JOHANNES KEPLER
UNIVERSITY LINZ

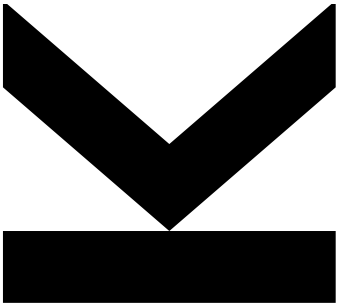
Submitted by
**Klaus Dworschak**

Submitted at
**Institute of Networks and Security**

Supervisor
**Assoz.-Prof. Mag. DI Dr. Michael Sonntag**

May 2018

# Automatische Anonymisierung von Urteilen



Master Thesis
to obtain the academic degree of
Diplom-Ingenieur
in the Master's Program
Computer Science

**JOHANNES KEPLER**
**UNIVERSITY LINZ**
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

# Contents

# Abstract

xyz

# Zusammenfassung

xyz

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation and Objectives

In this master's thesis we want to explore the possibilities of automatically detecting and replacing personal information (given name, surname, organization, dates, ...) from court rulings written in German language. It is commonplace to remove the personal data of plaintiff and defendant as well as witnesses from judgments. Since this is usually done manually it would be useful to have a tool that can with a high probability predict which words are personal data and which ones are not. If an entity is mentioned more than once the tool should also be able to link these entities to make it easier for the reader to grasp the meaning of the text with removed personal data.

The goal for this master's thesis is to implement an efficient command line tool that takes a text file as input and outputs a copy of that text file with annotations on all the words and phrases it classified as personal data. Both input and output will be plain text files. The proposed format for these annotations would put the original text and the replacement in curly braces, e.g. {Klaus Dworschak|PERSONAL_NAME}

What we want to do is a task that falls into the area of natural language processing. This area is all about the interactions between computers and human (natural) languages. The scientific name for the task we want to do is "named-entity recognition" which is a subtask of information extraction. All these terms will be discussed in greater detail in the next chapter.

The scientific name for identifying the names of entities in a sentence is Named-entity recognition (NER), a subtask of the task of information extraction. Information extraction in most cases concerns the processing of human language texts by means of natural language processing (NLP). Named entities are atomic elements in natural language belonging to pre-defined categories such as given names, dates, locations etc.

In simple terms NER, also called "sequence labeling" entails looking at a piece of text and deciding for each word which type of word it is. By using NER we can get an annotated version of the input text that highlights the names of entities.

Here is an example: $[\text{Klaus}]_{\text{Person}}$ did not buy 100 bitcoins from $[\text{coinbase}]_{\text{Organization}}$ in $[2013]_{\text{Time}}$.

For the English language there exist several NER systems that claim to be able to correctly identify entities in over 90 per cent of the cases for the popular English language benchmark CoNLL-2003 shared task. In English most of the time when a word is capitalized it indicates a proper noun, e.g. a personal or place name. The German language is trickier, though, since all nouns are always capitalized.

NER is a machine learning problem. Machine Learning tasks are typically classified in two major categories: Supervised learning and unsupervised learning. In supervised learning the computer is given example inputs with desired outputs and the goal is for it to derive general rules that map inputs to outputs. In unsupervised learning no such test data is provided, the computer must find patterns by itself. Additionally, there are some variants of supervised learning like semi-supervised learning (training data is missing many Inputs and desired outputs), Active learning (the computer can query the user for training data it desires) and Reinforcement learning (rewards and/or punishments are given as feedback to the program's actions).

At least traditionally NER is a case of supervised learning: The computer is provided with a big amount of labeled input files from which it derives a strategy to handle every future input file. Then it saves this strategy in a so called "model" file. To improve the outputs more input data for previously not covered words and phrases is provided to gradually improve the algorithm's accuracy. This is the process of "training" the computer. Sufficiently training a model for a new language can be a task that can take a linguist months, depending on the desired accuracy. Gazetteers, essentially lists of words of

a certain type (e.g. cities) are commonly used to reduce the work needed to properly train a model. This, however, might result in a NER system that is only effective on a reduced corpus (and much less effective with texts that were not used during training).

Regardless of how the NER process is handled programmatically an important part of this master's thesis will be to provide a sufficient set of training data. The accuracy of virtually all NER tools increases with the amount of training data. Usually libraries of 100,000s of sentences are used for this. The idea is to download hundreds of court rulings from RIS, EuGH, etc. and use them for training.

## 1.2 Outline

The rest of this document is structured as foll. Chapter 2 provides background information that serves as the foundation of our approach. In Chapter 3, the theory behind neural networks is presented, including algorithms and design decisions. Following is Chapter 4 which outlines approach that will be taken. In Chapter 5, implementation details are being discussed. Further, Chapter 6 compares our approach to related work. Finally our work is concluded by Chapter 7, which also provides an outlook for future work.

# Chapter 2

# Theoretical Background

This chapter tries to create a fundamental understanding of the background knowledge this work builds upon. In the first section German language legal texts, the texts that are analyzed in this work, are looked at in detail. Then it gives an overview of the natural language processing techniques useful for finishing the task at hand. How these techniques are related is discussed in detail. In this work, the focus will be on named-entity recognition, a subtask of information extraction.

## 2.1 Court judgments

Judgments delivered or published by the courts are usually anonymous and neutralized. That is, essentially the names of the parties to the proceedings as well as the other parties involved (e.g. witnesses and experts) have been blurred. This includes personal names, addresses, company names, etc. The names of judges, attorneys, persecutors and other people employed by the court are given in clear. This is done for privacy reasons and to protect other rights that are worthy of protection. [ACKE_2004]

The exception for this until quite recently was the European Court of Justice (ECJ). Texts published on the EU's dedicated site for publishing legal texts, Curia, had the names of all the involved people in clear. With the GDPR coming into effect this changed and as of July 2018 all the documents published on their publication site are anonymized. These court rulings are

usually available in many languages. German, as one of the three most spoken languages in the European Union is normally among them.

Some countries (e.g. Cyprus) have temporarily stopped publishing court judgments in fears of violating the GDPR. This comes as a surprise since everyone had five years to adapt for the GDPR to come into effect [40].

In Austria court rulings of all major courts are published in the internet at the Rechtsinformationssystem (RIS). This includes rulings of the Supreme Court (OGH), the Oberlandesgerichte (OLG), the Landesgerichte (LG), the Bezirksgerichte (BG) and selected decisions of foreign courts. Only a portion of all the court rulings are published, though. The reason given is that most of those court judgments are not relevant for the public and that there is not enough staff to get it done.[oeff]

The fact that human labor is a major restricting factor for the publication of court judgments is a good argument for a software system that makes anonymization easier.

### 2.1.1 Legal requirement of anonymization

The main focus of this work lies on the anonymization of German language Austrian court judgments. This section gives an overview of the legal basis of anonymization in local law.

In Austria § 15 (4) OGHG governs how anonymization of published court judgments has to be done. The law states that names, addresses and, if necessary, other names of localities and territories which allow conclusions to be drawn about the case in question shall be anonymized by letters, numbers or abbreviations in such a way that the traceability of the decision is not lost.

According to the OGH, the purpose of the provision is, on the one hand, to ensure that in the interests of personal protection of the parties, witnesses and "other participants" a corresponding anonymization of the decisions must be made. On the other hand, it should also be prevented that the text document is no longer understandable because of the anonymization [Felzmann/Danzl/Hopf, Oberster Gerichtshof2, § 15 OGHG Anm 7].

The above-mentioned authors also consider that the provision should ensure sufficient redaction and, in the standard case, permit anonymization by

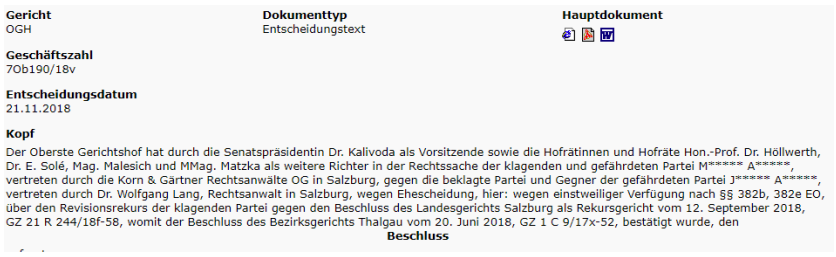
reducing the surname to the first letter and omitting the job title and the entire address of the person concerned. Under certain circumstances, it may be necessary to anonymize the first name of a person, especially if it is rather rare or otherwise conspicuous in the given context. In individual cases, however, even further steps to anonymization may be required if the identity of the person would would otherwise be apparent. In practice, however, some released texts are not as thoroughly anonymized.

