN or the max (or average) pooling from either RNNs hidden states or the last convolutional layers. Several models have been proposed using this technique [Ma et al., 2015, Mou et al., 2015, Yin and Schütze, 2015, Palangi et al., 2016, Tan et al., 2016, Tai et al., 2015].

Another approach is the encoder-decoder architecture, producing models also known as sequence-to-sequence models [Sutskever et al., 2014, Cho et al., 2014, Bahdanau et al., 2014]. In this architecture, an RNN (e.g. an LSTM) is used to produce a vector representation of the sentence, which is then fed as input into a decoder network that uses it to perform some prediction task (e.g. recreate the sentence, or produce a translation of it). The encoder and decoder networks are trained jointly in order to perform the final task. For some tasks, people propose to use attention mechanism on top of the CNN or LSTM model to introduce an extra source of information to guide the extraction of sentence embeddings [dos Santos et al., 2016].

In the next section, we present the model for creating sentence embeddings based on the supervised task of detecting an event mention in a sentence, similar to [Conneau et al., 2017] for the textual entailment task, which we think can further be transferred to other tasks (e.g. relation extraction).

4.2.2 Sentence encoder

Given a sentence representation method, we train a classifier to predict the existence of a trigger in a sentence. We then keep the dense layer before prediction and for every new sentence in test, we generate a fixed-sized real-valued vector, a sentence embedding. Thus, in this sentence embedding, the relationship among words in the sentence, i.e., the context information, is taken into consideration.

We tend to efficiently learn generic representation embeddings as wide sentential contexts, using a bidirectional recurrent neural network (RNN) that can subsequently be transferred to other tasks (e.g. relation extraction). Essentially, we are able to encode a sentence using the ACE dataset to learn a model that embeds the entire sentential context and the hint of the existence of an event mention. We aim at classifying them in sentences that contain events and sentences that do not, therefore it is a binary classification.

We motivate the use of a bidirectional RNN by the choice of taking into account the entire sentential context in both directions, to gain some dependency between adjacent words within a single sentence. We want to encode a variable length sentence into a fixed size embedding. This is achieved by choosing a linear combination of two RNN hidden vectors in one representation. The first is applied to the input sentence and the second one on a reversed copy of the input sequence, which reflects the *bidirectional* meaning.

The architecture of sentence encoding is depicted in Figure 4.4. That is, we train a model that can automatically transform a sentence to a vector that encodes the semantic meaning of the sentence by using the final hidden state of the last bidirectional RNN and then we consider the sentence embeddings as a larger context feature for the event detection task.

In the encoding phase, we first transform each token w_i into a real-valued vector x_i using the concatenation of the word embedding vectors. This is obtained by

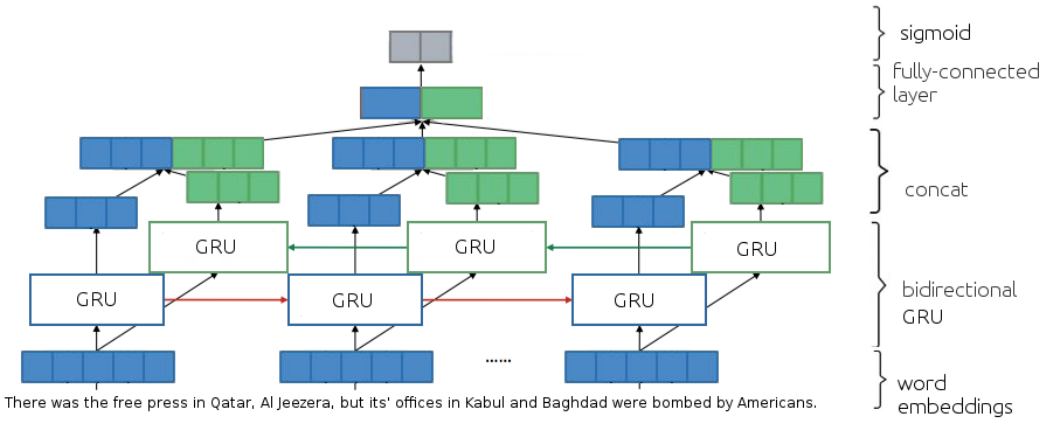


Figure 4.4 – RNN-based model for creating sentence embeddings

looking up a word embedding table (initialized or not by some pre-trained word embeddings).

Consider the input sequence $X = [x_1, x_2 \dots x_n]$. At each step i, we compute the hidden vector $h_t = \sigma\Big(W_f[h_{t-1}, x_t] + b_f\Big)$ based on the current input vector x_t and the previous hidden h_{t-1} using the non-linear transformation function σ (e.g. *sigmoid* or *tanh*). This recurrent computation is done over X to generate the hidden vector sequence $\overrightarrow{h} = [h_1, \dots, h_n]$. An important characteristic of the recurrent mechanism is that it adaptively accumulates the context information from the first position to t into the hidden vector. However, as we already said, h_t is not sufficient for the event trigger 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). Thus, a second RNN is run in the reverse direction from x_n to x_1 to generate the second hidden vector sequence $\overleftarrow{h} = [h_n, \dots, h_1]$ in which h_t summarizes the context information from position n to t. We obtain the new representation $[h_1, \dots, h_n]$ for X by concatenating the hidden vectors $\overrightarrow{h} = [h_1, \dots, h_n]$ and $\overleftarrow{h} = [h_n, \dots, h_1]$. h_t at time t basically encapsulates the context information for the whole sentence. This bidirectional representation $\overrightarrow{h} \parallel \overleftarrow{h}$, the concatenation of the hidden vectors in both directions, usually focuses on a specific component of the sentence, like a special set of related words. So it is expected to reflect an aspect, or component of the semantics in a sentence. Given

this representation, in the encoding phase, we apply a fully connected layer with a nonlinear activation and then, another fully-connected layer before prediction done with *sigmoid*, for binary classifying the sentences in sentences with and without events, as shown in Figure 4.4.

The model is trained off-line and saved, in order to be able to use this model for a transfer task. New sentences are passed through the model and the hidden representation produced by the fully-connected layer with a nonlinear activation is picked up for encoding the sentence into a fixed-sized vector.

4.2.3 Event Detection with sentence embeddings

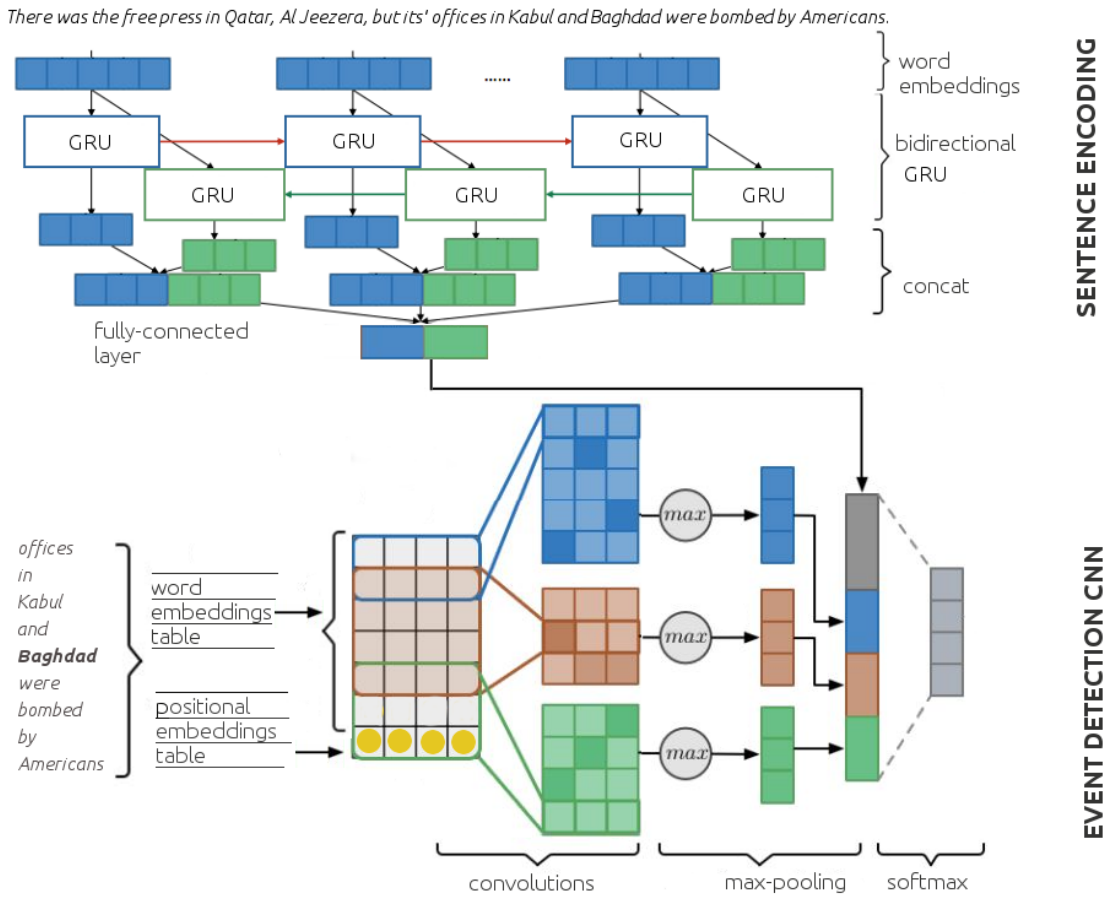


Figure 4.5 – Event detection model with sentence embeddings

The task of ED is modeled as a word classification task as described in Section 3.3.3.

As previously said, considering a sentence, we want to predict for each word of the sentence, if the current token is a trigger and decide what event type it represents. The current token is surrounded by a context that constitutes the main entry for the CNN model. For each trigger candidate $x^{(i)} \in \mathcal{T}$, we associate it a context window. We keep this model and modify it at prediction level by adding sentence embeddings as extra features. We consider that this larger context for a trigger candidate, made of the whole sentence, can bring more information about discriminant words by capturing associations between words beyond the range of the n-grams taken into account by the CNN model.

As depicted in Figure 4.5, for every window of words, we take the whole sentence, we look-up every word in the sentence in the word embedding table and we concatenate the real-valued vectors. Then, we feed these vectors into the sentence encoder and we pick up the hidden representation from the fully-connected layer as a sentence embedding. We concatenate the max-pooled word and position embeddings with the sentence embedding and feed this to a fully connected layer with a *softmax* activation function to generate the final prediction. One concern is that, for every sliding window of words with the focus on the middle word, the candidate trigger, the sentence representation is the same. We are counting in this situation on the representation of the context window to add relevant information in order to correctly detect and classify the trigger candidate.

4.2.4 Results

Hyperparameters

For the bidirectional GRU, we use 256 units for the GRU layers. We then apply a fully-connected layer of size 400 and *ReLu* activation and a final fully-connected layer with *sigmoid* activation, for binary classification. To avoid *overfitting*, we employ the use of early stopping [Prechelt, 1998] with a patience of 2 epochs, consisting

in stopping the training as soon as the error on the validation set is higher than it was in the previous epoch. We train the networks using stochastic gradient descent with shuffled mini-batches, the *Adam* update rule [Kingma and Ba, 2014] with a learning rate of $1e^{-3}$. During the training, the embedding tables (word and position embeddings) are optimized to achieve the optimal states. Finally, for training, we use the mini-batch size of 100.

Analysis

We decided to use a GRU network for creating the sentence embeddings by comparing it with an LSTM network. The results are shown in Table 4.1. GRUs usually run faster than LSTMs since they have fewer parameters, but LSTMs have a greater expressive power that may lead to better results. We can observe that the bidirectional GRU network is more balanced between precision and recall, while LSTM favors recall. Thus, we chose GRUs, which has also the benefit of having a smaller training time than LSTMs.

Table 4.1 – Performance on sentence classification: contains or not an event trigger

LSTM/GRU	**Precision**	**Recall**	**F1**
GRU	78.4	74.4	76.4
LSTM	67.6	86.3	75.9

As it can be observed from Table 4.2, our new NN architecture outperforms the baseline system by 3.2 points. It also gets results slightly better than the joint model (5). Thus, adding features corresponding to a sentence representation learned for the event prediction task has a very positive impact.

