NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INGORMATICS AND TELECOMMUNICATION

PROGRAM OF POSTGRADUATE STUDIES

MASTER'S THESIS

Named Entity Recognition using Neural Networks

Vasiliki D. Moschou

Supervisor: **Manolis Koubarakis,** Professor

ATHENS

MARCH 2020

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αναγνώριση Ονομασμένων Οντοτήτων με χρήση Νευρωνικών Δικτύων

Βασιλική Δ. Μόσχου

Επιβλέπων: **Μανόλης Κουμπαράκης,** Καθηγητής

ΑΘΗΝΑ

ΜΑΡΤΙΟΣ 2020

MASTER'S THESIS

Named Entity Recognition using Neural Networks

Vasiliki D. Moschou
ID: M1507

SUPERVISORS: **Manolis Koubarakis,** Professor

EXAMINING COMMITTEE: **Manolis Koubarakis,** Professor
Dimitrios Gunopoulos, Professor

Examination Date: March 9, 2020

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αναγνώριση Ονομασμένων Οντοτήτων με χρήση Νευρωνικών Δικτύων

Βασιλική Δ. Μόσχου
A.M.: M1507

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Μανόλης Κουμπαράκης,** Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Μανόλης Κουμπαράκης,** Καθηγητής
Δημήτριος Γουνόπουλος, Καθηγητής

Ημερομηνία Εξέτασης: 9 Μαρτίου, 2020

ABSTRACT

With the evolution of *Artificial Intelligence* (AI), every aspect in everyday life keeps changing. Multiple industries like education, transportation, health, gaming, social life, space industry and much more keep adjusting their way of working using AI techniques that provide automated procedures, less human errors and more efficiency and accuracy to delivering good products and services. The data plays a very important role to help the machine learn from the past, and to understand better the present and the future.

The role of deep learning shows up in the need to make good use of text documents, journals, papers, and everything textual that can be found in the web (or not) and that is difficult to extract knowledge from it. Neural networks are applied via deep learning, and aim to recognize patterns by interpreting data through machine perception, data pre-processing, clustering and classification.

Named Entity Recognition (NER) is one of the main tasks required for information extraction which targets on recognizing entities from unstructured text, and translating them into pre-defined subcategories, such as Person, Location, Organization, and more.

The research that has been done around this domain is large enough to question and analyse what is the best technique or combination of techniques applied to extract the best result on NER. The goal of this thesis, is to investigate the existing progress in NLP, aiming at NER. Following publicly available documents and applications, we are exploring the differences among them, and we evaluate the extracted final results. We analyse multiple Natural Language Processing (NLP) techniques that have been implemented for NER shared tasks.

We will compare and estimate state-of-the-art architectures for Neural networks (LSTM, CNN, RNN, CRF) aiming at identifying the following entity types: Person, Location, Organization, Misc (Miscellaneous, general term). Additionally, we will study the importance of using or not, hand-crafted or lexical features. For a fair comparison, we will use the same dataset to train the models, as described in the actual papers. Each approach is evaluated using precision, recall and F1 score.

SUBJECT AREA: Natural Language Processing, Artificial Intelligence

KEYWORDS: Named Entity Recognition, Entity Generation,

Feature Extraction, Neural Networks, Deep Learning

ΠΕΡΙΛΗΨΗ

Με την εξέλιξη της Τεχνητής Νοημοσύνης (ΤΝ), κάθε πτυχή της καθημερινότητας αλλάζει συνεχώς. Πολλοί τομείς του επαγγελματικού κλάδου, προσαρμόζουν συνεχώς τον τρόπο εφαρμογής των υπηρεσιών και προϊόντων τους με χρήση τεχνικών ΤΝ επιτυγχάνοντας πιο αυτοματοποιημένες διαδικασίες, λιγότερα ανθρώπινα σφάλματα, και μεγαλύτερη αποτελεσματικότητα και ακρίβεια. Τέτοιες εφαρμογές υπάρχουν στην εκπαίδευση, στους τρόπους μεταφοράς και μετακίνησης, στην υγεία, στην ψυχαγωγία, αλλά και σε οποιοδήποτε εργαλείο που μπορεί να συμβάλλει στην κοινωνική ζωή και δικτύωση. Τα δεδομένα παίζουν πολύ σημαντικό ρόλο, βοηθώντας τον υπολογιστή να μάθει από το παρελθόν, και να κατανοήσει καλύτερα το παρόν και το μέλλον.

Ο ρόλος της Πολυεπίπεδης ή Βαθιάς Μάθησης εμφανίζεται μαζί με την ανάγκη της αξιοποίησης της γραπτής πληροφορίας και κειμένου, εγγράφων, περιοδικών, και οτιδήποτε γραπτού υπάρχει στο διαδίκτυο. Τα Νευρωνικά Δίκτυα εφαρμόζονται μέσω της βαθιάς μάθησης και αποσκοπούν στην αναγνώριση προτύπων, μέσα από προ-επεξεργασία, ομαδοποίηση, και ταξινόμηση των δεδομένων.

Η Αναγνώριση Ονομασμένων Οντοτήτων αποτελεί μια από τις κύριες μεθόδους που απαιτούνται για την εξαγωγή πληροφοριών, η οποία στοχεύει στον εντοπισμό οντοτήτων μέσα από μη δομημένο κείμενο, και στην μετατροπή της σχετικής πληροφορίας σε συγκεκριμένες προ-δηλωμένες κατηγορίες, όπως για παράδειγμα Άτομο, Τοποθεσία, Οργανισμός και άλλα.

Το εύρος της έρευνας και των αλγορίθμων που έχουν εφαρμοστεί στον συγκεκριμένο κλάδο έχει εξελιχθεί σε τέτοιο βαθμό, που μας δίνει τη δυνατότητα να μελετήσουμε σε βάθος την κάθε εφαρμογή, και να εξάγουμε το καλύτερο δυνατό αποτέλεσμα. Στόχος της διπλωματικής εργασίας, είναι η μελέτη της προόδου που έχει γίνει στην Επεξεργασία Φυσικής Γλώσσας με σκοπό την Αναγνώριση Ονομασμένων Οντοτήτων. Ακολουθώντας ευρέως διαδεδομένα δημοσιεύματα και εφαρμογές, επιχειρούμε να βρούμε την εγκυρότητα των εφαρμοσμένων μοντέλων, τις διαφορές μεταξύ τους, και πως αξιολογείται το τελικό αποτέλεσμα.

Μελετάμε τις τεχνικές της Επεξεργασίας της Φυσικής Γλώσσας, που έχουν εφαρμοστεί για την κοινή έρευνα της Αναγνώρισης Ονομασμένων Οντοτήτων. Θα συγκρίνουμε και θα αποτιμήσουμε state-of-the-art αρχιτεκτονικές Νευρωνικών Δικτύων (LSTM, CNN, RNN, CRF) με σκοπό να αναγνωρίσουμε τους παρακάτω τύπους οντοτήτων: Άτομο, Τοποθεσία, Οργανισμός, Διάφορα (γενικός όρος). Ακόμα θα μελετήσουμε τη σημασία χρήσης ή μη, νέων χαρακτηριστικών μέσα από το σύνολο των δεδομένων, ειδικά εκείνων που δημιουργούνται από το ανθρώπινο χέρι. Για σωστή σύγκριση, χρησιμοποιούμε το ίδιο σύνολο δεδομένων για την εκπαίδευση του μοντέλου, με τον ίδιο τρόπο όπως χρησιμοποιείται και στα αντίστοιχα δημοσιεύματα. Η κάθε προσέγγιση αξιολογείται με βάση τις μετρήσεις ακρίβειας και ανάκλησης του F1-σκορ ανά τύπο οντότητας για κάθε νευρωνικό δίκτυο.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Επεξεργασία Φυσικής Γλώσσας, Τεχνητή Νοημοσύνη

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Αναγνώριση Ονομασμένων Οντοτήτων, Παραγωγή Οντοτήτων, Εξαγωγή νέων χαρακτηριστικών, Νευρωνικά Δίκτυα, Πολυεπίπεδη ή βαθιά μάθηση

ACKNOWLEDGEMENTS

I would like to thank my supervisor Manolis Koubarakis for his guidance and support for always being available to answer my questions and advise me during the implementation of my thesis, and for helping me improve my knowledge around the domain, and my organization skills during the research. Finally, I would like to thank my partner and my family, for all the love and support they offered me during this period of work.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον καθηγητή μου Μανόλη Κουμπαράκη για την καθοδήγηση και την υποστήριξή του και για το ότι ήταν πάντα διαθέσιμος ώστε να απαντήσει στις ερωτήσεις μου και να με συμβουλεύσει κατά την υλοποίηση της διπλωματικής μου, καθώς και για την βοήθειά του στο γνωστικό κομμάτι γύρω από αντικείμενο της εργασίας αλλά για και τις οργανωτικές δεξιότητές που ακολούθησα κατά τη διάρκεια της έρευνας. Τέλος, θα ήθελα να ευχαριστήσω τον σύντροφό μου και την οικογένειά μου για όλη την αγάπη και την υποστήριξη που μου προσέφεραν κατά τη διάρκεια της περιόδου της εργασίας.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

The present thesis is part of the requirements for the acquisition of a Master's degree in the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens. The main goal is to provide a way to extract (semi-) automatically entities from literature documents, of all types of fields, using the technologies of neural networks, and NLP. The number of datasets that was used is well-known to the wider area of public information, and contains enough annotations so that we can have a satisfying result of accuracy in the model.

The implementation of this thesis, is a study on existing implementations, trying to investigate the best approach out of which we can have the optimal accuracy by training approximately the same dataset.

The implementation and training of the investigated models was done on a Windows machine of a 32GB usage RAM with NVIDIA GPU. The selected models that were implemented were built using Python 3.7 and the features provided used *PyTorch*[1], *TensorFlow*[2], *NumPy*[3], *Scikit-learn*[4] and *Keras*[5], for the data preparation, training and prediction. The *CUDA*[6] library for NVIDIA was necessary to achieve a fast execution and a successful training of the data.

Working on this subject was a very interesting experience, as I managed to learn a lot on the field of neural networks and NER. In combination with my background knowledge which was mostly focused on software engineering, I was able to reproduce the applied models using machine learning libraries, and understand the big impact we can achieve by having available data, a big help for the field of science and text literature.

---

[1] https://pytorch.org/

[2] https://www.tensorflow.org/

[3] https://numpy.org/

[4] https://scikit-learn.org/stable/

[5] https://keras.io/

[6] https://docs.nvidia.com/cuda/

1. INTRODUCTION

In the last years, studies around Natural Language Processing (NLP) have been expanded a lot. End users are flooded with a huge amount of information and news of diverse nature, e.g. politics, physics, environmental, informatics, educational, historical, et cetera. The amount of data is big enough, that it can be delivered in a time-series mode, and thus, it requires a series of analysis and studies, in order to extract the proper knowledge depending on the domain. Via all the appropriate analysis that is constantly being achieved by scientists and researchers, a big number of documents, papers and books are published, providing all this information to the relevant users, industries, or services.

The big step that has been achieved with this data, is the contribution from multiple people and sources to annotate it and translate it to something more meaningful for the machine to read. Annotated data can connect the common parts of a variety of content around and help on the creation of a constant path from one data source to another. Figure 1 shows a representation of this need in a clearer way.

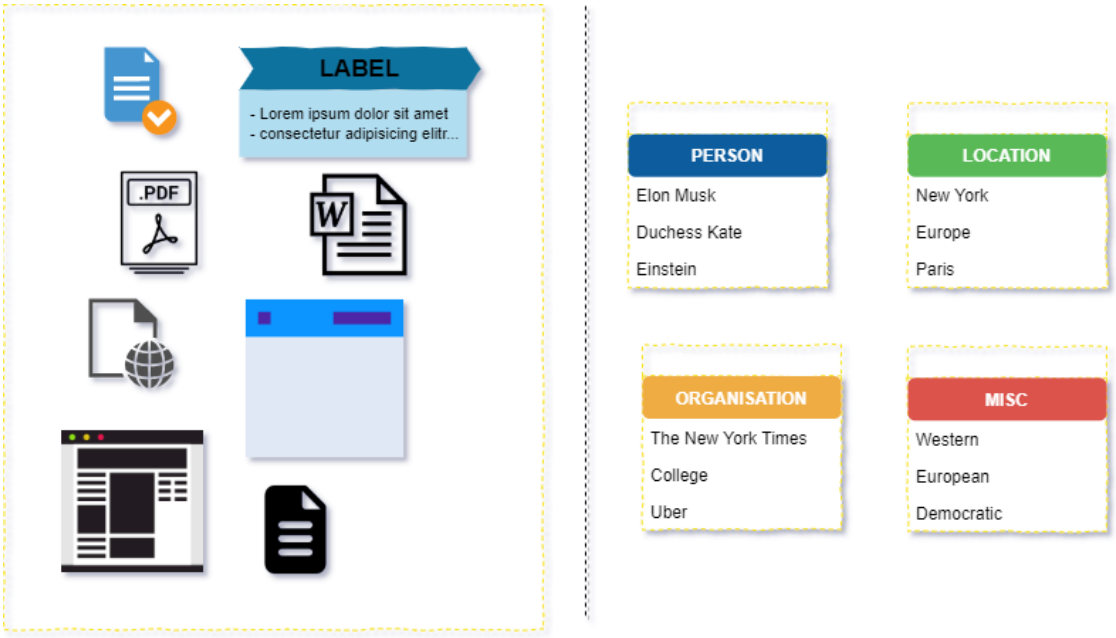


Figure 1: The main problem that NER tackles

How this annotated data is used though? Deep learning has already contributed a lot, with the pre-processing of different types of data, applied to multiple applications. Deep learning is a subarea of machine learning referring to the application of a set of algorithms called neural networks and their variants. In such techniques, one provides the network with a set of labeled examples which it learns, or trains on. Labeling these examples is done in many ways. Machine learning feature extraction is done manually, and

classification is done by machine. In deep learning both the feature extraction and the classification are done by machine [9].

The goal of this thesis, is to analyse studies of applied models that use deep learning techniques, in order to pre-process text, and recognize entities out of this text, creating a continuous source of information that can only be fed by more data which can be found everywhere.

In *Chapter 2*, we provide background information about the problem at hand, describing the main building blocks that compose our study.

In *Chapter 3*, we present our analysis of the related work and the main building blocks that compose our approach, showing the evolution of the task the thesis tackles more in detail.

In *Chapter 4*, we examine three alternatives and we provide all the theoretical approach followed by the authors, as well as the final presented results.

In *Chapter 5*, we reproduce and build the applied models of the three cases that were analyzed in Chapter 4. We conduct the training, testing and interlinking experiments according to the specifications given. We finally evaluate our approach, by analyzing the results of the models, and by comparing them across each other.

In *Chapter 6*, we summarize what was contributed and what could exist in a potential future work.

2. BACKGROUND ON NATURAL LANGUAGE PROCESSING AND NEURAL NETWORKS

In the previous chapter, we reviewed some main reasons for the importance to identify and map any literature part that is available and relevant to data coming from multiple sources. With the evolution of the technologies, we constantly come closer to more and more information of text coming from PDFs, websites, excel data files, social media comments, posts, emails and more. Named Entity Recognition (NER) is an important task for identifying all the necessary entities (e.g., persons, organizations, locations, et cetera) coming out of such text.

In this chapter we provide information on the historical background on NLP, and neural networks, their origins and evolution. Further on, we explain some of the main topics that will be used to address this thesis, focusing on an in-depth theory about their functionality.

2.1 Natural Language Processing

NLP is a subfield of linguistics, computer science, information engineering, and artificial intelligence, that deals with analyzing, understanding and generating the languages that humans use naturally in order to interface with computers in both written and spoken contexts. The machine interprets the important elements of the human language sentence, such as those that might correspond to specific features in a data set, and returns an answer. NLP can be used to interpret free text and make it decomposable and understandable.