In the European Union from 1 July 2018 the rule is "[...] to replace, in all its public documents, the name of natural persons involved in the case by initials. Similarly, any additional element likely to permit identification of the persons concerned will be removed"

In Germany there is no central, unified site where all court judgments are published as in Austria. A centralized on-line place for German court judgments has not been created yet due to a variety of reasons, among them Germany's highly federalized state system with many independent courts. There is no general rule for generalization. Some courts replace names by "First letter" + "...". Some other courts just put "plaintiff" instead of the plaintiff's actual name whenever he/she is mentioned.

### 2.1.2 Real world examples

Below is the heading of a court judgment released on RIS. In lines two and three the anonymized names of the parties to the proceedings can be seen, e.g. M***** A*****.



**Gericht**
OGH

**Dokumenttyp**
Entscheidungstext

**Hauptdokument**

**Geschäftszahl**
7Ob190/18v

**Entscheidungsdatum**
21.11.2018

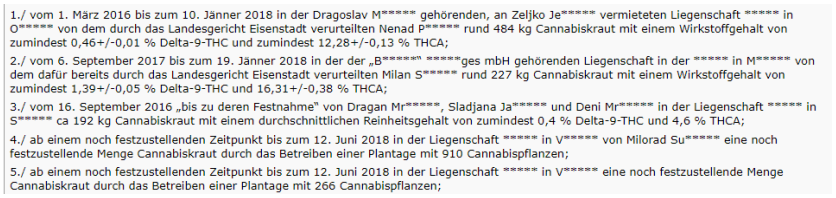
**Kopf**
Der Oberste Gerichtshof hat durch die Senatspräsidentin Dr. Kalivoda als Vorsitzende sowie die Hofrätinnen und Hofräte Hon.-Prof. Dr. Höllwerth, Dr. E. Solé, Mag. Malesich und MMag. Matzka als weitere Richter in der Rechtssache der klagenden und gefährdeten Partei M***** A*****, vertreten durch die Korn & Gärtner Rechtsanwälte OG in Salzburg, gegen die beklagte Partei und Gegner der gefährdeten Partei J***** A*****, vertreten durch Dr. Wolfgang Lang, Rechtsanwalt in Salzburg, wegen Ehescheidung, hier: wegen einstweiliger Verfügung nach §§ 382b, 382e EO, über den Revisionsrekurs der klagenden Partei gegen den Beschluss des Landesgerichts Salzburg als Rekursgericht vom 12. September 2018, GZ 21 R 244/18f-58, womit der Beschluss des Bezirksgerichts Thalgau vom 20. Juni 2018, GZ 1 C 9/17x-52, bestätigt wurde, den

**Beschluss**

Figure 2.1: RIS 5Ob15/18v

The next example below is meant to show that court judgments are harder to work with then texts from, say, a book. Twelve lines of text and there are

no clearly distinguishable sentences, just a numbered list with every point ending with a semicolon. Everything essentially being one long sentence. Also evidenced here is that the anonymization employed by RIS does not seem to be totally consistent: Even though the names imply certain countries of origin of the accused people they are not anonymized properly as in the previous example.



1./ vom 1. März 2016 bis zum 10. Jänner 2018 in der Dragoslav M***** gehörenden, an Zeljko Je***** vermieteten Liegenschaft ***** in O***** von dem durch das Landesgericht Eisenstadt verurteilten Nenad P***** rund 484 kg Cannabiskraut mit einem Wirkstoffgehalt von zumindest 0,46+/-0,01 % Delta-9-THC und zumindest 12,28+/-0,13 % THCA;

2./ vom 6. September 2017 bis zum 19. Jänner 2018 in der der „B*****" *****ges mbH gehörenden Liegenschaft in der ***** in M***** von dem dafür bereits durch das Landesgericht Eisenstadt verurteilten Milan S***** rund 227 kg Cannabiskraut mit einem Wirkstoffgehalt von zumindest 1,39+/-0,05 % Delta-9-THC und 16,31+/-0,38 % THCA;

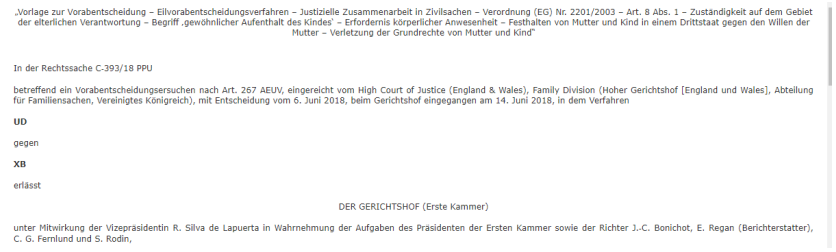
3./ vom 16. September 2016 „bis zu deren Festnahme" von Dragan Mr*****, Sladjana Ja***** und Deni Mr***** in der Liegenschaft ***** in S***** ca 192 kg Cannabiskraut mit einem durchschnittlichen Reinheitsgehalt von zumindest 0,4 % Delta-9-THC und 4,6 % THCA;

4./ ab einem noch festzustellenden Zeitpunkt bis zum 12. Juni 2018 in der Liegenschaft ***** in V***** von Milorad Su***** eine noch festzustellende Menge Cannabiskraut durch das Betreiben einer Plantage mit 910 Cannabispflanzen;

5./ ab einem noch festzustellenden Zeitpunkt bis zum 12. Juni 2018 in der Liegenschaft ***** in V***** eine noch festzustellende Menge Cannabiskraut durch das Betreiben einer Plantage mit 266 Cannabispflanzen;

Figure 2.2: RIS RS0119858

Below is the heading of a court judgment as published on the EU's Curia legal information system. As expected the names of parties to the proceedings have been replaced by initials, e.g. "UD" and "XD". Whenever the name of one of the anonymized people is mentioned further down in the text it is replaced by the same initials. Also worth mentioning: Most of the text is one long sentence, even the title "DER GERICHTSHOF (Erste Kammer)".



„Vorlage zur Vorabentscheidung – Eilvorabentscheidungsverfahren – Justizielle Zusammenarbeit in Zivilsachen – Verordnung (EG) Nr. 2201/2003 – Art. 8 Abs. 1 – Zuständigkeit auf dem Gebiet der elterlichen Verantwortung – Begriff ‚gewöhnlicher Aufenthalt des Kindes' – Erfordernis körperlicher Anwesenheit – Festhalten von Mutter und Kind in einem Drittstaat gegen den Willen der Mutter – Verletzung der Grundrechte von Mutter und Kind"

In der Rechtssache C-393/18 PPU

betreffend ein Vorabentscheidungsersuchen nach Art. 267 AEUV, eingereicht vom High Court of Justice (England & Wales), Family Division (Hoher Gerichtshof [England und Wales], Abteilung für Familiensachen, Vereinigtes Königreich), mit Entscheidung vom 6. Juni 2018, beim Gerichtshof eingegangen am 14. Juni 2018, in dem Verfahren

**UD**

gegen

**XB**

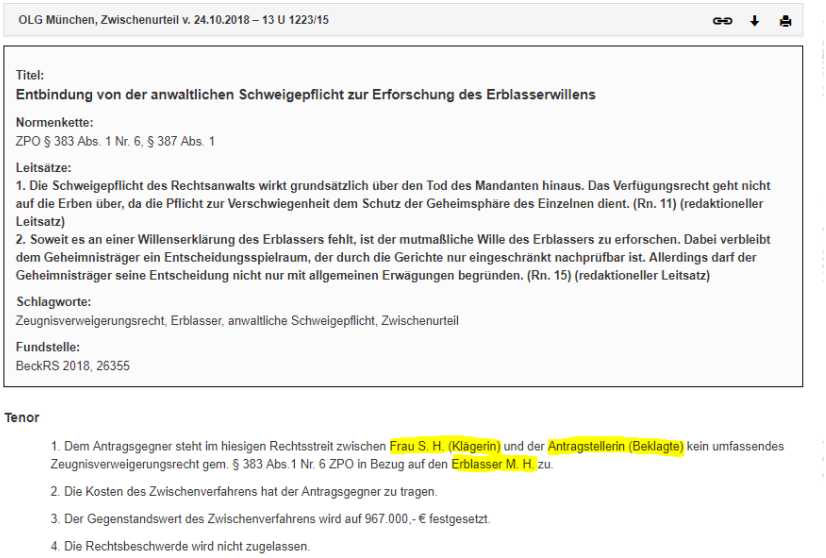
erlässt

DER GERICHTSHOF (Erste Kammer)

unter Mitwirkung der Vizepräsidentin R. Silva de Lapuerta in Wahrnehmung der Aufgaben des Präsidenten der Ersten Kammer sowie der Richter J.-C. Bonichot, E. Regan (Berichterstatter), C. G. Fernlund und S. Rodin,

Figure 2.3: CURIA ECLI:EU:C:2018:835

Lastly, a court ruling from a German court, in this case the OLG Munich. As mentioned before since there are no nationwide rules a legal text from a different German state would look very different. The parties to the proceedings are anonymized, both by avoiding to mention the names and by replacing them by initials. This time around the text consists of proper sentences that the text can easily be split into programmatically.