If we analyze the triggers returned by the baseline model, *CNN without any external features, our implementation* (1), and this new model with sentence embeddings (6), we notice that from a total of true triggers of 424, (1) returns 366 triggers from which 267 are correct and (6) returns only 276 from which 244 are correct. It explains why

Table 4.2 – Performance of the state-of-the-art neural-based systems for event detection on the blind test data

Number	Approaches	Precision	Recall	F1
1	CNN without any external features, our implementation	72.9	62.9	67.5
2	CNN without any external features [Nguyen and Grishman, 2015a]	71.9	63.8	67.6
3	CNN augmented with entity types [Nguyen and Grishman, 2015a]	71.8	66.4	69.0
4	Dynamic multi-pooling CNN in [Chen et al., 2015b]	75.6	63.6	69.1
5	Joint RNN in [Nguyen et al., 2016a]	66.0	73.0	69.3
6	CNN with sentence embeddings	**88.3**	58.9	**70.7**
7	Non-Consecutive CNN in [Nguyen et al., 2016b]	–	–	71.3
8	Hybrid neural network in [Feng et al., 2016]	84.6	64.9	73.4

the precision of (6), equal to 88.3, is the highest among all the neural-based models of Table 4.2. More qualitatively, from the true triggers returned by (6), the true triggers *leaving, War, demonstrate, discussions, launched, bloodshed, former, purge, jailed, launch, targeted, correspondence* are not detected by (1). Moreover, there are no true triggers correctly detected by (1) and not detected by (6), which means that even though the number of returned triggers returned is small, most of them are correct.

We can also remark that all models, with the exception of the *Joint RNN* (5) in [Nguyen et al., 2016a], tend to favor precision compared to recall, and our model gets the best one. At the same time, all models tend to suffer from an imbalance between precision and recall, which may be due to the unbalanced nature of the task, with many more words that are not event triggers than words that are event triggers. The Joint RNN seems to extract more triggers of lower quality, giving a higher cost to false positives and favoring in this way recall over precision, while our model has a slightly better performance by extracting smaller amounts of more accurate triggers, which proves that the *CNN with sentence embeddings* benefits

from its more global information. In comparison with *Joint RNN*, we are not using such a complex set of features (dependency-parse trees, information about entities). The *Non-Consecutive CNN* (7) of [Nguyen et al., 2016b] is another way to tackle the problem of the local nature of n-grams in CNN models by introducing non-consecutive n-grams, which are expected to help to capture the representation of all the arguments that are present in a sentence. In terms of comparison with our work, it should be noted that we do not use gold entity mentions and types, which proved to add 2 points to the F1 score of the baseline CNN. The joint RNN in [Nguyen et al., 2016a] (5) and the *Hybrid neural network* (8) in [Feng et al., 2016] use also a context representation for enriching the trigger candidate representation.

The model of [Feng et al., 2016] is more particularly interesting in terms of comparison with our model since it associates a bidirectional LSTM with a CNN. More precisely, the bidirectional LSTM is used for producing a representation of trigger candidates taking into account the preceding and following words in the sentence while the CNN is quite analogous to our baseline model and focuses on the detection of the most discriminant n-grams in a window surrounding trigger candidates. One can notice that there is an imbalance between precision and recall for both our system and the *Hybrid neural network*, which may denote the fact that encoding context at the sentence level benefits to the local features extracted by a CNN and leads to more precise predictions. Using the same sentence representation for each candidate trigger as we do seems to accentuate this trend towards high precision. Moreover, this option has a cost advantage both for training and application in comparison with [Feng et al., 2016] but at the expense of overall performance.

After having examined how to integrate information about the sentence context of triggers, in the next section, we will exploit the internal structure of words in order to include morphological and shape information in the features and will analyze the effects of this addition accordingly.

4.3 Exploiting the internal structure of words for Event Detection

So far, all state-of-the-art models presented are based mainly on word embeddings and the relative position of a possible trigger to the other words in a sequence. Some researchers studied the application of Convolutional Neural Networks (CNNs) to characters. These include using character-level n-grams with linear classifiers [Kanaris et al., 2007]. [Santos and Zadrozny, 2014] learn character-level embeddings, joins them with pre-trained word embeddings, and use a CNN for Part-of-Speech (POS) tagging. The same model was used for improving the performance of a named entity recognition (NER) system, with character-level embeddings, in [Santos and Guimaraes, 2015]. Character models have been used with success in several contexts, mostly in sequential tasks (POS, NER), for tackling the problem of the impossibility to have pre-trained embeddings for all words. [Zhang et al., 2015, Zhang and LeCun, 2015] explore the use of CNNs to learn directly from characters, without the need for any pre-trained embeddings. Notably, the authors use a relatively deep network and apply it to sentiment analysis and text classification tasks. [Kim et al., 2016] explore the application of character-level convolutions to language modeling, using the output of the character-level CNN as the input to an LSTM at each time step. The same model is easily applied to various languages since it does not rely on words.

In the context of event extraction, we also face the problem of missing word embeddings. In the ACE corpus, 14.8% of the words are not part of the pre-trained embeddings for the *Word2vec Google News* embeddings [Mikolov et al., 2013a], 1.5% for the *GloVe* embeddings [Pennington et al., 2014] and 4.5% for the fastText embeddings [Joulin et al., 2016b]). So, one possible advantage of character models might be the open vocabulary, due to the ability to deal with abnormal character combinations and misspellings. The ACE corpus comprises different types of dis-

course, professional news articles, and noisy forum discussions. Thus, there is an important possibility of the existence of misspelled or custom words.

We also consider that, while word-level embeddings are meant to capture syntactic and semantic information, character-level embeddings capture morphological and shape information. As an example, if we take a new word not present in the training data, *torturing*, but present in the validation data, given its root and suffix (i.e. *tortur-ing*), it is natural to assume that it is a variant of *torture* and the latter probably represents the same type of event, that being **Life.Injure**. If we take another example, a misspelled word, *travelling* from the validation data and *traveling*, the correct variant, present in the training data, with their suffixes *travel-l-ing* and *travel-ing*, we can conclude that probably *traveling* and its misspelled word might represent the same type of event, **Movement.Transport**.

Hence we decided to test if the use of character embeddings in the context of event extraction could be interesting. We study the effect of the character-level features on the extraction and classification of triggers by experimenting with word and character embeddings separately. We then propose different architectures for incorporating character-level features. We also propose some sort of data augmentation dedicated to increase the presence of these variants in the training corpus and apply our model on the two training datasets.

Our goal is to confirm the hypothesis that character information poses a valuable source of information for the event detection task.

4.3.1 Character-level CNN

We introduce the character-level CNN that generates a semantically-rich representation of a sequence of characters. The inputs are character embeddings that can essentially represent words that are not in the pre-trained embeddings vocabularies and deals with misspellings. Thus, the words not known in the sequence of words

can be taken into account more accurately with this level of representation. The character-level CNN basically applies a convolutional layer with a set of feature maps (filters) and a max-over-time over the concatenated character embeddings. See the top model in Figure 4.6.

For each trigger candidate $x^{(i)}$, we associate it a context window. As usual, in order to consider a limited sized context, longer sentences are trimmed and shorter ones are padded with a special token. For every word in the context, we generate the character embeddings and concatenate them. Given the character sequence $c = \{c_1, \ldots, c_n\}$ of a word w, we first transform each character c_i to its corresponding d-dimensional character embedding x by look-up in a char embedding table (drawn from a uniform distribution).

The obtained matrix representation C is then passed through a convolution layer. In the convolution layer, we have a set of feature maps (filters) $\{f_0 f_1, ..., f_n\}$ for the convolution operation. Each feature map f_i corresponds to some filter size k. A max-over-time pooling operation [Collobert et al., 2011] is applied over every feature map and the maximum value is taken as the feature corresponding to this particular filter.

4.3.2 Model

We make use of the character-level features as following:

— we apply the CNN for Event Detection (ED) presented in Chapter 3 for every trigger candidate surrounded by a fixed-sized context window and obtain the concatenation of the convolutional filters with different widths;

— we apply the character-level CNN, as mentioned before, on the sequence of characters corresponding to the words in the window and we obtain the local context character-level features.

We perform a late fusion of the results of these two models that separately learned different characteristics of the candidate trigger. The baseline CNN [Nguyen and

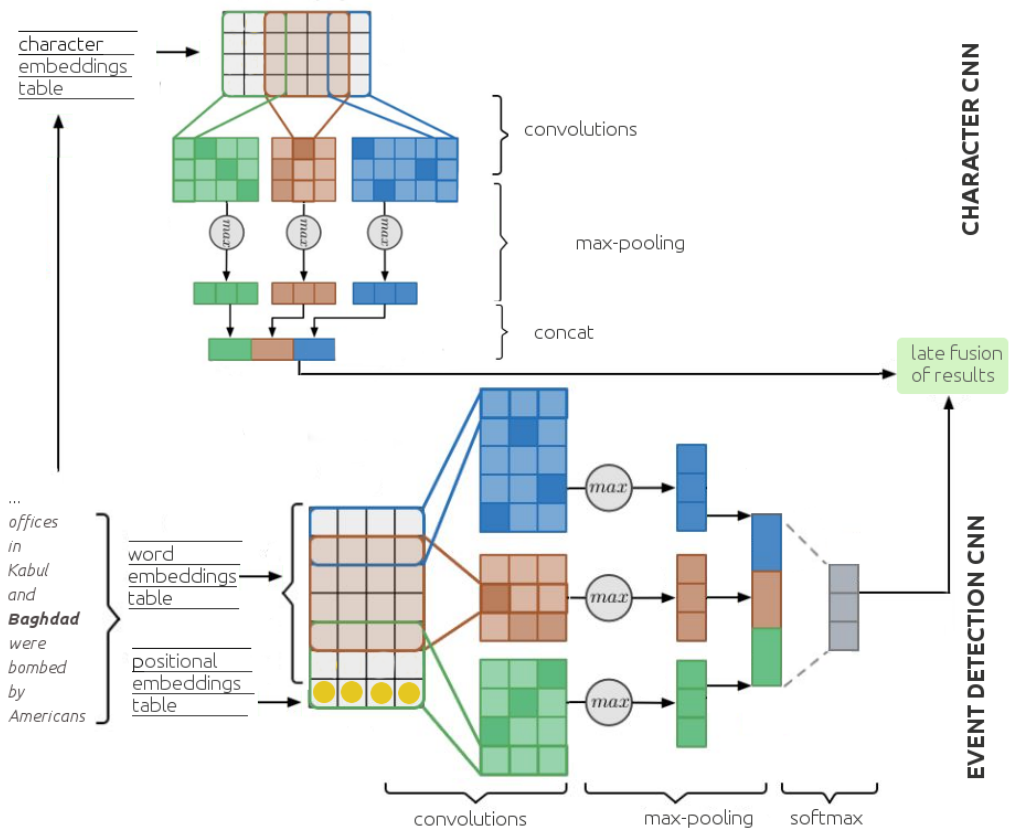


Figure 4.6 – Event detection model with character-level embeddings

Grishman, 2015a] combines word and positional embeddings that can capture syntactic and semantic information, and of course, the relative positions of words to the candidate trigger. The character-level CNN learns more local features from character n-grams and can capture morphological information. The late fusion focuses on the individual strength of these two models. We explain the methodology and our reasons for choosing this kind of late fusion in Section 4.3.4.

4.3.3 Data augmentation

Another possibility for better representing morphological variants in the corpus consists in adding examples. Thus, we propose adopting a type of data augmentation that relies on the generation of morphological derivations and inflections of triggers. By morphological derivation, we mean the generation of other words from given words, often by adding a prefix or a suffix and by inflection generation, we mean

the modification of verbs, nouns and adjectives to account for different grammatical features such as tense or gender. The inflections are basically a bundle of morpho-syntactic features. We decided to augment the true triggers from the training set of ACE 2015 Dataset, e.g. *become, force, hiring, name, sell, conviction.* For each of them, we manually generate different types of variations as in Table 4.3:

Table 4.3 – Examples of variants of true triggers

Triggers	**Event Type**	**Variants**
become	Personnel.Start-Position	became, becoming, becomes
force	Personnel.Start-Position & Conflict.Attack	forces, forced, forcing
hiring	Personnel.Start-Position	hires, hire, hired
name	Personnel.Nominate	names, named, naming
sell	Transaction.Transfer-Ownership	sells, sold, selling
conviction	Justice.Convict	convictions, convict, convicts, convicted, convicting

For every true trigger in the train set, we generate a context window as explained in Chapter 3. We create new instances of windows, by replacing the middle word (true trigger) with its morphological variants.

