Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

Projektarbeit Informatik

Automatic Detection of Company Names in Texts Using Deep Learning

Autoren	Fabian Camenzind Daniel Zürrer
Hauptbetreuung	Mark Cieliebak
Nebenbetreuung	Maurice Gonzenbach

Datum	15.01.2017

Zurich University
of Applied Sciences

School of
Engineering

DECLARATION OF ORIGINALITY

Project Work at the School of Engineering

By submitting this project work, the undersigned student confirm that this work is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

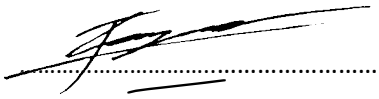
The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the project work have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

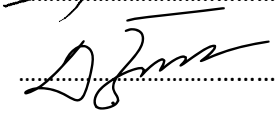
Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date:

Zürich, 15.01.2017

Signature:





The original signed and dated document (no copies) must be included after the title sheet in the ZHAW version of all project works submitted.

Zusammenfassung

In diesem Projekt soll ein künstliches neuronales Netzwerk trainiert werden, um Firmennamen in natürlicher Sprache zu erkennen. Das Erkennen von benannten Entitäten (Named Entity Recognition, kurz NER) wird oft durch linguistische Modelle erreicht. Solche Modelle sind komplex zum Erstellen und nur spezifisch für die jeweilige Domäne geeignet. Mit unserer NER basierend auf einem neuronalen Netzwerk möchten wir eine flexible Lösung erstellen, welche leicht für verschiedene Entitätstypen oder Sprachen angepasst werden kann.

Durch das Erstellen von unseren eigenen Trainingsdaten für das neuronale Netzwerk von frei erhältlichen Nachrichtentexten und Firmenlisten stellten wir sicher, dass die Daten für unseren Einsatzzweck geeignet sind. Dafür sammelten wir zuerst genügend Rohdaten, um Firmennamen in Texten mit natürlicher Sprache zu finden. Anschliessend erstellten wir eine Liste von markierten Sätzen als unsere Menge von Trainingsdaten. Dies erreichten wir durch einfaches suchen von Firmen aus unserer Firmenliste in den Nachrichtentexten. Wir nutzten eine textuelle Ausgabe dieser Daten, um die Qualität der Trainingsdaten abzuschätzen. Um das neuronale Netzwerk zu trainieren, wechselten wir zu einem auf Zahlen basierten Format. Sowohl die Texte wie auch die Markierungen werden darin als Glanzzahlen repräsentiert. Damit konnten wir beginnen, unser neuronales Netzwerk zu trainieren. Um die beste Konfiguration des Netzwerks zu finden, variierten wir sowohl die Trainingsdaten wie auch die Parameter des Netzwerks. Die Hauptparameter, mit denen wir experimentierten, waren die Grösse der Trainingsdaten und die Grösse und Anzahl der versteckten Ebenen im neuronalen Netzwerk.

Als Resultat haben wir einige fertig trainierte neuronale Netzwerke, mit denen über die Kommandozeile interagiert werden kann. Unser neuronales Netzwerk kann einige Firmennamen in natürlicher Sprache erkennen, ist aber nicht zuverlässig genug, um dies bei allen Eingabedaten zu erreichen. Die Qualität der Ausgaben des Netzwerks sind noch nicht gut genug, um sie in einem Produktionssystem einzusetzen. Die Ausgaben bieten aber eine gute Grundlage für weitere Experimente.

Abstract

This project aims to train a neural network to recognize company names in natural language. Named entity recognition (NER) is often achieved by using linguistic models which are complex to create and specific to a problem domain. With our NER based on a neural network we set out to create a flexible solution which can be adapted easily for different types of entities or languages.

By creating our own training data for the neural network from freely available news texts and lists of companies we ensured that the training data is tailored specifically to our purpose.

To this end we first gathered enough raw data to find companies in natural language texts and then created a list of labeled sentences as our set of training data. This was achieved by simply matching words from the natural language sample to our list of known companies. The text form of this data was used to manually observe the quality of the training data. To actually train the neural network we switched to an integer based format for the input data. In this format, the input text and labels are represented as integers. Next we started training our neural network with this input data. To find the best configuration we trained the neural network with different sets of input data as well as varying configurations. The main parameters with which we experimented were the size of the training data and the size and number of the hidden layers in the neural network.

The results are a few trained networks that can be interacted with using the command line. It can recognize some company names in natural language but the neural network fails to produce reliable output for all inputs. The quality of the network's output is not yet high enough to use in production but the results are a solid base for further experimentation.

Preface

Both team members have developed an interest in machine learning over the recent years. This interest lead to projects in the second year of studying at ZHAW which were first steps into machine learning or artificial intelligence as a whole. Because machine learning is such an expansive topic starting in the field is not trivial. In this project we were able to gain valuable insight into machine learning, and specifically deep learning, that will accompany us through the next years of our careers.

We would like to thank Mark Cieliebak and Maurice Gonzenbach for their support as well as the School of Engineering and their staff for the opportunity and hardware support to conduct this project.

Contents

1 Introduction

1.1 Goals

The goal of this project is to train a neural network to recognize company names in natural language. This neural network should then be able to accurately label words in a sentence as a name of an arbitrary company. There is no predefined goal regarding the performance of the resulting network. Rather we are trying to learn what accuracy we can get from this automatic natural language analysis.

For the official task description see section 7.1.

1.2 Overview of this Paper

This paper holds information and the experiments of the above project, concluded by Daniel Zürrer and Fabian Camenzind. With the enclosed CD one is able to reproduce a trained model and understand the workings of the named persons.

1.3 Prerequisites of the Reader

The reader of this document should be well-versed in programming (beneficially with knowledge of Python). Basic knowledge of the mechanics of neural networks is also required although specific terminology used in the document is explained.

2 Method

2.1 Realization

The project was divided in two phases. First we had to implement the tools to create our training data (see section 2.2) with which we then experimented in the second phase (see section 2.3). While there was some overlap between those two stages, they can be looked at as very much distinct.