OLG München, Zwischenurteil v. 24.10.2018 – 13 U 1223/15

Titel:
**Entbindung von der anwaltlichen Schweigepflicht zur Erforschung des Erblasserwillens**

Normenkette:
ZPO § 383 Abs. 1 Nr. 6, § 387 Abs. 1

Leitsätze:
**1. Die Schweigepflicht des Rechtsanwalts wirkt grundsätzlich über den Tod des Mandanten hinaus. Das Verfügungsrecht geht nicht auf die Erben über, da die Pflicht zur Verschwiegenheit dem Schutz der Geheimsphäre des Einzelnen dient. (Rn. 11) (redaktioneller Leitsatz)**
**2. Soweit es an einer Willenserklärung des Erblassers fehlt, ist der mutmaßliche Wille des Erblassers zu erforschen. Dabei verbleibt dem Geheimnisträger ein Entscheidungsspielraum, der durch die Gerichte nur eingeschränkt nachprüfbar ist. Allerdings darf der Geheimnisträger seine Entscheidung nicht nur mit allgemeinen Erwägungen begründen. (Rn. 15) (redaktioneller Leitsatz)**

Schlagworte:
Zeugnisverweigerungsrecht, Erblasser, anwaltliche Schweigepflicht, Zwischenurteil

Fundstelle:
BeckRS 2018, 26355

Tenor

1. Dem Antragsgegner steht im hiesigen Rechtsstreit zwischen Frau S. H. (Klägerin) und der Antragstellerin (Beklagte) kein umfassendes Zeugnisverweigerungsrecht gem. § 383 Abs.1 Nr. 6 ZPO in Bezug auf den Erblasser M. H. zu.
2. Die Kosten des Zwischenverfahrens hat der Antragsgegner zu tragen.
3. Der Gegenstandswert des Zwischenverfahrens wird auf 967.000,- € festgesetzt.
4. Die Rechtsbeschwerde wird nicht zugelassen.

Figure 2.4: OLG München, Zwischenurteil v. 24.10.2018 – 13 U 1223/15

## 2.2 Information Extraction

In information extraction, structured information is to be extracted from texts. Entities, events and relations between entities and events are extracted from texts.

Typically, methods of Natural Language Processing (NLP) are used.

### 2.2.1 Classification

The problem of Information Extraction can be classified into the field of **Natural Language Processing** (NLP). NLP is an area of computer science that deals with the analysis of large amounts of natural language texts. The goal is to name or extract text components.

The major evaluation and tasks of Natural language are categorized into the following categories:

- Syntax (e.g. Part-of-speech tagging, Lemmatization, ...)

- Semantics (e.g. Machine translation, Named entity recognition (NER), Natural language generation, ...)
- Discourse (e.g. Automatic summarization, Discourse analysis, ...)
- Speech (e.g. Speech recognition, Text-to-speech, ...)

Each of these categories contains many more tasks than the ones given in the examples.

In so-called **part-of-speech tagging** (POS tagging), assigned the part-of-speech (POS). Examples of part-of-speech tags (POS tags) are nouns, verbs, articles, adjectives, adverbs, prepositions, conjunctions, and so on. In nouns, for example, a distinction can be made between normal nouns in singular and plural as well as proper names. The same word may have a different part of speech in different sentences, depending on the context in the sentence. Therefore, simply looking up the word in a word list is not enough for POS tagging. Rather, POS taggers are trained on syntactically annotated text corpora and e.g. Hidden Markov Models or decision trees are used. The trained POS tagger can then be applied to new sentences as common in supervised learning.

POS tagging is usually based on statistical methods based on manually created training data. Different languages use tagsets that reflect the characteristics of the language. In German, the STTS (Stuttgart-Tübingen-Tagset) is often used. It contains eleven main parts of speech, which are divided into a total of 54 tags. Examples are NN for nouns, NE for proper names, ADJA for adjectives or ADV for adverbs.

[Example]

Tagging individual words does not or only partially work in languages where words define their meaning by suffixes (agglutinating languages). Another important part of natural language processing is the recognition of sentences or sentence boundaries (*sentence splitting*) and their grammatical structure (*sentence parsing*). In the next section the recognition of named entities as part of natural language processing is discussed.

## 2.3 Named-entity recognition

Named Entity Recognition (NER) refers to the detection of entities such as individuals, companies, geographic locations such as cities, or numbers such as monetary amounts. It is considered a sub task of Information Extraction and Natural Language Processing.

The task of NER was first published on the Message Understanding Conference 7 (MUC-7), which took place in 1997 [CR97]. Put simply, the task usually consists of two parts:

1. Identification of the words in a text.
2. Determining the class for each word (personal name, location or otherwise).

It is not just about recognizing the entities as such, but also assigning them to the right category. Which is often not easy for proper names of people, locations and organizations. Personal names are distinguished by first and last names, with some names used both as a first and last name. Unlike in the case of part-of-speech (POS) tagging, not every word should be explicitly defined in named entity recognition, but only entities should be tagged. Traditionally the recognition is based on defined or learned features such as capitalization, occurrences in dictionaries, characteristic word parts, etc. Furthermore, surrounding words or tags can be used as a feature. If a given name is recognized it is likely that it will be followed by a surname.

One of the primary reasons why named-entity recognition is harder for German than it is for English is that proper names and nouns can not be distinguished on the basis of upper and lower case. Thus, a number of rules for recognizing proper names that for English contribute to successful recognition can not be applied in German. An example of this is the rule “First name followed by uppercase word = Person name”. This rule works for English texts but in German contexts such as “schreibt Klaus Bücher?” or “Ist Petra Lehrerin?” it would falsely identify nouns as proper names.

The resolution of the reference or the co-reference, i.e. the union of all named entities (NEs) that relate to the same object, is not part of the task of NER. If, for example, the personal name "Huber" or "Josef Huber" appears several times in a text, the task of NER is limited to identifying all these

occurrences and identifying them as named entities of the type *person*. It is not the task of NER to identify that person by using data such as the place and date of birth or the social security number. It is also not the task of the NER to decide whether found named entities refer to the same person or several different persons with the same name. These tasks are assigned to subsequent processing steps.

The approaches to named entity recognition can be subdivided into rule-based methods and those that work with machine learning [10]. Solutions that combine these two approaches are also common.

### 2.3.1 Rule-based NER

Early systems for named-entity recognition were primarily rule-based. Rule-based entity recognition in texts is based on the definition of special sentence structures as well as word lists. The entity is then recognized as such because of its position in the sentence or its occurrence in the list. The advantage of rule-based systems is the traceability of results and the independence of training data. This allows them to adapt to new or changing requirements without having to retrain the model [10].

Rules can occur in different forms. In [CKL + 10] a language is developed in which rules for Named Entity Recognition can be defined. Often, regular expressions are used ([PM08], [CCM + 99]).

Most, if not all state-of-the-art results for NER tasks have been achieved with machine learning techniques rather than rule based ones [CHIT_2010].

### 2.3.2 Machine Learning based NER

In machine learning, the algorithm detects patterns in existing data and applies them to the target data set. A distinction is made between different learning methods such as supervised learning, unsupervised learning and others (see section 2.4).

Supervised learning algorithms use training data to learn a probable structure of the data. Typically these are labels such as POS tags that are applied to the real data based on the learned model. High accuracy can be achieved

like that. The training data set must be adjusted for each change in the labels. Also when the rough structure of the input data changes the algorithm has to be re-trained. However, learning algorithms on input sets structured similarly to the training data will usually yield better results than rule-based approaches (e.g. model trained to find geographic entities in message texts, applied to publications).

Algorithms for unsupervised learning are not given a fixed categorization. They subdivide the data according to patterns themselves. Since no specifications can be made, the found quantities / clusters can not be assigned to a unique category such as "organization name" or unique POS tags. The advantage of unsupervised learning is that the system does not have to be adapted to new inputs. For that neural networks are often used.