We motivate our choice by the fact that, in this way, we add morphological information to the models and this can increase the ability to distinguish different trigger forms.

In this work, we manually generated the inflections for the trigger words, which may not be a reliable technique when handling larger data. One interesting future approach can be the automatic generation of the variants. For example, the Special Interest Group on Computational Morphology and Phonology [1] (SIGMORPHON) and CoNLL have organized since 2016 two types of tasks in inflectional morphology that could be useful in that perspective: the first task implies the generation of inflected word forms based on labeled examples (e.g. the conversion of a lemma *become* to its present participle, *becoming*). The second one is more difficult, where

1. More information can be found at `http://www.sigmorphon.org/conll2017/`

the systems must infer the morphological features from the sentential context. More exactly, the systems have to provide the correct form of a lemma in context.

Next, we experiment with these two ways of adding morphological information to the models: character-level embeddings, with or without data augmentation.

4.3.4 Results

Hyperparameters

We consider a maximum length of 1024 for a sequence of characters. For every word in the sequence of 31 words previously presented, we look up every character in a character embeddings table and concatenate them. If the sequence is smaller than 1024, we pad the end with a special token.

The sequence of character embeddings is passed through a convolutional layer. The convolutional layer applies a set of 300 filters for each size in the set $\{2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Using the max-over-time pooling operation over the character filters of the word, we extract a fixed-size feature vector for the sequence that could cover the missing information brought by word embeddings.

We train the two networks (baseline CNN for ED and character-level CNN for ED) in the same manner, using stochastic gradient descent with shuffled mini-batches and Adam update rule [Kingma and Ba, 2014]. We employ dropout with a drop probability of 0.5 before the *softmax* layer. We also apply a dropout of 0.3 to the embeddings (word and character). During the training, we update the embedding tables (i.e., word, positional and character embeddings) to achieve the optimal states. Finally, for training, we use the mini-batch size of 128.

Analysis

For observing the influence that the character-level features have on the event detection and also, the effect of data augmentation on the performance of the model, we test the two models, with and without data augmentation.

Table 4.4 – Performance of the baseline CNN with word and positional embeddings on the blind test data. [1]: results with data augmentation

Approaches	**Precision**	**Recall**	**F1**
Baseline CNN	72.9	62.9	**67.5**
Baseline CNN[1]	63.4	63.9	63.6

Table 4.5 – Performance of the CNN with character embeddings for event detection on the blind test data. [1]: results with data augmentation

Approaches	**Precision**	**Recall**	**F1**
Character-level CNN	72.0	45.5	55.7
Character-level CNN[1]	67.6	47.8	56.0

First, we can observe from Table 4.4 and Table 4.5 that data augmentation hurts the performance of the baseline CNN model, but not the performance of the character-level CNN. This might be due to the type of data augmentation we use, which modifies the internal structure of words. We assume that, in the case of the baseline CNN, which relies on pre-trained word embeddings already capturing morpho-syntactic and semantic properties of words, adding new windows with trigger's variants may not be realistic, in the sense that these windows may correspond to word sequences that are not likely to be found in the dataset. We note, however, that at the character level, where no pre-trained vectors have been used, the model is able to learn the meaning of the context of words.

One could notice that the recall of all *Character-level CNN*s is considerably lower compared to the *Baseline CNN*s. We opted in favor of a late fusion of results, being motivated by the fact that the recall can be increased by the baseline model and the

precision by the character-level model. More precisely, this late fusion is performed by a kind of voting method, implemented as follows:

— if a trigger was detected by *Baseline CNN* and *Character-level CNN*, we keep the Character-level CNN label;
— if a trigger was detected by *Baseline CNN* but not by *Character-level CNN*, we keep the Baseline CNN label;
— if a trigger was detected by *Character-level CNN* but not by *Baseline CNN*, we keep the *Character-level CNN* label.

Moreover, we compare our late fusion approach with a model where we concatenate the output of the baseline CNN with the character-level CNN output as an extra feature before prediction. The two models are trained at the same time, sharing parameters. We also compute these results with the augmented data. Table 4.6 shows the results.

Table 4.6 – Performance of the CNN with character embeddings for event detection on the blind test data using two fusion methods. [1]: results with data augmentation

Approaches	**Precision**	**Recall**	**F1**
Baseline CNN	72.9	62.9	67.5
CNN with character-level features - joint[1]	82.2	62.5	71.0
CNN with character-level features - joint	84.8	63.4	72.6
CNN with character-level features - late fusion	87.8	63.2	**73.5**
CNN with character-level features - late fusion[1]	78.8	74.0	**76.3**

From Table 4.6, we can first outline that adding character embeddings outperforms the word embedding baseline CNN for all the models. Second, if we now consider the different results according to the data augmentation and the fusion method:

— joint & late fusion (not augmented): there is not a considerable difference between the two (only 0.9);
— joint & late fusion (augmented): in the case of joint prediction, with the two models trained at the same time, we notice that the recall is comparable to our baseline CNN while it is much higher for the late fusion approach. We

assume that in the joint approach, the power of representation provided by the words and positions from the *Baseline CNN* is overtaking the influence of the character embeddings representing morphological properties.

We notice that our best model manages to balance recall and precision using data augmentation and a late fusion of results (*CNN with character-level features - late fusion*[1]). The advantage of this type of voting is the increase in recall by trusting the *baseline CNN* while keeping a good level of precision by giving priority to the *CNN with character-level features*. It is clear that data augmentation further improves the CNN with character-level features by sharing the same purpose, the addition of more internal information to the features and respectively, the model.

Table 4.7 – Performance of the state-of-the-art neural-based systems for event detection on the blind test data. [1]: results with data augmentation

Number	**Approaches**	**Precision**	**Recall**	**F1**
1	CNN without any external features, our implementation	72.9	62.9	**67.5**
2	CNN without any external [Nguyen and Grishman, 2015a]	71.9	63.8	67.6
3	CNN with sentence embeddings[1]	**88.9**	54.9	**67.9**
4	CNN augmented with entity types [Nguyen and Grishman, 2015a]	71.8	66.4	69.0
5	Dynamic multi-pooling CNN in [Chen et al., 2015b]	75.6	63.6	69.1
6	Joint RNN in [Nguyen et al., 2016a]	66.0	73.0	69.3
7	CNN with sentence embeddings	88.3	58.9	**70.7**
8	Non-Consecutive CNN in [Nguyen et al., 2016b]	–	–	71.3
9	Hybrid neural network in [Feng et al., 2016]	84.6	64.9	73.4
10	CNN with character-level features - late fusion	87.8	63.2	**73.5**
11	CNN with character-level features - late fusion and sentence embeddings[1]	80.7	68.3	**74.0**
12	CNN with character-level features - late fusion and sentence embeddings	85.2	66.7	**74.8**
13	CNN with character-level features - late fusion[1]	78.8	**74.0**	**76.3**

We now compare different versions of our model with character-level features in

Table 4.7 with the neural-based models that have results above the baseline result, i.e., the *CNN model without any external features* (1, 2) in [Nguyen and Grishman, 2015a], the *Dynamic multi-pooling CNN* (5) model [Chen et al., 2015b], the *Joint RNN* (6) in [Nguyen et al., 2016a], the *Non-consecutive CNN* (8) in [Nguyen et al., 2016b], and the *Hybrid neural network* model (9) proposed by [Feng et al., 2016].

Combining word and positional embeddings with the *CNN with character-level features*'s output in a late fusion of results resulted in the best performance (76.3% F1 on the blind test set), bringing a gain of 8.8 points in F1 over the *Baseline CNN* (1). This demonstrates that our convolutional approach to learning character-level embeddings can be successfully applied to event detection, with no dependence on handcrafted features or language and the late fusion of results gives importance to the character-level model predictions. One weak point of the proposed approach is the introduction by the character-level CNN of additional hyperparameters to be tuned. However, we argue that it is generally preferable to tune hyperparameters than to handcraft features.

In order to understand how data augmentation helps the character-level CNN, we checked the triggers retrieved by *CNN with character-level features*[1]. Without augmentation, the model returns 305 from 424 total true triggers while with data augmentation, it returns 349. 62 of the triggers returned by the augmented model are not returned by the other. Between them, we note that at least half of them correspond to inflectional or morphological variants of triggers. Table 4.8 presents some examples of correct and incorrect triggers that come from the list of generated variants and that are not in the training set as a true trigger.

Since the obvious next step is to combine the two models we proposed, the *CNN with sentence embeddings* model of Section 4.2 and the *CNN with character-level features* model, with and without data augmentation, we report these results (11, 12). A late fusion of results is applied in the same manner as described in the previous section, where the vote of the *Baseline CNN* model is replaced by the *CNN with*

Table 4.8 – Examples of correct (correctly predicted) variants of true triggers retrieved, not present in the train set

Inflections retrieved	**Correct/Incorrect**	**Existing triggers**
formed	✓	form, former
formerly	✘	form, former
steps	✓	step
deploy	✓	deployed, deploying
extradited	✓	extradition, extraditing
wiped	✓	wipe
ousting	✓	ouster
funded	✓	fund

sentence embeddings model. We observe that, while this replacement leads to higher results without data augmentation (model 12), it does not reach the performance of the *Baseline CNN* model with data augmentation (model 13), even if the data augmentation has a positive impact on results. An unbalance between precision and recall is observed, favoring precision. Since we can observe from Table 4.7 that the highest precisions are obtained for the *CNN with sentence embeddings* models (3, 7), we assume that in both cases (augmented and not augmented: 11, 12) during the late fusion, the *CNN with character-level features* over-improves the quality of results.

4.4 Conclusions

We proposed two neural-based architectures for Event Detection (ED): *CNN with character-level features* and *CNN with sentence embeddings*, as improvements for the baseline model that we chose based on a CNN [Nguyen and Grishman, 2015a]. We analyze the effect of the technique of data augmentation and a prediction voting system that we proposed. The gain in performance can be explained by the following reasons:

— once again, CNNs are a good choice for event detection, since they semanti-

cally can understand a sequence of data by extracting the most informative parts for the sequence of words.

— the need for adding global information (sentence-level) for every trigger candidate is proved by the *CNN architecture with sentence embeddings* by the high quality of results (high precision). Moreover, we can confirm the effectiveness of Bi-GRUs to learn effective feature representations for sentences;

— comparing with other neural-based methods, the architecture that makes use of character-level features has the advantage of representing misspelled, custom or morphological variants of words from the dataset. When applying the late fusion of results that favors the *Character-level CNN* in deciding the labels of the triggers retrieved by the *Baseline CNN*, we conclude that the character-level features help the model in bringing a significant increase in performance and allow to get state-of-the-art results;

— the proposed data augmentation technique that increases the training dataset with variants of words (inflections) creates a balance between precision and recall and the augmented model with character-level features brings the best performance in comparison with the state-of-the-art approaches.

Chapter 5

Argument role prediction

This chapter focuses on the problem of argument role prediction, i.e., identifying the corresponding arguments to a trigger with a specific event type. This task comes as the second building block in the sequential pipeline that has as the first step the trigger detection and classification. In theory, the two tasks are highly interdependent but, in practice, they are applied as two separate steps: trigger identification is first treated as an independent task and argument finding and attribute assignment are each dependent on the results of trigger identification.

One challenging problem is the reliance on the event detection task, extracting arguments being extremely dependent on the event mentions (trigger words). Another phenomenon that cannot be ignored and has been studied by previous works in event extraction is the case of multiple-event sentences, where arguments can be common to different events, and thus can be mistakenly attached to the wrong event type. [Chen et al., 2015b] considers this sentence as an example: *In Baghdad, a cameraman died when an American tank fired on the Palestine Hotel.* This sentence has two event mentions: *Life.Die* event (trigger word: *died*) and *Conflict.Attack* event (trigger word: *fired*). *cameraman* has the role of a *Victim* for the *Life.Die* event and a *Target* for the *Conflict.Attack* event. The authors try to overcome this problem by modifying the max-pooling operation applied after the convolution: a dynamic

multi-pooling layer to obtain a maximum value for each part of a sentence, which is split by event triggers and event arguments. Their motivation is that, by applying the dynamic multi-pooling, the model captures the most valuable information with regard to the change of the candidate words.