2.1.1 Tools

All tools used in this project were written in Python. While most code used in our company finding tool is original code we did use part of the Natural Language Toolkit (NLTK). Namely the *Punkt sentence tokenization models* were used for splitting sentences into tokens (words)[1].

For our neural network we used an implementation based on TensorFlow[2] by Maurice Gonzenbach named *hippo*.

2.1.2 Responsibilities

Fabian Camenzind and Daniel Zürrer implemented the tool to create the training data for the neural network. They were also responsible for gathering the raw data and conducting the experiments in the second phase.

Maurice Gonzenbach provided the implementation of the neural network as well as the file writer for the input data. The setup of our specific server environments was also mainly provided by him.

2.2 Creation of Training Data

2.2.1 Gathering Raw Data

We created our own training data in a format inspired by the CoNLL-2003 shared task, but we only use the `I-ORG` named entity label [3]. The goal was to have a simple format

that marks all companies in a sentence to be used as a precursor for the input of the neural network.

```
We O
recognise O
companies O
such O
as O
Time I-ORG
Warner I-ORG
Inc. I-ORG
or O
Samsung I-ORG
. O
```

Figure 2.1: Label file with named entity labels `I-ORG` for companies

To create this input data we needed natural language input text, a list of company names and a program to search the input text and create the label file. A selection of natural language samples in the form of news texts are available at `http://www.statmt.org/wmt14/training-monolingual-news-crawl/` where we got our input text. We chose the latest file from 2013 containing 21'688'362 English sentences.

The list of companies used to create the label file was compiled by our team after querying the DBpedia Ontology. The resulting list contained about 20'000 entries which we could match with the input text. This list is of varying quality as many company names are also popular first or last names and thus results in many false positives when searching for matches in news texts.

2.2.2 Generating the Company List

Queries to the DBpedia Ontology are written in SPARQL. DBPedia offers an online query editor and executor at `http://dbpedia.org/snorql/` which we used to query the DBpedia Ontology. To gather the list of company names we used to create the input data for our neural network we used the query seen in figure 2.2.

Line 1 shows the selection of the DBpedia Ontology while the rest of the query specifies a selection of unique company names. The online query executor we used created a `JSON` file with all company names. We then simply had to extract the names from the surrounding data structure to create a list of company names separated by newline characters.

2.2.3 Creating a Company Finding Algorithm

Because of the large amount of data we had to work with we needed a fast algorithm to compare the words of the input text with our company list. To this end we used a Python

```
PREFIX dbpedia0:<http://dbpedia.org/ontology/>

SELECT distinct ?name WHERE
{
?body a dbpedia0:Company.
?body rdfs:label ?name.
}
```

Figure 2.2: Query used to retrieve a list of company names from DBpedia

`set` of the company list to look up company names. In a Python `set` the lookup of a member has an average time complexity of $\mathcal{O}(1)$[4]. To tokenize our input sentences into words we used the word tokenizer of the NLTK.

The main complexity of our company finding tool comes from the fact that many company names have multiple words and therefore a simple word to word comparison would miss many results. Our list of companies already had multi-word company names listed. As a result our tool needed to not only compare single words but also dynamically build multi-word phrases and compare them as well. As seen in figure 2.1 the output was generated with labels designating each token of the input text as part of a company name or not part of a company name. Our final tool to find company names as well as writing the label file can be seen in figure 2.3.

Later versions of the tool feature the generation of statistics or were written to create integer representations of the label file to be used as the input of our neural network. This first version was able to check about 1000 sentences per second.

Multi-Word Company Names

Matching multi-word company names to their corresponding company list entries is particularly performance intensive as we have to check two-word phrases, three-word phrases and so on starting at every token in our sentences. We achieve this by simply adding words to the currently looked at token in the sentence until a match is found or until the end of the sentence. Figure 2.4 shows an example of the phrases the tool generates and looks up in the company list.

This process could be sped up by limiting the number of words a phrase can contain but for our application the performance was sufficient.

2.2.4 Statistics

To better understand the nature of news texts and how company names are distributed within them, our company finding tool also creates a statistics file about the text. Note that any statistics described in this chapter are not based on the raw input data but on a set of sentences of which we know that 60% contain a company name. We biased our input

```
def searchCompany(sentenceWordList, sentenceWordCount, currentWordIndex,
↪ additionalWords):
    if (currentWordIndex < (sentenceWordCount - additionalWords)) and (' '.join(
↪ sentenceWordList[currentWordIndex:currentWordIndex+1+additionalWords]) in
↪ companySet):
        return additionalWords + 1
    else:
        return 0

def compareAndWrite(sentence):

    # tokenize sentence
    sentenceWordList = nltk.word_tokenize(sentence)
    sentenceWordCount = len(sentenceWordList)

    numberOfWordsInCompanyName = 0;
    # iterate over all tokens
    for currentWordIndex in range(0,sentenceWordCount):

        # write token in result file
        result.write(sentenceWordList[currentWordIndex] + ' ')

        # search for match in company list including multi-word phrases
        # until end of sentence
        for additionalWords in range(0,sentenceWordCount-currentWordIndex):
            # do not search for new company while in multi-word company name
            if numberOfWordsInCompanyName == 0:
                numberOfWordsInCompanyName = searchCompany(sentenceWordList,
↪ sentenceWordCount, currentWordIndex, additionalWords)

        # write labels in result file
        if numberOfWordsInCompanyName >= 1:
            result.write('I-ORG\n')
            numberOfWordsInCompanyName -= 1
            continue
        else :
            result.write('O\n')

    result.write('\n')
```

Figure 2.3: Algorithm to find company names in sentences and write the label file (file creation and reading of sources omitted)

```
...
as
as Time
as Time Warner
as Time Warner Inc.
as Time Warner Inc. or
as Time Warner Inc. or Samsung
as Time Warner Inc. or Samsung .
Time
Time Warner
Time Warner Inc.
or
or Samsung
...
```

Figure 2.4: Example of phrases that are being looked up in the company list using the sentence *We recognise companies such as Time Warner Inc. or Samsung.*

sets for the neural network to feature more sentences with company names to have more positive samples for the network to learn. Maurice Gonzenbach suggested the percentage of 60% and having seen promising results in our first experiments, we kept that percentage.

