A Multilingual Named Entity Recognition System based on Fixed Ordinally-Forgetting Encoding

Firas Dib

February 5, 2018

Master's thesis work carried out at the Department of Computer Science, Lund University.

Supervisor: Pierre Nugues, pierre.nugues@cs.lth.se

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

This thesis describes a system whose goal is to find named entities in text. The system uses an encoding method, called the fixed ordinally-forgetting encoding, to efficiently encode variable-length text. We applied this encoding to words and characters and we used the resulting vectors as features. The system is language agnostic, and has been evaluated and tested on multiple languages.

The system uses annotated data, which is supplied by third parties, as the knowledge source. The system parses any given text and outputs a list of entities found in the text with the given entity class and position in the text.

The system achieved an F1 score of 90.31 in the shared CoNLL2003 English task. In the TAC2017 competition, the system achieved a F1 score of 75.4 for English.

Keywords: MSc, NER, TAC, CoNLL, FOFE, named entities

Acknowledgements

I would like to thank Marcus Klang for all of his help and rubber ducking with the development of this system. I would also like to give Pierre Nugues, my supervisor, a special thanks for this opportunity and help throughout.

Contents

Chapter 1
Introduction

The field of *natural language processing* (NLP) has always been one of the corner stones in artificial intelligence and has gained a lot of popularity the recent years. The goal is to enable a computer to understand natural languages, such as the ones we speak every day. Because of the inherent ambiguity of natural languages, it is consequently difficult for computers to understand it. This thesis will focus on a subfield of NLP, where the goal is to find, disambiguate, and link named entities to unique IDs.

This process is often abbreviated as N.E.R.D.: *named entity recognition and disambiguation*. The first part, recognition, is the process of finding mentions of entities in text (such as person names, locations, etc). The second part, disambiguation, is the process of determining what or whom this mention refers to. This is especially difficult for a computer to determine as it typically requires a lot of previously gathered information and context to be accurately determined.

1.1 Problem Definition

Before I explain what this project does and how it does it, it is important to understand the difficulties it tries to solve.

Most humans have no trouble distinguishing words representing the mention of a name, a location, or any other entity. We also have the ability to disambiguate words, which are otherwise ambiguous, with the help of context. By looking at the sentence structure and the surrounding words, we can, with high precision, determine what this named entity is. This is, in large part, due to our previous knowledge of how language is structured and the large knowledge base we keep in our brain.

1.1.1 Entity Recognition

Let us analyze the following sentence:

His father, an executive with Marshall Field, often travelled to his home town of Syracuse, New York, on business.

(Douglas, 1994)

We can quite easily distinguish both mentions *Marshall Field* and *Syracuse*. Many of us have previously stumbled upon the name *Marshall* and have thus no trouble immediately identifying it to be a named entity of type *person name*. However, the entity *Syracuse* might not be as familiar as *Marshall*. Due to how the sentence is structured we know the verb *travel* must represent someone or something traveling somewhere, and thus we have deciphered that *Syracuse* must be a location. We recognize these words as mentions of entities and, given the context, we can determine their type.

1.1.2 Entity Disambiguation

Not only do these systems have to find all named entities (even those never before seen), they also need to be able to distinguish between them, since most are not unique. Take the following sentence as an example:

Mr. Bennett lives in France because he loves Paris in the summer

(Scott, 1908)

Here, again, we find the three entities *Mr. Bennett*, *France*, and *Paris* easily as well as the types of them: a *person name* and two *locations*. For a computer however, this process is not arbitrary, since the word *Paris* is ambiguous. Does it refer to the capital of France, Paris from the greek mythology, or Paris Hilton (Fig. 1.1)? We as humans can determine that it being a person or mythological creature is highly unlikely because only liking them over certain season of the year is dubious.

This is where computers start to struggle and what ultimate goal of NLP, and this system in particular, aims to solve.

1.1.3 Multi-language

Since more and more systems are distributed world wide, they need to be localized to the region they are being sold. Systems such as Apple's Siri support a multitude of languages today, and need to keep doing so. Creating language-specific systems is no longer viable, as the cost of the development and upkeep would be far too great.

A named entity recognition system would thus benefit greatly from being language agnostic, as it would be a system that would be easy to use for people all over the globe, regardless of their preferred language.

As such, the goal of the system described in this thesis is to be language agnostic, and in particular, be able to comprehend four key languages:

- English
- Spanish
- Chinese
- Swedish

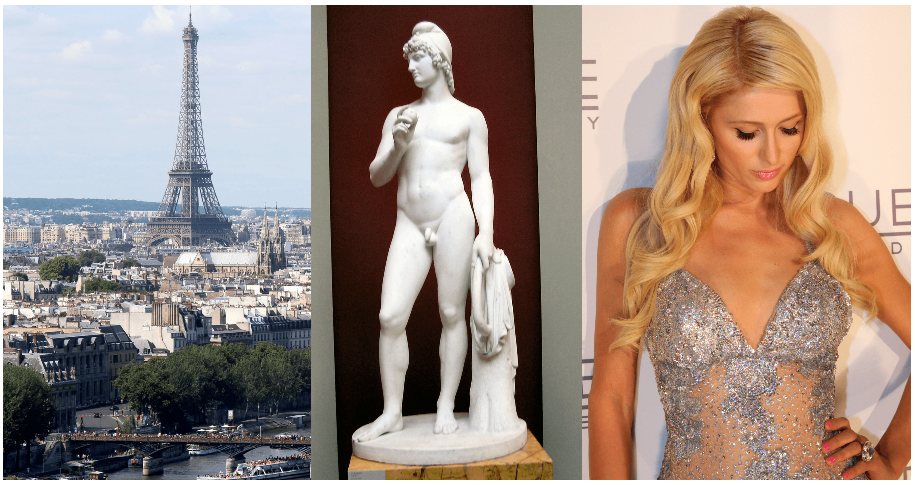


Figure 1.1: The ambiguity of Paris: the city capital of France, the mythological creature, and the person Paris Hilton. Images taken from Wikipedia.

1.2 Previous Work

The influential work of the CoNLL-2002 and CoNLL-2003 shared tasks (Tjong Kim Sang, 2002; Tjong Kim Sang and De Meulder, 2003) have become the defacto standard in the NER field of performance comparison. These tasks were originally aimed at language independent named entity recognition and have now instead been used as a benchmark.

Many of the best entity detection systems to this date use neural networks, and in particular, the long short-term memory (LSTM) network model. This architecture is especially good for variable input in which previous values influence future values, which is the case in named entitiy recognition. While these networks are extremely powerful, they are very demanding to train, and can take several days, even weeks, to finish. This slows down the development process which in turn affects experimentation and hyper parameter tuning, which both are crucial parts in neural network training and tuning.

Two state-of-the-art systems which use the LSTM architecture are those of Lample et al. (2016) and Ma and Hovy (2016). These systems are both based on the LSTM architecture, but also include convolutional neural networks (CNN) and conditional random fields (CRF). The great complexity involved in these systems is rectified with the great performance they showcase in many NER tasks.

In the TAC 2017 EDL competition there were 7 participants who used a system using neural networks based on LSTM, but with very different results. They ranked 1st, 2nd, 4th, 11th, 15th, 16th, and 21st, with a total F-score gap between the best and worst system of 24%. While there is no clear cut reason, a few possibilities might be hyper parameter tuning and training data.