We propose to model the task of argument role prediction as a relation extraction problem, where we pair the trigger and one argument in a relation, similar to the model proposed by [Chen et al., 2015b]. In this chapter, we firstly remind the common approaches to the relation extraction task and we describe a CNN based model for our task [Liu et al., 2013, Zeng et al., 2014, Nguyen and Grishman, 2015b]. An analysis of results and hyperparameters follows, proving that though not requiring complicated feature engineering, our approach outperforms the state-of-the-art feature-based methods extensively relying on the other supervised modules and manual resources for features. All the experiments are presented considering gold and predicted trigger words.

5.1 What are event arguments?

In the ACE 2005 dataset, an event argument is defined as an entity mention, a temporal expression or a value (e.g. *Crime*, *Sentence*, *Job-Title*) that is involved in an event (as participants or attributes with a specific role in an event mention). An event argument has a type and a role. For example, in a *Conflict.Attack* event type, one event argument can be an *Attacker* with three possible types: PER, ORG, GPE (Person, Organization, Geo-political Entity). The entity types detection has close ties to the named entity recognition (NER), work that is beyond the purposes of this thesis.

An entity mention (or as we will be referring to in the next sections, *an argument*) is a snippet of text that refers to something of an appropriate *role*.

Examples of argument roles are: *Attacker*, *Target* for an event of type *Conflict.Attack*.

All events can also have a *Time* and *Place* argument.

For example, for this sentence:

There was the free press in Qatar, Al Jazeera but its' offices in Kabul and Baghdad were bombed by Americans.

An argument detection and classification system should give as an output, for each argument, the following values:

— the event type: *Conflict.Attack*

— the argument role: *Attacker*

— the text snippet associated to the argument: *Americans*

One of the main challenges in evaluating such a system is that there are types of arguments that are common to many types of events. *Place* and *Entity* are common to most of the event types.

Identifying event arguments is a pair classification task. Each event mention is paired with each of the entity/timex/value mentions occurring in the same sentence to form a single classification instance. There are 36 classes in total: 35 role types and also an *Other* class. Again, the distribution of classes is skewed, though not as heavily as for the anchor task, with around 70% *Other* instances. The *Other* class is represented by the heads of all the other nouns, proper nouns or phrasal nouns that are not in relation with the trigger. One additional consideration is that no single event type allows arguments of all 36 possible roles, each event type has its own set of allowable roles.

Following previous work [Ji et al., 2008, Liao and Grishman, 2010, Hong et al., 2011, Nguyen and Grishman, 2015a, Chen et al., 2015b], an argument is correctly identified if its event subtype and offsets match those of any of the reference arguments. We consider the heads of all the nouns, proper nouns, and noun phrases and we consider the new offsets for the evaluation.

Finally, we use the same test set and the same training set as in previous work and

we evaluate the overall performance of the system with Precision (P), Recall (R) and F-measure (F1).

5.2 Links with Relation Extraction

In supervised approaches, the Relation Extraction (RE) and classification task specifically refers to the classification of an entity pair to a set of known relation types, using sentences containing mentions of the entity pair. The relation extraction task refers to predicting whether a given document contains a relation or not for the pair, modeled as a binary classification. Relation classification refers to predicting which relation class, coming from a given ontology, that document points to, given that it contains a relation (modeled as a multi-class classification problem). The two tasks can be combined by making a multi-class classification problem with an extra *no relation* class, thus, creating a tremendously unbalanced dataset where the number of the non-relation examples far exceeds the others, making relation extraction more challenging.

[Liu et al., 2013, Zeng et al., 2014, Nguyen and Grishman, 2015b] created similar models for relation extraction based on a Convolutional Neural Network (CNN) approach. [Liu et al., 2013] built a model with the inclusion of lexical features (Part-of-Speech (POS), entity types). With the motivation that the previous models that use word embeddings as input neglect semantic meaning among words, the authors assign a single vector to each synonym class rather than giving every individual word a vector. However, the model fails to exploit the real representational power of word embeddings.

Similarly to the previous model, [Zeng et al., 2014] used a CNN for encoding the sentence level features. But unlike [Liu et al., 2013], they used pre-trained word embeddings. The paper was the first work that used positional embeddings that we used for event detection task in Chapter 3.3, which were adopted as standard

in all subsequent RE and EE neural-based models. The sentence-level features are obtained through a non-linear transformation (activation function) applied on a max-pooled CNN. To replace the discrete features used by previous models, which were depending strongly on the results of existing NLP tools, they propose the lexical-level features that consist in the word embeddings of the targeted candidate entities and their context tokens. The lexical and sentence-level features are then concatenated to form a final feature vector and a *softmax* is applied to compute the confidence of each relation. One important contribution of this model was the use of a max-pooling layer after the convolutional layer.

The model proposed by [Nguyen and Grishman, 2015b] gets rid completely of external lexical features to enrich the representation of the input sentence and lets the CNN learn the required features itself. Their architecture is similar to [Zeng et al., 2014], consisting of the sentence-level word embeddings and positional embeddings followed by convolution and max-pooling. The positional feature is the combination of the relative distances of the current word to both entities of the relation. Additionally, they also incorporate convolutional kernels of varying sizes to capture wider ranges of n-gram features.

In Event Extraction (EE), this model became the base model for the argument role prediction neural-based pipelined approaches. For example, the authors of [Chen et al., 2015b] target one challenge of EE, that is to say, the fact that one argument can be common to multiple events, in the same sentence. After the prediction of the trigger word, the arguments can be mistakenly attached to the wrong event type. They try to overcome this problem by implementing a max-pooling CNN applied to the word embeddings of a sentence, to obtain, what they call, sentence-level features. As in [Zeng et al., 2014], they use the lexical-level features. They select the word embeddings of candidate words (candidate trigger, candidate argument) and the context tokens (left and right tokens of the candidate words) and concatenate them. This is injected in the main CNN, before prediction.

We will present in the next section a CNN based model for argument role labeling, that uses the minimum required features: word and positional embeddings, to analyze the dependency of arguments to the event detection task.

5.3 CNN model for argument role prediction

We base our model on a max-pooled CNN with word and positional embeddings as main features, which is the main architecture of the [Chen et al., 2015b, Nguyen and Grishman, 2015a, Nguyen and Grishman, 2015b]. The input of the model for argument role prediction consists of sentences marked with a trigger word, its type, and one argument of interest. We compute the maximal distance, in the training set, between the trigger and an argument linked by a relation (labeled with the event subtype) and choose an input width greater than this distance. We ensure that every input has this length by trimming longer sentences and padding shorter sentences with a special token.

Given a document, we first generate a set of argument candidates $\mathcal{T}$, considering only heads of the nouns, proper nouns and phrasal nouns as possible candidates. For each argument candidate $x^{(a)} \in \mathcal{T}$, we associate it with all the words in the sentence as a context, thus an argument candidate is represented as $x = [x^{(1)}, x^{(t)} \ldots, x^{(a)}, \ldots, x^{(n)}]$, where n is the length of the sentence and $x^{(t)}$ denotes the trigger and $x^{(a)}$ denotes the argument candidate. Each candidate $x^{(a)}$ or context word $x^{(i)}$ has as features the word itself, the relative distance of the token to the trigger $x^{(t)}$, the relative distance to the argument candidate $x^{(a)}$ and the event type detected in the ED step. Note that the relative distances only range from $-n+1$ to $n-1$. The position embedding matrix D has size $(2n-1) \times m_d$ where m_d is a hyperparameter indicating the dimensionality of the positional embeddings). Each feature is mapped to a vector retrieved from the following embedding tables:

— Word Embedding Table (initialized or not by some pre-trained word embed-

dings)

— Positional Embedding Table: to embed the relative distances. Each distance value is associated with a d-dimensional vector. In practice, the table is initialized randomly.

— Event Embedding Table: each type of event (*Life, Movement etc.*) or subtype (*Conflict.Attack, Life.Marriage, Life.Die etc.*) is associated with a d-dimensional vector. As in the case of positional embeddings, the table is initialized randomly. In this work, we experiment with both possibilities.

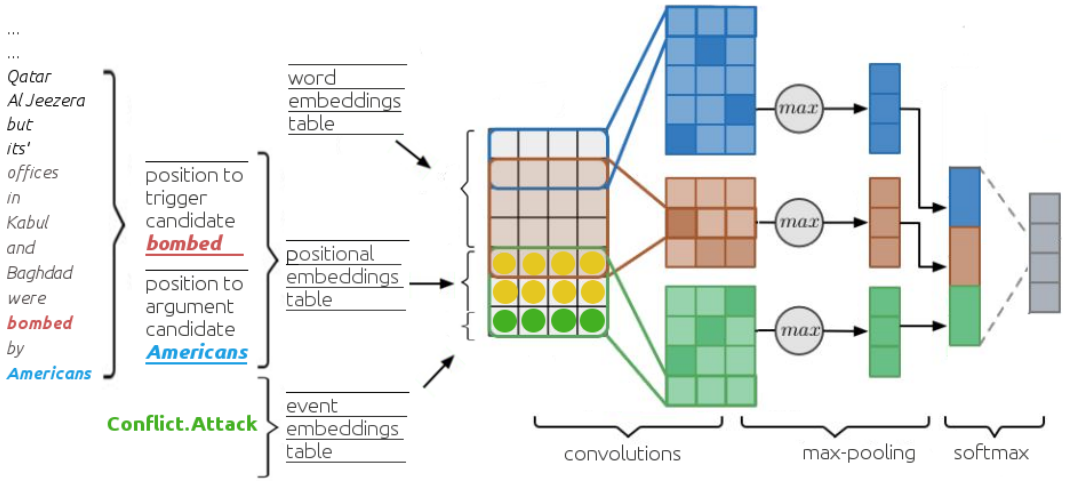


Figure 5.1 – CNN model for argument role prediction

The word embeddings w_i, the positional embeddings $d_i^{(1)}$, $d_i^{(2)}$ and the event embedding e_i are concatenated into a single vector $x_i = [w_i, d_i^{(1)}, d_i^{(2)}, e_i]$ to represent the word x_i. As a result, the original sentence x can now be viewed as a matrix X of size $(m_w + 2m_d + m_e) \times n$ where m_e is the dimensionality of the event embedding vectors, m_w the size of the word embedding and m_d, the size of the positional embedding.

The matrix representation X is then passed through a convolution layer. In the convolution layer, we have a set of feature maps (filters) $\{f_0 f_1, ..., f_n\}$ for the convolution operation. A max-over-time pooling operation [Collobert et al., 2011] is applied over every feature map and the maximum value is taken as the feature corresponding to this particular filter [Kim, 2014, Kalchbrenner et al., 2014]. These features form the penultimate layer and are passed to a fully connected *softmax*

layer whose output is the probability distribution over labels.

5.4 The influence of Event Detection on arguments

Since we formulated the event extraction as a two-stage, multi-class classification via two neural networks models with automatically learned features, we present our choices for these steps. In the first stage, event detection, we use the CNN with character-level features presented in Chapter 4, Section 4.3 (model 13 in Table 4.7), to classify each word of a sentence to identify trigger words. If one sentence has triggers, the second stage is conducted, which applies the CNN presented in the previous section, to assign arguments to triggers and assign the roles to the arguments.

We formalize the task as follows. Let $W = [w_1, w_2, \dots w_n]$ be a sentence where n is the sentence length and w_i is the i-th token.

For every sentence W, we check every w_i to find the trigger words. If w_i is a trigger word for some event of interest, we then need to predict the roles of every argument candidate. For every trigger detected and classified, we use the predicted event subtype and classify every argument candidate from the sentence (filtered by Part-of-Speech (POS): the heads of all the noun or proper noun chunks). Thus, for every pair (detected trigger, candidate argument), the argument role is predicted. The first N possible roles are kept with their probability. From these roles, only those that can be part of the subset roles possible for the detected event subtype and the one who has a maximum probability is considered as a result. If there are no possible roles, then the candidate argument is categorized as an *Other*.

For example, if we take this sentence: *There was the free press in Qatar, Al Jazeera but its' offices in Kabul and Baghdad were bombed by Americans.*

Suppose *bombed* has been detected and classified as *Conflict.Attack*. The argument candidates are: *press, Qatar, Al Jazeera, offices, Kabul, Baghdad*, and *Americans*.

For every pair (*bombed*, *Baghdad*), (*bombed*, *Americans*) etc., we classify the argument candidates. If for *Americans* two roles are detected: *Conflict.Defendant* and *Attack.Attacker*, with different probabilities, since the event subtype corresponds to the trigger *Conflict.Attack*, we consider the argument role with the greater probability. If, in case of one predicted role is *Die.Victim*, which corresponds to the *Life.Die* event subtype, then we keep the other one that corresponds to *Conflict.Attack* event subtype subset of argument roles.