We were mainly interested in two statistics. One statistic of interest was how many company names would be listed per sentence. The other statistic was the number of unique companies found in our input set. The following sections describe these statistics for a random sample of 200'000 sentences.

Company Names per Sentence

As can be seen in figure 2.5 most sentences only have one company name. With each additional company name the number of sentences decreases by a factor of ten. There were no sentences with more than six company names. Our own anecdotal experience of natural language matches the approximate scale of the distribution. This leads us to believe the training data is representative of natural language and suitable for the task. The result is calculated from the same data that was used to train our neural network so we expect the fully trained network to give a similar distribution when labeling new data.

Unique companies

It is also important that the training data contains a multitude of unique company names. Training the network with a small selection of company names could lead to a network that is very well calibrated to finding those names but not others. We want as many company names in our training data as possible so the neural network can learn the general

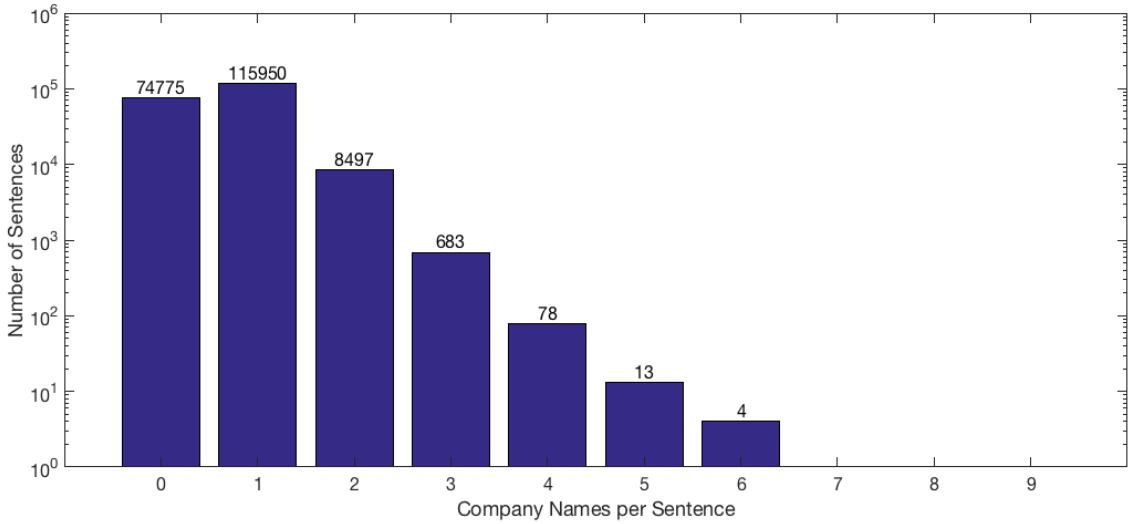


Figure 2.5: Number of companies per sentence.

characteristics of a company name. The set of 200'000 sentences yielded a total number of company names of 135'394 (distribution see figure 2.5). This included 1'982 unique names (about 1 in 68). We believe that this number is large enough to be representative of the general characteristics of a company name.

2.2.5 TensorFlow Input Data

We used TensorFlow as the framework for our neural network. For the first set of training, validation and testing data we split up the input text into single characters and labeled each character as part of a company name or not part of a company name. Since we only needed to designate two classes of characters we decided to use a binary label system consisting of `0` (not part of company name) and `1` (part of company name) as labels. The characters were also given a unique integer ID.

2.3 TensorFlow Experiments

2.3.1 Getting an Understanding

As described in the following section 2.3.3 the first few runs of our experiments were to see how everything behaves. After a couple of runs we were able to recognize first patterns accross our results. These patterns or tendencies of the neural network where given by the values that Maurice Gonzenbach printed into tenserflow.
We soon realized that we needed to add more information to our output and adjusted the TensorFlow code to also log the correct samples. Within the following experiments our

understanding grew in the matter of TensorFlow steadily.

2.3.2 Terminology

Epoch

An epoch is an iteration of training the neural network. One epoch corresponds the adjustments of weights within the neural network with the full training set. A typical training session in our project consists of 150 epochs. We found that number to be small enough to allow for efficient training and large enough to gain valuable insights from the results.

Overfitting

Overfitting is the term used when a neural network is trained too specifically on the training set. If the network learns the characteristics of the training set and does not perform well on other input data, the neural network is overfitted. During our experiments overfitting was a main concern and issues arising from overfitting are outlined in section 3.3.

Input Sets

To train our neural network we used three sets for each run. The input data consists of a training set, a validation set and a testing set. The training set is used to adjust the weights in the neural network. To counteract overfitting, the results of the training set are checked against the validation set. After every epoch the neural network is checked against the testing set. This set has no influence on the weights but is used to measure how well the neural network performs its task.

Hidden Layers

The hidden layers in a neural network are the layers that are neither input or output. All data is passed through those hidden layers and is ultimately passed to the output layer. With too few hidden layers, the neural network may not be adjustable enough to fulfill the given task. Too many hidden layers and it may overfit itself to the training set and not work on general data. For our experiments we usually chose two hidden layers of size 200.

Dropout

Dropout is used to counteract overfitting. When using dropout, a certain percentage of nodes is ignored for computing the output data at each step during training. This randomizes the neural network during training and only parts of it are trained at each step.

Dropout thus prevents an overfitting of the whole network. We typically used a dropout rate between 10% and 20% in our experiments.

Correct Samples

One important metric is the number of correct samples in the testing set. In our project this is expressed as a percentage of correctly labeled sentences in the set.

Loss

The loss of a neural network is a metric to discern how well it is performing its task. During training, the neural network is trying to minimize the loss. We expect a decrease in loss during the training session.

Perplexity

Perplexity is another metric to judge the neural network performance. It is optimally also minimized. Overfitting will cause an increase in perplexity. We use graphs of the perplexity to illustrate our experiments in this document because the performance can be easily judged from it.