This work is largely based on the previous work of Xu et al. (2017), where they used the same language model and similar network architecture as described in this thesis to find named entities in text.

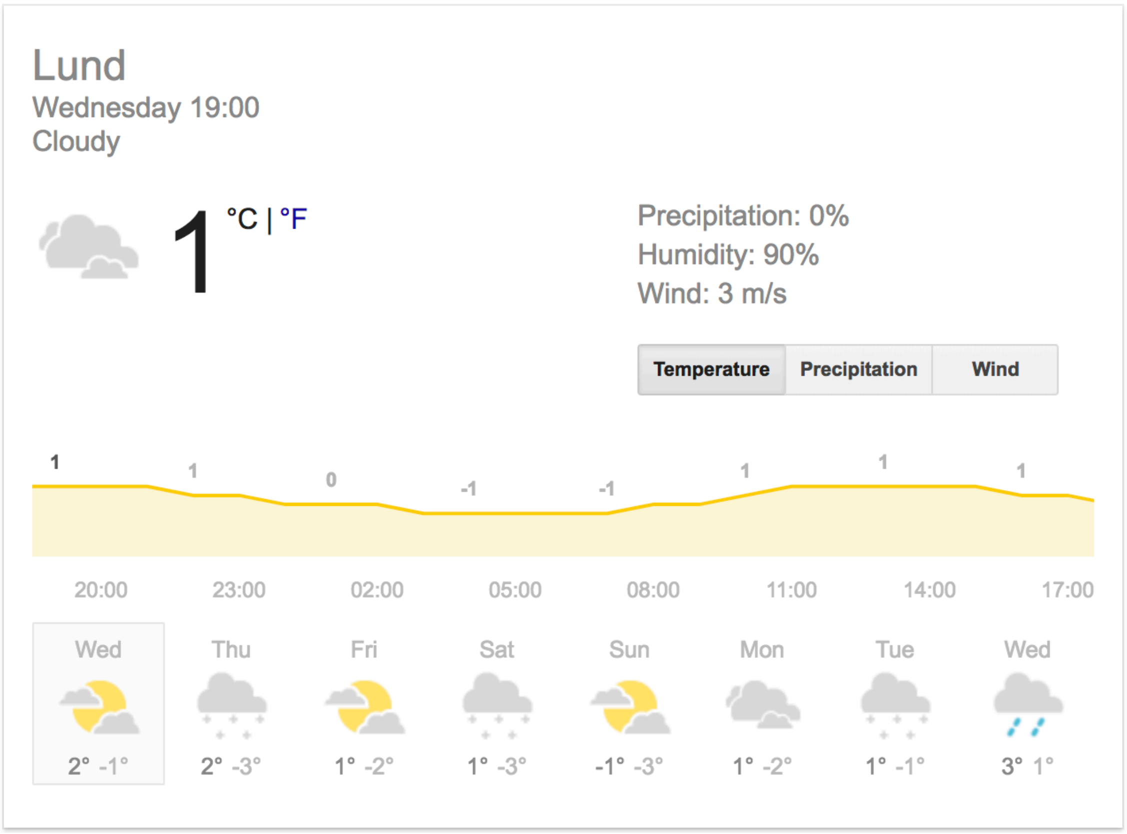
1.2.1 Text Analysis Conference

The Text Analysis Conference (TAC) (Ji et al., 2017; Ji and Nothman, 2016; Ji et al., 2015) is a series of evaluation workshops organized to evaluate the performance of NLP research. They provide a large test collection and a forum for organizations to share their results. It is the ultimate goal of this system to, in conjunction with other systems developed at Lund University, compete in this comprehensive task.

1.3 Applications

Many applications and devices we use on a daily basis make great use of NLP and NER in particular. It is most commonly found in search engines to try to disambiguate search queries and bring you the best results, but also in more recent technology like Apple's Siri and Amazon's Alexa. For all of these systems, it is critical that the computer properly understands what the user has input in order to produce relevant results, otherwise the system would be rendered useless.

Figure 1.2 shows an example from Google. Their system needs to understand what the subject of the query is, which in this case is *Lund*. After that it needs to classify the word, which in this case is a location. Finally it can determine the user wants to know what the weather is at a given location and the result is shown.


Lund
Wednesday 19:00
Cloudy

1 °C | °F

Precipitation: 0%
Humidity: 90%
Wind: 3 m/s

Figure 1.2: Example of named entity recognition and disambiguation on Google

1.3.1 Intended Usage

The system described in this thesis differs vastly from those previously mentioned as its purpose is only one: find named entities in text. The other systems have a larger goal in mind, such as delivering the best search results, finding the best translation, or answering a question, but are built on top of systems like this thesis describes.

As the system is intended to be language-agnostic it could ideally be used in many other applications

1.4 Contribution

This thesis contributes to the NLP research at Lund University with focus on entity extraction. It explores a promising method for multilingual entity extraction using no specific language features or other feature engineering.

Chapter 2
Approach

The goal of this thesis is to explore a method to structure natural language so a computer can understand it, and in particular, find named entities. This chapter will go into detail of how I have tackled this problem and which methods I have employed.

2.1 The Structure of Text

There are many different methods one could employ to perform the task of NER, as many have done. One thing most have in common is *context*. As John Rupert Firth elegantly put it:

> You shall know a word by the company it keeps (Firth, J. R. 1957:11)

By analyzing the context of words, we can often determine what they are, and that is the method employed in the system described in this thesis. The goal is not to teach a system specific answers, such as *Paris* is a city, but instead try to create a system that understands when *Paris* is found in conjunction with words such as *travel* or verbs of *going*, that it is a location.

2.2 Data Sets

In order to train an artificial intelligence (AI), we need to have data to do so. There are mainly two different methods of training an AI: unsupervised training and supervised training. In this paper, I will use the supervised training method, in which the model is trained with data which has been manually annotated or labeled with the correct answers.

Annotated data is a rare commodity and can cost several thousand dollars to get a hold of. Even then, the amount of data is often vary scarce. As if that was not enough, you will have a hard time using data produced by two different sources as it will often have been annotated with different rules. If the data is inconsistent enough, the algorithm will not learn anything meaningful and the results will be disappointing.

In this project, I have mainly utilized the data from SUCv3, CoNLL2002, and CoNLL2003 to benchmark my system against other systems, and data from TAC to train the final network used in the TAC2017 EDL competition.

The data is often split into two chunks: training set and evaluation set. This is so that you can evaluate the system to see how well it performs. During the development process you would train your system on the dedicated training data set and then evaluate on the evaluation data. This is done so that the evaluation data will be completely unseen to the system and thus test its true capabilities of classifying previously unseen data.

The annotated datasets follow a predefined syntax to facilitate parsing. Typically each line in the document is one word which has been tagged with *metadata*. All consecutive lines up until two consecutive new lines are considered one sentence. The words are often encoded as to easily determine which words form an entity and which don't. Two common encoding systems encountered are *IOB* (inside-outside-between) and *IOB2* (inside-outside-beginning). These encodings are very similar, with only a slight minor variation. In the *IOB* encoding scheme, each word which is a part of a named entity is encoded with an *I* unless the next word is part of another named entity of the same type, in which case it will be encoded with a *B*. In the *IOB2* scheme, the first word of a named entity is tagged with a *B* and all others with an *I*. In both systems, each word which is not a part of a named entity is tagged with an *O*.