5.5 Results

Hyperparameters

The parameters used for the CNN model for argument detection/classification are depicted as follows. The filter sizes used in the experiments are in the set $\{2, 3, 4, 5\}$ to generate feature maps and 300 feature maps are used for each filter size in this set as in [Nguyen and Grishman, 2015a].

The context is the whole sentence with a maximum size of 153 tokens (the length of the longest sentence) and shorter sentences are padded with a special token. The dimensionality for the positional embeddings is set to 50 [Nguyen and Grishman, 2015a] and 100 for event embeddings. We chose the event subtype due to the results of the experiments where we check which one from the event type or subtype is more suited for this task. These embeddings are drawn from a Gaussian distribution [Glorot et al., 2011]. The size of the mini-batch is set to 128 and we employed the pre-trained *Word2vec* [Mikolov et al., 2013a] word embeddings trained on Google News.

We use as regularization the early stopping [Prechelt, 1998], with a patience of 2 epochs, i.e. the number of epochs to wait before stopping the training if no progress on the validation set. Training is done via stochastic gradient descent with shuffled

mini-batches and the *Adam* update rule with a learning rate of $1e^{-3}$. As in previous experiments, the weights of the words, positional and event embedding tables are updated during the training of the model.

Analysis

In order to evaluate the effectiveness of the event embeddings, Table 5.1 reports the performance of the proposed CNN on the blind test set when the event subtype embeddings are either included or excluded from the systems. We make a difference between event type and subtype embeddings and compare them also.

Table 5.1 – Argument classification, results with gold triggers (gold relative distances to the trigger words) and event type/subtype, on the blind test set

Approaches	**Precision**	**Recall**	**F1**
− event embeddings	57.37	47.41	51.92
+ event type embeddings	58.35	49.34	53.46
+ event subtype embeddings	60.40	50.42	54.96

We always use positional embeddings in these scenarios. The reported results are obtained using gold triggers and event type/subtype. There is a 3 points performance gain with the event subtype embeddings over the model with only word and positional embeddings. Therefore this feature will be considered for the next experiments in the form *EventType.EventSubtype* (e.g. *Conflict.Attack*).

The Event Detection (ED) step predicts the *EventType.EventSubtype* and provides the position of the trigger word in the sentence. In order to evaluate the influence of the importance of the detected positions of the triggers (and thus, the importance of the positional embeddings) towards argument role classification, we report in Table 5.2 the performance on the blind test set, without the *EventType.EventSubtype* of the event and with/without the relative distances to the trigger. Note that in the experiment without the relative distances to the trigger, we also consider the sentences where no trigger is present, thus the results do not depend at all on

the trigger detection stage. With the large margin of performance, it is very clear

Table 5.2 – Argument classification, results without *EventType.EventSubtype* information and with/without relative distances to the trigger on the blind test set

Approaches	**Precision**	**Recall**	**F1**
− trigger distances	22.03	6.03	9.47
+ trigger distances	57.37	47.41	51.92

from the Table 5.2 that the positional embeddings relative to the trigger are very important and also very useful for CNNs for completing the argument role prediction task.

In Table 5.3, we present the results for argument role classification with provided gold-standard triggers and with predicted triggers.

Table 5.3 – Performance of the CNN for argument classification on the blind test data, with gold and predicted triggers

Approaches	**Precision**	**Recall**	**F1**
Our CNN with gold triggers	60.40	50.42	54.96
Our CNN with predicted triggers	47.46	40.08	43.46

Our CNN with predicted triggers has a considerable loss in F1, of 11.5%, which obviously means that arguments are dependent on a correct prediction of the trigger. Thus improving the model for trigger detection would probably lead to an increase in performance of the argument role prediction task.

We compare the proposed CNN with these state-of-the-art systems on the blind test set and compare with the following systems:

— The feature-based systems with local features (rich hand-designed feature sets) and global features such as cross-document, cross-sentence and cross-event information: the pipelined architecture in Cross-event in [Liao and Grishman, 2010] (3), both models proposed by [Li et al., 2013b] (2, 6) and [Yang and Mitchell, 2016] (5), the pipelined *MaxEnt*s (1) and the joint architecture (the structured perceptron model for joint beam search with local

Table 5.4 – Performance of the state-of-the-art systems for argument classification on the blind test data

Number	Approaches	Precision	Recall	F1
1	Our CNN with predicted triggers	47.4	40.0	43.4
2	MaxEnt with local features in [Li et al., 2013b]	65.4	33.1	43.9
3	Cross-event in [Liao and Grishman, 2010]	45.1	44.1	44.6
4	Cross-entity in [Hong et al., 2011]	51.6	45.5	48.3
5	Joint at document level in [Yang and Mitchell, 2016]	70.6	36.9	48.4
6	Joint beam search with local and global features in [Li et al., 2013b]	64.7	44.4	52.7
7	Dynamic multi-pooling CNN in [Chen et al., 2015b]	62.2	46.9	53.5
8	Joint RNN in [Nguyen et al., 2016a]	54.2	56.7	55.4

and global features in [Li et al., 2013b]) (6), the pipeline with cross-entity inference in [Hong et al., 2011] (4).

— The neural-based models: the dynamic multi-pooling CNN model [Chen et al., 2015b] (7) and the bidirectional joint Recurrent Neural Networks (RNNs) [Nguyen et al., 2016a] (8).

Our CNN with predicted triggers performs comparably with the MaxEnt with local features in [Li et al., 2013b] model (1) (a difference of 0.5%), which is also a joint model. In contrast with this system with discrete features, our system does not require any external features and only employs sentence-level information.

The joint RNN in [Nguyen et al., 2016a] (6) obtains the best results for argument role classification but heavily relies on lexical and syntactic features inherited from [Li et al., 2013b] (1, 4). It also has the highest recall because, besides the sentence-level features (sentence encoding), the model uses the dependencies among argument roles and among trigger subtypes, encoded by memory matrices. The trigger and the arguments are predicted simultaneously, in a joint approach, so that the model benefits of the memory matrices. For example, in the case of a 2-event sentence, an argument common to the two different events, can be easily attached to the right

event mention, knowing the previous detected triggers and arguments (e.g. one can notice that a *Victim* argument for a *Life.Die* event is often the *Target* argument for an *Conflict.Attack* event in the same sentence). In comparison with this system, the joint RNN [Nguyen et al., 2016a] (6), it is clear that a joint architecture augmented with discrete features can bring benefits.

5.6 Conclusions

We presented the argument detection and classification task as a use case of the Event Detection (ED) task. We considered an approach similar to the Relation Extraction (RE) methods, where we defined a relation between the trigger and every argument in an event mention. These are our conclusions regarding the results:

— The argument classification step uses the event subtypes predicted in the ED step and the position of the detected trigger (the relative distance of an argument candidate is computed in regards to this position). We saw a significant drop in performance between the cases where either gold triggers or the predicted triggers are used. Thus, an improvement of the trigger detection could facilitate the argument role prediction task. A correct detected trigger along with its position can clearly bring a gain in performance.

— Future work regarding this task may include the addition of character-level features and improvements to the model.

— Finally, the experiments proved that an argument role prediction system could benefit from a joint architecture, where previous detected arguments or triggers can influence the new predictions.

Chapter 6

Conclusions

This thesis has developed neural-based models for the Event Extraction task, more exactly for the sub-tasks Event Detection (ED) and Event Arguments Detection and Classification. Its main motivation is overcoming the difficulty of feature engineering for this type of tasks. The thesis has three main parts: firstly, the reproduction of a baseline neural-based system for ED and its in-depth study; secondly, the proposal of two new neural network architectures to take into account both the context at the sentence level and the morphological variants of words to improve the performance of event detection; thirdly, the argument identification and classification, presented as a use case for the event detection stage.

6.1 Summary of the contributions of the thesis

In Chapter 3, we re-implemented the baseline model for the Event Detection (ED) task based on a Convolutional Neural Network (CNN) proposed by [Nguyen and Grishman, 2015a]. We evaluated the model in a general setting and proposed a detailed analysis of its results according to its hyperparameters, especially its input word embeddings. Experiments showed that the choice of pre-trained embeddings and the embedding size have an important impact on the performance. The most

efficient embedding size we found was 300 and also, for ACE events, we proved that embeddings trained on large news datasets (large vocabulary) got the best performance.

Chapter 4 presents two new neural network systems that integrate and improve the baseline neural-based model for event detection. Firstly, we introduced a more global information for every trigger candidate, in order to help to recognize if a word is an event trigger or not and to disambiguate trigger words that can refer to several events. We proposed to build sentence representations relevant to the task of ED by training a Bi-GRU on a global classification task, to differentiate sentences that contain an event trigger and sentences that do not. This model is trained off-line and at the moment of trigger prediction (ED), we encode the sentence from which the context of the candidate trigger has been taken. This new representation is added before prediction in the baseline model. The high quality returned results are due to a high precision with an improvement of by 15.4 points over the chosen baseline model. The F1 is increased by 3.2 points. It compares favorably with the joint model in [Nguyen et al., 2016a] which also benefit from a sentence encoding done with a Bidirectional Gated Recurrent Units (Bi-GRU). We can thus confirm the effectiveness of RNNs to learn effective feature representations for sentences.

Secondly, we proposed two techniques for incorporating the inner information of words as new features. Firstly, we consider creating character-level features that can capture morphological and shape information of words, whereas word embeddings are effective to capture word-level syntactic and semantic information. For noticing the effect of the character-level features, we perform two types of experiments regarding the way the predictions are generated. Firstly, we jointly trained the baseline model and the character-level model together and predicted once, and secondly, we performed a late fusion of the results of the baseline CNN and the character-level model trained separately in order to benefit of the different learned characteristics of the candidate trigger.

The second technique is a type of data augmentation that relies on morphological derivation and inflection generation, which practically adds variants of known triggers in train, targeting the same impact as the character-level features. We noticed that data augmentation hurts the performance of the models that use word embeddings, but improves the performance of the models that use character embeddings, which we assume it is due to the type of data augmentation that modifies the internal structure of the word. At the character-level, where no pre-trained vectors have been used, the model is able to learn the meaning of the context of words.

The experiments show that data augmentation and a late fusion of results from the baseline CNN and the character-level features bring an increase of 8.8 points in F1 in comparison with the baseline model. As the late fusion of results focuses on the individual strength of the models, the precision and recall of this architecture are balanced and we obtain the highest results in comparison with the state-of-the-art results.

Naturally, we combined these models together: the model with sentence embeddings and the model with character-level features, in all the scenarios, with and without data augmentation, with and without late fusion of results. We did not get a better performance. An unbalance between precision and recall is observed, favoring excessively the precision, which probably results from the fact that both the model with sentence embeddings and the model with character-level features tend to over-improve the quality of results.

In Chapter 5, we presented the influence of ED on arguments detection and classification. We consider identifying event arguments as a relation extraction task, where the first argument is the event mention (trigger) detected in the first stage and the second is a candidate argument of the event mention. The event type predicted in the ED step is used as a feature for argument detection and classification, along with the position of the candidate argument to the predicted trigger. Our experiments show that the correct detection of the trigger is crucial, the predicted

event type being less important. At test phase, when predicting arguments with the already predicted trigger, we saw that the pipeline approach leads to error propagation. Nevertheless, the proposed approach performed slightly better than the feature-based systems and requires neither any elaborated hand-designed features nor features produced by existing supervised Natural Language Processing toolkits and resources.

In this thesis, we showed that neural-based models without any use of NLP tools allow encoding different kinds of features (local sentence structures, sentence-level context, morphological variations) while achieving very good performances for the ED task, and acceptable results for the argument detection and classification task. As a consequence, these models help to minimize the effort of feature engineering compared to the preceding approaches on the same dataset. We will make the code available on GitHub [1].

6.2 Future work

Following this dissertation, we envision future work for the Event Extraction (EE) task.

An important advantage of the proposals in this thesis is the non-reliance on features extracted from Natural Language Processing toolkits and rich hand-designed feature sets specific to this task.

Since we observed that the ED stage is crucial for argument detection and classification, we also envision the implementation of a joint model in order to capture the inter-dependencies between triggers and arguments.