In practice, the two approaches are combined into semi-supervised learning. The system uses a smaller training data set, from which, if necessary, initially a larger set of training data is generated. If necessary, the result can be checked and corrected in the intermediate step.

Traditionally in machine learning various different methods are being used. Most frequent are Support Vector Machines, Hidden Markov Models [6], their evolution Maximum Entropy Markov Models [7] and Conditional Random Fields (CRFs) [30].

CRFs (see section 2.5) were for a long time state of the art in machine-learning based named-entity recognition. Original CRFs utilize hand-crafted features which increases the difficulty of performance tuning and domain adaptation. In recent years neural networks with distributed word representations (i.e., word embeddings, see section 3.7) have been introduced to automatically calculate word values for CRFs [11, 26].

Recently, so to speak, a challenger has appeared: Solutions based on Artificial Neural Networks (ANN). Particularly recurrent neural networks (RNNs) such as long short-term nemory (LSTM) systems. These kinds of approaches are commonly called Deep Learning and looked at in detail in chapter 3.

### 2.3.3 German language

Named entity recognition for German only really started with the CoNLL-2003 shared task. At the CoNLL-2003 the system that performed best had

a F1 score of 72.41% [14]. Seven years later Faruqui and Pado presented a system that achieved a better score, at 78.2% [13]. The increased precision was achiebed by making use of semantic and morphological similarity of words. More recently, a GermEval NER shared task presented a series of systems [5]. The best of the 11 participating system sytems was ExB which had a F1 score of 79.08% [5].

These numbers are for named entity recognition of a standardized dataset. A system that has to be able to handle a dataset like this as well as text from a specific environment with more difficult sentence constructs (e.g. court judgments) would naturally perform worse.

### 2.3.4 Data Format

As discussed before NER classifies words according to a set of entities. In this thesis we will make use of training data distributed as proposed in the NoSta-D dataset [5]. This is based on the format the famous CoNLL2003 task uses.

The basic entities are these:

- **PER**: person
- **LOC**: location
- **ORG**: organization
- **OTH**: called miscellaneous in the CoNLL2003 task; for all names that are not already in the other categories. Examples are adjectives and events [52].
- **O**: no entity

And the following optional suffixes:

- **deriv**: token is derived from a name, e.g. *österreichisch*
- **part**: part of the token is a name, e.g. *österreichweit*

The named entities use the Inside–outside–beginning (IOB) tagging format as proposed by Sang and Veenstra [46]. Since some entities have multiple words sub-entities to distinguish between the beginning and the inside of an entity are needed. This brings us to a total of 9 entity classes: B-PER, I-PER, B-LOC, I-LOC, B-ORG, I-ORG, B-OTH, I-OTH and O.

In the annotated data each line is a word followed by the corresponding named entity category. For instance, the tag B-PER indicates the beginning of a person name, I-PER indicates inside a person name, and so forth.

## 2.4 Classification of learning algorithms

Machine learning algorithms can be roughly divided into the groups supervised learning, reinforcement learning and unsupervised learning [47]. The basis for classification in this work is supervised learning.

### 2.4.1 Supervised Learning

Supervised Learning is based on training "labeled" records, that is, records where the correct results are already assigned. Supervised learning techniques are used in feedforward and multilayer perceptron models. Learning in supervised models is also called an error backpropagation algorithm. The network is trained by the algorithm based on the error signal, so the difference between calculated and desired output values, the synaptic weights of the neurons adapts. The synaptic weight is proportional to the product of the error signal and the input of the synaptic weight. Based on this principle, each learning period consists of two steps, a forward step and a backward pass [47].

1. In Forward Pass, the network receives an input in the form of a vector which passes through the network neuron by neuron and appears as an output signal in the output layer. The output signal has the form $y(n) = \phi(v(n))$ with $v(n)$ as the local field of a neuron with $v(n) = \sum w(n)y(n)$. The output value $o(n)$ is compared with the actual value $d(n)$ and the error $e(n)$ is calculated. The synaptic weights of the network are not changed in this step.

2. In Backward Pass, the error, i.e. the output value of the layer to be corrected, is returned to the previous layer. This calculates the local gradient for each neuron in each layer so that the synaptic weights of the mesh are changed according to the delta rule:

$$\Delta w(n) = \eta \cdot \delta(n) \cdot y(n) \tag{2.1}$$

This recursive process is repeated until the network has converged.

### 2.4.2 Reinforcement Learning

Reinforcement Learning is a learning method applied to an agent in a dynamic environment that can perform various actions to achieve its goal. Thus, unlike other algorithms, a problem definition and a series of actions to be performed are not defined. Instead, the network must determine those actions that promise the best results [50]. In order to make the success measurable for the agent, he receives a *reward* for executed actions. On the one hand, the reward is given when changing to another state. On the other hand the expected total profit is calculated. The agent then tries to get a high reward from every action and maximize the total profit. The advantages of reinforcement learning in dynamic systems make it particularly interesting for usage in games. A combination of supervised and reinforcement learning is the basis for the well-known AlphaGo, an agent who mastered the board game Go and was able to defeat international top players such as Lee Sedol [48].

### 2.4.3 Unsupervised Learning

The success of machine learning systems often requires very large sets of labeled data. Creating this large amount of data is a major expense. Unsupervised learning is a learning method for inputs that contain patterns that are not or largely unclassified, which makes the learning process very difficult [21].

It is possible to access entirely unweighted records, e.g. large amounts of data from the Internet. In Unsupervised Learning structures such as corners

and edges or certain object classes (e.g. car tires or parts of a face) can be helpful for object recognition when a small, labeled record is used [33].

## 2.5 Conditional random field

Conditional random fields (CRFs) are a type of undirected, probabilistic model used in machine learning. They were introduced by John D. Lafferty in 2001 and are the "traditional" model that had been the state of the art for NER for nearly fifteen years [30]. CRFs can be seen as a sequential extension of the older Maximum Entropy Model (MEM).

Often the term CRF refers to a special form with a linear structure, the *linear chain CRF*. This is typically used to segment sequences. That is, the CRF receives a sequence $X$ as input and outputs an equal length sequence $Y$. Unlike Hidden Markov models (HMMs, a directed sequential data model), a CRF can access the complete information of the input sequence at any point, whereas an HMM sees only the current input. As a result, complex feature sets can be used.

Like all models of machine learning, CRFs must be trained, that is, their parameters must be estimated from data. There are various learning methods for this, such as gradient descent or the quasi-Newton method. The process used is supervised learning: Some sequences are specified, of which both the input and the desired output is known. The learning method then attempts to adjust the parameters in the CRF so that the correct output sequence is predicted for as many sequences in the training data as possible.

### 2.5.1 Features

In "classical" approaches to named entity recognition such as CRF the so called "features" are most important. They can be described as something similar to a human's senses. When CRFs are used at the beginning of a NER task the best possible features set has to be chosen from a wide range of options. It is worth mentioning that in a non-hybrid neural network no features are needed, since the network has to learn what aspects of the input data are most relevant during the training phase.

An example of features are so-called "word features", which state whether a word is capitalized, contains special characters or consists exclusively of uppercase letters.

# Chapter 3

# Artificial neural networks

The purpose of machine learning is to estimate the functional relationship between input and output data [49]. Since our brain is the most efficient machine of learning and cognition, it makes sense to replicate the structures of the brain.

This chapter explains, based on the biological basis, how some selected artificial neural networks are constructed and how they are trained.

## 3.1 Neurons

### 3.1.1 Biological inspiration

Artificial neural networks are modeled after the central nervous system and in particular the brain of humans or animals [36, 43]. The nervous system of living beings can be subdivided into three areas: 1) sensory information systems distributed throughout the body, 2) the central nervous system for information processing, and 3) motor systems for the control of movements. The information processing takes place mainly by the neurons [29]. Azevedo et al. [3] found that the human brain contains an estimated 86 billion neurons which is more than the brains of other primates.

Due to the complexity of these biological nervous systems, the following description is essentially limited to the aspects that are also depicted in

artificial neural networks.

Figure 3.1 shows the simplified structure of such a neuron, which is the basis for artificial neurons.