2.2.1 CoNLL

The CoNLL2002 and CoNLL2003 data sets are great starting points as they have been used as benchmarks for many systems throughout the years. The data is also annotated in a clear and concise manner with few errors which gives your network the best chance of success. The data has also already been split into a train, test and development set which means your system will have the exact same prerequisites as all the other systems which have participated in this task.

The CoNLL2002 data uses the *IOB2* encoding system while CoNLL2003 uses *IOB*. See an example of this in Table 2.2, where the final column determines which entity type the word is. The possible entity types in this corpus can be found in Table 2.1.

Entity Type	**Meaning**
PER	Person name
LOC	Any location
ORG	Names of organizations
MISC	Entities which don't fit in the other groups
O	Not an entity (other)

Table 2.1: The available entity types in the CoNLL corpus.

Word	Part-of-Speech	Chunk	Named Entity
EU	NNP	I-NP	I-ORG
rejects	VBZ	I-VP	O
German	JJ	I-NP	I-MISC
call	NN	I-NP	O
to	TO	I-VP	O
boycott	VB	I-VP	O
British	JJ	I-NP	I-MISC
lamb	NN	I-NP	O
.	.	O	O

Table 2.2: Example of CoNLL2003 data

2.2.2 TAC

TAC has since 2014 held the EDL competition in which they have provided training and evaluation data to the participants. The competition was initially only focused on English, which means year 2014 only has training data for English, while year 2015 and 2016 have training data for all three languages: English, Spanish, and Chinese. Over the years, the structure of the documents has changed drastically, which made working with the data quite difficult. Because TAC uses a very specific set of rules to annotate their data, it was non-trivial to include data from other sources (such as CoNLL) which meant every piece data I could scavenge from the supplied documents was crucial. This data also contains nominal mentions which no other data set does. A nominal mention is a word or a group of words which refers to a named entity, which can be seen in Table 2.4 (index 12-13).

The data is given in an XML format along with a single file which contains information about which words, in which files, at what positions, are named entities. This is quite tedious to work with which is why we wrote a Python script which converted this scattered structure to a more pleasant format, that of CoNLL. An example of this can be seen in Table 2.4 where the second to last column is the entity type and the last column is the nominal type. This follows the IOB2 encoding system. The available entity types can be found in Table 2.3.

Entity Type	Meaning
PER	Person name
ORG	Names of organizations
GPE	Geo-political entities (such as countries)
LOC	Other locations
FAC	Facilities
TTL	Titles
O	Not an entity (other)

Table 2.3: The available entity types in the TAC corpus. Each entry has a named and nominal version, except that of *O* and *TTL*. TTL only exists for English documents and is equivalent to a nominal entity of person name.

Index	Word	Lemma	Named Entity	Type (Named/Nominal)
0	After	After	O	O
1	a	a	O	O
2	week	week	O	O
3	of	of	O	O
4	acrimonious	acrimonious	O	O
5	talks	talks	O	O
6	in	in	O	O
7	Brussels	Brussels	B-GPE#404	B-NAM#404
8	,	,	O	O
9	where	where	O	O
10	Tsipras	Tsipras	B-PER#405	B-NAM#405
11	dismissed	dismissed	O	O
12	proposals	proposals	O	O
13	from	from	O	O
14	the	the	O	O
15	lenders	lenders	O	O
16	as	as	O	O
17	”	”	O	O
18	blackmail	blackmail	O	O
19	”	”	O	O
20	,	,	O	O
21	the	the	O	O
22	40	{NUMBER}	O	O
23	-	-	O	O
24	year	year	O	O
25	-	-	O	O
26	old	old	O	O
27	prime	prime	B-PER#406	B-NOM#406
28	minister	minister	I-PER#406	I-NOM#406
29	said	said	O	O
30	parliament	parliament	B-ORG#407	B-NAM#407
31	would	would	O	O
32	meet	meet	O	O
33	on	on	O	O
34	Saturday	Saturday	O	O
35	to	to	O	O
36	approve	approve	O	O
37	holding	holding	O	O
38	a	a	O	O
39	referendum	referendum	O	O
40	on	on	O	O
41	July	July	O	O
42	5	{NUMBER}	O	O
43	.	.	O	O

Table 2.4: Example of TAC data after it has been ran through our script

2.2.3 Stockholm-Umeå Corpus

The Stockholm-Umeå Corpus (SUC) (Gustafson-Capková and Hartmann, 2006) is provided in a similar fashion to that of CoNLL: a training set, development set and test set. The data inside the files is also provided in a similar structure. The corpus is IOB2-encoded (like CoNLL2002) and a concatenation of the 3rd and 2nd column from the right gives the encoding and entity class. The available entity types in this corpus can be found in Table 2.5.

Entity Type	**Meaning**
person	Person name
place	Any location
inst	Organizations (and sometimes countries)
animals	Names of animals
myth	Mythological creatures
product	Products
work	Creations (e.g. a book)
event	Events
other	Entites which don't fit in the other groups
O	Not an entity (other)

Table 2.5: The available entity types in the SUC corpus.

Index	Word	Lemma	POS	POS	Morphological Features	Dep. Link	Dep. Label	Chunk Tag	Chunk Type	Named Entity Tag	Named Entity Type	ID
1	Gamla	gammal	JJ	JJ	POS|UTR/NEU|SIN|DEF|NOM	_	_	_	_	B	work	aa01a-005:26
2	testamentet	testamente	NN	NN	NEU|SIN|DEF|NOM	_	_	_	_	I	work	aa01a-005:27
3	kan	kunna	VB	VB	PRS|AKT	_	_	_	_	O	_	aa01a-005:28
4	fortfarande	fortfarande	AB	AB	_	_	_	_	_	O	_	aa01a-005:29
5	ge	ge	VB	VB	INF|AKT	_	_	_	_	O	_	aa01a-005:30
6	en	en	DT	DT	UTR|SIN|IND	_	_	_	_	O	_	aa01a-005:31
7	anvisning	anvisning	NN	NN	UTR|SIN|IND|NOM	_	_	_	_	O	_	aa01a-005:32
8	om	om	PP	PP	_	_	_	_	_	O	_	aa01a-005:33
9	bitterheten	bitterhet	NN	NN	UTR|SIN|DEF|NOM	_	_	_	_	O	_	aa01a-005:34
10	i	i	PP	PP	_	_	_	_	_	O	_	aa01a-005:35
11	konflikterna	konflikt	NN	NN	UTR|PLU|DEF|NOM	_	_	_	_	O	_	aa01a-005:36
12	i	i	PP	PP	_	_	_	_	_	O	_	aa01a-005:37
13	området	område	NN	NN	NEU|SIN|DEF|NOM	_	_	_	_	O	_	aa01a-005:38
14	-	-	MID	MID	_	_	_	_	_	O	_	aa01a-005:39
15	och	och	KN	KN	_	_	_	_	_	O	_	aa01a-005:40
16	om	om	PP	PP	_	_	_	_	_	O	_	aa01a-005:41
17	tidsperspektivet	tidsperspektiv	NN	NN	NEU|SIN|DEF|NOM	_	_	_	_	O	_	aa01a-005:42
18	.	.	MAD	MAD	_	_	_	_	_	O	_	aa01a-005:43

Table 2.6: Example of SUC data

2.3 Neural Networks

The system described in this thesis uses a deep neural network to achieve its goal. A neural network, or NN for short, is a classifier which means it is a method for assorting objects into distinct classes. In this case, the classes are different named entity types (person names, locations, organizations, etc), or possibly none.