A few examples where NLP is daily used is on practices like spell checking, autocomplete, spam filters, extracting information, classifying, machine translation, voice text messaging, complex question answering, and more.

2.2 Deep Learning

Most of the NLP technologies are powered by deep learning – a subfield of machine learning. Deep learning facilitates a lot any kind of implementation that requires large amounts of training data, as well as faster machines and multicore CPU/GPUs. New models are extracted with advanced capabilities and improved performance. More flexible learning of intermediate representations, more effective learning methods for using context and transferring between tasks, and also, better regularization and optimization methods.

Machine learning includes a lot of manually designed features, which are often over-specified, incomplete, or time consuming. Deep learning on the opposite, can automatically learn good features or representations from raw inputs. It makes it easy for the model to adapt fast and learn.

Deep learning provides a very flexible, universal, and learnable framework for representing the world, for both visual and linguistic information. The contribution of deep learning in NLP tasks, provides the ability to build well trained models that do not require traditional, task-specific feature engineering.

2.2.1 Feature representation for NLP tasks

In machine learning, *feature learning* or *representation learning* [24] is a set of techniques that allows a system to automatically discover the representations needed for feature detection or classification from raw data. This replaces manual feature engineering and allows a machine to both learn the features and use them to perform a specific task.

Feature learning is motivated by the fact that machine learning tasks such as classification [25] often require input that is mathematically and computationally convenient to process. However, real-world data such as images, video, and sensor data has not yielded yet to algorithmically define specific features. An alternative way is to discover such features or representations through examination, without relying on explicit algorithms.

Feature learning can either be:

- *Supervised*: features are learned using labeled input data
- *Unsupervised*: features are learned with unlabeled input data

Neural networks are a family of learning algorithms that use a “network” consisting of multiple layers of inter-connected nodes. Multilayer neural networks can be used to perform feature learning, since they learn a representation of their input at the hidden layer(s) which is subsequently used for classification or regression at the output layer.

2.2.2 NLP techniques

One of the first steps that need to be performed on the analysis of our data, depend a lot on what type of data we have, meaning what is the format of the data to be trained, and where we extract it from. The main steps to apply a successful NLP technique are:

- *Tokenization*: breaking text into words and sentences
- *Stop-word Removal*: filtering common words
- *N-Grams*: identifying commonly occurring groups of words
- *Word-sense disambiguation*: identifying he context in which the word occurs
- *Part-of-speech*: identifying Part-of-Speech
- *Stemming*: removing ends of the words

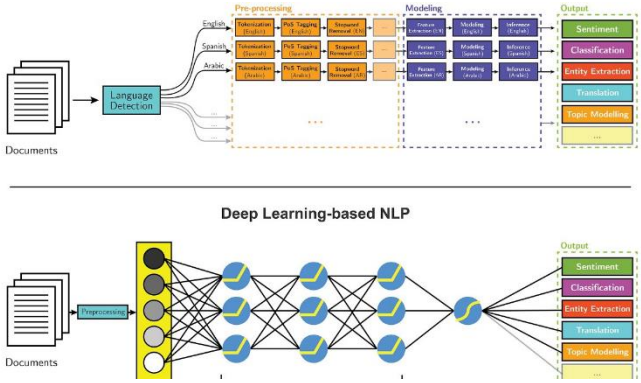


Figure 2: Simple NLP approach vs. Deep Learning NLP approach

In Figure 2 (taken from [50]) we see the difference between a classical and a deep learning NLP approach. In classical NLP techniques (upper part of the figure) we pre-process the data in the early stages before generating features out of the data. In the next phase, we use hand-crafted features that are generated using NER tools, POS taggers, and parsers. We feed these features as input to the ML algorithm and train the model. We check the accuracy, and if it is not good, we optimize some of the parameters of the algorithm and try to generate a more accurate result. Depending on the NLP application, we can include the module that detects the language and then generates features.

In case of deep learning techniques (lower part of the figure) for an NLP application, we do some basic pre-processing on the data we have. Then, we convert our text input data to a form of dense vectors. To generate the dense vectors, we use word-embedding techniques such as Word2Vec, GloVe and so on, and feed these dense vectors embedding to the deep neural network. Here we do not use hand-crafted features but different types of deep neural networks as per the NLP application. The multiple layers of deep neural networks generalize the goal and learn the steps to achieve the defined goal. In this process, the machine learns the hierarchical representation and gives us the result that we validate and tune the model as per the necessity.

2.3 Text Embeddings

Any classification method [25] uses a set of features or parameters to characterize each object, where these features should be relevant to the task in hand. Sentences are classified according to their syntax or structure, their form and how complete they are.

Depending on the type of each sentence, the model can separate the declarative sentences, from the interrogative and the imperative sentences.

There are several rule-based approaches that classify text into organized groups by using a set of handcrafted linguistic rules. These rules set the system in a way so that it can use semantically relevant elements of a text to identify relevant categories based on its content.

Text embeddings [27] are real values of vector representations of strings. We build a dense vector for each word, chosen in a way that it's similar to vectors of words that appear in similar contexts.

Word embeddings are considered a great starting point for most deep NLP tasks. They are the result of unsupervised learning of word representation and they can be considered as a distributed semantic model. It is globally assumed, that words that belong to similar context share semantic meaning. They allow deep learning to be effective on smaller datasets, as they are often the first inputs to a deep learning architecture and the most popular way of transfer learning in NLP. Each word can be represented as a vector in power of *N*, where *N* is a unique number of words in the given dictionary.

The most popular names in word embeddings are *Word2Vec* by Google [28], and *GloVe* by Stanford [19]. Additionally, there are more applied techniques, like *DFastText*[7], *Poincare Embeddings*[8], *sense2vec*[9], *Skip-Thought*[10], *Adaptive Skip-Gram*[11].

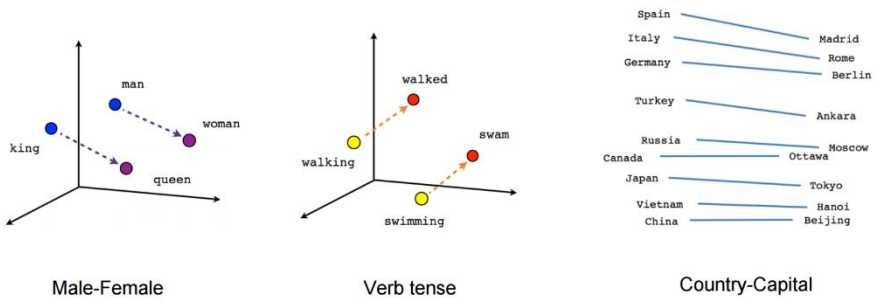


Figure 3: Examples of Word Representations

In Figure 3 (taken from [28]) we see some interesting word associations captured by embeddings. Once the word vectors are reduced to two dimensions, it is possible to see relationships between certain words. Examples of a semantic relationship are

[7] https://fasttext.cc/

[8] https://github.com/facebookresearch/poincare-embeddings

[9] https://github.com/explosion/sense2vec

[10] https://pypi.org/project/skip-thoughts/

[11] https://github.com/sbos/AdaGram.jl

male/female designations and country/capital relationships, while an example of syntactic relationship is past vs. present tense. Words that share semantic or syntactic relationships will be represented by vectors of similar magnitude and be mapped in close proximity to each other in the word embedding. This actually opens up an entirely new dimension of possibilities for finding patterns or other insights in the data.

2.3.1 Word2Vec Embeddings

In Word2Vec [28], we have a large corpus of text in which every word in a fixed vocabulary is represented by a vector. We then go through each position *t* in the text, which has a center word *c* and context words *o*. Next, we use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa). We keep adjusting the word vectors to maximize this probability.

For efficient training of Word2vec, we can eliminate meaningless (or higher frequency) words from the dataset (e.g. *a*, *the*, *of*, *then*). This helps improve model accuracy and training time. Additionally, we can use negative sampling for every input by updating the weights for all the correct labels, but only on a small number of incorrect labels.

Word2vec has 2 model variants worth mentioning:

- **Skip-Gram:** [22] We consider a context window containing *k* consecutive terms. Then we skip one of these words and try to learn a neural network that gets all terms except the one skipped and predicts the skipped term. Therefore, if two words repeatedly share similar contexts in a large corpus, the embedding vectors of those terms will have close vectors.
- **Continuous Bag of Words:** [28] We take multiple sentences in a large corpus. Every time we see a word, we take the surrounding word. Then we input the context words to a neural network and predict the word in the center of this context. When we have thousands of such context words and the center word, we have one instance of a dataset for the neural network. We train the neural network and finally the encoded hidden layer output represents the embedding for a particular word. It so happens that when we train this over a large number of sentences, words in similar context get similar vectors.

Both the variants are window-based models, meaning that the co-occurrence statistics of the corpus are not used efficiently, resulting in suboptimal embeddings.

2.3.2 GloVe Embeddings

The *GloVe* [19] model captures the meaning of one word embedding with the structure of the whole observed corpus. For that reason, the model trains on global co-occurrence counts of words and makes a sufficient use of statistics by minimizing least-squares error and, as a result, produces a word vector space with meaningful substructure. With this way, an outline sufficiently preserves words' similarities with vector distance.

2.4 POS Tagging

POS tagging [29] is essential for building *lemmatizers*[12] which are used to reduce a word to its root form. POS tagging is the process of making up a word in a corpus to a corresponding part of speech tag, based on its context and definition.

2.5 Named Entity Recognition

NER [17] is the task of identifying textual mentions and classifying them into a predefined set of types of various approaches which have been proposed to tackle the task, from hand-crafted feature-based machine learning models, like conditional random fields [37] and perceptron [36], to deep neural models [15, 7].

In order to properly map and recognize label entities, we need to apply NER on text. Named entities provide a small, tractable set of elements carrying a well-defined semantics. Generic named entities are names of persons, locations, organizations, phone numbers, and dates, while domain-specific named entities include names of like, sensor, modern, nature, media, et cetera.

For the requirements of the thesis, this step plays a very important role in the high quality of the search results and the better detailed way of expressing desired knowledge graphs. In order to have proper results, the amount of data that is required, is huge. Hence, the identification of entities like the ones mentioned above, must be done in an automatized way, which will help achieving the optimal output of NER. There are several techniques, which need to be analyzed, and be combined in such a way that the best results will come out after the implementation.

2.6 Neural Networks

Neural Networks (NN) [30] have been around since 1943 when the subject was first initialized. They can be described as a finite subset of units (or nodes or neurons) $N = \{n_1, n_2, \ldots,\}$ and a finite set $H \subseteq N \times N$ of directed edges or connections between nodes. *Forward-Neural-Networks* (FNNs) [31] are acyclic graphs, though *Recurrent Neural Networks* (RNNs) [48] are cyclic. There are several layers that define a NN. The first layer is always the input layer, and it is represented as a set of input units, a subset of N.

This artificial neuron inside a NN is modelled mathematically, as shown in Table 1.

Table 1: Mathematical logic of an artificial neuron

1. Each neuron is modeled as a set of inputs to the neuron (x_1, x_2, x_3), and as a set of outputs y to other neurons
2. Numeric values are fed into the neuron, which represent the strength of the input signal

[12] https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

3. The neuron produces a numeric output value, which represents the strength of the signal leaving the neuron
4. Information flow through the neuron from its inputs to its output
5. The connections leading into the neuron have weights that can be either increased or decreased *(w_1, w_2, w_3)*
6. When a neuron received input from its connections, it sums together all of the input values
7. The neuron has an activation function which determines how much output it will produce given the summed input values. There are several types of activation functions, each one being appropriate for various applications. Activation functions can be considered as mathematical ways to squash the output so that they stay within a certain range of values.

2.6.1 Convolutional Neural Networks

A *Convolutional Neural Network* (ConvNet / CNN) [42] is a deep learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects / objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods, filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters / characteristics.

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or another operation (dot product). The activation function is commonly a *RELU* layer [26], and is subsequently followed by additional convolutions such as pooling layers, fully connected layers, and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution. The final convolution, in turn, often involves backpropagation in order to weight the end product more accurately.

CNNs can use more hyper-parameters than a standard multilayer perceptron. While the usual rules for learning rates and regularization constants still apply, the following should be kept in mind when optimizing.

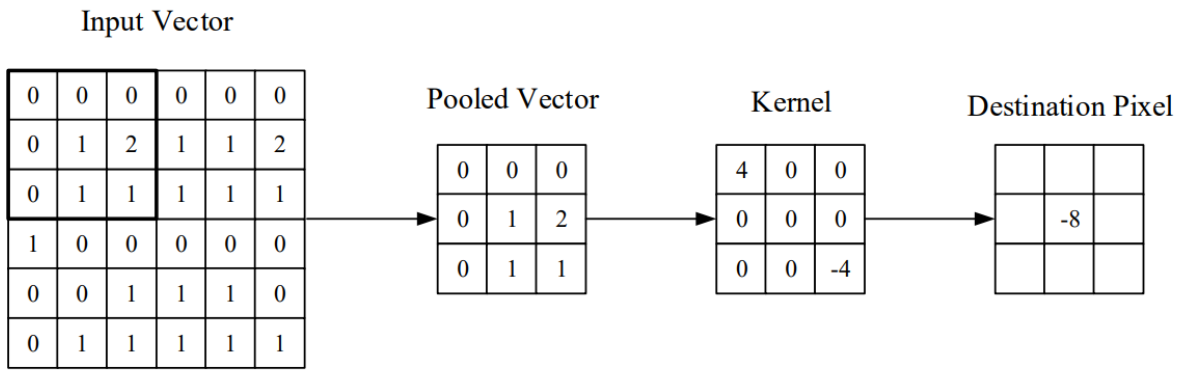


Figure 4: A visual representation of a convolutional layer

In Figure 4 (taken from [42]), we see how the movement of the first part of the convolutional layer (kernel), is moving in carrying out the operation of an image pre-processing. The center element of the kernel is placed over the input vector, of which is then calculated and replaced with a weighted sum of itself and any nearby pixels.

2.6.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) [48] simply address the issue of classifying what kind of event is the next one to occur based on reasoning on previous events. RNN is a looping mechanism that acts as a highway to allow information to flow from one step to the next. It is a neural sequence model that achieves state-of-the-art performance on important tasks that include language modeling [32], speech recognition [34], and machine translation [33]. The RNN dynamics can be described using deterministic transitions from previous to current hidden states.

In practice, RNNs accept an input vector *x*, and give an output vector *y*. However, this output vector's contents are crucially influenced not only by the input that was fed in, but also by the entire history of inputs that were fed in the past.

In an example of character-level language models (Figure 5[13]), RNN receives a huge chunk of text, and tries to calculate the probability distribution of the next character in the sequence given a sequence of previous characters. This will then allow to generate new text one character at a time.

Each character will be encoded into a vector using 1-of-k encoding, and will be fed into the RNN one at a time with the help of a step function. We can then observe a sequence of 4-dimensional output vectors (one dimension per character), which interpret as the confidence the RNN currently assigns to each character coming next in the sequence.