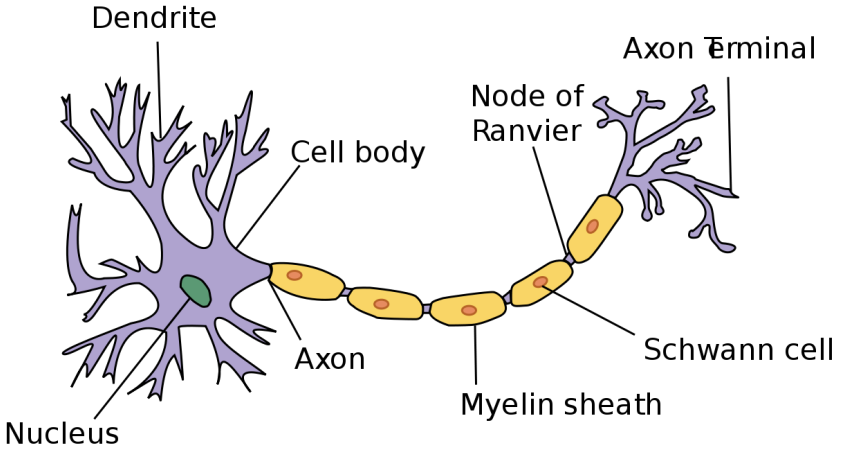



Figure 3.1: Simplified representation of a neuron

Neural networks consist of a large number of neurons. Neurons are cells that can accumulate and transmit electrical activity. On the cell body (soma) are the dendrites (left side of figure 3.1) and the *axon* with the *axon terminals* (right side). The axon terminals lead to the dendrites of other neurons, but do not touch them and form the 10nm to 50nm synaptic cleft. In this way, a human neuron has about 1000 incoming and 1000 outgoing connections to other neurons. These junctions are called *synapses*.

Due to the higher concentration of negatively charged ions on the inside of the cell membrane and positively charged ions on the outside, at rest there is a charge difference (action potential) between the neuron and its environment of about -70mV. This difference can be increased (excitatory connection) or reduced (inhibitory connection) by the discharge of neurotransmitters at the adjacent axon terminals. The effects on a neuron (exciting and inhibiting) add up. When the voltage in the cell exceeds a threshold, the neuron itself generates a pulse that is passed through the axon and synapses to all connected neurons. The rate of propagation of this signal is influenced by the myelin sheath. The stronger an axon is myelinated, the faster the signal spreads. The impulse also causes the action potential to be reset to approximately -70mV [2, 29].

Information can be stored in neural networks in several places. Further-

more, the action potentials of the neurons differ. Depending on how big the fundamental voltage difference is, higher or lower pulses from other neurons are necessary for the neuron itself to generate a pulse. In the form of this threshold, information can be coded [2].

Another form of information storage is the conductivity of the synapses, which increases as more voltage pulses are transmitted. The electrical conductivity decreases over time when no pulses are transmitted [12].

Multiple interconnected neurons can learn and store information by changing these parameters. How this principle can also be used for computers is described in the following section.

### 3.1.2 Theoretical model

As mentioned in the previous section the implementation of artificial neurons is based on the basic principles of biological neurons. The structure and operation of an artificial neuron will now be explained with reference to the following figure and set in relation to the biological inspiration (see 3.1.1).

The biological model is transferred to a theoretical model by Warren S. McCulloch and Walter H. Pitts with the following components [36, 8]:

- an input vector $x = [x_0 x_1 x_2 \dots x_n]$, where $x_0 = 1$ represents an additional input for the on-neuron (*bias*), which can shift the activation function and in some cases represent a threshold [56, 8].
- a vector of weights $w = [w_0 w_1 w_2 \dots w_n]$
- the weighted sum of the inputs $z = \sum_{i=0}^{n} w_i x_i$, called *logit of the neuron* [8, p. 8].
- the activation function $f(z) = y$, where $y$ is the output of the neuron.

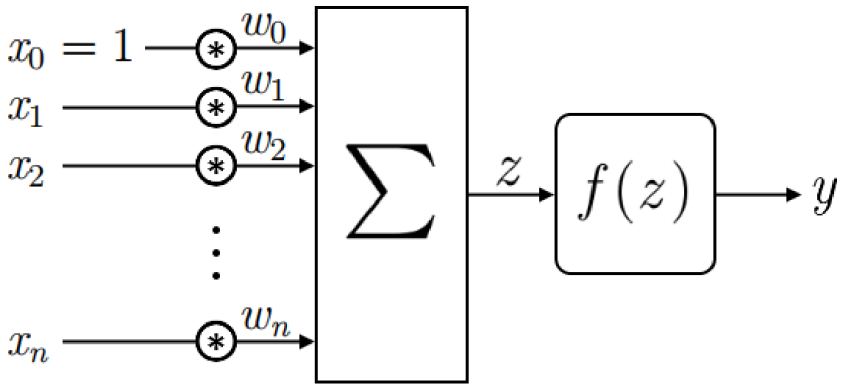



Figure 3.2: vossneural

The input is first mapped to a vector $x$ of length $n$. $x_i$ is multiplied by the corresponding weight $w_i$, which represents a learned weight, where $0 \leq i \leq n$. Here, $x_0$ represents the constant value bias. The products are added up and this sum is called $z$. Then an activation function $f(z)$ is applied, which in the simplest case is a linear activation (identity). In this case $f(z) = z = \sum_{i=0}^{n} w_i x_i$ applies. The output $y$ can be used as an input to several other neurons, in the recurrent case even as an input to itself.

With the theoretical neuron defined above and a corresponding activation function, a linear perceptron according to Rosenblatt can be described. [8]. Thus, according to the *universal approximation theorem*, even neurons with linear activation are capable of arbitrarily approximating continuous functions in the compact space of $\mathbb{R}^n$ [25].

The neurons in the human visual cortex are arranged in layers that, with increasing depth, are able to recognize increasingly complex features (in perceived stimuli). Between an input layer and an output layer there are so-called *hidden layers* in theoretical models, which are mainly responsible for the neural network being able to recognize properties on its own - see 3.4. When neurons always transmit their stimuli to next-layer neurons, they are called *feed-forward* and, with enough depth, *deep neural networks*. However, it is not absolutely necessary to link every neuron with all of the next layer or that all layers have the same number of neurons [8].

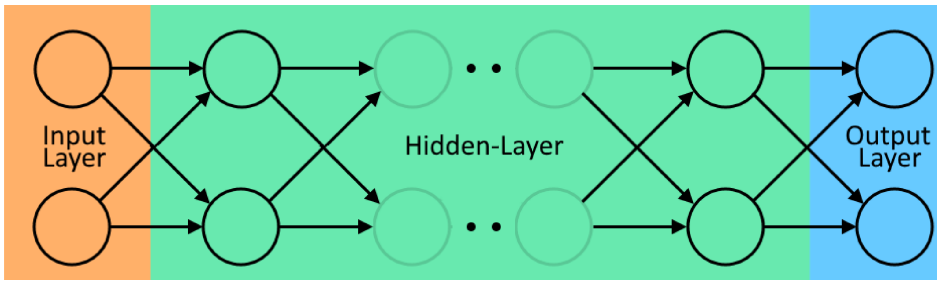



Figure 3.3: vossnetworks

### 3.1.3 Activation functions

As already mentioned, there are several activation functions. Depending on the specific application, some functions are better suited than others. There are also neural networks that use different functions. In order to be able to select one or more activation functions for a neural network, the explanation of some functions follows here.

The simplest form of the activation function is a **Binary step function**. If the input value is below the defined threshold $\theta$, the function returns zero, otherwise one (see also equation 3.1). The threshold is the point at which the activation function has the largest increase [28]. This comes very close to the biological model with inhibitory and excitatory inputs. As an illustration, Figure XXXXX uses a threshold of $\theta = 1$.

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \tag{3.1}$$

Nevertheless, this activation function may have unwanted effects in terms of error correction. With binary output, small changes in weights can affect outputs in ways that make the effects in subsequent neurons unpredictable. In most cases, a learning process in a neural network can only function adequately if small adjustments to the weights of a neuron cause even a small change in the final result.

In the following, $x$ as a function parameter is related to an activation function for the scalar product between the weight and input vector and the threshold value. The function parameter of the *activation x'* should not be

confused with the weights $x$.

$$\sum_{i=1}^{n} w_i x_i = w \cdot x \tag{3.2}$$

$$f(x') = \ f(w \ \cdot x + b) \tag{3.3}$$

A simple representative of these non-binary activation functions is the **identity function** (also called linear activation function). Due to the constant slope corresponding to the value of the input or a multiple thereof, this function can be easily calculated.

$$f(x) = x \tag{3.4}$$

Nevertheless, the use of such a feature is not always ideal. A network that consists of linear layers only acts like a network with only one layer, even if the network has multiple layers. This is because the composition of two linear images is itself linear. With a purely linear function, however, it is not possible to learn complex relationships and not represent linear function mappings. For this reason, neural networks should always contain at least one nonlinear layer. In addition, when optimizing the fault of a network, the derivative is usually used (see chapter XXXXX). However, the derivative of a linear function is always a constant. Thus, the correction does not depend on the delta of the input but always on the same gradient (slope).