The NN accepts a number of inputs, commonly referred to as *features* which in some shape or form describe the subject which is to be classified, and outputs a probability distribution containing all classes. Typically the highest probability class from the output is chosen if its greater than a certain threshold.

There are many different types of neural networks, but the most common are: feed-forward, recurrent, and convolutional neural networks. In this work, I have used the feed-forward variety, which has the benefit of being very simple, but not as powerful as it can only handle fixed length input.

2.3.1 Language Model

I applied a *fixed ordinally-forgetting encoding* (FOFE) (Xu et al., 2017; Zhang et al., 2015) as a method of encoding variable-length contexts to fixed-length features. This encoding method can be used to model language in a suitable manner for feed-forward neural networks without compromising on context length. It is derived from the *bag-of-words model* (BoW) in which text, such as a sentence or document, is represented as the *bag of its words*. This model completely disregards both word order and grammar, but retains the multiplicity of words.

In a document with the following two sentences:

(1) In January Peter travelled to Birzh for a meeting with Augustus.

(2) Peter travelled by ship from 9 to 13 october.

(Pushkin, 1999)

They are represented in the bag-of-words model as shown in Listing 2.1.

```
{
  "In": 1,
  "January": 1,
  "Peter": 2,
  "travelled": 2,
  "to": ,2
  "Birzh": 1,
  "for": 1,
  "a": 1,
  "meeting": 1,
  "with": 1,
  "Augustus": 1,
  ".": 2,
  "by": 1,
  "ship": 1,
  "from": 1,
  "9": 1,
  "13": 1,
  "october": 1
}
```

Listing 2.1: Example of the bag-of-words model. The number represents how many times each word has been encountered.

The FOFE model can be seen as a weighted BoW-model in which the same concept is used, but instead of only retaining the word count, we also retain the word position relative to a focus word. This is a great improvement since word order often has large influence of the meaning of words.

Following the notation of Xu et al. (2017), given a vocabulary V, where each word is encoded with a one-hot encoded vector and $S = w_1, w_2, w_3, ..., w_n$, an arbitrary sequence of words, where e_n is the one-hot encoded vector of the nth word in S, the encoding of each partial sequence z_n is defined as:

$$z_n = \begin{cases} 0, & \text{if n = 0} \\ \alpha \cdot z_{n-1} + e_n, & \text{otherwise,} \end{cases} \tag{2.1}$$

where the α constant is a weight/forgetting factor which is picked such as $0 \leq \alpha < 1$. The result of the encoding is a vector of dimension $|V|$, whatever the size of the segment.

Let's take two simple examples with the vocabulary $V = [A, B, C]$. Each word can be encoded using one-hot-encoding as $[1, 0, 0]$, $[0, 1, 0]$, and $[0, 0, 1]$ respectively. With the sequence *ABC*, the encoding becomes $[\alpha^2, \alpha, 1]$, while the sequence *ABBAC* is encoded as $[\alpha^4 + \alpha, \alpha^3 + \alpha^2, 1]$.

2.3.2 Word Embeddings

Word embeddings are a type of representation of words that allow words with similar meaning to have a similar representation. Each word is represented as real valued vectors in a predefined vector space of typically 50 to 500 dimensions, which are randomly initialized and learned through some method, such as *word2vec*. This also has the benefit of reducing the dimensions of the BoW representation.

Word2vec is a tool developed by Mikolov et al. (2013), which is used to produce said word embeddings by using a shallow two-layer feed-forward neural network. It takes as its input a large corpus of text and produces a vector space of several hundred dimensions, with each unique word being assigned a corresponding vector in this space. As the network is being trained, the vectors are repositioned such that words that share common contexts in the corpus are located in close proximity to one another in the space.

2.3.3 Training the Network

Training a neural network is not an easy task, since there are many hyper parameters available for tuning. I used an iterative process where I would adjust one hyper parameter at a time and then see how the system performed over a few epochs. Since an epoch only took about one minute to finish it was not too bad, even though a lot of time was wasted waiting.

To aid with understanding the intricate details of the network, I used the Deeplearning 4 java (DL4J) Web UI (Gibson and Patterson, 2017) which, among other things, shows how well the system is learning and the magnitude of the weight adjustments for each layer. Figure 2.1 shows an early stage in the development process where the network was still not stable.

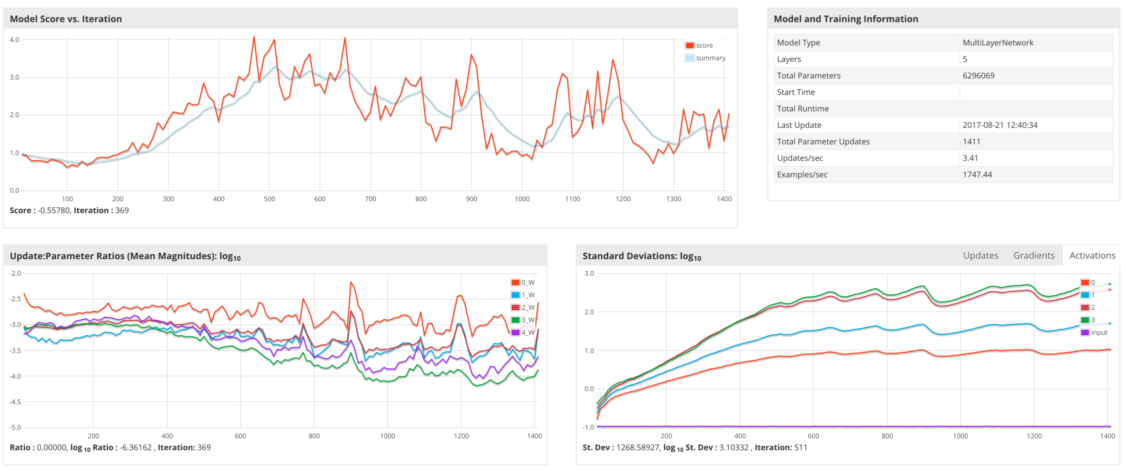
The image displays four independent boxes of information:

Top Left: This graph is showing the *score*, which is the result from the loss function. This measure is used to evaluate how well the network is learning and should be trending downwards. The image is showing quite a bit of fluctuation which indicates the network is not learning as well as it should, which could be because of poor features or hyper parameter selection.

Top Right: This table is just displays general information about the network.

Bottom Left: This graph shows with which *magnitude* each layers parameters are being updated. This is a measure of how much the network is learning. A good rule of thumb is that this number should hover somewhere around $1E-3$ (Gibson and Patterson, 2017). From the image one can deduce that the network is starting out well but slowly deteriorates over time with some weird spikes. This could indicate poor feature choice or weird input data.

Bottom Right: Finally this graph is showing the standard deviation of the *activations* for each layer. This gives an insight to how large the activations within the network are, which can help identify issues such as neurons dying off (activations becoming 0) or activations exploding. A good value to aim for is somewhere around -1 and 1 in the graph (Gibson and Patterson, 2017). Values growing too large (as seems to be the indication in this case) could be due to poorly normalized input.