[13] https://karpathy.github.io/2015/05/21/rnn-effectiveness/

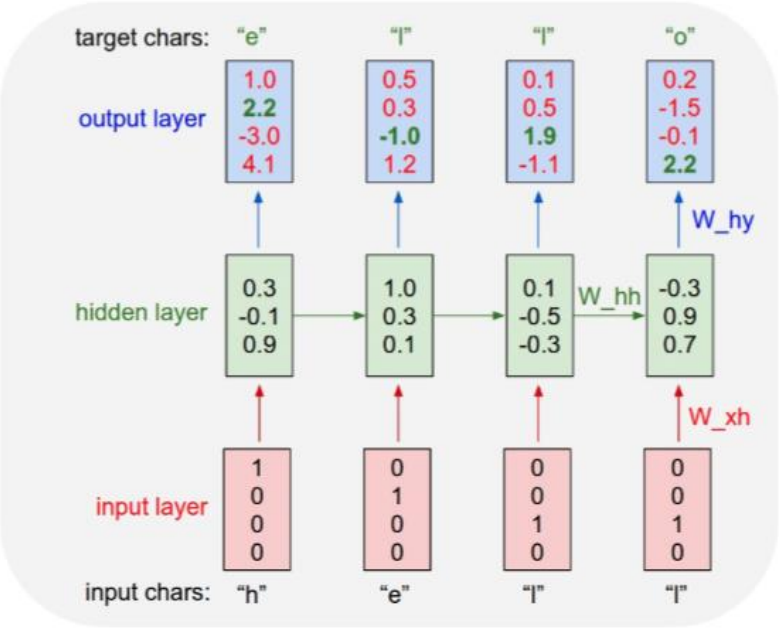


Figure 5: RNN with 4-dimensional I/O layers, and a hidden layer of 3 units (neurons)

The diagram in Figure 5, shows the activations in the forward pass when the RNN is fed the characters *"hell"* as input. The output layer contains confidences the RNN assigns for the next character (the vocabulary is *"h,e,l,o"*). The green numbers are expected to be high and the red numbers to be low.

This process is repeated over and over many times, until the network converges and its predictions are eventually consistent with the training data in that correct characters are always predicted text.

2.6.3 Long Short-Term Memory Neural Networks

Long Short-Term Memory networks – usually just called *LSTMs* – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by [41], and were refined and popularized by many people in following work. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior. LSTMs support the structure of a chain of repeating modules, but each module, like in RNNs, with the difference that each module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way (Figure 6[14]).

[14] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

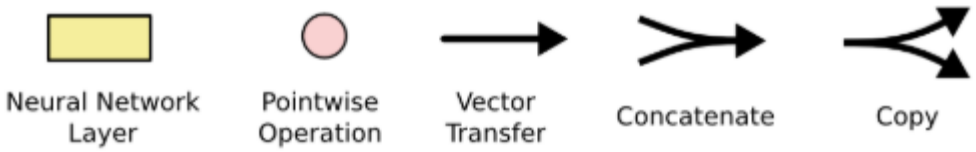


Figure 6: The four neural network layers in LSTM

There are several architectures of LSTM units. A common LSTM architecture is composed of a *"cell"* (the memory part of the LSTM unit), and three *"regulators"*, usually called gates of the flow of information inside the LSTM unit: an *"input gate"*, an *"output gate"*, and a "*forget gate"*. Some variations of the LSTM unit do not have one or more of these gates or maybe have other gates.

The *cell*, is responsible for keeping track of the dependencies between the elements in the input sequence. The *input gate*, controls the extent to which a new value flows into the cell, and the *forget gate* controls the extent to which a value remains in the cell. Finally, the *output gate* controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit. The activation function of the LSTM gates is often the logistic sigmoid function.

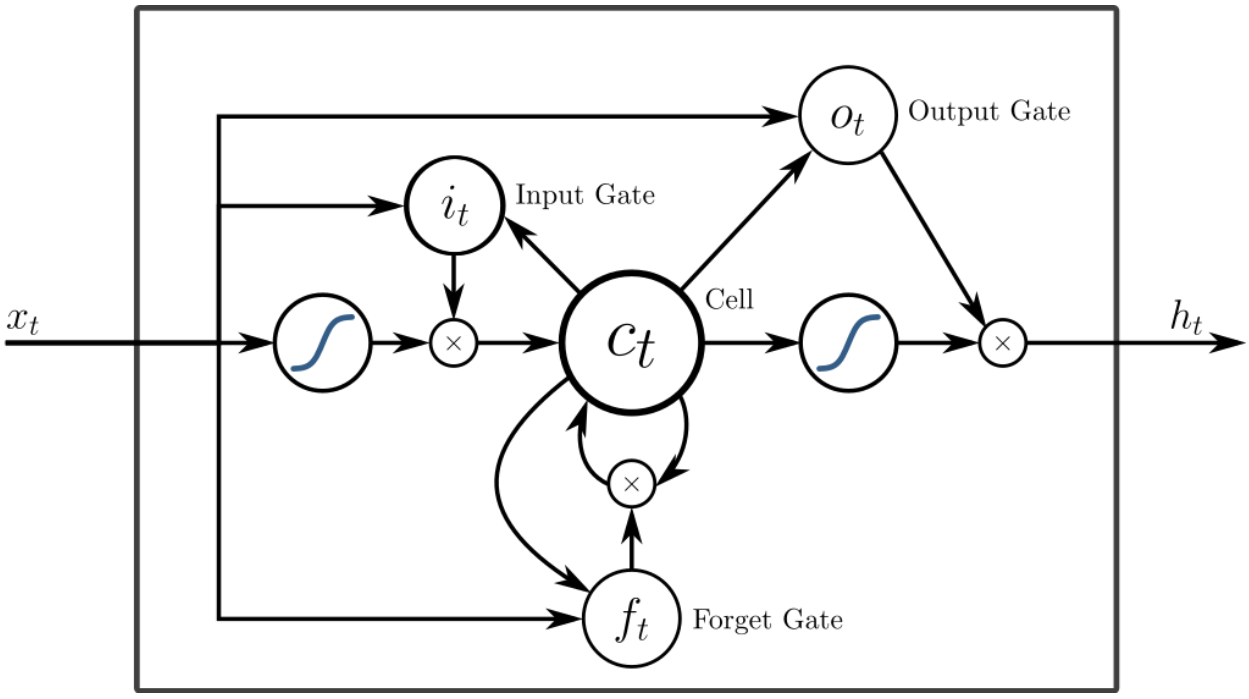


Figure 7: A peephole LSTM unit with input, output, and forget gates

In Figure 7 (taken from [34, 51]) we see an example of peephole LSTM unit with input, output and forget gates. We observe that each one of the gates can be thought as a "standard" neuron in a feed-forward (or multi-layer) neural network. They compute an activation (using an activation function) of a weighted sum, and represent the activations

of respectively the input, output, and forget gates, at time step. The 3 exit arrows from the memory cell to the 3 gates and represent the peephole connections. These peephole connections denote the contributions of the activation of the memory cell at time step. The little circles containing a symbol represent an element – wise multiplication between its inputs. The big circles containing an S-like curve represent the application of a differentiable function to a weighted sum.

2.6.4 Bi-LSTM

Bi-directional Recurrent Neural Networks (BRNNs) [35] connect two hidden layers of opposite directions to the same output. With this form of generative deep learning, the output layer can get information from past (backwards) and future (forward) states simultaneously.

BRNNs were introduced to increase the amount of input information available to the network. Standard RNNs may have restrictions as the future input information cannot be reached from the current state. On the contrary, BRNNs do not require their input data to be fixed. Moreover, their future input information is reachable from the current state.

In comparison with unidirectional (Figure 8[14]) is that in the LSTM that runs backwards, we preserve the information from the future. In bidirectional though, we use two hidden states combined so that we are able to preserve information from both past and future in any point in time.

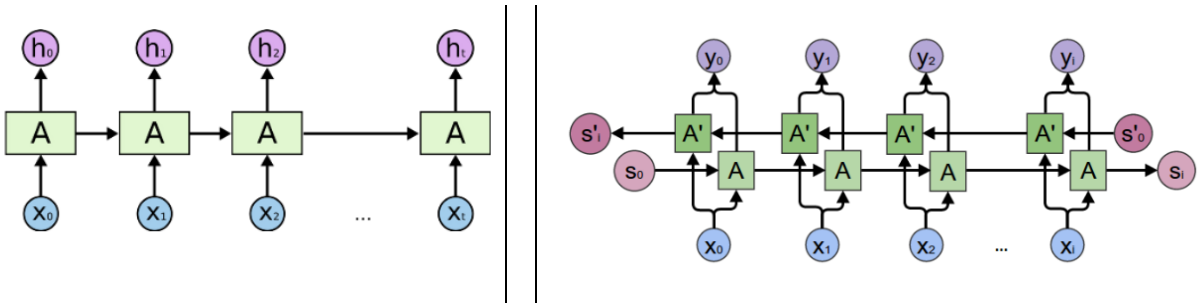


Figure 8: LSTM vs Bi-LSTM views

2.7 Activation functions

In artificial neural networks, the activation function of a node defines the output of that node given an input or set of inputs. Activation functions are mathematical equations that determine the output of a neural network. The function is attached to each neuron in the network, and determines whether it should be activated or not, based on whether each neuron's input is relevant for the model's prediction. Activation functions also help normalize the output of each neuron to a range between *1* and *0* or between *-1* and *1*.

An additional aspect of activation functions is that they must be computationally efficient because they are calculated across thousands or even millions of neurons for each data

sample. Modern neural networks use a technique called backpropagation to train the model, which places an increased computational strain on the activation function, and its derivative function.

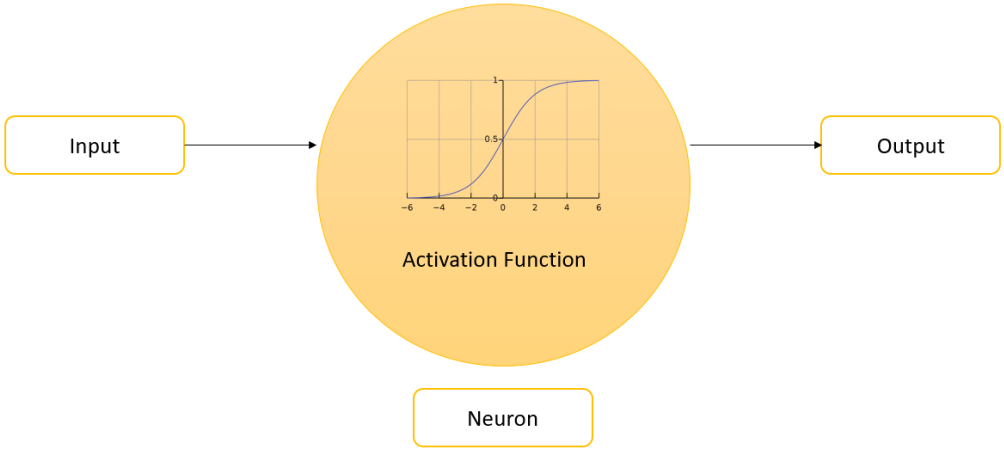


Figure 9: The activation function in a neural network

Figure 9 shows how the activation function is structured with regards to the neural network. It is a mathematical "gate" in between the input feeding the current neuron and its output going to the next layer. It can be as simple as a step function that turns the neuron output on and off, depending on a rule or a threshold. It can also be a transformation that maps the input signals into output that are needed for the neural network to function.

The need for speed has led to the development of new functions such as sigmoid / logistic, tanh / hyperbolic tangent, ReLU (Rectified Linear Unit), leaky ReLU, parametric ReLU, SoftMax, swish. There are three types of Activation Functions:

- The *binary step function*, which is a threshold-based activation function, but it doesn't allow multi-value outputs.
- The *linear activation function*, which takes the inputs, multiplied by the weights for each neuron, and created an output signal proportional to the input. In one sense, a linear function is better than a step function, because it allows multiple outputs, not just yes and no. It is not possible to use backpropagation to train the model, and all layers of the neural network collapse into one.
- The *non-linear activation functions*, which are mostly used by modern neural network models. They allow the model to create complex mappings between the network's inputs and outputs, which are essential for learning and modeling complex data, such audio, video, or even datasets which are non-linear or have high dimensionality. Non-linear functions address the problems of a linear activation function:

- They allow backpropagation because they have a derivative function which is related to the inputs.
- They allow "stacking" of multiple layers of neurons to create a deep neural network. Multiple hidden layers of neurons are needed to learn complex datasets with high levels of accuracy.

2.8 Dropout

Dropout[15] [40] is a regularization technique patented by Google for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term *dropout* refers to dropping out units (both hidden and visible) in a neural network.

A good example is during training, where dropout layer randomly sets some of the inputs it received to *0* – effectively removing the contribution of these inputs to the network. This has the effect of generalizing the model, which reduces the overfitting and results on better performance on real word data. The percentage of the inputs that get to 0 can also be set. The setting of values to 0 is done in a random fashion as sets of data are passed. This ensures the model is forced to learn the relationship between all possible paths of data through the network.

2.9 Logistic Regression

With the regression, we attempt to predict a numeric outcome based on one or more input variables. The "classic" application of logistic regression model is binary classification (binomial classifier) – we translate input into one of two categories. Neural networks are used for the purpose of clustering through unsupervised learning, classification through unsupervised learning, or regression. That is, they help group unlabeled data, categorize labeled data or predict continuous values. Classifiers typically use a form of logistic regression in the net's final layer to convert continuous data into dummy variables like *0* and *1*. For example, given someone's height, weight and age, he might be bucketed as a heart-disease candidate or not.

The results of the logistic regression can be interpreted as the likelihood that the data in question is the targeted value or not. The main category is known as the default class, usually the represented as the *1* in the classifier. Logistic regression can also be expressed in a Bayesian way[16].

[15] https://en.wikipedia.org/wiki/Dropout_(neural_networks)

[16] https://en.wikipedia.org/wiki/Bayesian_probability

2.10 Conditional Random Fields

Conditional random fields (CRFs) are a class of statistical modeling method often applied in pattern recognition and machine learning, and used for structured prediction. CRFs fall into the sequence modeling family. Whereas a discrete classifier predicts a label for a single sample without considering *"neighboring"* samples, a CRF can take context into account. An example to this explanation, is the linear chain CRF, shown in Figure 10, which predicts sequences of labels for sequences of input samples.

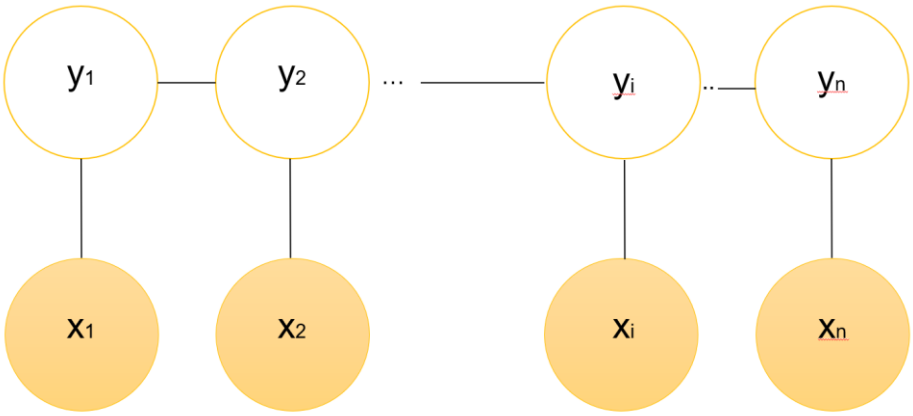


Figure 10: Structure of linear chain conditional random field

CRFs are a type of discriminative undirected probabilistic graphical model. They are used to encode known relationships between observations and construct consistent interpretations and are often used for labeling or parsing of sequential data, such as NLP shared tasks. Specifically, CRFs find applications in POS tagging, shallow parsing, named entity recognition, dense finding, and more.

Conclusion

In this chapter, we provided background information about the subject being studied, describing the main components that play an important role to the next steps of our analysis.

Specifically, we focused on the theory of NLP, neural networks, and deep learning. With regards to deep learning, we presented different NLP techniques, in combination with feature representation NLP tasks. In addition, we explained how text embeddings work, emphasizing to the Word2Vec and GloVe embeddings, as well as POS tagging.

We gave a thorough explanation on some neural networks, that will be also used in our analysis in the next chapters. Important types of neural networks, like CNNs, RNNs, LSTMs, Bi-LSTMs. We also provided information about necessary functions and

methods, that are used in neural networks, and affect critically the result of the training data. Methods like *dropout*, *logistic regression*, CRFs.

3. BACKGROUND AND RELATED WORK

In this chapter, we provide a short review of the related work around NER, for multiple tagging entities and training datasets. The main datasets used in the models we analyse are the CoNLL-2003 [18] dataset and the OntoNotes v5 [5, 6] dataset, both in the English language.