Considering the architecture of the model, we consider experimenting replacing the Convolutional Neural Networks (CNNs) with Capsule Networks proposed by [Sabour

1. https://github.com/EmanuelaBoros/event-detection

et al., 2017]. They are meant to overcoming one of CNN major drawbacks, the fact that the max-pooling operation disregards positional information. This operation basically helps in creating positional invariance which can lead to triggering false positive for a sequence of words. For example, as explained in [Chen et al., 2015b], there can be cases in event extraction, where one sentence may contain two or more events, and these events may share arguments with different roles. A percentage of 27.3 of the sentences in the ACE dataset contain more than one trigger words. [Chen et al., 2015b] considers this sentence: *In Baghdad, a cameraman died when an American tank fired on the Palestine Hotel.* This sentence reflects two event mentions: *Life.Die* event (trigger word: *died*) and *Conflict.Attack* event (trigger word: *fired*). A max-pooling layer keeps the most important information in the sentence, thus, it can obtain *a cameraman died*, and disregard *an American tank fired*. This can lead to predicting the sentence as a *Conflict.Die* event, and totally ignoring the *Attack* event. Also, these events share an argument with different roles. *cameraman* has the role of a *Victim* for the *Life.Die* event and a *Target* for the *Conflict.Attack* event. In this case, a missed event mention leads to error propagation in the event argument roles classification stage. The authors of [Chen et al., 2015b] tried to overcome this problem by implementing the max-pooling operation as a multi-pooling convolutional network. One drawback of their solution is the fact that this model is highly dependent on the task (trigger and argument position).

Until now, we considered event detection as a word classification task. Candidate triggers in the form of multi-word expressions were stripped of the components that are not verbs (e.g. adverbs or prepositions). One possibility of eliminating the constraint of predicting every word is the employment of the BIO annotation schema to assign the labels to each word in the sentences. This technique has already been applied by [Nguyen and Grishman, 2015a] only for argument classification.

We also envision a work on one major drawback of the EE, that is to say, the requirement of large training datasets which require a significant work on manual

annotation and a considerable amount of expert domain knowledge and expertise. Solutions could include unsupervised or weakly-supervised techniques (e.g. bootstrapping, that has been one trend in augmenting data in case of the pattern-based methods). Unsupervised learning models have been applied successfully in other problems such as autoencoder models [Xu et al., 2017], Generative Adversarial Networks (GANs) [Goodfellow et al., 2014] etc. GANs are basically neural networks that learn to create synthetic data similar to a known input data, having as components a generator and a discriminator. A vast palette of GANs has been proposed towards data augmentation. At first, the main drawbacks of this GAN architecture were: one cannot control what data to generate, and cannot generate categorical data. As explained by the authors of [Goodfellow et al., 2014], *If you output the word penguin, you cannot change that to penguin + .001 on the next step, because there is no such word as penguin + .001. You have to go all the way from penguin to ostrich.* [2]. Solutions to these drawbacks were implemented in [Chen et al., 2016, Gulrajani et al., 2017, Donahue et al., 2016]. Very recently, a first attempt at applying a kind of GAN method to event extraction has been proposed in [Hong et al., 2018].

One other possible perspective can be the formulation of the ED task as a sequential learning problem to capture temporal dependencies in sequences, potentially of arbitrary length. These models usually rely on Recurrent Neural Networks (RNNs). One improvement could be the implementation of an *attention mechanism* for a better modeling of long-term dependencies [Luong et al., 2015, Yang et al., 2016, dos Santos et al., 2016]. Attention mechanisms allow for a more direct dependence between the states of the model at different points in time. Usually, a model that could benefit from such attention mechanism is of an *encoder-decoder* type. A Convolutional Neural Network (CNN) generates semantically-rich representations of text (the *encoder*) and an RNN (the *decoder*) is applied after, for predicting the next sequence. At this moment, the attention mechanism is focusing on the relevant

2. Information available at `https://www.reddit.com/r/MachineLearning/comments/40ldq6/generative_adversarial_networks_for_text/`

information generated by the CNN, so the decoder only uses specific information and finally makes the prediction. An attention mechanism allows the model to learn how to generate a context vector for each time step, and, thus, in the context of event extraction, it can be useful in the case of multiple event sentences.

Bibliography

[Abend and Rappoport, 2017] Abend, O. and Rappoport, A. (2017). The state of the art in semantic representation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 77–89.

[Ahn, 2006] Ahn, D. (2006). The stages of event extraction. In *Proceedings of the Workshop on Annotating and Reasoning about Time and Events*, pages 1–8. Association for Computational Linguistics.

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[Baker et al., 1998] Baker, C. F., Fillmore, C. J., and Lowe, J. B. (1998). The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics.

[Banko et al., 2009] Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., and Etzioni, O. (2009). *Open information extraction for the web*. University of Washington.

[Banko et al., 2008] Banko, M., Etzioni, O., and Center, T. (2008). The tradeoffs between open and traditional relation extraction. *Proceedings of ACL-08: HLT*, pages 28–36.

[Bansal et al., 2014] Bansal, M., Gimpel, K., and Livescu, K. (2014). Tailoring continuous word representations for dependency parsing. In *ACL (2)*, pages 809–815.

[Bengio, 2012] Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., and Vincent, P. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

[Bengio et al., 1993] Bengio, Y., Frasconi, P., and Simard, P. (1993). The problem of learning long-term dependencies in recurrent networks. In *Neural Networks, 1993., IEEE International Conference on*, pages 1183–1188. IEEE.

[Bojanowski et al., 2017] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

[Bordes et al., 2014] Bordes, A., Chopra, S., and Weston, J. (2014). Question answering with subgraph embeddings. *arXiv preprint arXiv:1406.3676*.

[Bottou, 2010] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

[Bronstein et al., 2015] Bronstein, O., Dagan, I., Li, Q., Ji, H., and Frank, A. (2015). Seed-based event trigger labeling: How far can event descriptions get us? In *ACL (2)*, pages 372–376.

[Brown et al., 1992] 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. 18:467–479.

[Chen et al., 2015a] Chen, W., Grangier, D., and Auli, M. (2015a). Strategies for training large vocabulary neural language models. *arXiv preprint arXiv:1512.04906*.

[Chen et al., 2016] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180.

[Chen et al., 2015b] Chen, Y., Xu, L., Liu, K., Zeng, D., and Zhao, J. (2015b). Event extraction via dynamic multi-pooling convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, volume 1, pages 167–176.

[Chieu et al., 2003] Chieu, H. L., Ng, H. T., and Lee, Y. K. (2003). Closing the gap: Learning-based information extraction rivaling knowledge-engineering methods. In *41st international Annual Meeting on Association for Computational Linguistics (ACL-2003)*, pages 216–223.

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[Collobert, 2011] Collobert, R. (2011). Deep learning for efficient discriminative parsing. In *14th International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*.

[Collobert and Weston, 2008] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *25th International Conference of Machine learning (ICML-08)*, pages 160–167. ACM.

[Collobert et al., 2011] Collobert, R., Weston, J., Battou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

[Conneau et al., 2017] Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*.

[Dahl et al., 2013] Dahl, G. E., Sainath, T. N., and Hinton, G. E. (2013). Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613. IEEE.

[Dai et al., 2016] Dai, J., He, K., and Sun, J. (2016). Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3150–3158.

[Deerwester et al., 1990] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391.

[Djemaa et al., 2016] Djemaa, M., Candito, M., Muller, P., and Vieu, L. (2016). Corpus annotation within the french framenet: a domain-by-domain methodology. In *Tenth International Conference on Language Resources and Evaluation (LREC 2016)*.

[Doddington et al., 2004] Doddington, G., Mitchell, A., Przybocki, M., Ramshaw, L., Strassel, S., and Weischedel, R. (2004). The automatic content extraction (ace) program–tasks, data, and evaluation. In *Proceedings of LREC*, volume 4, pages 837–840. Citeseer.

[Donahue et al., 2016] Donahue, J., Krähenbühl, P., and Darrell, T. (2016). Adversarial feature learning. *arXiv preprint arXiv:1605.09782*.

[Dos Santos and Gatti, 2014] Dos Santos, C. N. and Gatti, M. (2014). Deep convolutional neural networks for sentiment analysis of short texts. In *COLING*, pages 69–78.

[dos Santos et al., 2016] dos Santos, C. N., Tan, M., Xiang, B., and Zhou, B. (2016). Attentive pooling networks. *CoRR, abs/1602.03609*.

[Duan et al., 2017] Duan, S., He, R., and Zhao, W. (2017). Exploiting document level information to improve event detection via recurrent neural networks. In *Eighth International Joint Conference on Natural Language Processing (IJCNLP 2017)*, pages 352–361. Asian Federation of Natural Language Processing.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

[Ergun and Sert, 2016] Ergun, H. and Sert, M. (2016). Fusing deep convolutional networks for large scale visual concept classification. In *Multimedia Big Data (BigMM), 2016 IEEE Second International Conference on*, pages 210–213. IEEE.

[Etzioni et al., 2005] Etzioni, O., Cafarella, M., Downey, D., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D. S., and Yates, A. (2005). Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134.

[Everingham et al., 2010] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338.

[Faruqui et al., 2015] Faruqui, M., Dodge, J., Jauhar, S. K., Dyer, C., Hovy, E., and Smith, N. A. (2015). Retrofitting Word Vectors to Semantic Lexicons. In *2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT 2015)*, pages 1606–1615, Denver, Colorado.

[Feng et al., 2016] Feng, X., Huang, L., Tang, D., Ji, H., Qin, B., and Liu, T. (2016). A language-independent neural network for event detection. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 66–71.

[Feng et al., 2017] Feng, Y., Zhang, H., Hao, W., and Chen, G. (2017). Joint extraction of entities and relations using reinforcement learning and deep learning. *Computational intelligence and neuroscience*, 2017.

[Finkel et al., 2005] Finkel, J. R., Grenager, T., and Manning, C. (2005). Incorporating non-local information into information extraction systems by gibbs sampling. In *43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370.

[Fonseca and Rosa, 2013] Fonseca, E. R. and Rosa, J. L. G. (2013). A two-step convolutional neural network approach for semantic role labeling. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–7. IEEE.

[Freitag, 1998] Freitag, D. (1998). Information extraction from HTML: Application of a general machine learning approach. In *AAAI'98*, pages 517–523.

[Ghannay et al., 2016] Ghannay, S., Favre, B., Esteve, Y., and Camelin, N. (2016). Word embedding evaluation and combination. In *LREC*.

[Glorot et al., 2011] Glorot, X., Bordes, A., and Bengio, Y. (2011). Domain adaptation for large-scale sentiment classification: A deep learning approach. In *28th International Conference on Machine Learning (ICML-11)*, pages 513–520.

[Goldberg, 2016] Goldberg, Y. (2016). A primer on neural network models for natural language processing. *J. Artif. Intell. Res.(JAIR)*, 57:345–420.

[Goller and Kuchler, 1996] Goller, C. and Kuchler, A. (1996). Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pages 347–352. IEEE.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Graves, 2012] Graves, A. (2012). Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*.

[Grishman and Sundheim, 1996] Grishman, R. and Sundheim, B. (1996). Message understanding conference-6: A brief history. In *COLING 1996*, pages 466–471.

[Grishman et al., 2005] Grishman, R., Westbrook, D., and Meyers, A. (2005). Nyu's english ace 2005 system description. *ACE*, 5.

[Gulrajani et al., 2017] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*.

[Gupta and Sarawagi, 2009] Gupta, R. and Sarawagi, S. (2009). Domain adaptation of information extraction models. *ACM SIGMOD Record*, 37(4):35–40.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hill et al., 2016] Hill, F., Cho, K., and Korhonen, A. (2016). Learning distributed representations of sentences from unlabelled data. *arXiv preprint arXiv:1602.03483*.

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

[Hobbs et al., 1992] Hobbs, J. R., Appelt, D., Tyson, M., Bear, J., and Israel, D. (1992). SRI International: Description of the FASTUS system used for MUC-4. In *4th Conference on Message understanding*, pages 268–275.

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

[Hong et al., 2011] Hong, Y., Zhang, J., Ma, B., Yao, J., Zhou, G., and Zhu, Q. (2011). Using cross-entity inference to improve event extraction. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1127–1136. Association for Computational Linguistics.

[Hong et al., 2018] Hong, Y., Zhou, W., jingli, Zhou, G., and Zhu, Q. (2018). Self-regulation: Employing a generative adversarial network to improve event detection. In *56th Annual Meeting of the Association for Computational Linguistics (ACL 2018)*, pages 515–526. Association for Computational Linguistics.