Score : -0.55780, **Iteration :** 369

Model and Training Information

Model Type	MultiLayerNetwork
Layers	5
Total Parameters	6296069
Start Time	
Total Runtime	
Last Update	2017-08-21 12:40:34
Total Parameter Updates	1411
Updates/sec	3.41
Examples/sec	1747.44

Ratio : 0.00000, **log $_{10}$ Ratio :** -6.36162 , **Iteration:** 369

St. Dev : 1268.58927, **log $_{10}$ St. Dev :** 3.10332 , **Iteration:** 511

Figure 2.1: A screenshot of the DL4J UI showing an early stage debugging session of the network.

2.3.4 Feature Selection

A lot of thought has gone into feature selection as this is ultimately what the system will use to base its prediction on. If the features are poor, the classification will not be any better.

The goal is to present the system with as much information about the sentence as possible so it will be able to pick up on distinguishing features. In order to do so, I have decided to include two different feature classes: word and character features. The word features will model the entire sentence, including the focus words, while the character features will only model the focus words. This is done both case-sensitive and case-insensitive.

2.4 The Development Process

When I started this project, I had little to no knowledge of how systems like these operated, and let alone the technologies I would come to employ. But as with any unknown problem, the first step is research followed by creating a baseline.

The process started by reading the paper by Zhang et al. (2015) and trying to understand the techniques they had employed and how their system worked. The baseline was in turn largely based on their system and their findings alone. This was an iterative process where I would re-read the paper and modify my system, trying to mimic their results and using their result as guidance for whether my system was functioning properly or not.

A large portion of the time was devoted to research and trial and error. Before I could begin implementing the system I would need to learn how a neural network worked. I started by implementing a simple tax, the XOR-problem. To solve this simple problem I wrote everything from scratch myself, this helped me get to terms with the mathematics and understand the vocabulary used in this field of science.

Once I felt I had a good understanding of neural networks, I went through the paper again and highlighted all the terms I was unsure about, and did my research on them. This included understanding word embeddings, what they were, why they were used and how they helped solve this task. This of course included reading another paper, this time a paper about word2vec. There were a lot of hard concepts to grasp here but eventually it clicked.

Finally when I had a more solid ground to stand on, I started developing the actual system. This went through hundreds of iterations before it started performing adequately, and during this time I was iteratively learning new things and going through all the previous material.

To evaluate the system, I employed both manual and statistical methods. I would use Matlab to plot different aspects of the system and analyze the curves, as well as evaluating against manually annotated data and comparing key values. These key values were precision, recall and F1 score, which I calculated myself as well as using third party tools. The results were then written down and kept track of, in conjunction with the features and other valuable information used to achieve it, in order to draw conclusions as to which configurations worked best. One of the most valuable tools were that of the framework I used (deeplearning4java) which has a built in user interface that shows in depth details about the neural network. This shows the magnitude of the activations, how much the network is learning, and much more. This was a crucial tool to analyze the network and find which ideas worked better than other.

Chapter 3
Implementation

The NER system is a feed-forward neural network which takes as input a sentence and outputs the highest probability class. Multiple features are used, both on the word and character level. The network is language-agnostic, which means it can be trained to find named entities in any language given there is data for it to be trained on. This system has so far been successfully used to find named entities in English, Spanish, Chinese and Swedish.

3.1 Text Parsing

Each dataset used to train has been tokenized in a certain way, typically not identical to one another. To facilitate this I created a simple interface to be able to plug in custom parsers to handle all the different data sets in a simple manner. Typically data sets were split up on a sentence basis, which is what I used to feed the network as well. It would not be completely unreasonable to use multiple sentences as input to the network as previous sentences might have relevant context. I have elected not to do this because I wanted to explicitly model the beginning and end of sentence and keep this information constant across all training samples.

3.1.1 Annotation Rules

The different data sets use widely differing annotation rules which makes using them in conjunction hard or in some cases even impossible. The CoNLL data does not contain any overlapping mentions, while the data supplied by TAC does. An overlapping mention is when the same word creates two different entities. When an overlapping mention is found, the system will simply add it as a variant and allow the network to train on it as well.

Everest Base Camp and other treks in the area – despite significant damage to lodges the main trails to EBC are open to trekkers.

In the quote above, taken from the TAC2016 data, the word “Everest” is tagged as a location, while “Everest Base Camp” is tagged as a facility, thus creating an overlapping mention. Out of a total of 8286 annotated mentions in the TAC2016 English data, there are only 146 overlapping mentions (~1.76%).

An experiment I carried out when training for the TAC2017 ERD competition was to convert CoNLL2002 Spanish data into a TAC-compatible format, as we had very little data to train on in this language. We went over each tagged mention in the CoNLL corpus and looked up which was the most likely mention class on wikidata and used that, and ensured it conformed to the TAC rules. The data set nearly tripled in size, but unfortunately the results were worse than when we trained with only the TAC data. This is likely due to the fact that the different annotation rules are clashing, which results in inconsistent data being fed to the network, which in turn just ends up confusing it.

3.2 Neural Network

The system uses a simple feed-forward neural network which is trained from labeled data (supervised learning). In recent years neural networks have shown great promise in solving tasks of this nature and most of the papers being released on the subject are doing just that. However, almost none use the network type I have elected to use, which is a feed-forward neural network.

This network type is the most basic and straightforward one to implement, but has its limitation of not being able to accept variable width input. This means that regardless of the size of the sentence being fed to the network, it needs to become a fixed size before being fed into the network. Other network types, such as recurrent neural networks (RNNs), do not have this limitation which is often why they are been favored for tasks of this nature.

3.2.1 The Architecture

The architecture is based on the previous work of Xu et al. (2017) and best practices recommended by the CS231n[1] course by Stanford University.

The network architecture can be described as two parts; the first part converts sparse input into dense, while the second does the actual classification. The first part of the system is not necessarily required and can be computed outside of the network, but I have found that keeping them as a part of the network helps the system learn further and allows it to back propagate to a more finely tuned solution.

3.2.2 Feature Projection

As previously mentioned the first part of the network is responsible for taking a one-hot sparsely encoded input and projecting it into a dense representation.