The CoNLL-2003 task consists of newswire text from the Reuters RCV1 corpus tagged with four different entity types (PER, LOC, ORG, MISC). Models are evaluated based on a span-based F1 on the test set, where both the train and development splits are used for training.

The OntoNotes corpus v5 [5, 6] is a richly annotated corpus with several layers of annotation, including named entities, coreference, part of speech, word sense, propositions, and syntactic parse trees. These annotations are over a large number of tokens, a broad cross-section of domains, and 3 languages (English, Arabic and Chinese). The NER dataset (of interest here and only in English) includes 18 tags, consisting of 11 types (Person, Organization, et cetera) and 7 values (Date, Percent, et cetera), and contains 2 million tokens.

3.1 Word representations

Pre-trained word representations [20] play an important role in many neural language understanding models. [10] provide a deeper analysis on the topic by learning high quality representations. They introduce a new type of deep contextualized word representation that significantly improves the state-of-the-art in every considered case across a range of challenging language understanding problems. They model complex characteristics of word use, like syntax and semantics, and evaluate how these uses vary across linguistic contexts, such as model polysemy.

The vectors that are used are referred as *Embeddings from Language Models* (ELMo) representations, expressing that they derive a bidirectional LSTM that is trained with a coupled *language model* (LM) objective on a large text corpus. Extensive experiments show that ELMo representations work extremely well in practice, as (a) they can be easily added to existing models for language understanding problems, (b) they improve the state-of-the-art in every case, including up to 20% relative error reductions, and (c) they outperform the representations derived from the just the top layer of an LSTM.

3.2 Feature Engineering and hand-crafted features

A big part of what deep learning contributes to, is feature engineering. The NER task requires both lexical and syntactic knowledge, something that creates a huge need of manually engineered features to most statistical sequential labeling models. The authors

of paper [15] actually started the trend of feature engineering-free modelling, by learning internal representations of compositional components of text (e.g. word embeddings)

However, the general opinion supports that hand-crafted features are redundant for deep learning models, as they already learn adequate representations of text automatically from corpora. [9] proposes a hybrid neural architecture method for exploring handcrafted features applied to NER tasks. They extend a *Bi-LSTM-CNN-CRF* [7] model by incorporating an auto-encoder loss to take manual features as input, and then reconstruct them. IOBES was used as a tagging scheme, and word embeddings were initialized with GloVe [19]. As a result, they present significant improvements over a collection of competitive baselines. They obtain 91.89 F1 score for CoNLL-2003 English shared task, which outperforms a collection of highly competitive baseline models. Additionally, they present an abstract study which analyses the impacts of each manually hand-crafted feature, showing a performance degradation when eliminating POS, word shape and gazetteer features. This means that each feature contributes to NER performance beyond what is learned through deep learning alone.

Another approach was applied by [14] in order to present a multilingual and robust *NER and Classification* (NERC) system, which depends on general features to apply word representation features for high performance. Three types of simple clustering features are used, based on unigram matching: (a) Brown, (b) Clark and (c) Word2vec features. All three types contribute to very competitive results in both in-domain and out-of-domain settings. The study reports state-of-the-art results in multiple languages (Dutch, English, German, Spanish and Basque). It is also reported that the system's performance remains high even when reducing the supervised data by half or more.

Keeping in mind that language-specific resources and features are costly to develop in new languages and new domains, NER finds it challenging to adapt. Unsupervised learning from unannotated corpora offers an alternative strategy for obtaining better generalization from small amounts of supervision. Character-based word representations learned from a supervised corpus, in combination with unsupervised word representations learned from unannotated corpora, contribute to the approach presented by [16] where two neural models are presented that don't use at all any language-specific knowledge or even resources. The models are designed to capture two needs:

- Reasoning jointly over tagging decision for each token
    - A Bi-LSTM with a sequential conditional random factor layer above it (LSTM-CRF), is compared with a new model that constructs and labels chunks of input sentences using an algorithm inspired by transition-based parsing with states represented by *stack LSTMs* (s-LSTM).
- If token-level evidence for "being a name" includes both orthographic and distributional evidence
    - A character-based word representation model [21] is used to capture orthographic sensitivity

- A combination of representations with distributional representations [22] is used to capture distributional sensitivity

The first model is based on Bi-LSTMs and CRFs, and requires less orthographic information since it gets more contextual information out of the bidirectional LSTMs. A state-of-the-art performance is obtained in Dutch, German and Spanish, and a near state-of-the-art performance is obtained in English, without any hand-engineered features or gazetteers.

The second model constructs and labels segments using a transition-based approach inspired by shift-reduce parsers. It is more dependent on character-based representations, in order to achieve a competitive performance. It consumes the words one by one and it just relies on the word representations when it chunks words. In this case, the best previously published results are surpassed in several languages, but the performance is less high than the LSTM-CRF model. Near state-of-the-art results are reported when compared to systems that do not use external data.

3.3 Sequence labeling on neural models

[2] present an open-source neural sequence labeling toolkit, where three layers are used for the implementation: (a) a character sequence layer, (b) a word sequence layer, and (c) an inference layer. NCRF++ is designed for quick implementation of different neural sequence labeling models with a CRF inference layer. Built with *PyTorch* and a custom model structure, it can be configured very easily, and can be very flexible with features. It supports user-defined features via distributed representations, and it also integrates several state-of-the-art automatic feature extractors, such as CNN and LSTM. NCRF++ uses batch calculation, which helps on the acceleration of the GPU, and can provide fast results which are equally comparable to other state-of-the-art neural models.

The Character Sequence Layer integrates several typical neural encoders for character sequence information, such as RNN and CNN. The Word Sequence Layer supports both RNN and CNN as the word sequence feature extractor. It can be stacked, building a deeper feature extractor. The Inference Layer takes the extracted word sequence representations as features and assigns labels to the word sequence. Both softmax and CRF are supported as the output layer.

Regarding the evaluation, the experiments were conducted on (a) CoNLL-2003 dataset with the standard split, for the NER task, and on (b) CoNLL-2000 shared task with the split of [23], for the chunking task. It is shown that it can achieve stat-of-the-art results with an efficient running speed.

3.3.1 Contextual string embeddings for sequence labeling

Using string embeddings for sequence labeling is another approach followed by [8] where they present a method for producing them using neural character LM. In more details, the study suggests to leverage the internal states of a trained character language model, to

product a novel type of word embedding which we refer to as *contextual string embeddings*. These embeddings are trained without any explicit notion of words and model words, and are contextualized by their surrounding text, meaning that the same word will have different embeddings depending on its contextual use.

They are also highly useful for downstream tasks, by outperforming the previous state-of-the-art word on English and German NER tasks, using the CoNLL-2003 shared task.

3.3.2 Empower sequence labeling with task-aware language model

A different study explores once more the impact of sequence labeling in the use of neural networks. [3] proposes an effective sequence labeling framework, called ML-LSTM-CRF, which extracts knowledge from self-contained order information of training sequences. It leverages both word-level and character-level knowledge in an efficient way.

In deeper analysis, for *character-level* knowledge, a neural language model is incorporated, with the sequence labeling task, and multi-task learning guides the language model towards task-specific key knowledge. For *word-level* knowledge, there is fine-tune applied for pre-trained word embeddings, without co-training or pre-training of the word-level layers

With regards to the evaluation, the CoNLL-2003 NER task was used for the experiments, reaching a F1 score of 91.71 without the addition of any extra annotations.

3.3.3 Hybrid semi-Markov CRF for sequence labeling

[1] present the improvement of SCRF methods by employing word-level and segment-level information in parallel, for segment score calculation. A joint decoding algorithm is proposed for neural sequence labeling. The two basic steps that are applied are (a) the word-level labels that are utilized to derived the segment scores in SCRFs, and (b) the CRF and SCRF output layers that are integrated into a unified neural network on CoNLL-2003 NER shared task. The final result gives a state-of-the-art performance when no external knowledge is used.

3.3.4 Semi-supervised sequence tagging with bidirectional language models

In addition to the aforementioned practices, another study investigates an alternate semi-supervised approach which does not require additional labeled data. [12] presents a simple and general semi-supervised method for augmenting token representations in sequence tagging models. A neural language model (LM) is used, pre-trained on a large, unlabeled corpus, to compute an encoding of the context at each position in the sequence and use it in the supervised sequence tagging model. This makes the context sensitive representation captured in the LM embeddings, very useful for the supervised sequence tagging schema.

By applying this method and adding LM embeddings in the main system, the overall performance is increased with a total F1 score of 91.93 on the CoNLL-2003 shared NER task. A new state-of-the-art F1 score of 96.37 is also achieved on the CoNLL-2000 Chunking task.

The analysis shows as well, that adding a backward LM in addition to traditional forward LMs consistently robust, even when the LM is trained on unlabeled data from a different domain, or when the baseline model is trained on a large number of labeled examples.

3.3.5 Transfer learning for sequence tagging with hierarchical recurrent networks

Last but not least, it makes sense to mention one more practice applied for sequence tagging by [13]. The main task challenges how to transfer knowledge from one task to another – often referred as *transfer learning*. The problem is originated from the need of knowledge transfer for neural sequence taggers where a source task with plentiful annotations is used to improve performance on a target task with fewer available annotations.

The transfer learning approach that is presented, depends on a deep hierarchical recurrent neural network, which shares the hidden feature representation and part of the model parameters between the source task and the target task. The new approach combines the objectives of the two tasks and uses gradient-based methods for efficient training. Three neural network architectures are designed for the settings of *cross-domain*, *cross-application*, and *cross-lingual* transfer.

Evaluating the performance, the new transfer learning method achieves significant improvements on various datasets under low-resource conditions, as well as new state-of-the-art results on some of the benchmarks. In the performance, (a) the label abundance for the target task, (b) the relatedness between the source and target tasks, and (c) the number of parameters that can be shared, are the main factors that eventually contribute to the final outperforming results.

Table 2: F1 scores in related work using the CoNLL-2003 dataset

Index	Model	F1 score
Model1	Deep contextualized word representations	92.22
Model2	Evaluating the Utility of Hand-crafted Features in Sequence Labeling	91.89
Model3	Robust multilingual Named Entity Recognition with shallow semi-supervised features	91.36
Model4	Neural Architectures for Named Entity Recognition	90.94

Model5	NCRF++: An open-source Neural Sequence Labeling Toolkit	91.35
Model6	Contextual String Embeddings for Sequence Labeling	93.09
Model7	Empower Sequence Labeling with Task-Aware Language Model	91.71
Model8	Hybrid semi-Markov CRF for Neural Sequence Labeling	91.38
Model9	Semi-supervised sequence tagging with bidirectional language models	91.93
Model10	Transfer Learning for Sequence Tagging with Hierarchical Recurrent Networks	91.26

Table 2 shows an overview of the related F1 scores that explain the performance results on each one of the above analyzed models using the same training dataset. It is clear that there are minor differences in some of the results, and it is proven that the impact of sequence labeling and managing properly hand-crafted features and word embeddings, can contribute to high performances and new defines state-of-the-art results, compared to previous works. We can also confirm the variation of the F1 scores in Figure 11, where the use of *contextual string embeddings* provides a clear difference in the final result.

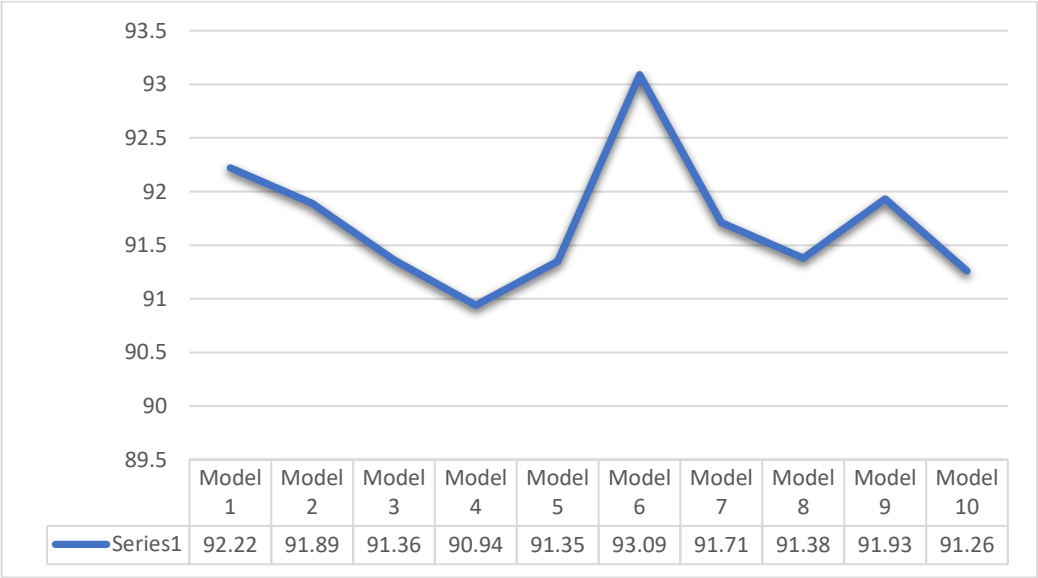


Figure 11: Evolution of the F1 scores in the related work models

Conclusion

In this chapter we provided all the necessary information about the related work around NER, which will be used in the next chapters. We reviewed how word representations contribute to neural language understanding models, and how important feature engineering and hand-crafted features are for the model. Finally, we analyzed the techniques of sequence labeling on neural models, focusing on contextual string embeddings, hybrid semi-Markov CRFs, task-aware and bidirectional language models, and transfer learning.

4. AN EXPERIMENTAL COMPARISON OF THREE ALTERNATIVES

In this chapter, we focus on describing the workflow of three different approaches with regards to NER. We provide the theoretical analysis that each approach follows and we present the neural network components that define the implementation of the final model and conduct the training.

We also give all the necessary information with regards to the system configuration, and the settings that were applied to the experiments, as described by the authors, including all the technical characteristics that comprise the model, along with the final results and the duration of the training for each dataset.

The datasets that were used for the training are the same for all the three studies, CoNLL-2003 and OntoNotes 5.0. This makes the analysis and comparison equal with regards to the annotated data, and gives us room to focus only on the different techniques applied.

4.1 Robust lexical features for improved neural network Named-Entity Recognition

The authors of [11], focus on the importance of hand-crafted features in neural network approaches to named entity recognition. It is shown that lexical features can be quite useful.

A new method is presented, which suggests to embed words and entity types into a low-dimensional vector space which is trained from annotated data produced by distant supervision from Wikipedia. As an outcome, an offline feature vector is computed, representing each word.

The published results of the study show a new state-of-the-art F1 score of 87.95 on OntoNotes 5.0 dataset, and a state-of-the-art performance with a F1 score of 91.73 on the over-studied CoNLL-2003 dataset.

4.1.1 Problem definition

There are some main limitations on the use of gazetteer features. Specifically:

- The binary representation of features is not always successful as the gazetteer features only encode the presence of an *n-gram* in each list and omit its relative frequency. As a result, the preference of a word is not always captured correctly. For example, “Greece” can be considered either as a *person*, *organization* or *location* in a text, even if it mainly refers to the country most of the time.
- The generation of entries is a time-consuming approach, as it depends on matching every *n-gram* in a sentence against entries in the lexicons.
- There are a lot of non-entity words that might be completely skipped from the gazetteer features, even though it can be proven the specific words can appear after the mention of a type *PER*.

4.1.2 Method analysis

One of the first steps to be defined, is the way to *embed words and entity types*. Wikipedia plays a very important role on this, as it is helped on the generation of WiNER [46] and WiFiNE [45], two approaches for generating named entities, which produce coarse (4 classes) and fine-grained (120 labels) named entity annotations.