2.3.3 Experiments

Our first experiments with TensorFlow provided us a platform to get familiar with the system and see if the neural network can be trained at all with our input data. The first set of training data had a size of 20'000 samples and was trained with the following parameters:

```
python hippo.py lstmseq train v1/dc.json -eval-train -ne 100 -lstm-mode
bidir -hidden 100 100 -tb
```

The first results already looked promising. What we are looking for is a small perplexity on our validation set which indicates the accuracy of our trained neural network. After 100 epochs of training we arrived at a minimal perplexity of about 0.033. This was a good first result and showed us that we are on the right track. The change of the perplexity value through the 100 epochs can be seen in figure 2.6

This curve is roughly what we have been expecting. Ideally the perplexity would sink to 0 after we find optimal parameters for our problem. The next two experiments were run on a training set with 40'000 samples and hidden layers of the sizes (200 200) and (50 50). We also increased the number of epochs to 150 to get a better understanding of the effects of longer term training. With larger hidden layers we expect to see a faster initial drop in perplexity but with eventual overfitting. By training the neural network with larger hidden layers the risk of learning the specific characteristics of the training data increases. This

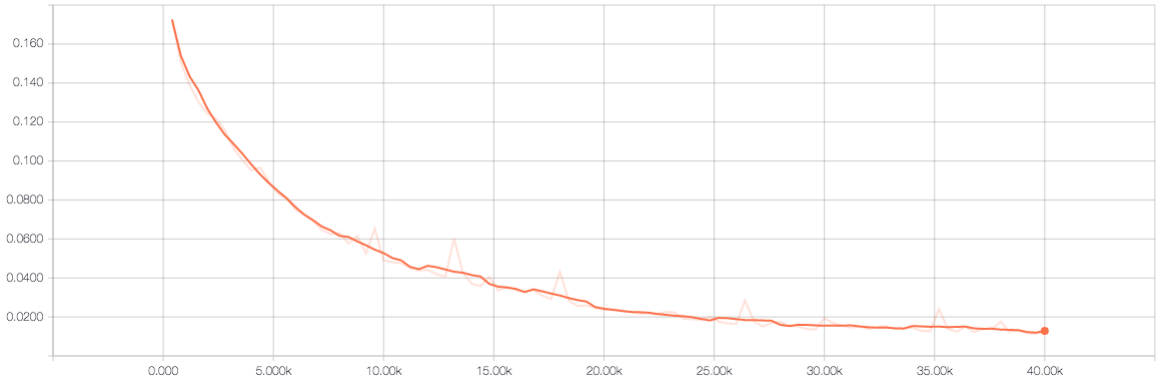


Figure 2.6: Smoothed perplexity over time for the test-set of our first experiment. Graph generated by TensorBoard.

makes the results for general application, as simulated by the validation set, less accurate. The overfitting can be clearly seen in figure 2.7.

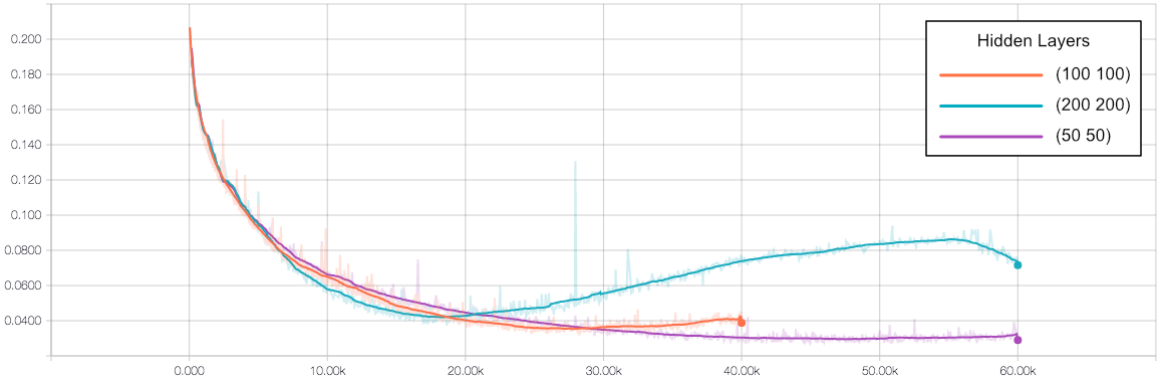


Figure 2.7: Smoothed perplexity over time for the test-set of our first three experiments. Graph generated by TensorBoard.

Our longest run at this point took just over 19 hours to complete. Because we also intended to increase the size of the training set to 80'000 and eventually 160'000, these times would only increase. This meant we had to carefully think about our experiments. For our next experiments we increased the training set size to the aforementioned 80'000 sentences. This time we were interested in how a dropout would affect the overfitting that clearly occured. We kept the hidden layers at (200 200) for these experiments.

While testing our trained models, which reached scores of up to 84%, we saw that the neural network only recognized a couple of characters sporadically throughout the input string. It looked like the model did not know the simplest rule of a what a word is. At this moment we realized that our company finder (see 2.2.3) removed all spaces from the input line. This error prevented the neural network from learning the definition of a word thus only recognizing companies within sentences without spaces. After we fixed the bug by adding a space at the end of each word (excluding punctuation) we came closer to the expected results in our tests. As of now the neural network recognizes some companies

(mostly from our DBpedia list) if there is a space at the end, probably but not proven, due to an error in the company finding tool. For more information see the results in chapter 3.

2.3.4 Impediemts

The following subsection tries to explain some problems as well as impediments we came across following the data-gathering.

Shared Server

After the first few weeks of creating training data sets some runs needed to be done. The ZHAW provided a server which was shared among four teams. The server was strong but very restricted. After a couple of short (five to six hour runs) it seemed to be realistic to start a bigger training run with more data. In the following days it became clear that the server was not fit for multiple accounts/projects running and it failed, thus leading to the first loss of a nearly trained model. The attempt of organizing the projects and their GPU time on the server failed which caused our team to abandon the shared server in the end.