[1]http://cs231n.stanford.edu/

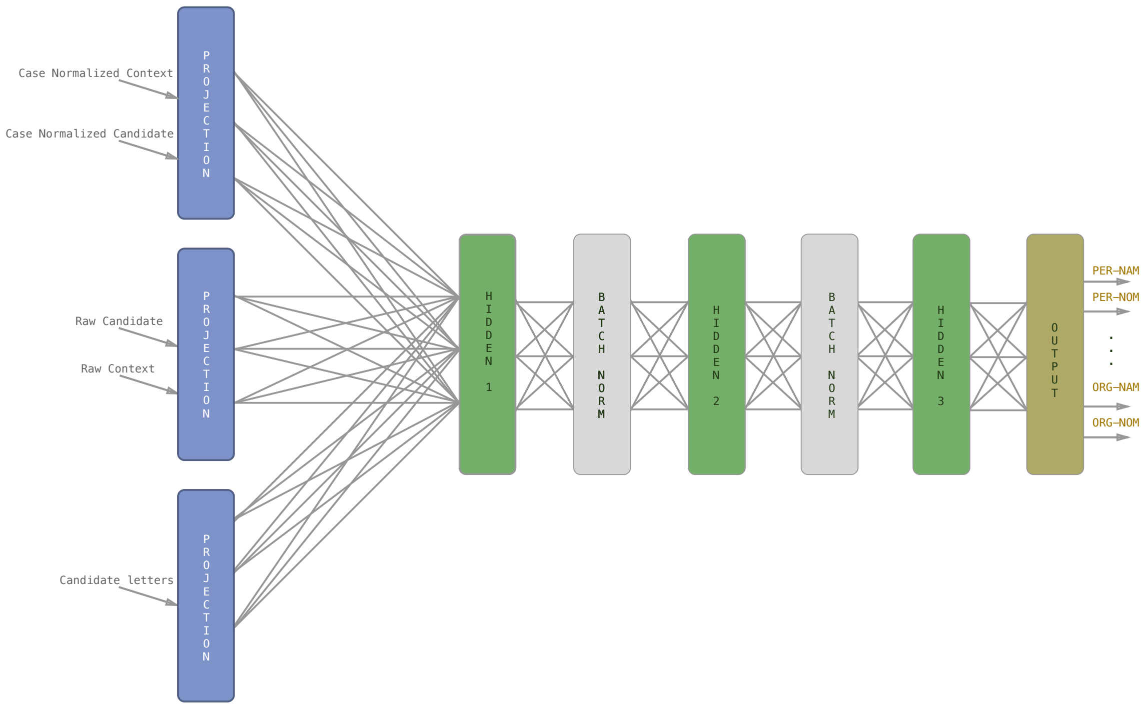


Figure 3.1: Overview of the full network architecture

The projection is done through a simple layer which is initialized with the weights from a previously trained word2vec model, commonly referred to as a projection or embedding layer. The word2vec model has been trained outside of the network on an extremely large corpus for many hours, such as the entirety of wikipedia or the Gigaword corpus. Once training has finished we have obtained a model that will convert a sparse one-hot encoded input into a dense vector of N dimensions.

There are many benefits to doing this projection, some of which are:

- Reduce the amount of mathematical operations.
- Create a more dense representation of the data.
- The output vectors from the word2vec model will give us a better representation of the words and similar words will have similar vectors, which gives the network a better intuition about them.

The reference C implementation for word2vec was used with adjusted hyper parameters, which can be found in Table 3.1. Unless explicitly listed in the table, the default value was used.

3.2.3 Classification

The second part of the network is where all the grunt work is done. These layers are what will eventually classify a given word or fragment as a certain class. The classifier is made up of three hidden layers with batch normalization layers in between.

Parameter	**Value**
Hierarchical Softmax	Enabled
Window Size	10
Negative Examples	10
Iterations	5
Size	256
Sample	1e-5
Continuous Bag of Words	Disabled (use Skipgram)

Table 3.1: Customized word2vec hyperparameters

When classifying a sentence the system will exhaustively classify all fragments possible in the string. The fragments are generated by using a sliding window technique over the sentence. This algorithm goes through each sentence, word for word, with a sliding window of increasing width. It starts off at 1 in width and increases to a max size of 7. This value was determined by looking at the statistics of how many mentions of a given size existed in the TAC data (see table 3.2). The majority of them (99.9%) were 7 or fewer words. The reason for this limitation is because for each positive sample we generate a negative one, and with a too large window, there will be too many irrelevant negative samples which we have to choose from. This would skew the data sampling process negatively.

The sliding window algorithm is very simple to both implement and visualize. An example of the algorithm in action can be seen in Listing 3.1.

```
The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.

⋮

The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.
```

Listing 3.1: Example sentence. Each bold/underlined chunk becomes a fragment.

Only fragments which only consist of words that form a complete named entity are kept; all other are either tagged as disjoint or, if they are overlapping with an actual mention, overlapping negative examples. For example, `University of Lund` is one mention, but the algorithm will return multiple candidates: `University`, `University of`, `of`, `of Lund`, `Lund` which all consist of words which are tagged as mentions, but will be converted into negative overlapping examples since they form the complete named entity which in this case is `University of Lund`.

The output is a probability distribution for each class, whose sum is 1. The class which had the highest probability was selected and kept only when the probability was greater than a fixed cutoff value. If there was no class greater than this threshold, the fragment would be assigned to the `NONE` class.

Two different methods were created and evaluated against each other for this task: The highest probability and the longest match:

Size	Count	Approximate total %
1	27088	87.9
2	3070	10
3	468	1.5
4	114	0.37
5	39	0.13
6	30	0.1
7	12	< 0.04
8	5	< 0.02
9	4	< 0.01
10	1	< 0.01
11	2	< 0.01

Table 3.2: The mention size distribution (in words) for TAC2015 english corpus.

- The highest probability algorithm orders the list of fragments first by highest probability, then by fragment length.
- The longest match algorithm first orders the fragments first by their length, then by their probability.

The complete algorithm is described in Listing 3.2.

```
Collect all potential fragments
Run one of the two sorting algorithms

for each sorted fragment:
  Ensure probability > threshold:
    Then ensure fragment does not overlap with any other fragment:
      Set the entity class of this fragment to the predicted class
    Otherwise:
      Set the entity class of this fragment to NONE
  Otherwise:
    Set the entity class of this fragment to NONE
```

Listing 3.2: Fragment pruning algorithm.

During testing, the highest probability algorithm produced the best results, a few points greater than the longest first. The output was also visually cleaner upon manual inspection. We did a grid search for the cutoff value and found that 0.5 produced the best results. Nonetheless, both 0.4 and 0.6 yielded similar results and would be reasonable choices as well.

3.3 Training the Network

In order to train the network, I needed to create features in a sufficiently fast manner to not have the training process be bottle-necked by the feature creation. I created a very simple class that easily allowed me to turn features on or off to carry out experiments to see what worked and what did not.

During training we use pre-tokenized sentences which are run through the feature generation code and fed to the network for classification. This is done in batches of size 500.

3.3.1 Word-level Features

The word-level features use bags of words to represent the focus words and FOFE to model the focus words as well as their left and right contexts. As context, I used all the surrounding words up to a maximum distance, defined by the floating point precision limits using the FOFE α value as a guide. This was based on the fact that every number in a neural network is represented by a float, which consequently means every mathematical operation will use a float. Floats are used instead of doubles to reduce the amount of system memory required.

By exploiting the fact that floating point numbers only can represent a finite amount of decimals, we can reduce the amount of work the system has to put into generating features by finding the maximum context size. To do this, we simply need to calculate to which power we may raise α before the value will be rounded off to zero. A floating point can represent values up to around 10^{-37} (cppreference.com, 2017), which means we simply need to solve the following equation to figure out how many words the context may consist of:

$$\alpha^n = 10^{-37} \iff n = \frac{\log(10^{-37})}{\log(\alpha)} \tag{3.1}$$

Each word feature is used twice, both in raw text and normalized lower-case text. The FOFE features are used twice, both with and without the focus words. For the FOFE-encoded features we used $\alpha = 0.5$ which results in a context size of maximum ~122 words.

The beginning and end of sentence are explicitly modeled with `BOS` and `EOS` tokens, which have been added to the vocabulary list.

The complete list of features is then the following:

- Bag of words of the focus words;
- FOFE of the sentence:
  - starting from the left, excluding the focus words.
  - starting from the left, including the focus words.
  - starting from the right, excluding the focus words.
  - starting from the right, including the focus words.

In total the system input consists of 10 different word feature vectors; five of which are generated from the raw text, and five generated from the lowercase text. The resulting vector for each word feature is 256 dimensions.

3.3.2 Character-level Features

The character-level features only model the focus words from left to right and right to left. Two different types of character features were used: One that models each character and one that only models the first character of each word. The FOFE encoding was used here as well as it enables us to weight the characters and model their order. For these features, I used $\alpha = 0.8$.