This method uses WiFiNE as the source of annotations. Each entity mention is mapped to a pre-defined set of 120 entity types. Types are stored in a 2-level hierarchical structure (e.g. */person*, and */person/teacher*). The corpus consists of 3.2M Wikipedia articles, comprising 1.3G tokens that were annotated with 157.4M named entity mentions and their types. Leveraging WiFiNE into a joint vector space, we embed words and entity types. Via that, we compute for each word a 120-dimensional vector, where each dimension encodes the similarity of the word with an entity type. We call this vector a *LS representation*, for lexical similarity. For instance, the embedding for */product/software* will be trained using context words that surround all entities that were (automatically) labelled as */product/software* in Wikipedia. Figure 12 (taken from [11]) shows that combining words and entity types can become really helpful for word embeddings.

(v1) On **October 9, 2009**, the **Norwegian Nobel Committee** announced that **Obama** had won the **2009 Nobel Peace Prize**.
(v2) On /date, the /organization/government_agency announced that /person/politician had won the /award.

Figure 12: Example of the two variants of a given sentence

The FastText toolkit [38] is used to learn the uncased embeddings for both words and entity types. A *skip-gram* [22] model is trained to learn 100-dimensional vectors with a minimum word frequency cutoff of 5, and a windows size of 5. It is observed that mentions often annotated by a given type, tend to cluster around this entity type (e.g *"firefox"* close to */product/software*, or *"enzyme"* close to */biology*). Additionally, words that are labelled with different types, tend to appear between types they were annotated with (e.g.*"gpx2"* which can be considered both as software and as a gene, has its embedding between */product/software* and */biology*).

The lexical representation that is proposed, is computed offline, without adding any computation burden at test time. This representation encodes the preference of an entity-mention word for a given type, an information out of which binary gazetteer features are extracted.

4.1.3 NER system

For the purposes of this implementation, the popular Bi-LSTM-CRF [7] architecture system is adopted (Figure 13 taken from [11]), which is mostly used in many sequential tagging tasks. Lexical vectors are computed offline and are not adjusted during training.

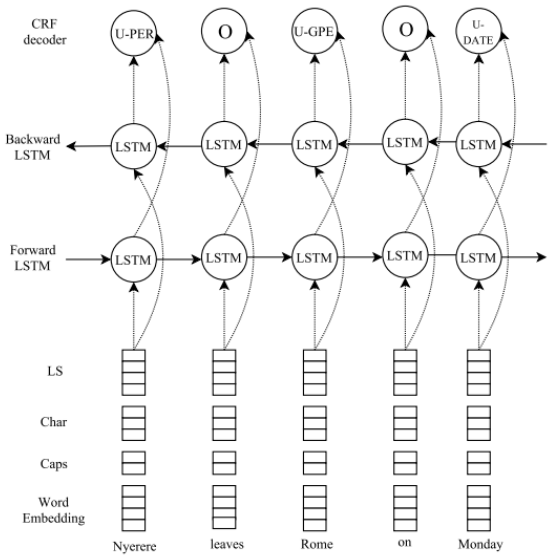


Figure 13: Main architecture of the NER system

In addition to the LS vector, the paper incorporates publicly available pre-trained embeddings, as well as character-level, and capitalization features. Publicly available word embeddings were also used for the experiments, such as Senna[17], Word2Vec [28], GloVe [19], and SSKIP [39], where the latter performs the best as indicated from the reports. SSKIP embeddings are 100-dimensional case sensitive vectors trained on n-skip-gram model. CoNLL and OntoNotes also use these embeddings, which are adjusted during training and report good performance.

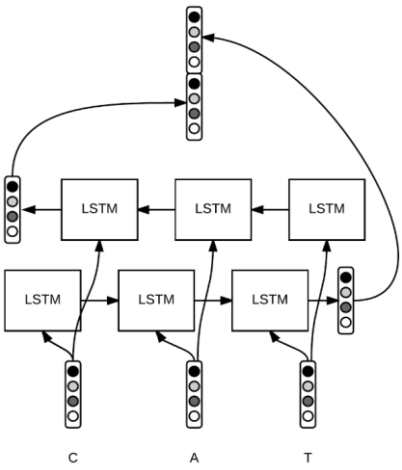


Figure 14: Character representation of the word "CAT" to the word-level Bi-LSTM

[17] https://ronan.collobert.com/senna/

A forward and a backward LSTM is used, to derive a representation of each word from its characters (Figure 14 taken from [11]). A character lookup is randomly initialized, then trained at the same time like the Bi-LSTM model.

In order to characterize certain categories of capitalization patterns, the system uses capitalization features (*allUpper*, *allLower*, *upperFirst*, *upperNotFirst*, *numeric* or *noAlphaNum*).

4.1.4 Experiments and evaluation

The main NER benchmarks where used with regards to the data: CONLL-2003 [18] and OntoNotes 5.0 [5]. For both datasets, the *IOB* encoding is converted to *BILOU* for better performance [49].

Table 3: Statistics of the CoNLL-2003 and OntoNotes 5.0 datasets

Dataset		Train	Dev	Test
CoNLL-2003	*#tok*	204,567	51,578	46,666
	#ent	23,499	5,942	5,648
OntoNotes 5.0	*#tok*	1,088,503	147,724	152,728
	#ent	81,828	11,066	11,257

CoNLL-2003 is annotated for four entity types: Person *(PER)*, Location *(LOC)*, Organization *(ORG)*, and Miscellaneous *(MISC)*. The four entity types are fairly evenly distributed, and the train/dev/test datasets present a similar type distribution. OntoNotes 5.0 dataset is annotated with 18 entity types, and is much larger than CoNLL, as shown in Table 3 (taken from [11]) and explained the task definition. Hence, more data is used for training and evaluation in comparison to CoNLL.

Table 4: System characteristics for CoNLL and OntoNotes datasets

Characteristic	**CoNLL-2003**	**OntoNotes 5.0**
Mini-batch	10	10
Learning rates	0.009	0.013

Hidden dimensions	128	256
Character Embedding	25	25
Forward and Backward character LSTMs hidden dimension	50	50
Capitalization embeddings dimension	25	25
Epochs	50	50

The implementation uses TensorFlow[18] library and the model run on a GeForce GTX TITAN Xp GPU.

The model characteristics for the training and implementation, are described below, combined with the different values applied for CoNLL and OntoNotes (Table 4):

- A mini-batch stochastic gradient descent (SGD) with a momentum of 0.9 and a gradient clipping of 5.0 is used for training.
- A dropout mask [40] is applied on the input / output vectors of the Bi-LSTM with a probability of 0.5, to migrate overfitting.
- The hyper-parameters are tuned by grid-search, and early stopping is used based on the performance on the development set.
- The applied variations are:
    - Dropout: [0.25, 0.5, 0.65],
    - Hidden units: [50, 128, 256, 300]),
    - Capitalization: [10, 20, 30],
    - Char embedding dimensions: [25, 50, 100],
    - Learning rate: [0.001, 0.015] by step 0.002,
    - Optimization algorithms,
    - Fixing the other hyper-parameters
- The training time took about 2.5 hours for CoNLL and 8 hours for OntoNotes.

[18] https://www.tensor-flow.org/

4.2 Named Entity Recognition with Bidirectional LSTM-CNNs

The authors of [4], focus on the need of feature engineering. A neural network architecture is presented which automatically detects word – and character – level features using a hybrid bidirectional LSTM and CNN architecture. A novel method of encoding partial lexicon matches in neural networks is proposed and is compared to existing approaches.

The published results of the paper show that the system is competitive on the CoNLL-2003 dataset. Using little feature engineering - two lexicons constructed from publicly – available sources, a new state-of-the-art performance is established, with a F1 score of 91.62 on CoNLL-2003 dataset and 86.28 on OntoNotes – surpassing systems that employ heavy feature engineering, proprietary lexicons, and rich entity linking information.

4.2.1 Method analysis

The whole approach is inspired by the work of [15], who proposed an effective neural network model that requires little feature engineering. More in details, we work with lookup tables that transform discrete features such as words and characters into continuous vector representations, which are then concatenated and fed into a neural network. Instead of a feed-forward neural network, the Bi-directional Long-Short Term Memory network (Bi-LSTM) is used (Figure 15 taken from [4]). In order to induce character-level features, a Convolutional Neural Network (CNN) is used, which has been successfully applied to Spanish and Portuguese NER, and German POS tagging.

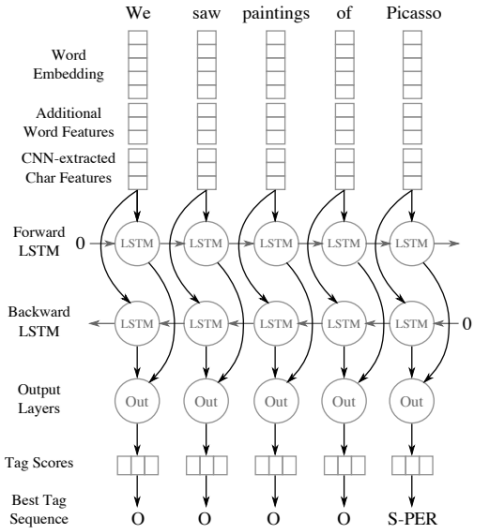


Figure 15: The (unrolled) Bi-LSTM for tagging named entities

The main steps that were applied to the model are described separately below.

Sequence labeling with Bi-LSTM

Deployment of a stacked bi-directional recurrent neural network, with long short-term memory units to transform word features into named entity tag scores. The extracted features of each word are fed into a forward LSTM network and backward LSTM network. *Linear* and *log-softmax* layers helped decoding the output of each network at every time step into log-probabilities. These two vectors are then simply added together to produce the final output.

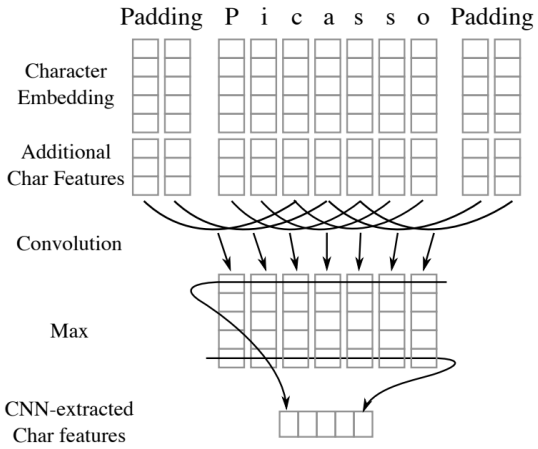


Figure 16: Example of a CNN extracting character features from each word

Character features extraction using CNN

Employing a *convolution* and a *max* layer to extract a new feature vector from the per-character feature vectors. Words are padded with a number of special *padding* characters on both sides depending on the window size of the CNN. The hyper-parameters of the CNN are the windows size and the output vector size. In Figure 16 (taken from [4]) we are able to see an example of the CNN with padding, for the word “Picasso”.

Word embeddings

Several publicly available word embeddings were used for the models as listed below:

- 50-dimensional word embeddings released by [15], which contributed the most for the best generated model.
- GloVe embeddings [19] trained on 6 billion words from Wikipedia and Web text, and Google’s word2vec embeddings [28], trained on 100 billion words from Google News.
- All words are lower-based before passing through the lookup table to convert to their corresponding embeddings. The pre-trained embeddings are allowed to be modified during training.

Character embeddings

A lookup table is randomly initialized with values from a uniform distribution with range [-0.5, 0.5] to output a character embedding of 25 dimensions. The character set includes all unique characters in the CoNLL-2003 dataset plus the special tokens *padding* (used for CNN) and *unknown* (used for all the other characters in OntoNotes).

Word-level capitalization feature

A separate lookup table is used to add a capitalization feature with the following options: *allCaps*, *upperInitial*, *lowercase*, *mixedCaps*, *noinfo*.

Word-level lexicons feature

For each of the four categories (*PER*, *LOC*, *ORG*, *MISC*), a list of known named entities is compiled from DBpedia[19], by extracting all descendants of DBpedia types corresponding to the CoNLL categories. Figure 17 (taken from [4]) below shows an example of how the lexicon features are applied.

Text	Hayao	Tada	,	commander	of	the	Japanese	North	China	Area	Army
LOC	–	–	–	–	–	B	I	–	S	–	–
MISC	–	–	–	S	B	B	I	S	S	S	S
ORG	–	–	–	–	–	B	I	B	I	I	E
PERS	B	E	–	–	–	–	–	–	S	–	–

Figure 17: Example of how lexicon features are applied

For each token in the match, the feature is encoded in *BIOES* annotations (Begin, Inside, Outside, End, Single), indicating the position of the token in the matched entry.

Additional character-level features

A lookup table is used to output a 4-dimensional vector representing the type of the character (upper-case, lower-case, punctuation, other).

4.2.2 NER system

The training and evaluation of the neural network are done on a per-sentence level. The initial states of the LSTM are zero vectors, and all lookup tables are randomly initialized with values drawn from the standard normal distribution.

The network is trained to define a tag-transition matrix A where $A_{i,j}$ represents the score of jumping from tag i to tag j in successive tokens, and $A_{0,i}$ as the score for starting with tag i. θ is defined as the set of parameters for the neural network, and $\theta' = \theta \cup \{A_{i,j} \; \forall i, j\}$ as the set of parameters to be trained. Figure 18 (taken from [4]) shows an example of the sum of network and transition scores.

[19] https://wiki.dbpedia.org/

$$S([x]_1^T, [i]_1^T, \theta') = \sum_{t=1}^{T} \left(A_{[i]_{t-1},[i]_t} + [f_\theta]_{[i]_t,t} \right)$$

Figure 18: sum of network and transition scores

The above objective function and its gradients can be efficiently computed by dynamic programming [15]. During the inference time, the Viterbi algorithm is applied in order to maximize the final score.

Training is done by mini-batch stochastic gradient (SGD) with a fixed learning rate. Each mini-batch consists of multiple sentences with the same number of tokens. Dropout is also used to the output node of each LSTM layer, as it becomes very effective in reducing overfitting.

4.2.3 Experiments and evaluation

The datasets that were used for the training, as mentioned above, are the CoNLL-2003 dataset and the OntoNotes 5.0. The sizes in number of tokens (entities) is shown in Table 3 (taken from [11]), as it the same one used in the previous analysis.

For all datasets, all digit sequences are replaced by a single *"0"*. In addition, before training, the sentences are grouped by word length into mini-batches, and then are shuffled. As CoNLL-2003 is small, compared to OntoNotes, the model was trained on both the training and development sets after performing hyper-parameter optimization on the development set.

Table 5: Hyper-parameter search space and final values

Hyper-parameter	CoNLL-2003 (Round 2)		OntoNotes 5.0 (Round 1)	
	Final	Range	Final	Range
Convolution width	**3**	$[3, 7]$	**3**	$[3, 9]$
CNN output size	**53**	$[15, 84]$	**20**	$[15, 100]$
LSTM state size	**275**	$[100, 500]$	**200**	$[100, 400]$[10]
LSTM layers	**1**	$[1, 4]$	**2**	$[2, 4]$
Learning rate	**0.0105**	$[10^{-3}, 10^{-1.8}]$	**0.008**	$[10^{-3.5}, 10^{-1.5}]$
Epochs[11]	**80**	-	**18**	-
Dropout[12]	**0.68**	$[0.25, 0.75]$	**0.63**	$[0, 1]$
Mini-batch size	**9**	-[13]	**9**	$[5, 14]$

As far as it concerns the hyper-parameter optimization, two rounds of optimization are performed, and based on the development set performance, the best settings are selected. Table 5 (taken from [4]) shows the final hyper-parameters, and Table 6 (taken from [4]) shows the dev set performance of the best models in each round.

Table 6: Development set F1 score performance of the hyper-parameter settings

Round	CoNLL-2003	OntoNotes 5.0
1	93.82 (± 0.15)	**84.57** (± 0.27)
2	**94.03** (± 0.23)	84.47 (± 0.29)

In the first round, random search is performed, and 500 hyper-parameter settings are evaluated on CoNLL-2003. Afterwards, with the same settings, the learning rate is being tuned on the OntoNotes development set.

In the second round, independent hyper-parameter searches are performed for dataset using Optunity's implementation which becomes more effective than random search. 500 hyper-parameters were also evaluated in this round.