Amazon EC2

The following solution was to use Amazon Web Services EC2 server instances. The first images were created by Maurice Gonzenbach and dispached on multiple instances. Since GPU based On-Demand EC2 instances can cost up to 3.08 $ per hour, ZHAW opted for EC2 Spot Instance. The problem with those instances became apparent a few days later when we lost all our data again. The Amazon Web Services knowledge on the project wasn't the highest in the beginning which led to the mistake of backing up all data on EC2 Spot Instance to a S3 bucket. A couple of days later the prices for spot instances skyrocketed and we decided to switch to an on demand instance. Subsequently this wasn't as easy as hoped and the project account first needed to wait for approval for new instances. The approval took nearly a week in which no computations could be made.

3 Results

3.1 Final Experiments

After getting to know the neural network we started our last batch of experiments with set sizes of 60′000, 2′000 and 1′000 for the training, validation and testing sets respectively. We found those set sizes to give us diverse input data while maintaining reasonable and manageable run times.

In these final experiments we varied the number and size of the hidden layers as well as the dropout rate to try and counteract the negative effects we have seen in previous experiments. In this chapter we will discuss the outcome of the following four experiments:

Reference	Command
Experiment 1	`python hippo.py lstmseq train v3/dc.json -eval-train -ne 150 -save -best 20 -lstm-mode bidir -hidden 150 150 -dropout 0.1 -tb`
Experiment 2	`python hippo.py lstmseq train v3/dc.json -eval-train -ne 150 -save -best 20 -lstm-mode bidir -hidden 150 -dropout 0.1 -tb`
Experiment 3	`python hippo.py lstmseq train v3/dc.json -eval-train -ne 150 -save -best 20 -lstm-mode bidir -hidden 150 -dropout 0.2 -tb`
Experiment 4	`python hippo.py lstmseq train v3/dc.json -eval-train -ne 150 -save -best 20 -lstm-mode bidir -hidden 200 -dropout 0.15 -tb`

Table 3.1: The commands used to start our final four experiments.

Experiment 1 yielded acceptable results but after testing a few sentences against those results we realized that the neural network was too specifically trained on our input data. By reducing the number of hidden layers in Experiment 2 we tried to restrict the network's capacity to save specific characteristics. However Experiment 2 resulted in a very similar behaviour to Experiment 1. This is what led us to increase the dropout rate to 20% for Experiment 3. While this increased dropout rate severely decreased overfitting, the performance of our neural network in recognizing company names only slightly increased. As a last effort for Experiment 4 we increased the size of the hidden layer to 200 and set a dropout rate of 15%. We believe that dropout rate to be a good compromise between overfitting and performance for our task. The larger hidden layer should enable the network to learn more characteristics of company names.

All four experiments resulted in a network that is able to recognize some company names, mostly limited to the ones in our training set. The changes we made during the last four experiments had a disappointingly small impact on the performance. The actual performances will be discussed in section 3.2 and examples of the results can be found in section 3.3.

3.2 Scores

For our final four experiments we plotted the number of correct samples per batch as well as the perplexity of all runs. The neural network used a batch size of 150 to train. A correct sample is any sentence for which each character is correctly identified as part of a company name or not part of a company name. As displayed in figure 3.1 all experiments had very similar performance in regards to the correct samples. Each time the network was able to correctly label about 87% of samples.

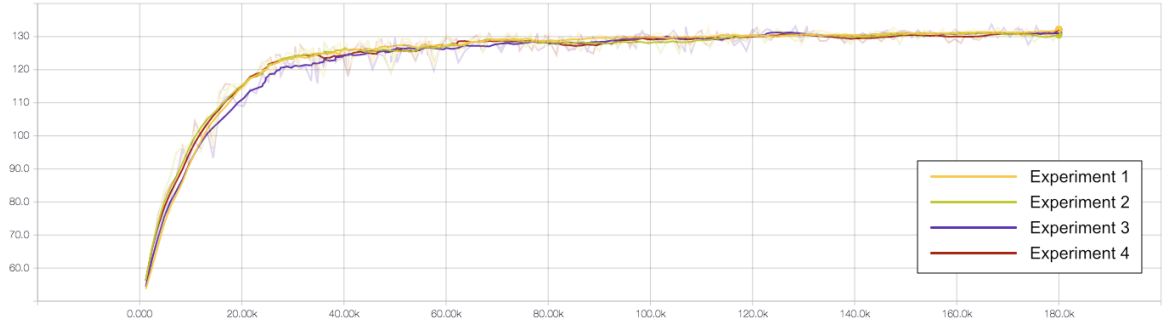


Figure 3.1: Smoothed number of correct samples over time for the test-set of our final experiments. Graph generated by TensorBoard.

These resulting numbers are however deceptively large. When manually testing (see section 3.3) we quickly found that only few samples containing company names not in the training data are labeled correctly. Figure 3.3 shows the perplexity values for the final experiments. Comparing the graphs with table 3.1 shows that only the large dropout rate of 20% in Experiment 3 had a strong impact on the overfitting.

Compared to the numbers in figure 2.7 the perplexity values and progression look more moderate which leads us to the conclusion that we are close to optimal values with these experiments.

3.3 Examples

To compare the performance of the resulting networks from our final experiments we used the interactive mode of the our neural network with a few sample sentences:

- We recognise companies such as Time Warner Inc. or Samsung.