[Honnibal and Johnson, 2015] Honnibal, M. and Johnson, M. (2015). An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378.

[Huang et al., 2012] Huang, E. H., Socher, R., Manning, C. D., and Ng, A. Y. (2012). Improving word representations via global context and multiple word prototypes. In *50th Annual Meeting of the Association for Computational Linguistics*, pages 873–882.

[Huang and Riloff, 2011] Huang, R. and Riloff, E. (2011). Peeling back the layers: Detecting event role fillers in secondary contexts. In *ACL 2011*, pages 1137–1147.

[Huang and Riloff, 2012a] Huang, R. and Riloff, E. (2012a). Bootstrapped training of event extraction classifiers. In *13th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2012)*, pages 286–295.

[Huang and Riloff, 2012b] Huang, R. and Riloff, E. (2012b). Modeling textual cohesion for event extraction. In *26th Conference on Artificial Intelligence (AAAI 2012)*.

[Huffman, 1952] Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

[Jaderberg et al., 2014] Jaderberg, M., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Synthetic data and artificial neural networks for natural scene text recognition. *arXiv preprint arXiv:1406.2227*.

[Jagannatha and Yu, 2016] Jagannatha, A. N. and Yu, H. (2016). Bidirectional rnn for medical event detection in electronic health records. In *2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 2016, pages 473–482. NIH Public Access.

[Ji et al., 2008] Ji, H., Grishman, R., et al. (2008). Refining event extraction through cross-document inference. In *ACL*, pages 254–262.

[Joulin et al., 2016a] Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. (2016a). Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

[Joulin et al., 2016b] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016b). Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

[Joulin et al., 2017] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2017). Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431. Association for Computational Linguistics.

[Kalchbrenner et al., 2014] Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

[Kanaris et al., 2007] Kanaris, I., Kanaris, K., Houvardas, I., and Stamatatos, E. (2007). Words versus character n-grams for anti-spam filtering. *International Journal on Artificial Intelligence Tools*, 16(06):1047–1067.

[Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

[Kim et al., 2016] Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2016). Character-aware neural language models. In *AAAI*, pages 2741–2749.

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kingsbury and Palmer, 2002] Kingsbury, P. and Palmer, M. (2002). From treebank to propbank. In *LREC*, pages 1989–1993. Citeseer.

[Kiros et al., 2015] Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Krupka et al., 1991] Krupka, G., Jacobs, P., Rau, L., and Iwańska, L. (1991). GE: Description of the NLToolset System as Used for MUC-3. In *3rd Conference on Message understanding*, pages 144–149.

[Lample et al., 2016] Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

[Lazaridou et al., 2013] Lazaridou, A., Marelli, M., Zamparelli, R., and Baroni, M. (2013). Compositional-ly derived representations of morphologically complex words in distributional semantics. In *ACL (1)*, pages 1517–1526.

[Le and Mikolov, 2014] Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196.

[Leacock and Chodorow, 1998] Leacock, C. and Chodorow, M. (1998). Combining local context and Wordnet similarity for word sense identification. In Fellbaum, C., editor, *WordNet: An electronic lexical database.*, pages 265–283. MIT Press.

[LeCun et al., 1990] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404.

[LeCun et al., 1998a] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[LeCun et al., 1998b] LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. (1998b). Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer.

[Levy and Goldberg, 2014] Levy, O. and Goldberg, Y. (2014). Dependency-based word embeddings. In *ACL (2)*, pages 302–308.

[Levy et al., 2015] Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

[Li et al., 2015] Li, J., Luong, M.-T., and Jurafsky, D. (2015). A hierarchical neural autoencoder for paragraphs and documents. *arXiv preprint arXiv:1506.01057*.

[Li et al., 2013a] Li, P., Zhu, Q., and Zhou, G. (2013a). Argument inference from relevant event mentions in chinese argument extraction. In *ACL (1)*, pages 1477–1487.

[Li, 2015] Li, Q. (2015). *Joint Information Extraction*. PhD thesis, Rensselaer Polytechnic Institute.

[Li et al., 2014] Li, Q., Ji, H., Hong, Y., and Li, S. (2014). Constructing information networks using one single model. In *EMNLP*, pages 1846–1851.

[Li et al., 2013b] Li, Q., Ji, H., and Huang, L. (2013b). Joint event extraction via structured prediction with global features. In *ACL (1)*, pages 73–82.

[Liao and Grishman, 2010] Liao, S. and Grishman, R. (2010). Using document level cross-event inference to improve event extraction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 789–797. Association for Computational Linguistics.

[Lin et al., 2003] Lin, W., Yangarber, R., and Grishman, R. (2003). Bootstrapped learning of semantic classes from positive and negative examples. In *Proceedings of ICML-2003 Workshop on The Continuum from Labeled to Unlabeled Data*, volume 1, page 21.

[Liu et al., 2013] Liu, C., Sun, W., Chao, W., and Che, W. (2013). Convolution neural network for relation extraction. In *International Conference on Advanced Data Mining and Applications*, pages 231–242. Springer.

[Luong et al., 2015] Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

[Luong et al., 2013] Luong, T., Socher, R., and Manning, C. D. (2013). Better word representations with recursive neural networks for morphology. In *CoNLL*, pages 104–113.

[Ma et al., 2015] Ma, M., Huang, L., Xiang, B., and Zhou, B. (2015). Dependency-based convolutional neural networks for sentence embedding. *arXiv preprint arXiv:1507.01839*.

[Ma et al., 2016] Ma, Y., Kim, J.-j., Bigot, B., and Khan, T. M. (2016). Feature-enriched word embeddings for named entity recognition in open-domain conversations. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 6055–6059. IEEE.

[Maas et al., 2011] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 142–150. Association for Computational Linguistics.

[Melamud et al., 2016] Melamud, O., McClosky, D., Patwardhan, S., and Bansal, M. (2016). The role of context types and dimensionality in learning word embeddings. *arXiv preprint arXiv:1601.00893*.

[Meyers et al., 2001] Meyers, A., Kosaka, M., Sekine, S., Grishman, R., and Zhao, S. (2001). Parsing and glarfing. *Proceedings of RANLP-2001*.

[Meyers et al., 2004] Meyers, A., Reeves, R., Macleod, C., Szekely, R., Zielinska, V., Young, B., and Grishman, R. (2004). The nombank project: An interim report.

In *Proceedings of the Workshop Frontiers in Corpus Annotation at HLT-NAACL 2004*.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. In *International Conference on Learning Representations (ICLR 20013), workshop track*.

[Mikolov et al., 2010] Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech*, volume 2, page 3.

[Mikolov et al., 2013b] Mikolov, T., Yih, W.-t., and Zweig, G. (2013b). Linguistic regularities in continuous space word representations. In *Hlt-naacl*, volume 13, pages 746–751.

[Miller et al., 1990] Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. J. (1990). Introduction to wordnet: An on-line lexical database. *International journal of lexicography*, 3(4):235–244.

[Mnih and Hinton, 2007] Mnih, A. and Hinton, G. (2007). Three new graphical models for statistical modelling. In *24th International Conference of Machine learning (ICML 2007)*, pages 641–648. ACM.

[Mooney, 1999] Mooney, R. (1999). Relational learning of pattern-match rules for information extraction. In *Sixteenth National Conference on Artificial Intelligence*, pages 328–334.

[Moschitti et al., 2003] Moschitti, A., Morarescu, P., and Harabagiu, S. (2003). Open-domain information extraction via automatic semantic labeling. In *Proceedings of FLAIRS*, pages 397–401.

[Mou et al., 2015] Mou, L., Peng, H., Li, G., Xu, Y., Zhang, L., and Jin, Z. (2015). Discriminative neural sentence modeling by tree-based convolution. *arXiv preprint arXiv:1504.01106*.

[Nam and Han, 2016] Nam, H. and Han, B. (2016). Learning multi-domain convolutional neural networks for visual tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4293–4302.

[Nguyen et al., 2016a] Nguyen, T. H., Cho, K., and Grishman, R. (2016a). Joint event extraction via recurrent neural networks. In *Proceedings of NAACL-HLT*, pages 300–309.

[Nguyen et al., 2016b] Nguyen, T. H., Fu, L., Cho, K., and Grishman, R. (2016b). A two-stage approach for extending event detection to new types via neural networks. *ACL 2016*, page 158.

[Nguyen and Grishman, 2015a] Nguyen, T. H. and Grishman, R. (2015a). Event detection and domain adaptation with convolutional neural networks. In *ACL (2)*, pages 365–371.

[Nguyen and Grishman, 2015b] Nguyen, T. H. and Grishman, R. (2015b). Relation extraction: Perspective from convolutional neural networks. In *Proceedings of NAACL-HLT*, pages 39–48.

[Nguyen and Grishman, 2016] Nguyen, T. H. and Grishman, R. (2016). Modeling skip-grams for event detection with convolutional neural networks. In *Proceedings of EMNLP*.

[Nguyen and Grishman, 2018] Nguyen, T. H. and Grishman, R. (2018). Graph convolutional networks with argument-aware pooling for event detection. In *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*.

[Nguyen et al., 2016c] Nguyen, T. H., Sil, A., Dinu, G., and Florian, R. (2016c). Toward mention detection robustness with recurrent neural networks. *arXiv preprint arXiv:1602.07749*.

[Palangi et al., 2016] Palangi, H., Deng, L., Shen, Y., Gao, J., He, X., Chen, J., Song, X., and Ward, R. (2016). Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 24(4):694–707.

[Patwardhan, 2010] Patwardhan, S. (2010). *Widening the field of view of information extraction through sentential event recognition*. PhD thesis, University of Utah.

[Patwardhan and Riloff, 2007] Patwardhan, S. and Riloff, E. (2007). Effective information extraction with semantic affinity patterns and relevant regions. In *2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2007)*, pages 717–727.

[Patwardhan and Riloff, 2009] Patwardhan, S. and Riloff, E. (2009). A unified model of phrasal and sentential evidence for information extraction. In *2009 Conference on Empirical Methods in Natural Language Processing (EMNLP 2009)*, pages 151–160.

[Peng and Dredze, 2016] Peng, N. and Dredze, M. (2016). Learning word segmentation representations to improve named entity recognition for chinese social media. *arXiv preprint arXiv:1603.00786*.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

[Pradet et al., 2014] Pradet, Q., Danlos, L., and de Chalendar, G. (2014). Adapting verbnet to french using existing resources. In *Ninth International Conference on Language Resources and Evaluation (LREC'14)*.

[Prechelt, 1998] Prechelt, L. (1998). Early stopping-but when? *Neural Networks: Tricks of the trade*, pages 553–553.

[Riloff, 1996a] Riloff, E. (1996a). Automatically generating extraction patterns from untagged text. In *AAAI'96*, pages 1044–1049.

[Riloff, 1996b] Riloff, E. (1996b). An empirical study of automated dictionary construction for information extraction in three domains. *Artificial intelligence*, 85(1):101–134.

[Rumelhart et al., 1985] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

[Rumelhart et al., 1988] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). *Learning representations by back-propagating errors.* MIT Press, Cambridge, MA, USA.

[Ruppenhofer et al., 2016] Ruppenhofer, J., Ellsworth, M., Petruck, M. R., Johnson, C. R., and Scheffczyk, J. (2016). *FrameNet II: Extended theory and practice.* Institut für Deutsche Sprache, Bibliothek.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

[Ryan, 2011] Ryan, R. J. (2011). *Groundtruth Budgeting: A technique for Weakly-Supervised Relation Extraction of Medical.* PhD thesis, MIT CSAIL.

[Sabour et al., 2017] Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3857–3867.

[Santos and Zadrozny, 2014] Santos, C. D. and Zadrozny, B. (2014). Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826.

[Santos and Guimaraes, 2015] Santos, C. N. d. and Guimaraes, V. (2015). Boosting named entity recognition with neural character embeddings. *arXiv preprint arXiv:1505.05008.*

[Schnabel et al., 2015] Schnabel, T., Labutov, I., Mimno, D., and Joachims, T. (2015). Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 298–307.

[Schuler, 2005] Schuler, K. K. (2005). Verbnet: A broad-coverage, comprehensive verb lexicon.

[Shaoul, 2010] Shaoul, C. (2010). The westbury lab wikipedia corpus. *Edmonton, AB: University of Alberta.*

[Shen et al., 2014a] Shen, Y., He, X., Gao, J., Deng, L., and Mesnil, G. (2014a). A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 101–110. ACM.