In order to ensure the characters fall into an appropriate range, they were encoded with a simple modulo hash. Each characters ASCII value is normalized to be within the range 0 and 128. This limitation is reasonable since most characters of English and Spanish are in the ASCII table. The Spanish characters in the range 128:256 are confused with unaccented ASCII characters, for instance *ñ* with *q*.

In total the system input consists 4 different character feature vectors. The resulting vector for each character feature is projected to 64 dimensions.

3.3.3 Balancing the Dataset

In order for the network to not only learn the most dominant class in the dataset, it has to be balanced. Because most of the words do not form a named entity, the `NONE` class is over-represented. If this is not adjusted the network will classify almost everything as not a named entity, which is not correct in the real world.

This is done by ensuring the data which the network is seeing adheres to a distribution where 10% of all the fragments must be valid mentions, and the rest must not be. This 90-10 split was empirically found by checking the frequency of overall sentences and seeing how many named entities they contained.

The 90% of the remaining data is then further split up into two chunks: one 60% which contains overlapping negative samples and one 30% which contains disjoint negative samples. This is because the network seems to benefit from learning the mention boundaries, where a boundary begins and where it ends. This split has also been found by trial and error, and can likely be fine tuned for better performance.

All of the positive examples are kept, and the remaining negative examples are then randomly sampled respectively until the correct distribution has been achieved. The final set is then shuffled and the training can begin. This process is repeated for every epoch during training of the network which has proven to be beneficial for the network. Because the network is introduced to random noise every new epoch (due to the random sampling of the negative examples), the process has been self-regularizing and prevented overfitting.

3.3.4 Hyperparameters

The hyperparameters used for the system were based on those of Xu et al. (2017) and have been fine tuned throughout the process of developing the system. These have more or less been identical for all languages, with minor variations to the epoch count. The learning rate was decayed by a constant each epoch until it reached the final value.

Parameter	**Value**
Initial learning rate	0.1024
Final learning rate	0.0625
Batch size	512
Hidden layers	3
Neuron count	512
Dropout	0.1024
Optimizer	ADAM
Epoch count	128 - 160

Table 3.3: Hyperparameters used when training the network

Chapter 4
Results and Discussion

I evaluated the system continuously throughout the development process against different data sets and different languages to ensure consistency and no over-fitting of specific data-sets or languages.

4.1 Datasets

There are many different datasets one can use to evaluate against. The system has been evaluated using four independent datasets:

CoNLL2002: This data set was used to evaluate the system in Spanish and compare the results with overall results from other participants throughout the years. The Spanish data is a collection of news wire articles made available by the Spanish EFE News Agency from May 2000. In this task, the contestants received one training file and one evaluation file, which is used during development to evaluate the system. The final evaluation is done against a held back data set which the systems have not previously seen.

CoNLL2003: The 2003 competition is very similar to the 2002, but different in the sense that it focuses on the English language instead. The data is a collection of news wire articles from the Reuters Corpus.

TAC: The text analysis conference held by NIST has included training data for NER since 2014. These data sets included three languages: English, Spanish, and Chinese. The 2014 data only included the English language.

SUC: The Stockholm-Umeå Corpus is a collection of Swedish texts from the 1990's. There are multiple versions of this corpus, but the one used here is version 3. The corpus contains 10 different entity classes, but because some are very infrequent and imbalanced I have decided to group them into broader classes following the style of `CoNLL`. See Table 4.1 for the class conversion.

Original Class	**New Class**
person	PER
place	LOC
inst	ORG
animal	MISC
myth	
product	
work	
event	
other	

Table 4.1: New arrangement for entity classes of SUC corpus

4.2 Evaluation Metrics

The system has been evaluated using a custom built test tool as well as the test tool of the corpus (if any exists).

One major difference between my custom tool and all external tools is that it does not only check each word, but also each fragment. The sliding window algorithm will produce every permutation of words possible in the sentence (up to a certain length). These are then all classified and taken into account when evaluating the system and calculating the final score. However, when using an external tool, this is not possible. They typically take their input in the form of a file where each word has been tagged with an entity class, which they then compare with the correct answers (line by line). This means that my method is more robust, more thoroughly tests the network against all possible inputs and produces more accurate scoring.

The evaluation used metrics are precision, recall and F1.

Precision: This is the measurement for how many relevant samples were selected.

Recall: This is the measurement for how many selected samples are relevant.

F1: This score is a micro-averaged harmonic mean of precision and recall which is calculated as such:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{4.1}$$

4.3 Results

The results below are from the tests conducted in my development environment as well as the official results from the TAC2017 EDL competition. The results which are produced by my custom evaluation tool will be marked with an asterisk (*).

Each language uses its own pre-trained word embeddings trained either from the giga-words corpus (if available) or the corresponding wikipedia dump for that language.

4.3.1 CoNLL2002

The network was trained and evaluated on the Spanish dataset from CoNLL2002. The top official result from the task has been added as a comparative result, as well as that of the Stanford NER parser and other state of the art systems. See Table 4.2.

System	**Precision**	**Recall**	**F1**
This work	87.55	80.82	**84.05**
This work (*)	80.13	84.33	82.15
CMP02 (Carreras et al., 2002)	81.38	81.40	81.39 ± 1.5
Stanford NER (Finkel et al., 2005)	81.24	81.03	81.14
LSTM-CRF (Lample et al., 2016)	n/a	n/a	**85.75**

Table 4.2: CoNLL2002 results

4.3.2 CoNLL2003

The network was trained and evaluated on the English dataset from CoNLL2003. The top official result from the task has been added as a comparative result, as well as that of the Stanford NER parser and other state of the art systems. See Table 4.3.

System	**Precision**	**Recall**	**F1**
This work	91.34	89.30	**90.31**
This work (*)	87.98	90.50	89.21
FIJZ03 (Florian et al., 2003)	88.99	88.54	88.76 ± 0.7
Stanford NER (Finkel et al., 2005)	88.21	87.68	87.94
LSTM-CRF (Lample et al., 2016)	n/a	n/a	**90.94**
BRNN-CNN-CRF (Ma and Hovy, 2016)	91.35	91.06	**91.21**

Table 4.3: CoNLL2003 results

4.3.3 SUC

The network was trained the original classes as well as the condensed version. Both results have been included in Table 4.4 below to showcase the difference. Since there was no official evaluation tool, I have opted to use my own which means the results below are very pessimistic. The comparative result of Salomonsson (2012) uses the condensed 4 classes instead of all 9.

4.3.4 TAC

The results in Table 4.5 include the results from the development phase where we trained on data from 2014-2015 and evaluated on 2016, as well as the official results from TAC2017 where we trained on all available data.

System Variant	Precision	Recall	F1
This work 9 classes (*)	49.23	57.14	58.86
This work 4 classes (*)	70.56	79.83	**74.56**
LIBLINEAR-NER (Salomonsson, 2012)	75.97	72.80	**74.35**

Table 4.4: SUC results

Evaluation Corpus	Precision	Recall	F1
TAC2016 English	80.1	67.6	73.3
TAC2016 Spanish	76.9	61.2	68.2
TAC2016 Chinese	73.6	56.7	64.0
TAC2017 English	80.2	71.1	75.4
TAC2017 Spanish	76.9	67.2	71.7
TAC2017 Chinese	77.5	56.0	65.0

Table 4.5: TAC results

4.4 Discussion

Overall I think the system has performed admirably. The simplicity of the design in conjunction with how easy it is to train are two great features. Due to its simplicity, innovation is facilitated which in turn might result in an even better system.