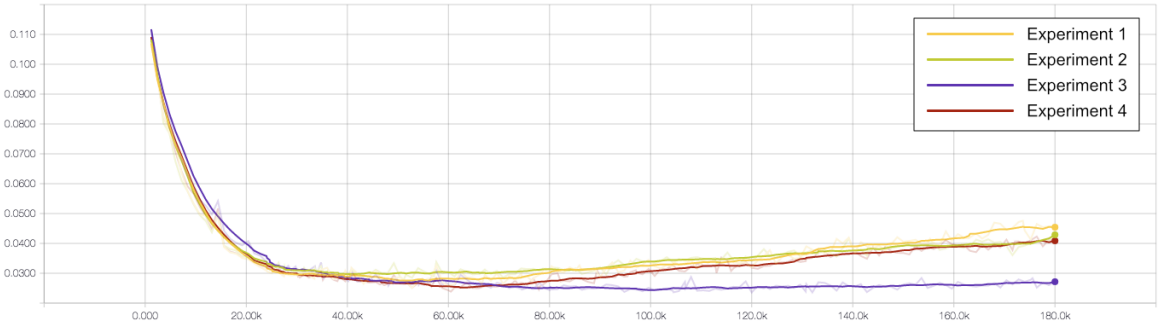


Figure 3.2: Smoothed perplexity over time for the test-set of our final experiments. Graph generated by TensorBoard.

- Apple is reportedly dropping Samsung as its chip manufacturer.
- If Ford Wants to Beat Tesla, It Needs to Go All In

These samples include two standard sentences as well as a headline. The headline is an interesting challenge for the network because one of the main features of a company name is the capitalization. In the headline most words are capitalized and the network needs to recognise different features of a company name. The output of our network labels characters with 1 if they are part of a company name and 0 if they are not. Figure 3.3 lists the output for each of our final experiments.

3.4 Analysis

When examining the results in figure 3.3 it becomes apparent that the quality of the output is not high enough for a production system. At the same time the neural network correctly identifies some company names which is promising for further experimentation with the system. As described in section 3.1 the network is still trained too specifically to the input data. In the samples this manifests itself by the network always being able to identify `Samsung` while missing `Apple` in all experiments. After Experiment 3, which had the highest dropout rate and experienced the smallest amount of overfitting, the network was able to correctly identify `Time Warner Inc.` but also yielded the most false positives in the sample headline.

One issue all our experiments suffered from is the recognition of an extra space after company names. This is due to the way our training data is created. Because the spaces are lost during the tokenisation of the input text we manually add it back with the same label as the word. This means that the space after a company name is also labelled as part of that name in the input data. We only identified this issue after our final experiments and were unable to correct it due to time constraints.

With some clean-up work the output of the network could be improved in a production system. The first step of clean-up would be to remove the extra spaces described above.

```
We recognise companies such as Time Warner Inc. or Samsung.
00000000000000000000000000000000000011000000000000000000111

Apple is reportedly dropping Samsung as its chip manufacturer.
00000000000000000000000000000011111111000000000000000000000000

If Ford Wants to Beat Tesla, It Needs to Go All In
00000000000000000000000000000010000000000000000000
```

(a) Output of the neural network trained in Experiment 1.

```
We recognise companies such as Time Warner Inc. or Samsung.
00000000000000000000000000000000000000000000000010000011111111

Apple is reportedly dropping Samsung as its chip manufacturer.
00000000000000000000000000000011111111000000000000000000000000

If Ford Wants to Beat Tesla, It Needs to Go All In
00000000000000000000000000000000000000000000000000
```

(b) Output of the neural network trained in Experiment 2.

```
We recognise companies such as Time Warner Inc. or Samsung.
00000000000000000000000000000011111111111111111100000001111

Apple is reportedly dropping Samsung as its chip manufacturer.
00000000000000000000000000000011111111000000000000000000000000

If Ford Wants to Beat Tesla, It Needs to Go All In
00000000000000000000000000000011000000000000111100
```

(c) Output of the neural network trained in Experiment 3.

```
We recognise companies such as Time Warner Inc. or Samsung.
00000000000000000000000000000000000000000000000011000011111111

Apple is reportedly dropping Samsung as its chip manufacturer.
00000000000000000000000000000011111111000000000000000000000000

If Ford Wants to Beat Tesla, It Needs to Go All In
00000000000000000000000000000000000000000000000000
```

(d) Output of the neural network trained in Experiment 4.

Figure 3.3: Sample output from our trained neural network.

You could then expand partial labelling to full words or remove positive labels when most of the word is labelled with 0.

In the end we did expect to reach a more intelligent network that would be able to recognize different companies. Even if the network was able to recognize some companies, our `companyFinder` did it better. We were able to see some recognition for example if there is *corp* written after a word but those occurrences where small in numbers and are not reproducible

4 Discussion

4.1 Reflection

The given task for this project consists of three main parts. The following sections contain a reflection of the achieved results in regards to each part.

4.1.1 Generating Annotated Corpus

In terms of programming and testing this was the most straightforward part. A basic version of our company finding tool was implemented within a few weeks after the project kickoff. Since then the tool has been refined and refactored for better performance, easier readability, maintainability and extendability. We think that the tool is a solid piece of software which could be used in other projects. For a more detailed discussion of the possibilities this tool offers see section 4.2.1.

The second subtask of this part was much more involved than initially expected. While we did receive a link to a set of news text which we were able to use throughout the project, the generation of a list of company names was unexpectedly difficult. We contacted the Swiss Federal Commercial Registry Office and as well as the Federal Statistical Office to receive a list of Swiss companies. Unfortunately the access to that data is very restricted with the best offer being a list of 328 companies. That number is nowhere near large enough to create a representative training set for our neural network so we had to create the list ourselves with freely available data.

The details of the list generation are explained in section 2.2.2. The resulting data was of sufficient quality to conduct our experiments. However the data could be improved by removing more peculiar entries such as numbers. It should also be noted that some entries contain a suffix such as *Ltd.* or *Inc.* while others do not. This meant that our training data may not be as precise as it could be with a more expansive set of entries. At the same time we felt that this was not a pressing issue because we would generate few false positives which we judged more important than false negatives.

4.2 Prospects

In this section we briefly discuss how our results can be used in future projects. We also provide hints where the process might be improved. For a concrete description of how to

use our software see chapter 5.

4.2.1 Company Finding Tool

Our company finding tool is written for general search of a list of expressions (in our case company names) within a list of sentences. It can be used without much modification to find and label any data in input text as long as the data can be represented in list form. This means it can be used to create training data for a wide array of tasks.