The training takes about 6 hours for CoNLL-2003 and 10 for OntoNotes, whereas tagging set takes about 12 seconds for ConLL-2003 and 60 for OntoNotes.

4.3 End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF

The authors of [7] introduce a novel neural network architecture that benefits from both word-and-character-level representations automatically. This network uses a combination of Bi-LSTM, CNN, and CRF. The system doesn't require neither feature engineering not data pre-processing. Hence, it can be applicable to multiple sequence labeling tasks.

The system is evaluated on two datasets for two sequence labeling tasks. CoNLL-2003 dataset is used for NER, and Penn Treebank WSJ corpus, for part-of-speech (POS) tagging. State-of-the-art performance is obtained for both datasets, 97.55% for POS tagging, and 91.21% for NER. In this analysis, we will focus more on the approach targeted to the NER task.

4.3.1 Method analysis

Despite the multiple developments of NER systems, hand-crafted features are still hard to be replaced. The performance of every model can drop rapidly when it only depends on neural embeddings.

What is proposed, is a neural network architecture for sequence labeling. What is described, is an end-to-end model requiring no task-specific resources, feature engineering, or data pre-processing apart from pre-trained word embeddings on unlabeled corpora. This configuration makes the model applicable to a wide range of sequence labeling tasks, languages, and domains.

The main components used for the definition of the model are *CNNs* which are used for encoding character-level representation, *character - and word-level representations* which are fed into a *Bi-LSTM* to model context information of each word, and a *sequential CRF* which is used on top of Bi-LSTM, to jointly decode labels for the whole sentence.

4.3.2 NER system

In this section, the neural architecture of every component used in the model, is explained more in details.

CNN for Character-level representation

Figure 19 (taken from [7]) shows the CNN that is used for this approach and that helps extracting character-level representation of a given word. In this version of CNN, the authors use only character embeddings as the inputs, without character type features. A dropout layer [40] is applied before character embeddings are input to CNN.

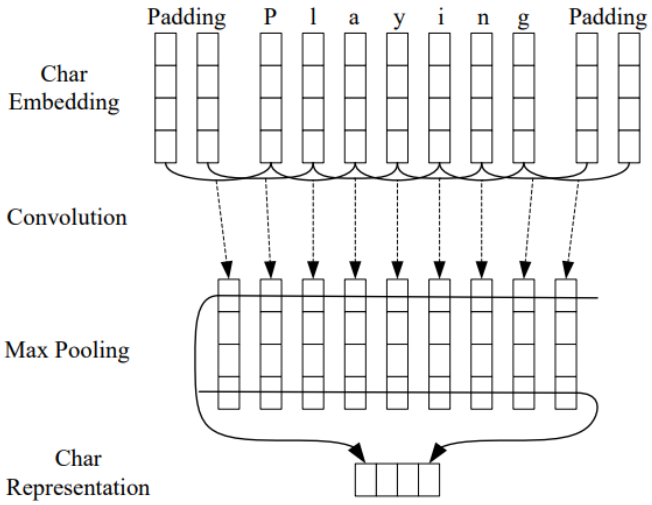


Figure 19: CNN for extracting character-level representations of words

Bi-directional LSTM & CRF

Figure 20 (taken from [7]) shows the basic structure of an LSTM unit.

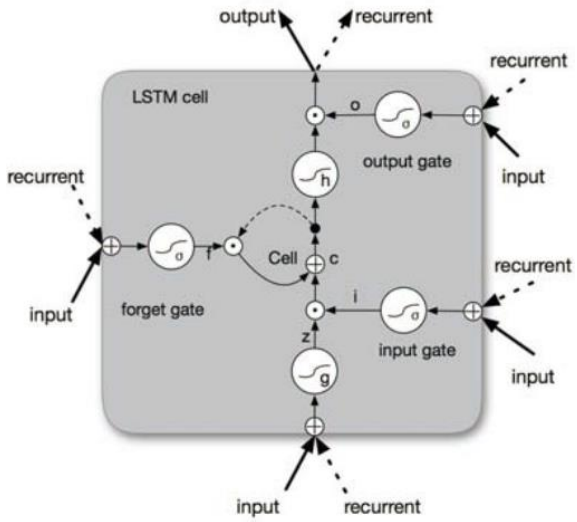


Figure 20: Schematic of LSTM unit

The basic goal for Bi-LSTM is to present each sequence forwards and backwards to two separate hidden states to capture past and future information, respectively. Then the two hidden states are concatenated to form the final output.

For CRF training, the maximum conditional likelihood estimation is used. Maximum likelihood chooses parameters such that the log-likelihood *(L(W, b)* is maximized. The logarithm of the likelihood is given by:

$$L(\mathbf{W}, \mathbf{b}) = \sum_i \log p(\boldsymbol{y}|\mathbf{z}; \mathbf{W}, \mathbf{b})$$

where $\{(z_i, y_i)\}$ is the training set. Decoding is to search for the label sequence *y** with the highest conditional probability. For a sequence CRF model, training and decoding can be solved efficiently by adopting the Viterbi algorithm.

The final constructed neural network model, is fed by the output vectors of Bi-LSTM into a CRF layer. The whole architecture can be displayed in Figure 21 (taken from [7]). The dashed arrows indicate the dropout layers that are applied on both the input and output vectors of Bi-LSTM.

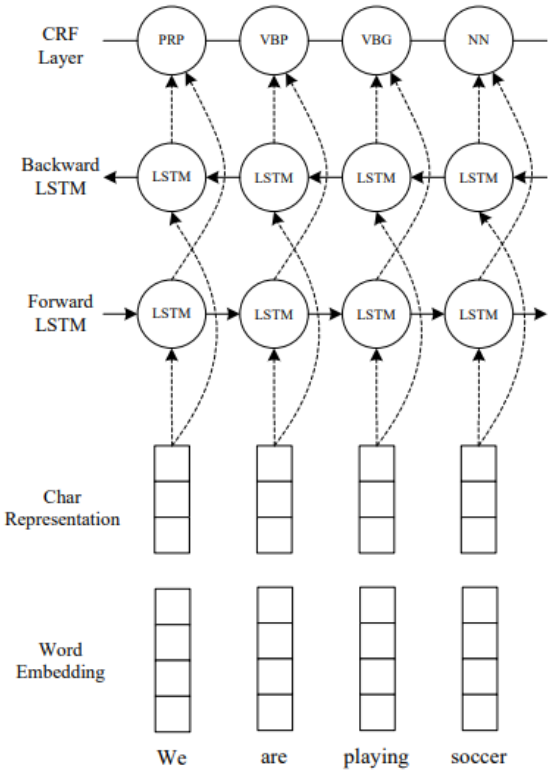


Figure 21: Main architecture of the neural network

4.3.3 Experiments and evaluation

In order to validate the effectiveness of pre-trained word embeddings, randomly initialized embeddings were used with 100 dimensions, sampled based on the analogy of the dimension of embeddings. Matrix parameters are randomly initialized following the approach of [47].

Mini-batch stochastic gradient descent (SGD) was used in order to optimize the parameters. Table 7 shows more details on the configuration of SGD.

Table 7: Optimization Algorithm Parameters

Field	Value
Batch size	10
Momentum	0.9
Learning rare	$\eta_0 = 0.015$ for NER
Gradient clipping	5.0

Additional methods that were used for better optimization are:

- **Early Stopping** [43]: It is based on performance on validation sets. Based on the presented experiments, 50 epochs can lead very close to optimal parameters.
- **Fine Tuning** [44]: The initial embeddings are fine-tuned, and are modified during gradient updates of the neural networks model, by back-propagating gradients.
- **Dropout Training** [40]: It is applied in order to control overfitting and to regularize the model. It is applied to character embeddings before inputting to CNN, and on both the input and output vectors of Bi-LSTM. The dropout rate is fixed at 0.5 for all dropout layers through all the experiments.

The tuning of hyper-parameters contributes as well to the algorithm optimization, via a random search. The hyper-parameters are almost the same for both the tasks of POS tagging and NER, except the initial learning rate. The state size of LSTM is set to 200.

For NER, the experiments are performed on the English data from CoNLL-2003 shared task [18] as mentioned above, containing the four main types of entities, PER, LOC, ORG, MISC. The BIOES tagging scheme is used as well here.

Table 8: F1 score on NER with and without dropout

	Train	Dev	Test
No	99.97	93.51	89.25
Yes	99.63	94.74	91.21

The neural network was implemented with the use of Theano library[20]. The computations ran on a *GeForce GTX TITAN X GPU* machine. The model training requires about 8 hours for NER, resulting to a F1 score of 91.21 using GloVe embeddings. Table 8 shows the final results with and without the use of dropout.

Conclusion

In this chapter, we focused on describing the workflow of the three different approaches we have chosen, with regards to NER, as instructed by the authors in their official publications. We provided the theoretical analysis that each approach follows and we presented the neural network components that define the implementation of the final model and conduct the training.

Finally, we presented the system configuration of each neural network, together with any possible adjustments that we need to be aware of. The published final scores for each one of the trained datasets are also presented in this chapter.

[20] https://github.com/Theano/Theano

5. EVALUATION RESULTS AND DISCUSSION

Having completed the analysis of each of the three studies in the previous chapter, Chapter 5, represents the main contribution of this thesis, by applying all the three techniques in practice, following the shared information provided by the authors of each study.

Each study, provides a publicly available *GitHub* link with the source code of the implementation for every study. We provide in details all the steps that were followed for the execution of the models, and we describe every adjustment that was required to be done in our system, in order to properly apply the training of each model.

Even though the studied and selected datasets (CoNLL-2003, OntoNotes 5.0), are the same for all the three cases, this evaluation uses only the CoNLL-2003 shared dataset, for the main reason that it is the only one openly available for usage. OntoNotes *5.0* on the other hand, is not taken into consideration, as its' content is not available to all the users in the web, and thus, it restricts us from analyzing the training using its' data.

We follow the steps and the tests to-be-performed as described in every repository, in order to replicate a similar procedure, so that we have a fair comparison among the results as presented in the paper, in the actual repository, and in our own implementation. Finally, we provide and evaluate the results of the experiments as generated in our own system, and we compare the validity of the execution and the F1 score.

The experiments were performed on a Windows machine of a 32GB usage RAM with NVIDIA HD Graphics P530 Quadro M2000M GPU. Given the fact that the machine specifications are not the same, as the ones used for each approach, we take the difference of results into consideration for our final conclusions.

5.1 System configuration

For the purpose of every implementation, a set of libraries needs to be installed. All the three repositories use *Python 3.x*[21] as the main programming language, and either

[21] https://www.python.org/

PyTorch[22] or *TensorFlow*[23] for the neural network procedures. Keras[24], which is running on top of *TensorFlow*, is also applied by some models, facilitating the usage of the neural network functionalities. Table 9 shows the detailed list for the setup of the libraries for every project.

Table 9: Version of installed libraries per project

# *name*	Project name	GitHub repository Name	Packages (among other dependencies)
Repo1	Robust Lexical Features for Improved Neural Network Named-Entity Recognition	NER-with-LS	python 3.7.0, keras 2.x, pyhocon, tensorflow / tensorflow-gpu 1.14
Repo2	Named Entity Recognition with Bidirectional LSTM CNNs	Named-Entity-Recognition-with-Bidirectional-LSTM-CNNs	python 3.6.8, keras 2.x, tensorflow 1.4.1, nltk, numpy
Repo3	End-to-end Sequence Labeling via Bi-directional LSTM CNNs CRF tutorial	End-to-end-Sequence-Labeling-via-Bi-directional-LSTM-CNNs-CRF-Tutorial	python 3.6.8, torch 1.2.0, numpy

As we see in the above table, we give an index number to each one of the studies. For the rest of our analysis, we will refer to each one of these repositories, as *Repo1*, *Repo2*, *Repo3* respectively, so that we are able to mention quickly the one needed.

The procedure that was followed for each of the three projects, is similar. After cloning each repository into our system, we installed the necessary libraries, as mentioned above, together with additional dependencies that turned up to be required during the execution. As a next step, and following the instructions of every project, we downloaded the data sources of CoNLL-2003.

[22] https://pytorch.org/

[23] https://www.tensor-flow.org/

[24] https://keras.io/

The replicated repositories which include the additional to-be-installed libraries, as well as some instructions on the initial steps to follow, are available in their respective repositories in GitHub.

5.2 Dataset annotations

Having cloned the ***Repo1*** GitHub repository, and having installed the technical requirements, which are present in our replicated version, it's time to download our data. We follow the steps as provided from the *README* instructions, and we navigate to the respective link to download the data. We move the data to the already existing */data* directory. The downloaded files are *conll.joblib*, used to provide a set of lightweight pipelining in Python, helping with an easy parallel computing[25], and *conlleval*, which represents a *Perl*[26] script that will be used for the evaluation of the model. What is currently missing from the directory, are the main data files of CoNLL-2003 for the language of English. As both of the next two studies (*Repo2*, *Repo3*) already provide the CoNLL-2003 dataset in their corresponding repositories, we decided to use their datasets for the case of *Repo1* too, given the fact that in *NER-with-LS*, the data was not included in the downloaded file. Plus, it is important to have the exact same dataset to train, aiming to a final fair comparison.

As already mentioned in previous chapters, this shared dataset, is already annotated using the tags PER, ORG, LOC, MISC which are encoded with the BIOES annotation scheme. In the CoNLL-2003 dataset, we observe that each tag is labelled with either the prefixed letter *B*, *I*, or *O*. *B-* denotes the beginning, and *I-* the inside of an entity. All other words, which don't refer to entities of interest, are labelled with the O- tag.

In the dataset provided for *Repo3* model, the annotated data labels a lot of tags with the prefix I- instead of B-, even if the annotated text indicates that the word to be annotated, is in the beginning of the text. In addition, the respective miscellaneous label is *-X-* instead of *O*. *Repo2* is properly set-up with the *B-* and *O-* prefix wherever is needed. The same structure of labels exists in all the three datasets for training, validation and testing.

[25] https://en.wikipedia.org/wiki/Parallel_computing

[26] https://www.perl.org/

The interest that is raised here, is to evaluate how much will this discrepancy affect the final result of each of the trainings. As a final decision, we train *Repo1* with both the two versions that are used in *Repo2* and *Repo3*. Furthermore, we train *Repo2* with the annotations of *Repo3*, and *Repo3* with the annotations of *Repo2* so that we are able to have our complete analysis between the studies. In each switch of annotated data, it is important to remember to rename the files, so that they can be recognized by each model appropriately.

5.3 Model implementation and training

5.3.1 Repo1 model configuration[27]

Continuing on *Repo1*, with the annotated data properly stored in the relevant directory, we check and confirm that the configuration file of the project (*experiments.conf*) has the correct paths setup for reading the data. Navigating in the main code where the data will be processed and the model will be trained, we realize that we need to rename our annotated data to *conll.train.txt*, *conll.dev.txt*, *conll.test.txt*, and there is no use anywhere of the suggestion of the authors to use the prefix ”*eng*”, instead of ”*conll*”.

In the same file, we also confirm the hyper-parameters configuration, regarding the embeddings dimension, the number of epochs, the batch size, and more.

First, we experiment the results of the model, by using the annotations as *Repo2*. We copy the annotated *train/valid/test* files into the respective */data/conll-2003* directory of *Repo1*, and we rename them accordingly.

The next step is the pre-processing of the data. The different annotations play a very important role here, as the part of pre-processing is to read the annotated data, and transform it from IOB to IOB2, or else BILOU encoding, as mentioned in the previous chapter during the analysis of this study. This will improve the total performance hence it is interesting to observe any possible differences due to the discrepancies of the two annotated datasets.

[27] https://github.com/vasilikivmo/repo1_ner_with_ls_ghaddarAbs

With the parameters configured as proposed by the authors, the model was trained within 3-4 hours. Changing the annotated data to the one used by *Repo3*, we pre-processed the data again, and ran the training once more. The training this time took about 3 hours. The final F1 scores for both annotations are shown in Table 10 for both *Repo2* and *Repo3* respectively.