A modification of this function is the **rectified linear unit** (ReLU), which has a modified slope from zero (usually a slope of zero). Thus, the ReLU is a simple non-linear function.

A popular and commonly used non-linear function is the **logistic function** (a.k.a. Sigmoid or Soft step function). This function is a kind of smoothed threshold function and has some interesting features.

$$f(x) = \ \sigma(x) = \ \frac{1}{1 + e^{-x}} \tag{3.5}$$

Unlike many other activation functions, the function value of the sigmoid function always moves between 0 and 1 instead of between negative infinity

and positive infinity. The larger $x$ becomes, the farther the e-function value approaches zero, and thus the value of the sigmoid function approaches unity. Conversely, the value of the e-function becomes larger and smaller as $x$ becomes smaller, which causes the value of the sigmoid function to approach zero. Thus, the sigmoid function is well suited as a classification function, since there is a strong slope especially at the $x$ values between -2 and 2. However, the clearer the trend becomes, the further the slope decreases. For very large or small values, a phenomenon called "Vanishing Gradient Problem" can occur for this reason. This can lead to the fact that the network no longer learns or only very slowly. More about this problem can be found in section 3.4.1.

The tangent hyperbolicus function (**TanH**), like the logistic function, is a sigmoid function. However, the range of values is between $-1$ and 1 ($W = [-1; 1]$). The function also approaches these limits asymptotically.

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.6}$$

The **Softmax activation function** is a generalization of the sigmoid function and is always needed when no binary classification, but a selection of one out of several classes must be made. The classification is achieved via the output of probabilities. Unlike the previous activation functions, a $J$-dimensional vector $x$ is passed as input, since the outputs of the neurons must be related by weighted mean to each other. Where $j$ is the number of categories and $i$ is the currently considered neuron.

$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \tag{3.7}$$

If all the outputs of a Softmax layer are summed, the result is again a probability of 1. Especially in the output layer, the Softmax function is often used because the probability distribution makes classification easy.

Which of the activation functions presented here should be used depends mainly on the use case and the data to be processed. Both LeCun and Karpathy advise against the use of the logistics function and recommend tanh or if possible the ReLU [32].

## 3.2 Deep Learning

Deep Learning has become a very popular topic lately. It has been, however, a challenge to define it for many because it has changed forms slowly over the past decade. One useful definition specifies that deep learning deals with a "neural network with more than two layers." Neural networks had to transcend architecturally from the earlier network styles (in conjunction with a lot more processing power) before showing the spectacular results seen in more recent years. [DeepL_Oreilly] It can be argued that "Deep Learning" is just a fancy new term describing artificial neural networks. [SpracTec].

Following are some of the facets in this evolution of neural networks:

- More neurons than previous networks
- More complex ways of connecting layers/neurons in NNs
- Explosion in the amount of computing power available to train
- Automatic feature extraction

O'Reilly defines deep learning as "Neural networks with a large number of parameters and layers in one of four fundamental network architectures":

- Unsupervised pre-trained networks
- Convolutional neural networks
- Recurrent neural networks
- Recursive neural networks

Variations of the aforementioned architectures exist as well.

## 3.3 Feedforward networks

Feedforward networks are characterized by the fact that a layer is only connected to the following layer. Information always moves in one direction and

never goes backwards. They consist of three parts: an input vector $\mathbb{R}^m$, a certain number of hidden layers, and an output vector $\mathbb{R}^n$ (see figure 3.4).

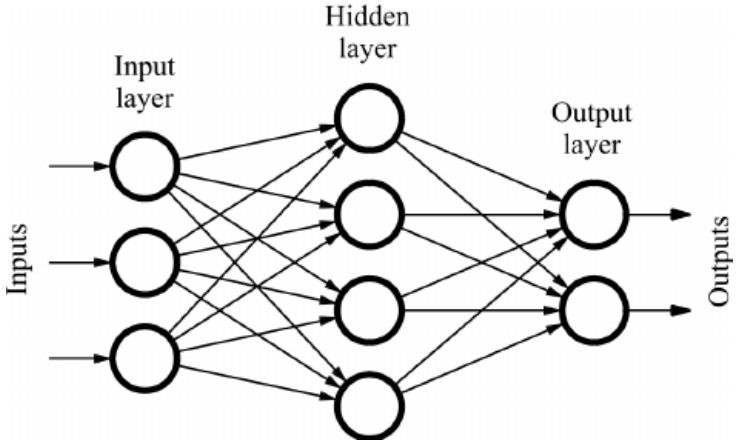



Figure 3.4: ffnetwork

Individual values of these components are called neurons (nodes). To calculate the output, input neurons are linked to a *hidden layer* using trained weights. Likewise, consecutive hidden layers are linked together, and the last hidden layer is the output neurons. Edge values entering a node are summed up and restricted with an activation function.

Let's assume that we have a feedforward network with only one *hidden layer*. Then the network can be written in matrix form:

$$h = \phi(W \cdot x) \tag{3.8}$$

$$y = V \cdot h \tag{3.9}$$

Here is what the variables mean:

- $x$ is the input vector $\mathbb{R}^m$
- $W$ is a weighting matrix of the form $\mathbb{R}^{k \times m}$
- $\phi$ is the activation function for the values of the *hidden layer* $h$.
- $V$ is the second weighting matrix $\mathbb{R}^{n \times k}$ to associate the *hidden layer* with output $y$

Three activation functions are most commonly used. First, the sigmoid function that maps the inputs to a range of (0, 1). Next, the hyperbolic function that has the value range of (-1, 1). And thirdly the Rectified Linear Unit (ReLU).

1. $\sigma(x) = \frac{1}{1+e^{-x}}$
2. $tanh(x)$
3. $ReLU(x) = max(0, x)$

If there are no hidden layers a feedforward network is called *single-layer feedforward network*. It consists of an input layer with $n$ inputs and $m$ outputs only. The information flows in one direction - from the input to the output layer.

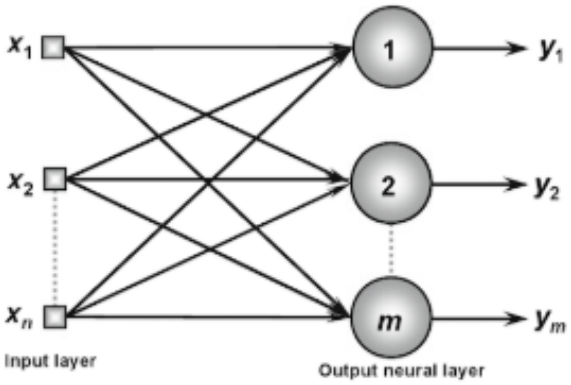



Figure 3.5: Example of a single-layer feedforward network

### 3.3.1 Single-layer perceptron

The simplest form of a feedforward neural network is the *single-layer perceptron*. It consists of input neurons and a single layer of output neurons

At the beginning of the sixties, the model of the perceptron presented here was developed by Frank Rosenblatt, which is characterized by the following structure [56]:

- a layer of input neurons permanently connected to input cells, which are usually assumed to be binary inputs
- weighted connections $w_1$,. , , , $w_n$, which may take any real values, and lead from the above input neurons to one or more output neurons

Each output neuron represents a classifier and fires exactly (i.e. outputs 1) if the sum $s$ of the $n$ different $w_i$-weighted inputs $x_i$ exceed a certain threshold $\theta$. Otherwise, the output $o$ is 0:

$$s = \sum_{i=1}^{n} x_1 \cdot w_i \tag{3.10}$$

$$o = f(s) = \{ \begin{cases} 1 & if\ s > \theta, \\ 0 & otherwise \end{cases} \tag{3.11}$$

This binary classifier is commonly known as a *threshold function*. Various other implementations of the threshold function $f$ are conceivable.

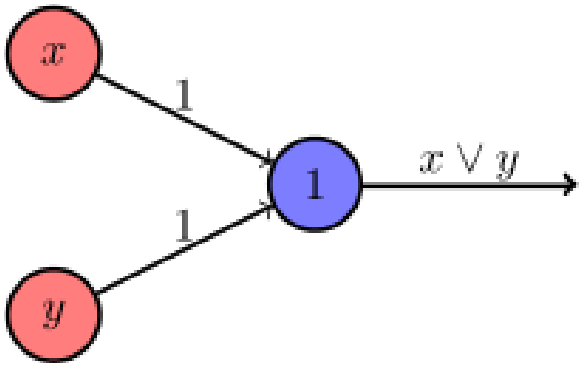



Figure 3.6: Simple perceptron that realizes a logical OR.