Currently the tool offers options to safe the training data in human readable form (see figure 2.1) as well as in a variable width matrix (VWM). Since the writing of the results to a file is handled by an independent class it can easily be extended to fit the specific needs of a project.

The quality of the training data is mainly a function of the raw input data. At the core of the tool is simple string matching so the more complete and clean the list of expressions the better the resulting training data. Depending on the task this can be a challenge that has no universal solution.

4.2.2 Pipeline

We did not use any automation to make the resulting training data from our company finding tool available to the neural network. Each time we created a new set of training data it had to be manually uploaded to the off-site hardware. This meant manually organizing folder structure which is error prone by nature. In a larger project we would recommend to automate this step for more efficient data handling.

In addition to automating the data handling the creating of training data would preferably be handled on the same hardware that is used to train the network. This allows immediate access to all data by the team members as well as the neural network. Splitting tasks onto different hardware entails a greater need for arrangements between team members which could lead to a loss in efficiency.

4.2.3 Resulting Neural Network

In our project we only tested the resulting network by direct command line interaction. For the network to be used in an actual application there would need to be additional tools. Such tools are not part of this project and have thus not been created by us. We offer no recommendation on how to integrate the trained neural network into an application.

Due to the size of a trained model we won't be providing a trained model as well. You'll find instructions on how to train your own model in the Documentation Chapter (see 5).

5 Documentation

This chapter describes how the tools we created and used can be used to either replicate our results or to conduct your own research. All software and input data we used can be found on the enclosed DVD.

5.1 Gathering Raw Data

5.1.1 Input Text

As explained in section 2.2.1 our input text comes from `http://www.statmt.org/wmt14/training-monolingual-news-crawl/`. This directory offers news texts split up into sentences separated by newline characters in various languages including German, French and English. The texts offer a wide variety of natural language samples and were very well suited for our task. The specific English data set we used (`http://www.statmt.org/wmt14/training-monolingual-news-crawl/news.2013.en.shuffled.gz` [GZ archive, 1.1GB]) is also available on the enclosed DVD.

5.1.2 List of Entities

Gathering the companies for our companies file we used `DBpedia` (for further detail on how to download the exact JSON List see 2.2.2). To convert the `JSON` file from the SPARQL result, we created the `extractCompanyFromDBPedia` Python tool. Before using the tool you need to prepare the JSON by removing everything outside of the company array. The prepared JSON file should start and end with a square bracket. It then saves the given JSON as a simplified list. If the `extractCompanyFromDBPedia` tool is used for other entries, the name of the extracted object may need to be changed in the code.

5.2 Creation of Input Data for the Neural Network

Now that we have our news.txt and companieslist.txt files the next step is to create input data for the neural network to train. Call `companyFinder.py` from your console of choice three times with varying `setSize` and `startLine` as well as the name of the set. In our case we used the following calls for our first sets:

```
python companyFinder.py companylist.txt news.txt 20000 100000 Vali
```

```
python companyFinder.py companylist.txt news.txt 1000 1 Test

python companyFinder.py companylist.txt news.txt 100 1000000 Train
```

These three calls result in three folders, `Vali`, `Test` and `Train` which can now be moved into the `hippo` directory under *inputdir/vX/* with `X` being any number of your version scheme choice. With the folders in place make sure that the right files are linked in the `dc.json` file which should look like figure 5.1.

```
{
   "given_params": {"char_embeddings" : "char_embeds.npy"},

   "ds":{
      "train":{
         "char_indexes": "train/IDs.npz",
         "labels": "train/tags.npz"
      },
      "valid":{
        "char_indexes": "vali/IDs.npz",
        "labels" : "vali/tags.npz"
      },
      "test": {
        "1": {
          "char_indexes" : "test/IDs.npz",
          "labels" : "test/tags.npz"
        }
      }
   }
}
```

Figure 5.1: Example of the dc.json file

Inside the *inputdir/vX/* directory should also be the `char_embeds.npy` file which can be found on the CD.

5.3 Training the Neural Network

Provided the hippo directory from the CD is on the server and the python installation works correctly, follow the instructions below.

To start training the neural network we can now use the following call on the `hippo.py` tool from Maurice Gonzenbach. (The given arguments are exemplary. Further arguments can be found in the `hippo/src/args` directory)

```
python hippo.py lstmseq train v3/dc.json -eval-train -ne 150 -save -best
20 -lstm-mode bidir -hidden 150 150 -dropout 0.1 -tb
```

Following is as short explanation of the above arguments. We are not able to provide

more than the following information about `hippo.py` since we don't know the whole extent of the tool.

Argument	Description
`train`	Starts the TensorFlow lstmseq in training mode in which the model learns. For the interactive mode 5.3.1 the argument must be *interactive*.
`v3/dc.json`	The path to the `dc.json file` for your current training set (consistent of three sub-folders).
`-eval-train`	No explanation possible.
`-ne 150`	The amount of epochs (see 2.3.2) that the neural network will run.
`-save`	Save the model for each epoch.
`-best n`	Can only be used in conjunction with `-save` to only save the best model after the nth epoch.
`-lstm-mode bidir`	Set the mode of the neural network. We only worked with bidir thus no further explanation possible.
`-hidden 150 150`	Set the amount and size of the hidden layers. The example will result in two hidden layers with each 150 nodes.
`-dropout 0.1`	Set the dropout of the neural network in percent (1 being 100%). For more information see 2.3.2.
`-tb`	Save statistics in the output folder to view on TensorBoard.

Table 5.1: hippo.py training arguments explanation.

5.3.1 Manually Interacting with the Neural Network

One way to test your best neural network models is to use TensorFlow interactive mode. In interactive mode you can input strings of text via the command line. The neural network will then calculate the character labels and output them on the command line.

```
  python hippo.py lstmseq interactive v3/dc.json ../outputdir/17-01-11_09-15-18/
-lstm-mode bidir -hidden 200
```

Argument	Description
`interactive`	Starts the TensorFlow lstmseq in interactive mode.
`v3/dc.json`	The `dc.json` used during the training of the model to access the correct structure an embeddings.
`../outputdir/17-01-11_09-15-18/`	Path to the model
`-lstm-mode bidir`	Set the mode of the neural network. Must be equal to the mode during training.
`-hidden 150 150`	Set the amount and size of the hidden layers. Must be equal to the `-hidden` argument in training.