The results from the CoNLL2002 and CoNLL2003 task shows this system is comparable to that of the other state of the art systems. These two datasets are great for showing this because the training and test data is the same for everyone, and many results have accumulated over the years.

The TAC competition is a different story though. Many of these systems use third party data to train their system, such as manually annotated files or even silver standard corpus. The results in Table 4.5 show how all the F1 scores improved from 2016 to 2017 on the same system, where the only difference was the amount of training data. This goes to show how important training data is, and it is not unrealistic to think that with even more training data the system might have performed even better. Beyond just training data, many of the other participants have more mature systems on which they have worked for many years, and are backed by larger teams and large corporations.

The systems by Lample et al. (2016) and Ma and Hovy (2016) both use a LSTM architecture which is an order of magnitude more complex than the FFNN architecture described in this thesis, and they only achieved a tiny bit better F1 score. The result of this complexity was highlighted by the organizers of TAC in their summary report, in which they discussed the duplicability problem of deep neural networks (DNN) (Ji et al., 2017). Many participants used DNNs but with very varying results, finishing all the way from 1st to 21st with a total F1-score gap of 24%.

4.4.1 Multilingual Performance

This system used no language specific feature engineering. All of the features are used for all languages, except those which have unreliable word segmentation, such as Chinese.

These languages are instead modeled at the character level, which means the character features are disabled and word features model characters instead. Other than that the system is identical for each language.

By having a language agnostic solution such as this, training for new languages is trivial and highly effective.

Chapter 5
Conclusion

The system presented in this thesis uses nothing but the most simple feed-forward neural network architecture to produce results very close to some of the best, and most complicated, systems in the world today. The biggest limitation to finding the true potential of the system was training data, which showed its importance with the TAC results. By merely adding a few thousand more entities to the training data for each language, the F1 scores increased from 73.3 for English, 68.2 for Spanish, and 64 for Chinese to 75.4 (+2.1), 71.7 (+3.5), and 65 (+1.0) respectively. Overall I believe this system has proven itself to be a success.

5.1 Improvements

Even though the system performed well, it still has plenty of room to improve. There are numerous techniques that can be employed to improve performance, and a numerous tests to be conducted to establish the best hyperparameters and featurues. These suggestions below have not been performed because of limitations in time and scope of this project.

Gazetteer: A gazetteer is a geographical dictionary, which can be used to further improve the entity recognition of the system. This can be achieved, either by using the gazetteer as input to the system in the form of a feature, or as a mean to prune the results.

Word embeddings: In this system, I have elected to use word2vec to generate the embeddings of 256 dimensions. But I have not conducted any conclusive experiments to determine that word2vec is the best tool for this, or that 256 dimensions is the optimal amount. Alternatives such as GloVe, and fewer or more dimensions, could potentially improve system performance.

Alternative learning schedules: Neural networks are often regulated by learning schedules, in which the learning rate is often set high in the beginning and slowly lowered to a tiny value by the end of the training process. The idea is that we will find the best global minima and then optimize it. This worked quite well for me, but studies have shown alternative methods might improve results even further. Fairly new techniques such as the Yellowfin Optimizer (Zhang et al., 2017) and Cyclic Learning Rates (CLR) (Smith, 2015) have shown faster convergence and better optimization of the global minima.

Distribution of Training Samples: The importance of balancing training data is very important, and while my fixed distribution (10% named entities, 60% overlapping fragments, and 30% disjoint fragments) worked well, I believe it can still be improved. Instead of sampling all of the data at once, one could use *stratified sampling* instead. This will allow each entity type to be sampled independently, relative to its abundance in the training data.

Multi System Merge: Instead of solely relying on this system to find everything, using multiple (existing) systems in tandem might improve performance further, and create one system with exceptional results.

Word Context Length: As described in Section 3.3.1, a maximum of around 120 words can be used as context within one sentence. I went with the assumption that more words in context were better, but perhaps there is a sweet spot with fewer words. Or instead of limiting context to the current sentence, it could span multiple sentences backwards and forwards.

Bibliography

Carreras, X., Màrques, L., and Padró, L. (2002). Named entity extraction using adaboost. In *Proceedings of CoNLL-2002*, pages 167–170. Taipei, Taiwan.

cppreference.com (2017). Numerical limits in c. Last accessed: 2017-12-26.

Douglas, A. (1994). *Radio Manufacturers of the 1920's*. Number v. 2 in Radio Manufacturers of the 1920's. Sonoran Pub.

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

Florian, R., Ittycheriah, A., Jing, H., and Zhang, T. (2003). Named entity recognition through classifier combination. In Daelemans, W. and Osborne, M., editors, *Proceedings of CoNLL-2003*, pages 168–171. Edmonton, Canada.

Gibson, A. and Patterson, J. (2017). *Deep Learning: A Practitioner's Approach*. O'Reilly.

Gustafson-Capková, S. and Hartmann, B. (2006). Manual of the stockholm umeå corpus version 2.0. In *SUC*.

Ji, H. and Nothman, J. (2016). Overview of TAC-KBP2016 Tri-lingual EDL and Its Impact on End-to-End KBP. In *Proceedings of the Ninth Text Analysis Conference (TAC 2016)*, Gaithersburg, Maryland. National Institute of Standards and Technology.

Ji, H., Nothman, J., Hachey, B., and Florian, R. (2015). Overview of tac-kbp2015 trilingual entity discovery and linking. In *TAC*.

Ji, H., Pan, X., Zhang, B., Nothman, J., Mayfield, J., McNamee, P., and Costello, C. (2017). Overview of tac-kbp2017 13 languages entity discovery and linking.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. *CoRR*, abs/1603.01360.

Ma, X. and Hovy, E. H. (2016). End-to-end sequence labeling via bi-directional lstm-cnns-crf. *CoRR*, abs/1603.01354.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

Pushkin, A. (1999). *History of Peter the Great, and other historical prose*. The Complete Works of Alexander Pushkin. Milner and Company Limited.

Salomonsson, A. (2012). *Entity-based information retrieval.* LU-CS-EX: 2012:05. Lund : Department of Computer Science, Faculty of Engineering, LTH, Lund University, 2012.

Scott, F. (1908). *Pamphlets in Philology and the Humanities*. Number v. 20 in Pamphlets in Philology and the Humanities. publisher not identified.

Smith, L. N. (2015). No more pesky learning rate guessing games. *CoRR*, abs/1506.01186.

Tjong Kim Sang, E. F. (2002). Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2002*, pages 155–158, Taipei.

Tjong Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2003*, pages 142–147, Edmonton.

Xu, M., Jiang, H., and Watcharawittayakul, S. (2017). A local detection approach for named entity recognition and mention detection. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1237–1247, Vancouver, Canada. Association for Computational Linguistics.

Zhang, J., Mitliagkas, I., and Ré, C. (2017). Yellowfin and the art of momentum tuning. *arXiv preprint arXiv:1706.03471*.

Zhang, S., Jiang, H., Xu, M., Hou, J., and Dai, L. (2015). The fixed-size ordinally-forgetting encoding method for neural network language models. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 495–500, Beijing, China. Association for Computational Linguistics.