However, because there is only one layer of trainable weights, the single-layer perceptron can only represent linear functions, and thus it is applicable to far fewer problems than a multilayered network. Only with a *multilayer perceptron* it is possible to map non-linear functions.

### 3.3.2 Multilayer perceptron

The limitation of the single-layer perceptron could later be solved with the multilayer perceptron, in which there is at least one hidden layer in addition to the output layer. All neurons in a layer are fully linked to the neurons of the next layer.

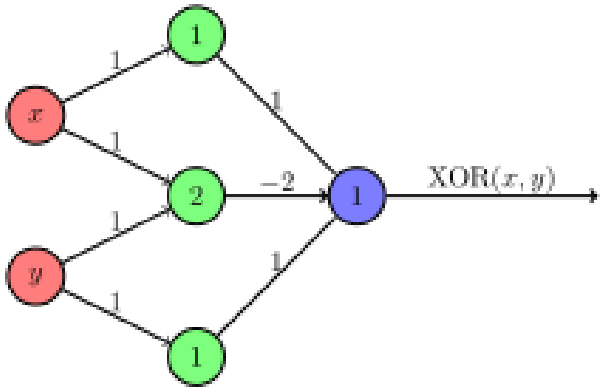



Figure 3.7: Two-layer perceptron for calculating the XOR function

Other network topologies also exist:

- Fully connected: The neurons of one layer are connected to the neurons of all following layers
- Shortcuts: Some neurons are not only connected to all neurons of the next layer, but also to neurons of layers beyond that.

A multilayer perceptron can be trained, among other things, with the backpropagation (see chapter 3.6.3). For that the weights of the connections are adapted so that the network can classify the desired patterns with supervised learning.

The extension of these network topologies by adding more hidden layers and introduction of other architectures (e.g. recurrent neural networks), which are also usually trained by means of backpropagation, is today summarized under the buzzword *Deep Learning* (see section 3.2).

### 3.3.3 Convolutional neural networks

Convolutional Neural Networks (CNN) are powerful representatives of the feed-forward networks. They emerged from the study of the brain's visual cortex and have been in use for tasks such as image recognition since the 1980s. The first practical application of a CNN was introduced in 1989 by Bell Labs. Yann LeCun combined the older concepts of CNN and backpropagation and applied them to the classification of handwritten numbers. The resulting network, nicknamed LeNet, was used by the United States Postal Service in the 1990s to automate the reading of postal codes on envelopes [31].

In principle, CNNs can be considered as a kind of neural network that uses many identical copies of the same neuron [1]. As a result, the network can consist of many neurons and computationally large models can be expressed. At the same time, however, the number of actual parameters to be learned - the values which describe the behavior of the neurons - remains small.

The idea of using multiple copies of the same neuron is similar to the abstraction of functions in computer science and mathematics. In programming a function is written once and used it in many places. Not writing the same code many times in different places makes programming faster and results in fewer bugs. Similarly, a CNN can learn a neuron once and use it in many places, making it easier to learn the model and reduce errors.

A classical convolutional neural layer consists of one or many *convolutional layers*, followed by a *pooling layer*:

A convolutional layer reduces the number of outputs relative to input using convolution. For this purpose, a so called kernel (a filter), which is smaller or equal to the input matrix, is moved over the input step by step. The elements of the matrix are calculated with the weights of the mask, added up and then give an element in the output matrix. The number of times the mask was applied to the input is the number of output elements. You can apply multiple masks in one layer. A mask is used for the entire input with the same weights. This is also referred to as shared weights, since the weights remain the same for all applications of the mask. In image processing, this ensures that the same patterns are recognized throughout the image. Another advantage is that fewer parameters need to be optimized, which speeds up

[1] Other neural networks exist that make use of copies of the same neuron such as recurrent neural networks and recursive neural networks

the processing.

In the pooling step, unnecessary information is discarded. There are different types of pooling. By far the most widespread is *max-pooling*, whereby from each 2×2 square of neurons of the convolutional layer only the activity of the most active neuron is retained for the subsequent calculation steps. The activity of the remaining neurons is discarded.

In recent years, CNNs have achieved superhuman performance in some complex visual tasks, due to increased computational power, the amount of training data available, and the development of improved training algorithms. They operate image search services, self-driving cars, automatic video classification systems and much more. In addition, CNNs are not limited to visual perception: they are also successful in other tasks such as speech recognition and natural language processing.

## 3.4 Recurrent neural networks

Recurrent Neural Networks (RNNs) are a special architecture of neural networks, that can effectively incorporate temporal dependencies within the input data. They are distinguished by connections of neurons of one layer to neurons of the same or a previous layer. This is done to detect mostly time-coded information in the data. The existence of such feedbacks is the main differentiating factor from feedforward networks. Recurrent networks also allow variable-length input to be processed by feeding back data from the last calculation through feedback links. Recurrent networks are trained using *backpropagation through time* (see section 3.6.4).

As with feedforward networks, recurrent networks use information from the current time $t$. Additionally they use information from the previous time $t-1$. Thus, there are two inputs per label to be calculated: the standard input $x_t$ and information from the previous hidden layer $h_{t-1}$.

These values can provide a wide range of information of the sequence as the respective information $h_{t-1}$ is passed on for each input in the RNN. They make it possible to find correlations between events that are far apart, these correlations being called *long-term dependencies*.

Let's assume - as with feedforward networks - that we have a recurrent

neural network with only one hidden layer. Then the output can be calculated as follows:

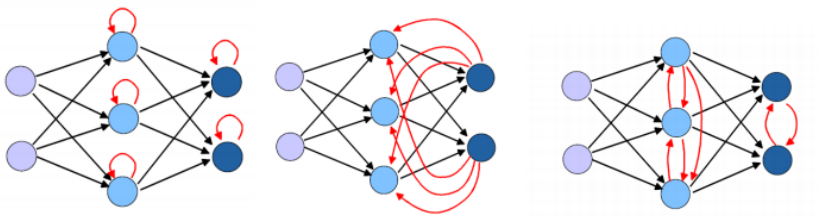
$$h = \phi(W \cdot x_t + U \cdot h_{t-1}) \tag{3.12}$$

$$y_t = V \cdot h_t \tag{3.13}$$

The only difference to the feedforward network is that $U \cdot h_{t-1}$ is added to the new hidden layer. Only for the first value of an input sequence this summand is omitted. $h_{t-1}$ denotes values of the previous hidden layer with respect to $t$. $U$ is a new weighting matrix between hidden layers that has to be trained exactly like $W$. It indicates how strong the influence of past events is on a new input.

Recurrent networks can be subdivided into neural networks with:

1. **direct feedback**: Here, connections from the output to the input of the same unit exist. This means that the activity level or the output of the unit becomes an input of the same unit.
2. **indirect feedback**: In this case the activity is sent back to previous layers of the neural network.
3. **lateral feedback**: Here the feedback of the information of a unit takes place at neurons, which are in the same layer. An example of such lateral feedbacks are the horizontal cells in the human eye.
4. **complete connections**: These networks have connections between all neurons.



(a) direct feedback (b) indirect feedback (c) lateral feedback

Figure 3.8: Representation of the different types of feedback

Recurrent neural networks are most commonly used in problems that require the processing of sequences. Examples include handwriting recognition, speech recognition and machine translation. Additionally, RNNs are a

reasonable approach for tasks that need to model sequences with different length, as the network can be dynamically unrolled according to the length of the input sequence.

Unlike feedforward networks, recurrent networks can capture sequences and use them to generate their output. This is useful, for example, in automatic text generation, where a subsequent letter always depends on the previous one and can not be arbitrarily chosen. An RNN is able to intentionally follow one specific letter with another to form meaningful words, a feedforward network can not.

The predominant type of recurrent neural networks are LSTMs (see section 3.5), or similar variants based on direct feedback.

### 3.4.1 Vanishing gradient problem

The gradient represents the change in all weights in relation to the change in error. The problems of vanishing and exploding gradients occur when the errors are transmitted over many time steps. If the gradient is unknown, one can not bring about a reduction of the error by changing the weights and the network is unable to learn. It can come to unknown gradients, as information flowing through a network is multiplied many times. If you multiply an amount repeatedly with a value just above 1, the result can become unmanageable and in this case, it is called an **exploding gradient**. Conversely, if any number is repeatedly multiplied with a factor smaller than 1 the result will be a very small number. The value can become so small that it can no longer be learned by a network. This is called a **vanishing gradient**.