Table 5.2: hippo.py interactive arguments explanation.

6 References

6.1 References

[1] NLTK Project. (2016). NLTK 3.0 documentation - nltk.tokenize package, [Online]. Available: `http://www.nltk.org/api/nltk.tokenize.html` (visited on 12/08/2016).

[2] (n.d.). TensorFlow, [Online]. Available: `https://www.tensorflow.org` (visited on 12/08/2016).

[3] CLiPS Research Center. (2003). Language-Independent Named Entity Recognition (II), [Online]. Available: `http://www.cnts.ua.ac.be/conll2003/ner/`.

[4] (n.d.). TimeComplexity, [Online]. Available: `https://wiki.python.org/moin/TimeComplexity` (visited on 11/14/2016).

6.2 Glossary

Amazon Web Services AWS provides a broad spectrum of cloudbased computation and storage solutions alongside of other services like database, analysis, network, etc .

DBpedia DBpedia is a RDF based knowledge base with aver 4.5 million entries (mostly from Wikipedia). Most of the entries are classified in a consistent ontology and can be accessed via SPARQL.

DBpedia Ontology The DBpedia Ontology is a shallow, cross-domain ontology, which has been manually created based on the most commonly used infoboxes within Wikipedia. The ontology currently covers 685 classes which form a subsumption hierarchy and are described by 2,795 different properties.

EC2 EC2 is a Amazon Web Services cloudbased service that provides scaleable computation.

EC2 Spot Instance Spot instances are instances on a free market where one can bid for computation time. The cost are usually below 10% of normal cost but can skyrocket due to high demand on the market.

JSON JSON is an open-standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs.

NER Named entity recognition is a form of information extraction that seeks to locate and classify named entities in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

neural network A neural network is a computer system modeled on the human brain and nervous system to solve a given task.

NLTK NLTK is a platform for building Python programs to work with human language data.

Python Python is a high-level, general-purpose, interpreted, dynamic programming language.

RDF RDF is a family of specifications originally designed as a metadata data model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax notations and data serialization formats.

S3 bucket Amazons Simple Storage Service (S3) is a Amazon Web Services Service and provides flexible storage.

set A set object is an unordered collection of distinct hashable objects.

SPARQL SPARQL(pronounced "sparkle") is a query language designed to retrieve and manipulate data stored in RDF format.

TensorBoard TensorBoard provides the visual counterpart to TensorFlow.

TensorFlow TensorFlow is an open source software library for numerical computation using data flow graphs. It was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization.

tokenize In lexical analysis, tokenization is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens.

VWM A variable width matrix is a data type that allows efficient storage of rows of variable width in a matrix.

6.3 List of Figures

6.4 List of Tables

7 Appenidx

7.1 Official Task Description

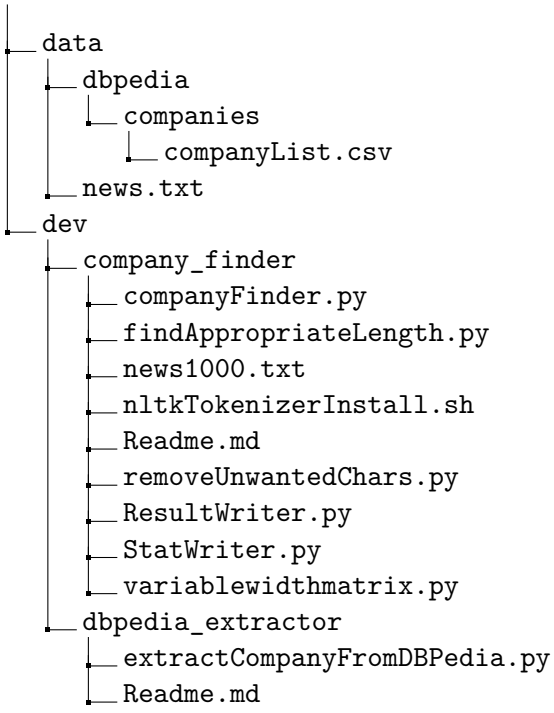
NER is an important task in automatic text analytics, where entities such as company names, persons, or locations should be found in natural language texts. This project thesis focuses on finding company names using deep learning technologies.

This thesis consists of following tasks:

- Generation of an annotated corpus with texts and positions of company names
- Setup of a deep neural network for NER
- Optimizing the network for detecting company names

7.2 DVD Directory

The enclosed DVD has the following folder structure.



```
hippo
├── docker
├── inputdir
│   ├── v1
│   │   ├── char_embeds.npy
│   │   ├── dc.json
│   │   ├── newsTest
│   │   │   ├── annotated.txt
│   │   │   ├── IDs.npz
│   │   │   ├── longSentences.txt
│   │   │   ├── stats.txt
│   │   │   └── tags.npz
│   │   ├── newsTrain
│   │   │   ├── annotated.txt
│   │   │   ├── IDs.npz
│   │   │   ├── longSentences.txt
│   │   │   ├── stats.txt
│   │   │   └── tags.npz
│   │   ├── newsVali
│   │   │   ├── annotated.txt
│   │   │   ├── IDs.npz
│   │   │   ├── longSentences.txt
│   │   │   ├── stats.txt
│   │   │   └── tags.npz
│   │   └── old_char_embeds.npy
│   ├── v2
│   │   └── ..
│   └── v3
│       └── ..
├── run
│   ├── tensorboard
│   │   ├── 16-11-15_12-13-27
│   │   └── ..
│   └── tensorboard.sh
├── src
│   ├── architectures
│   ├── args
│   │   └── ..
│   ├── datahandlers
│   │   └── ..
│   ├── evaluations
│   ├── hippo.py
│   └── ..
└── word_embeds.npy
pa_arbeit
└── pa_camenfa1_zuerrdan_HS16.pdf
```