Table 10: Training results on Repo1 using either the Repo2 or Repo3 annotated data

Dataset Annotations	F1-score DEV	F1-score TEST
Repo2	94.75	91.87
Repo3	94.91	91.75

We can confirm that the execution performed equally with what that was described in the paper analysis. The F1 score differs slightly from the values presented in the paper, for both the annotated datasets. In addition, given the different machine where the training was applied, difference can be skipped and we can assume, that the results are equally good as the paper presents.

5.3.2 Repo2 model configuration[28]

As mentioned in the main paper of ***Repo2***, the authors compare the results of the model against various dropout values for each dataset. The models are trained using only the training set for each dataset to isolate the effect of dropout on both dev and test sets. When using a dropout value of 0.50, the F1 score in test set for CoNLL is 91.14 (±0.35). Specifically, the model uses Bi-LSTM – CNN with embeddings and capitalization.

The repository instructions depend on this set of model configurations, and on top of it, they adjust part of the settings by:

- Not considering lexicons at all,

[28] https://github.com/vasilikivmo/repo2_Named-Entity-Recognition-with-Bidirectional-LSTM-CNNs_amalkraj

- Using bucketing to speed up the training, and
- Using *nadam* optimizer instead of *SGD*

Based on that, they present a final F1 score of 90.9% running on approximately 70 epochs for CoNLL-2003 dataset.

Figure 22 (taken from the repository of [4]) shows the actual constructed network model that was used, with the commands of the *Keras* API, which facilitated the implementation and ran on top of *TensorFlow*.

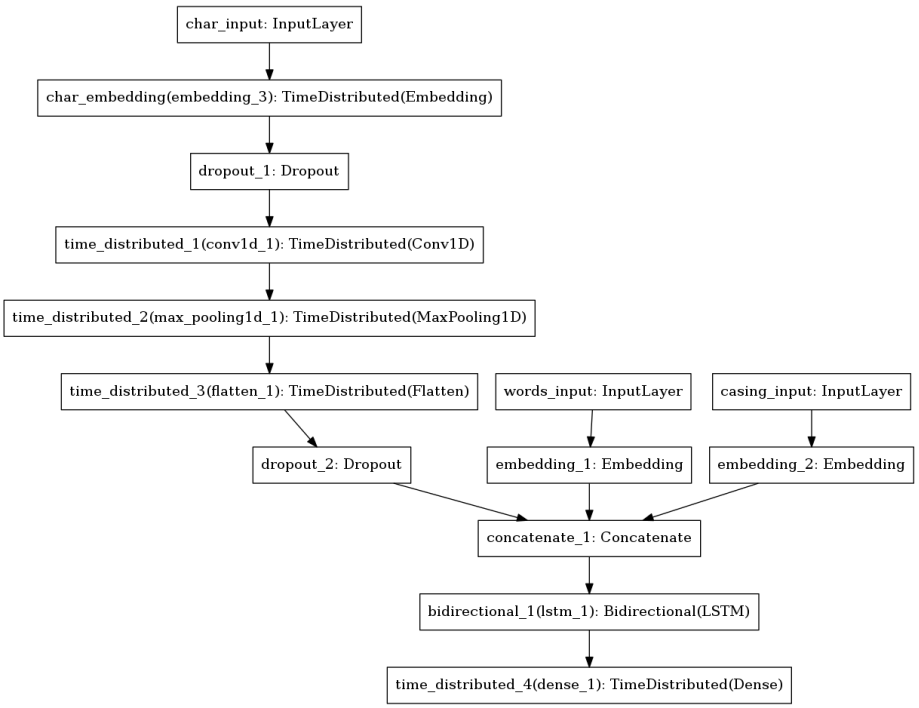


Figure 22: Network Model Constructed using Keras

Continuing to the analysis and implementation from our side, we first observe that the annotated dataset is already existing in the repository. So, it makes sense to use this one for our training immediately. However, we will run the model two times, in order to change the annotated data, to the other version as mentioned above. This will help having a complete overview of the effect of the two types of annotations to the three models. Following the repository's instructions, we first download the embedding files so that they are properly considered during the training.

Small adjustments needed to be made in the source files for the model, always by respecting the original configurations from the developers. We changed the number of

epochs from *50* to *70*, so that we are able to compare the final result with the one indicated in the description of the repository. In addition, we adjusted the part where the trained models are loaded, mostly to respect the packages versions, and to avoid having errors in the system, without causing any effect to the F1 scores.

The training of the model with its' original annotated data, took about 1 hour with approximately 70 epochs, with the flexibility to complete the training sooner on successful results. The F1 score on dev data was 92.7% and on test data 89.8%. Using the dataset annotated as the one from Repo3, the training took about 1 hour as well with approximately 70 epochs, and the final F1 score to be 93.1% on dev data and 90.1% on test data. Table 11 show all the results for both cases.

Table 11: Training results on Repo2 using either the Repo2 or Repo3 annotated data

Dataset Annotations	F1-score DEV	F1-score TEST
Repo2	92.7	89.8
Repo3	93.1	90.1

This indicates that the adjustments occurred in the repository can affect a lot the final result compared to the published papers, but on the other hand, the difference of the score, can explain the importance of the additional configuration settings that were skipped in the implementation. The repository declares a F1 score of 90.9%, even though in our implementation the final score is 89.8, where both are a bit less than the F1 score presented in the study, which is 91.14. We can obviously confirm that the score values are close to each other, which means that the model won't be influenced a lot in the actual entities prediction, but still, it is hard confirm the stability of these values, something that makes Repo2 less trust-worthy.

In addition to the training we tried to have a final inference on the validity of the model, by testing some main examples of named entities. Table 12 shows this evaluation.

Table 12: Inference table examples for Repo2 using both annotations

Sentence	Original annotations	Repo3 annotations
Steve went to Paris	Steve PER	Steve PER
	Paris LOC	Paris LOC
Steve Went to Paris	Steve PER	Steve PER
	Went PER	Went PER
	Paris LOC	Paris LOC
Jay is from India	Jay PER	Jay PER
	India LOC	India LOC
Donald is the president of USA	Donald PER	Donald PER
	USA LOC	USA LOC
European authorities fined Google a record $5.1 billion on Wednesday for abusing its power in the mobile phone market and ordered the company to alter its practices	European MISC	European MISC
	Google ORG	Google ORG
Dr. Doull from the Royal College of Paediatrics in Wales backed the Fresh Start	Dr. PER	Dr. PER
	Doull PER	Doull PER
	Royal ORG	Royal ORG
	College ORG	College ORG
	Of ORG	of ORG
	Paediatrics ORG	Paediatrics ORG
	Wales LOC	Wales LOC
	Fresh MISC	Fresh MISC
	Start **ORG**	Start **MISC**
The ceremony at Auschwitz culminated a week of events	Auschwitz MISC	Auschwitz **LOC**

around the world, including a commemoration in Jerusalem attended by dozens of world leaders, who urged collective vigilance against a resurgence of anti-Semitism worldwide	Jerusalem LOC anti-Semitism MISC	Jerusalem LOC anti-Semitism MISC
As a result, Russia was not invited to the anniversary of the liberation of Auschwitz, even though the Soviet Army liberated the camp.	Russia LOC Auschwitz MISC Soviet MISC Army **MISC**	Russia LOC Auschwitz MISC Soviet MISC Army **ORG**
NAEYC asked two researchers about what their work tells us about toys, children, and play.	NAEYC ORG	NAEYC ORG

5.3.3 Repo3 model configuration[29]

Following the instructions of the public repository provided by the authors, we were able to download the GloVe vectors, and the respective data files of the CoNLL-2003 dataset. *Pytorch* is one of the main plugins that require the execution and the training of the model, together with some more libraries, as shown in Table 9.

The repository contains a *Jupyter[30]* notebook file, which demonstrates step-by-step how to construct the model and implement the state-of-the-art Bi-directional LSTM-CNN-CRF architecture as described from the study analysis. This helps us become more familiar with the whole logic of the model, and more comfortable with the use of *PyTorch*. The essential parts that are covered through this implementation are:

- The preparation of the final dataset, which includes:
    - The pre-processing of the data, where all the digits in the words are replaced by 0, so that we are able to concentrate more on only important alphabets

[29] https://github.com/vasilikivmo/repo3_End-to-end-Sequence-Labeling-via-Bi-directional-LSTM-CNNs-CRF-Tutorial_jayavardhanr

[30] https://jupyter.org/

    - The update of the tagging scheme – The authors in the paper use the BIOES tagging Scheme, in opposition to the BIO that is used in the dataset. Thus, we convert the tagging scheme from BIO to BIOES
    - The creation of mappings for words, characters and tags
- The loading of the word embeddings (100-dimension GloVe vectors containing 6 billion words)
- The network definition, which includes:
    - The initialization of weights and the initialization scheme for the LSTM layers
    - The CNN encoder for character level representation
    - The Bi-directional LSTM for word level encoding
    - The CRF layers for output decoding
    - The addition of helper functions for further calculation of the score
    - The implementation of the Forward and the Viterbi algorithms
- The definition of the training parameters
- The training, evaluation and testing of the model

Accordingly, we create the respective file in the source code, containing all the parts that are necessary for the implementation. The only adjustments needed to be done, is to choose to train the model from the beginning, so that we generate our own results, without using the existing pre-trained model. We also add more examples at the last part of the testing, so that we can have a more thorough list of the performance of the model.

The training of the model took about 7 hours with the final F1 score being 92.3% on DEV dataset. The higher result than what the papers present, can be explained, from the continuous improvements on the model, as shown in the public repository. In addition to that, we can clearly see that the time of training, is less than what is presented in the paper, even though the machine we used is not qualified for such procedures.

Changing the annotated data to the one coming from *Repo2*, the training took about 7 hours as well, having a final F1 score of 93.1% on DEV dataset. Table 13 shows the final scores by using these two types of datasets. We can also view similar inference validity outputs of example sentences on Table 14.

Table 13: Training results on Repo3 using either the Repo2 or Repo3 annotated data

Dataset Annotations	F1-score DEV	F1-score TEST
Repo2	93.1	88.1
Repo3	92.3	87.3

Table 14: Inference table on Repo2 examples using both annotations

Sentence	Original annotations	Repo3 annotations
Steve went to Paris	Steve PER Paris LOC	Steve PER Paris LOC
Steve Went to Paris	Steve PER Paris LOC	Steve PER Paris LOC
Jay is from India	Jay PER India LOC	Jay PER India LOC
Donald is the president of USA	Donald PER USA LOC	Donald PER USA LOC
European authorities fined Google a record $5.1 billion on Wednesday for abusing its power in the mobile phone market and ordered the company to alter its practices	European MISC Google **PER**	European MISC Google **ORG**
Dr. Doull from the Royal College of Paediatrics in Wales backed the Fresh Start	Royal ORG Fresh **MISC**	Doull **ORG** Royal ORG Paediatrics **LOC** Wales **LOC** Fresh **ORG, MISC**

The ceremony at Auschwitz culminated a week of events around the world, including a commemoration in Jerusalem attended by dozens of world leaders, who urged collective vigilance against a resurgence of anti-Semitism worldwide	Auschwitz LOC Jerusalem LOC anti-Semitism **ORG**	Auschwitz LOC Jerusalem LOC
As a result, Russia was not invited to the anniversary of the liberation of Auschwitz, even though the Soviet Army liberated the camp.	Russia LOC Auschwitz **PER** Soviet **LOC**	Russia LOC Auschwitz **ORG, LOC** Soviet **ORG**
NAEYC asked two researchers about what their work tells us about toys, children, and play.	NAEYC ORG	NAEYC ORG

Having completed the training of all the three models, covering every case of different annotation data and respecting the configuration settings of each repository, we come to a conclusion that the results may vary across the different annotations, but to a level that the final model performance will be affected. Figure 23 shows a comparison of the final F1 scores of the TEST dataset, for all the three repositories, using either the annotated data of Repo2, or Repo3. We can see that indeed there is a difference, in the final F1 scores for each Repo, however, the range between our implementation and the paper's implementation remains approximately the same. This indicates, that all the three models can perform almost equally as presented by each study, and the focus only remains in using the appropriate parameters for an optimal tuning.

Comparing the inference validity examples though, we can see that the understanding of some words within a sentence might change (e.g. *Went* becomes PER instead of NA, *Google* becomes PER instead of ORG, *Auschwitz* becomes PER instead of ORG/LOC), which shows the importance of well pre-processed documents, with proper capitalization, and a good text structure. The annotations affecting the final output of the sentences can create a lot of confusion, and have a bigger impact to the final results of the model.

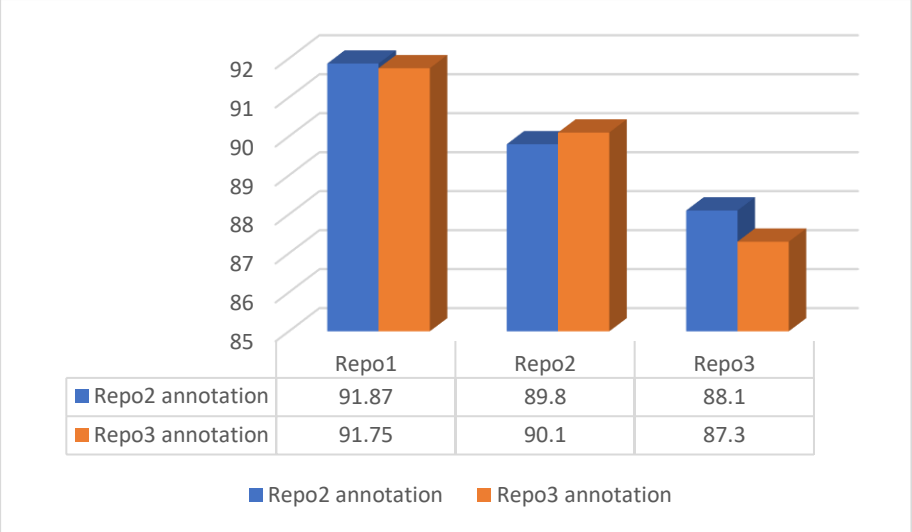
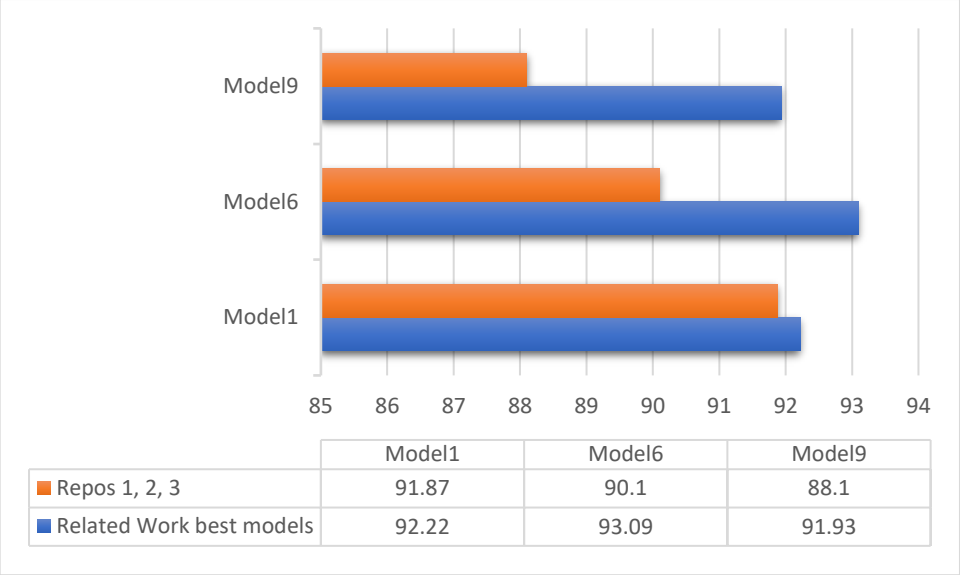


Figure 23: Comparison of F1 scores, testing two versions of annotated data

5.4 Final evaluation of the models

Having analyzed and re-produced the three studied papers and their respective implementations, we come to an understanding, that all the three evaluations can represent the main study followed by the authors. This makes the comparison across the three models to be even, so that we can focus only on the approach followed for each case, as described in the previous chapter. As a result, we observe that the use of lexical features can indeed be very useful, when we embed words and entity types into a low-dimensional vector space, as presented in *Repo1*. This approach using the same training dataset, seems to brings very clear improvements in the entity recognition, in comparison to the word- and character-level features detection using the Bi-LSTM and CNN architectures as explained in Repo2. Similarly, it seems to perform better than Repo3 as well, where the latter depends on a similar architecture as Repo2, combining Bi-LSTM, CNN and CRF techniques.