[Shen et al., 2014b] Shen, Y., He, X., Gao, J., Deng, L., and Mesnil, G. (2014b). Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 373–374. ACM.

[Socher et al., 2011] Socher, R., Pennington, J., Huang, E. H., Ng, A. Y., and Manning, C. D. (2011). Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Conference on Empirical Methods in Natural Language Processing*, pages 151–161.

[Socher et al., 2013] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

[Stevenson, 2006] Stevenson, M. (2006). Fact distribution in information extraction. *Language Resources and Evaluation*, 40(2):183–201.

[Stevenson and Greenwood, 2005] Stevenson, M. and Greenwood, M. A. (2005). A semantic approach to ie pattern induction. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 379–386. Association for Computational Linguistics.

[Sudo et al., 2003] Sudo, K., Sekine, S., and Grishman, R. (2003). An improved extraction pattern representation model for automatic ie pattern acquisition. In *41st Annual Meeting on Association for Computational Linguistics (ACL-03)*, pages 224–231.

[Surdeanu et al., 2003] Surdeanu, M., Harabagiu, S., Williams, J., and Aarseth, P. (2003). Using predicate-argument structures for information extraction. In *Pro-*

ceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1, pages 8–15. Association for Computational Linguistics.

[Surdeanu et al., 2006] Surdeanu, M., Turmo, J., and Ageno, A. (2006). A hybrid approach for the acquisition of information extraction patterns. In *EACL-2006 Workshop on Adaptive Text Extraction and Mining (ATEM 2006)*, pages 48–55.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[Tai et al., 2015] Tai, K. S., Socher, R., and Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

[Talukdar et al., 2008] Talukdar, P. P., Reisinger, J., Paşca, M., Ravichandran, D., Bhagat, R., and Pereira, F. (2008). Weakly-supervised acquisition of labeled class instances using graph random walks. In *Conference on Empirical Methods in Natural Language Processing*, pages 582–590.

[Tan et al., 2016] Tan, M., dos Santos, C. N., Xiang, B., and Zhou, B. (2016). Improved representation learning for question answer matching. In *ACL (1)*.

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

[Tolias et al., 2015] Tolias, G., Sicre, R., and Jégou, H. (2015). Particular object retrieval with integral max-pooling of cnn activations. *arXiv preprint arXiv:1511.05879*.

[Turian et al., 2010] Turian, J., Ratinov, L., and Bengio, Y. (2010). Word representations: a simple and general method for semi-supervised learning. In *48th international Annual Meeting on Association for Computational Linguistics (ACL 2010)*, pages 384–394.

[Turney and Pantel, 2010] Turney, P. D. and Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37:141–188.

[Wattarujeekrit et al., 2005] Wattarujeekrit, T. et al. (2005). *Exploring Semantic roles for Named Entity Recognition in the Molecular biology domain*. PhD thesis, Ph. D. diss., Department of Informatics, School of Multidisciplinary Sciences, The Graduate University for Advanced Studies.

[Werbos, 1990] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[Williams and Zipser, 1995] Williams, R. J. and Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, 1:433–486.

[Wu et al., 2015] Wu, Y., Xu, J., Jiang, M., Zhang, Y., and Xu, H. (2015). A study of neural word embeddings for named entity recognition in clinical text. In *AMIA Annual Symposium Proceedings*, volume 2015, page 1326. American Medical Informatics Association.

[Xu et al., 2017] Xu, W., Sun, H., Deng, C., and Tan, Y. (2017). Variational autoencoder for semi-supervised text classification. In *AAAI*, pages 3358–3364.

[Yang and Mitchell, 2016] Yang, B. and Mitchell, T. (2016). Joint extraction of events and entities within a document context. *arXiv preprint arXiv:1609.03632*.

[Yang et al., 2016] Yang, Z., Yang, D., Dyer, C., He, X., Smola, A. J., and Hovy, E. H. (2016). Hierarchical attention networks for document classification. In *HLT-NAACL*, pages 1480–1489.

[Yangarber et al., 2000] Yangarber, R., Grishman, R., Tapanainen, P., and Huttunen, S. (2000). Automatic acquisition of domain knowledge for information

extraction. In *18th Internation Conference on Computational Linguistics (COLING 2000)*, pages 940–946.

[Yih et al., 2014] Yih, S. W.-t., He, X., and Meek, C. (2014). Semantic parsing for single-relation question answering.

[Yin and Schütze, 2015] Yin, W. and Schütze, H. (2015). Convolutional neural network for paraphrase identification. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 901–911.

[Yin et al., 2016] Yin, W., Yu, M., Xiang, B., Zhou, B., and Schütze, H. (2016). Simple question answering by attentive convolutional neural network. *arXiv preprint arXiv:1606.03391*.

[Zaremba et al., 2014] Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

[Zeiler, 2012] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

[Zeng et al., 2014] Zeng, D., Liu, K., Lai, S., Zhou, G., Zhao, J., et al. (2014). Relation classification via convolutional deep neural network. In *COLING*, pages 2335–2344.

[Zeng et al., 2016] Zeng, W., Luo, W., Fidler, S., and Urtasun, R. (2016). Efficient summarization with read-again and copy mechanism. *arXiv preprint arXiv:1611.03382*.

[Zhang and LeCun, 2015] Zhang, X. and LeCun, Y. (2015). Text understanding from scratch. *arXiv preprint arXiv:1502.01710*.

[Zhang et al., 2015] Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657.

[Zhao et al., 2018] Zhao, Y., Jin, X., Wang, Y., and Cheng, X. (2018). Document embedding enhanced event detection with hierarchical and supervised attention.

In *56th Annual Meeting of the Association for Computational Linguistics (ACL 2018)*, pages 414–419. Association for Computational Linguistics.

[Zhu et al., 2015] Zhu, C., Qiu, X., Chen, X., and Huang, X. (2015). A re-ranking model for dependency parser with recursive convolutional neural network. *arXiv preprint arXiv:1505.05667*.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Méthodes Neuronales pour l'Extraction d'Événements

Mots clés : extraction d'information, extraction d'événements, réseaux de neurones, plongements de mots

Résumé : Du point de vue du traitement automatique des langues (TAL), l'extraction des événements dans les textes est la forme la plus complexe des processus d'extraction d'information, qui recouvrent de façon plus générale l'extraction des entités nommées et des relations qui les lient dans les textes. Le cas des événements est particulièrement ardu car un événement peut être assimilé à une relation n-aire ou à une configuration de relations. Par rapport aux relations ne faisant intervenir que deux entités, s'ajoute donc une dimension nouvelle obligeant à sortir bien souvent du cadre de la phrase, ce qui constitue une difficulté supplémentaire. En pratique, un événement est décrit par un déclencheur (le mot ou l'expression qui évoque l'événement) et un ensemble de participants à cet événement (c'est-à-dire des arguments ou des rôles) dont les valeurs sont des extraits de texte.
Alors que la recherche en extraction d'information a largement bénéficié des jeux de données étiquetés manuellement pour apprendre des modèles permettant l'analyse des textes, la disponibilité de ces ressources reste un problème important. En outre, de nombreuses approches en extraction d'information fondées sur l'apprentissage automatique reposent sur la possibilité d'extraire à partir des textes de larges ensembles de traits définis manuellement grâce à des outils de TAL élaborés. De ce fait, l'adaptation à un nouveau domaine constitue un défi supplémentaire.
Cette thèse présente plusieurs stratégies pour améliorer la performance d'un système d'extraction d'événements en utilisant des approches fondées sur les réseaux de neurones et en exploitant les propriétés morphologiques, syntaxiques et sémantiques des plongements de mots. Ceux-ci ont en effet l'avantage de ne pas nécessiter une modélisation a priori des connaissances du domaine et de générer automatiquement un ensemble de traits beaucoup plus vaste pour apprendre un modèle.
Nous avons proposé plus spécifiquement différents modèles d'apprentissage profond pour les deux sous-tâches liées à l'extraction d'événements: la détection d'événements et la détection d'arguments. La détection d'événements est considérée comme une sous-tâche importante de l'extraction d'événements dans la mesure où la détection d'arguments est très directement dépendante de son résultat. La détection d'événements consiste plus précisément à identifier des instances d'événements dans les textes et à les classer en types d'événements précis. Classiquement, un même événement peut apparaître sous la forme de différentes expressions et ces expressions peuvent elles-mêmes représenter des événements différents dans des contextes différents, d'où la difficulté de la tâche. La détection des arguments s'appuie sur la détection de l'expression considérée comme déclencheur de l'événement et assure la reconnaissance des participants de l'événement. Parmi les difficultés à prendre en compte, il faut noter qu'un argument peut être commun à plusieurs événements et qu'il ne s'identifie pas nécessairement à une entité nommée facilement reconnaissable.
En préalable à l'introduction de nos nouveaux modèles, nous commençons par présenter en détail le modèle de l'état de l'art qui en constitue la base. Des expériences approfondies sont menées sur l'utilisation de différents types de plongements de mots et sur l'influence des différents hyperparamètres du modèle en nous appuyant sur le cadre d'évaluation ACE 2005, standard d'évaluation pour cette tâche.
Nous proposons ensuite deux nouveaux modèles permettant d'améliorer un système de détection d'événements. L'un permet d'augmenter le contexte pris en compte lors de la prédiction d'une instance d'événement (déclencheur d'événement) en utilisant un contexte phrastique, tandis que l'autre exploite la structure interne des mots en profitant de connaissances morphologiques en apparence moins nécessaires mais dans les faits importantes. Nous proposons enfin de reconsidérer la détection des arguments comme une extraction de relation d'ordre supérieur et nous analysons la dépendance de cette détection vis-à-vis de la détection d'événements.

Title: Neural Methods for Event Extraction

Keywords: information extraction, event extraction, neural networks, word embeddings

Abstract: With the increasing amount of data and the exploding number data sources, the extraction of information about events, whether from the perspective of acquiring knowledge or from a more directly operational perspective, becomes a more and more obvious need. This extraction nevertheless comes up against a recurring difficulty: most of the information is present in documents in a textual form, thus unstructured and difficult to be grasped by the machine. From the point of view of Natural Language Processing (NLP), the extraction of events from texts is the most complex form of Information Extraction (IE) techniques, which more generally encompasses the extraction of named entities and relationships that bind them in the texts. The event extraction task can be represented as a complex combination of relations linked to a set of empirical observations from texts. Compared to relations involving only two entities, there is therefore a new dimension that often requires going beyond the scope of the sentence, which constitutes an additional difficulty. In practice, an event is described by a trigger (the word or phrase that evokes the event) and a set of participants in that event (that is, arguments or roles) whose values are text excerpts.

While IE research has benefited significantly from manually annotated datasets to learn patterns for text analysis, the availability of these resources remains a significant problem. These datasets are often obtained through the sustained efforts of research communities, potentially complemented by crowdsourcing. In addition, many machine learning-based IE approaches rely on the ability to extract large sets of manually defined features from text using sophisticated NLP tools. As a result, adaptation to a new domain is an additional challenge.

This thesis presents several strategies for improving the performance of an Event Extraction (EE) system using neural-based approaches exploiting morphological, syntactic, and semantic properties of word embeddings. These have the advantage of not requiring a priori modeling domain knowledge and automatically generate a much larger set of features to learn a model. More specifically, we proposed different deep learning models for two sub-tasks related to EE: event detection and argument detection and classification. Event Detection (ED) is considered an important subtask of event extraction since the detection of arguments is very directly dependent on its outcome. ED specifically involves identifying instances of events in texts and classifying them into specific event types. Classically, the same event may appear as different expressions and these expressions may themselves represent different events in different contexts, hence the difficulty of the task. The detection of the arguments is based on the detection of the expression considered as triggering the event and ensures the recognition of the participants of the event. Among the difficulties to take into account, it should be noted that an argument can be common to several events and that it does not necessarily identify with an easily recognizable named entity.

As a preliminary to the introduction of our proposed models, we begin by presenting in detail a state-of-the-art model which constitutes the baseline. In-depth experiments are conducted on the use of different types of word embeddings and the influence of the different hyperparameters of the model using the ACE 2005 evaluation framework, a standard evaluation for this task.

We then propose two new models to improve an event detection system. One allows increasing the context taken into account when predicting an event instance (event trigger) by using a sentential context, while the other exploits the internal structure of words by taking advantage of seemingly less obvious but essentially important morphological knowledge. We also reconsider the detection of arguments as a high-order relation extraction and we analyze the dependence of arguments on the ED task.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France