Figure 24 shows the evaluation of the three models, in comparison to the 3 best selected techniques, that we described in Chapter 3.



	Model1	Model6	Model9
Repos 1, 2, 3	91.87	90.1	88.1
Related Work best models	92.22	93.09	91.93

Figure 24: Comparison of the main repos with the best 3 models from related work

For the above comparison, we used the best F1 score as it was extracted for the three repositories, from either the annotated data on *Repo2*, or *Repo3*. We can observe that the performance followed by Model6 remains high in difference even in comparison with our own implementation. Learning to predict the next character on the basis of previous characters (contextual string embeddings) contributes to high F1 score results of 93.09 for CoNLL-2003. An equal comparison can only be made between Repo3 Model1 or Model 9, where Model1 focus on a new type of deep contextualized word representation, and Model9 focus on a general semi-supervised approach, for adding pre-trained context embeddings.

6. CONCLUSION AND FUTURE WORK

In this study, we reviewed and analyzed how Natural Language Processing (NLP) techniques are applied for Named Entity Recognition (NER) tasks. Deep learning seems to play a very important role on the implementation of neural network models, that aim to process and manage data from text documents, journals, papers. NER is one of the main tasks required for information extraction, which focus on identifying entities from unstructured text, and translating them into pre-defined sub-categories, like the main ones used, Person, Location, Organization, or Miscellaneous. Based on the existing studies and implementations, we investigated the current progress in NLP for NER usage. We followed publicly available documents and applications, and we explored the differences among them, through evaluation of the presented results.

As a start, we needed to cover any background information required, so that we can understand the concepts throughout the whole study. We described the main building blocks that compose our approach, GloVe embeddings, POS tagging, Recurrent Neural Networks (RNNS), Long Short-Term Memory Neural Networks (LSTMs), Bidirectional LSTMs (Bi-LSTMs), Conditional Random Fields (CRFs). As a next step, we presented our analysis on the related work with regards to current models, and the different techniques, that try to tackle the problem of the thesis. We reviewed and explained NER systems that use feature engineering, and hand-crafted features to train the model. We also explored how sequence labeling works on neural models, focusing on contextual string embeddings, hybrid semi-Markov CRFs, task-aware and bidirectional language models, and transfer learning.

For our main contribution for this thesis, we applied an experimental comparison of three alternative techniques for NER tasks. We explained the approach that each one of the studies followed, and what were the final results presented by the authors in each case. The first study that was analyzed, uses hand-crafted features, by embedding word and entity types into a low-dimensional vector space, trained by annotated data. The second study, focus on detecting word- and character - level features using a hybrid bidirectional LSTM and CNN architecture. Finally, the third approach, studies a similar technique of word-and-character-level feature engineering, by combining a bidirectional LSTM-CNN-CRF. We evaluated the F1 score of each of the three techniques by re-producing them. Regarding the dataset to be trained, we focused on the CoNLL-2003 shared task, and

with this type of annotations, we trained the models, extracting all the necessary information indicating the validity of each model, compared to the results of the paper.

The evaluation showed that the papers support the actual F1 scores, with small differences. For some of the repositories, we needed to make small adjustments, by adjusting the configuration settings, or by adjusting the source code so that it can be compatible with our machine. For some cases, we needed to respect the initial tuning of hyper-parameters, which could be slightly different from the one presented in the paper. As a final result, we can say that the use of low-dimensional vector space, creates an interest for further analysis, as it is an indicator of the best F1 score on the same annotated dataset.

Our future plans include further experiments and analysis on the current progress, focusing on re-producing some more techniques of interest, like the use of *contextual string embeddings* for sequence labeling [8], which proposes embeddings that are trained without any explicit notion of words, and that are contextualized by their surrounding text. As this related work seems to produce much higher results on the same CoNLL-2003 dataset, it raises a lot of interest to examine and evaluate it in comparison to our first study. This could provide the ability to combine these techniques, and potentially produce better results.

Furthermore, we could review potential improvements on the fine-tuning of the hyper-parameters for each one of the studies, which could improve the performance of the pre-processing of the data. On top of that, we could experiment by training all our models with the OntoNotes 5.0 dataset as well, in order to compare and evaluate the produced results with the published ones.

REFERENCES

[1] Zhixiu Ye, Zhen-Hua Ling, Hybrid semi-Markov CRF for Neural Sequence Labeling in: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, January 2018, Melbourne, Australia, pp. 235-240.

[2] Jie Yang, Yue Zhang, NCRF++: An Open-source Neural Sequence Labeling Toolkit, in: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL), July 2018, Melbourn, Australia.

[3] Liu Liyuan, Shang Jingbo, Xu Frank, Ren Xiang, Gui Huan, Peng Jian, Han Jiawei, Empower Sequence Labeling with Task-Aware Neural Language Model in: Proceedings of the Annual Conference of the Association for the Advancement of Artificial Intelligence (AAAI), September 2017.

[4] Jason P.C. Chiu, Eric Nichols, Named Entity Recognition with Bidirectional LSTM-CNNs in: Proceeedings of Transactions of the Association for Computational Linguistics, November 2015, pp. 357-370.

[5] Weischedel Ralph, Hovy Eduard, Marcus Mitchell, Palmer Martha, Belvin Robert, Pradhan Sameer, Ramshaw Lance, Xue Nianwen, OntoNotes: A Large Training Corpus for Enhanced Processing in: Proceedings of the Annual Conference on Computational Natural Language Learning (CoNLL), January 2011.

[6] Pradhan Sameer, Moschitti Alessandro, Xue Nianwen, Ng Hwee, Björkelund Anders, Uryupina Olga, Zhang Yuchen, Zhong Zhi, Towards Robust Linguistic Analysis using OntoNotes in: Proceedings of the Association for Computational Linguistics, August 2013, Sofia, Bulgaria, pp. 143-152.

[7] Xuezhe Ma, Eduard Hovy, End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF in: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, March 2016, Berlin, Germany, pp. 1064-1074.

[8] Alan Akbik, Duncan Blythe, Roland Vollgraf, Contextual String Embeddings for Sequence Labeling in: Proceedings of the Association for Computational Linguistics, August, 2018, Santa Fe, New Mexico, USA.

[9] Minghao Wu, Fei Liu, Trevor Cohn, Evaluating the Utility of Hand-crafted Features in Sequence Labeling in: Proceedings of the Annual Conference on Empirical Methods in Natural Language Processing, January 2018, Brussels, Belgium, pp. 2850-2856.

[10] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer, Deep contextualized word representations in: Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, June 2018, New Orleans, Louisiana, pp. 2227-2237.

[11] Ghaddar Abbas, Langlais Philippe, Robust Lexical Features for Improved Neural Network Named-Entity Recognition in: Proceedings of the Association for Computational Linguistics, June 2018, New Mexico, USA, pp. 1896-1907.

[12] Peters Matthew, Ammar Waleed, Bhagavatula Chandra, Power Russell, Semi-supervised sequence tagging with bidirectional language models in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, July 2017, Vancouver, Canada, pp. 1756-1765.

[13] Yang Zhilin, Salakhutdinov Ruslan, Cohen William, Transfer Learning for Sequence Tagging with Hierarchical Recurrent Networks in: Proceedings of the 5th International Conference on Learning Representations (ICLR), March 2017.

[14] Agerri Rodrigo, Rigau German, Robust multilingual Named Entity Recognition with shallow semi-supervised features in: Proceedings of the Journal in Artificial Intelligence, September 2016, pp. 63-82.

[15] Collobert Ronan, Weston Jason, Bottou Leon, Karlen Michael, Kavukcuoglu Koray, Kuksa Pavel, Natural Language Processing (almost) from Scratch in: Proccedings of the Journal in Machine Learning, Computing Research Repository (CORR), March 2011, pp. 2493-2537.

[16] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, Chris Dyer, Neural Architectures for Named Entity Recognition in: Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, June 2016, San Diego, California, pp. 260-270.

[17] Palshikar Girish, Techniques for Named Entity Recognition: A Survey in: Proceedings of the Journal on Bioinformatics: Concepts, Methodologies, Tools, and Applications, January 2012, pp. 191-.

[18] Erik F. Tjong Kim Sang, Fien De Meulder, Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition in: Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL, May 2003, pp. 142-147.

[19] Pennington Jeffrey, Socher Richard, Manning Christoper, Glove: Global Vectors for Word Representation in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), October 2014, Doha, Qatar, pp. 1532-1543.

[20] Turian Joseph, Ratinov Lev-Arie, Bengio Yoshua, Word Representations: A Simple and General Method for Semi-Supervised Learning in: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 2010, Uppsala, Sweden, pp. 384-394.

[21] Ling Wang, Luís Tiago, Marujo Luís, Astudillo Ramon, Amir Silvio, Dyer Chris, Black Alan, Trancoso Isabel, Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation in: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, August 2015, Lisbon, Portugal.

[22] Mikolov Tomas, Chen Kai, Corrado G.s, Dean Jeffrey, Distributed Representations of Words and Phrases and their Compositionality in: Proceedings of the Intetrnational Conference on Neural Information Processing Systems (NIPS), January 2013 pp.1-9.

[23] Yang Jie, Liang Shuailong, Zhang Yue, Design Challenges and Misconceptions in Neural Sequence Labeling in: Proceedings of the 27th International Conference on Computational Linguistics, June 2018, Santa Fe New Mexico, USA.

[24] Bengio Yoshua, Courville Aaron, Vincent Pascal, Representation Learning: A Review and New Perspectives in: Proceedings of the IEEE transactions on pattern analysis and machine intelligence, August 2013, pp. 1798-1828.

[25] Kotsiantis Sotiris, Supervised Machine Learning: A Review of Classification Techniques in: Proceedings of Informatica, the International Journal of Computing and Informatics, October 2007, Ljubljana, Slovenia.

[26] Agostinelli Forest, Hoffman Matthew, Sadowski Peter, Baldi Pierre, Learning Activation Functions to Improve Deep Neural Networks in: Proceedings of the International Conference on Learning Representations (ICLR), December 2014.

[27] Naili Marwa, Habacha Anja, Ben Ghezala Henda, Comparative study of word embedding methods in topic segmentation in: Proceedings of the International Conference on Knowledge Based and Intelligent Information and Engineering Systems, December 2017, Marseille, France, pp. 340-349.

[28] Mikolov Tomas, Corrado G.s, Chen Kai, Dean Jeffrey, Efficient Estimation of Word Representations in Vector Space in: Proceedings of the International Conference on Learning Representations (ICLR), January 2013, pp. 1-12.

[29] Jatav Vishaal, Teja Ravi, Bharadwaj Srini, Srinivasan Venkat, Improving Part-of-Speech Tagging for NLP Pipelines in: Proceedings of Computational Linguistics, August 2017.

[30] Zhang Peter, Neural Networks for Classification: A Survey in: Proccedings of the IEEE Transactions on Systems, Man, and Cybernetics, December 2000, pp. 451 - 462.

[31] Sazli Murat, A brief review of feed-forward neural networks in: Communications, Faculty of Science, University of Ankara, January 2006, pp. 11-17.

[32] Mikolov Tomas, Karafiát Martin, Burget Lukas, Cernocký Jan, Khudanpur Sanjeev, Recurrent neural network based language model in: Proceedings of the 11th Annual Conference of the International Speech Communication Association, INTERSPEECH January 2010, pp. 1045-1048.

[33] Kalchbrenner Nal, Blunsom Phil, Recurrent continuous translation models in: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), October 2013, pp. 1700-1709.

[34] Graves Alex, Mohamed Abdel-rahman, Hinton Geoffrey, Speech Recognition with Deep Recurrent Neural Networks, in: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP), January 2013, pp. 6645-6649.

[35] Schuster Mike, Paliwal Kuldip, Bidirectional recurrent neural networks in: Proceedings of the IEEE Transactions on Signal Processing, December 1997, pp. 2673-2681.

[36] Ratinov Lev, Roth Dan, CoNLL ’09 Design Challenges and Misconceptions in Named Entity Recognition in: Proceedings of the 13th Annual Conference on Computational Natural Language Learning (CoNLL), June 2009.

[37] Finkel Jenny, Grenager Trond, Manning Christoper, Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling in: Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics, January 2005.

[39] Bojanowski Piotr, Grave Edouard, Joulin Armand, Mikolov Tomas, Enriching Word Vectors with Subword Information in: Proceedings of the Transactions of the Association for Computational Linguistics, July 2016.

[39] Faruqui Manaal, Tsvetkov Yulia, Yogatama Dani, Dyer Chris, Smith Noah, Sparse Overcomplete Word Vector Representations in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing, June 2015, pp. 1491-1500.

[40] Srivastava Nitish, Hinton Geoffrey, Krizhevsky Alex, Sutskever Ilya, Salakhutdinov Ruslan, Dropout: A Simple Way to Prevent Neural Networks from Overfitting in: Proceedings of the Journal of Machine Learning Research, June 2014, pp. 1929-1958.

[41] Hochreiter Sepp, Schmidhuber Jürgen, Long Short-term Memory in: Proceedings of the Journal on Neural computation, December 1997, pp. 1735-1780.

[42] O'Shea Keiron, Nash Ryan, An Introduction to Convolutional Neural Networks in: Proceedings of National and Evolutionary Computing (NE), November 2015.

[43] C. Lee Giles, Steve Lawrence, Rich Caruana, Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping in: Proceedings of the Intetrnational Conference on Neural Information Processing Systems (NIPS), March 2001, pp. 402-408.

[44] Käding Christoph, Rodner Erik, Freytag Alexander, Denzler Joachim, Fine-Tuning Deep Neural Networks in Continuous Learning Scenarios in: Asian Conference on Computer Vision (ACCV) International Workshops, March 2017, pp. 588-605.

[45] Nothman Joel, Curran James R., Murphy Tara, Transforming Wikipedia into Named Entity Training Data in: Proceedings of the Australian Language Technology Workshop, January 2008, Hobart, Australia, pp. 124-132.

[46] Ghaddar Abbas, Langlais Phillippe, WiNER: A Wikipedia Annotated Corpus for Named Entity Recognition in: Proceedings of the 8th International Joint Conference on Natural Language Processing, November 2017, Taipei, Taiwan, pp. 413-422.

[47] Glorot Xavier, Bengio Yoshua, Understanding the difficulty of training deep feedforward neural networks in: Proceedings of the Annual International Conference on Artificial Intelligence and Statistics, January 2010, pp. 249-256.

[48] Bill G. Horne, C. Lee Giles, An experimental comparison of recurrent neural networks in: Proceedings of the 7th Intetrnational Conference on Neural Information Processing Systems (NIPS), July 1997.

[49] Ratinov Lev, Roth Dan, Design Challenges and Misconceptions in Named Entity Recognition in: Proceedings of the Annual Conference on Computational Natural Language Learning (CoNLL), June 2009.

[50] Sk Ahammad Fahad, Inflectional Review of Deep Learning on Natural Language Processing in: Proceedings of the 2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE), July 2018.

[51] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutnik, Bastiaan Steunebrink, Jurgen Schmidhuber, LSTM: A Search Space Odyssey in: Proceedings of the IEEE Transactions on Neural Networks and Learning Systems, October 2017, pp. 2222-2232