Which of the two phenomena occurs depends on whether the weight of the recurrent transitions $w_{jj} > 0$ or $w_{jj} < 1$ and how the activation function is designed in the hidden node. With a sigmoidal activation function, the vanishing gradient problem is stronger, but with a rectified linear unit $max(0; x)$ it is easier to reach the exploding gradient. Pascanu et al. gives a thorough mathematical treatment of vanishing and exploding gradient problems that characterizes the exact conditions under which these problems can occur [38]. Given these conditions, they propose an approach to training via a regularization expression that forces the weights to values where the gradient neither disappears nor explodes.

The problem of exploding gradients can be overcome by a reasonable upper limit. For the vanishing gradients finding a solution appears to be much more difficult and this topic is still the subject of research.

### 3.4.2 Long-term dependencies problem

As mentioned earlier, recurrent neural networks are able to recognize sequences and work with dependencies. Unfortunately, this ability is limited. If there is only a small time gap between interdependent data, an RNN will be able to recognize this relationship and draw the right conclusions. However, if the time interval between the input of the data and the time at which they are needed for a result becomes very large, an RNN can no longer establish this relationship. As an example, Olah gives a language model that predicts the next word depending on the previous one. An RNN is able to predict the last word in the sentence "The clouds are in the sky." Since the distance between sky and clouds is very small. It is much more difficult to predict the last word in the text "I grew up in Greece. I speak Greek.". The previous words merely suggest that the name of a language must follow. The context that it is most likely "Greek" is obtained only through the first sentence. However, an RNN can not remember a whole text and thus does not learn the connection between Greece and Greek.

In theory RNNs should be able to handle these "long-term dependencies". Sadly, in practice this is not the case. Hochreiter and Bengio explored this problem in their papers and found some fundamental reasons for that [22, 4].

To solve the problem of long-term dependencies, *Long Short-Term Memory* networks are used.

## 3.5 Long Short-Term Memory

Long Short-Term Memory (LSTM) networks are a special type of recurrent network that can work with long-term dependencies. They were first introduced by Hochreiter and Schmidhuber [24] and further refined by many other researchers (e.g. Felix Gers [15]). LSTMs work tremendously well on a large variety of problems and are central to the boom in machine learning that is happening in the last years.

LSTMs have been designed to specifically solve the problem of long-term dependencies. Storing information over a long period of time is their default behavior and not something that has to be learned with difficulty.

They consist of memory cells into which information can be written and read from again. With the help of so-called gates, which are opened or closed, a cell decides what is stored and when a read, write and delete is allowed. These gates are analogous and implemented by a sigmoid function, resulting in a range of 0 to 1. Analogous has the advantage over digital that it is differentiable and therefore suitable for backpropagation. Just like the inputs in the feedforward and recurrent networks, the gates have weights. These weights are also adjusted during the learning process so that the cell learns when to insert, read or delete data.

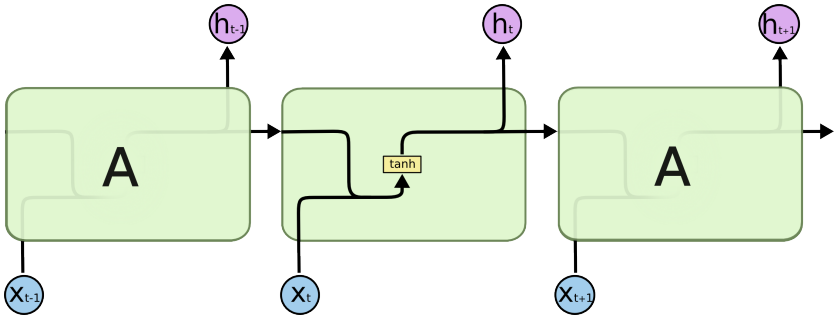



Figure 3.9: The repeating module in a standard RNN contains a single layer.

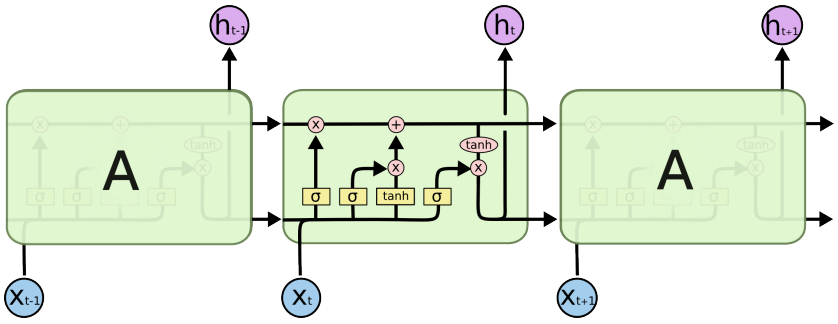



Figure 3.10: The repeating module in an LSTM contains four interacting layers.

Figures 3.9 and 3.10 show a simple recurrent network at the top and an LSTM network at the bottom. Both are represented over three time steps, whereby the second step reflects their inner life. While in the RNN a simple

structure with only one function (yellow box in the figure) is responsible for the result, an LSTM uses four functions. How these functions interact with each other and get to a result is explained step by step in the next section.

### 3.5.1 Structure of a memory cell

#### Cell state

The cell state is the actual location or memory of the LSTM. Figure 3.11 shows the progression through a memory cell. On the left, the cell state is taken from the previous time step and passed on to the next. In the middle are two operations that can change the state during this time step. What task they have follows in the section cell state update.

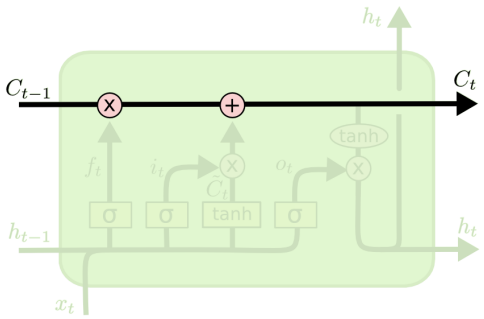



Figure 3.11: LSTM cell state

#### Forget gate

The Forget Gate uses the sigmoid function to decide which information to delete. It looks at the old output $h_{t-1}$ and the new input $x_t$ and gives a value between 0 and 1 for each information in the cell state $C_{t-1}$. A 1 means "completely keep this" and a 0 “completely get rid of this”. This is done since sometimes it can be useful to forget things, e.g. the memory cell can be reset if it is known that the following data is unrelated to the previous.

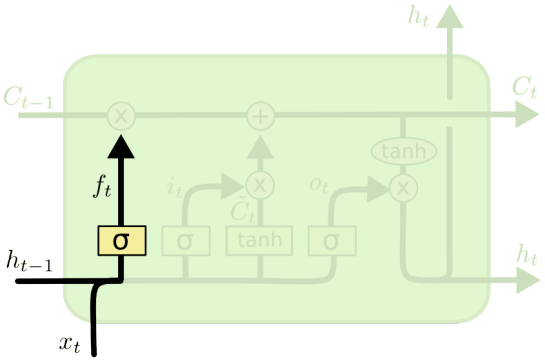



Figure 3.12: XXXXX

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.14}$$

### Input

The decision as to which data should be stored consists of two parts. First the sigmoid layer $i_t$ called "input gate layer" gives a result between 0 and 1. By that it decides which cell state we will update. In addition, a tanh layer creates a vector of candidate values, $\tilde{C}_t$, that could be stored in the state.

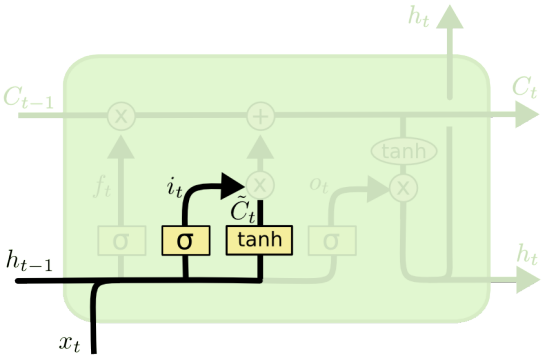



Figure 3.13: XXXXX

$$i_t = \sigma(W_i \cdot [h_{t-1},\ x_t]\ +\ b_i) \tag{3.15}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1},\ Xt]\ +\ b_C) \tag{3.16}$$

### Update of cell state

After the forget gate and the input gate have decided what to do with the data, the old cell state $C_{t-1}$ is updated and becomes $C_t$. For this $C_{t-1}$ is multiplied by the result $f_t$ of the forget gate, thus erasing everything that should be forgotten. Subsequently, the data scaled by the input gate and prepared by the tanh function are added to the cell state.

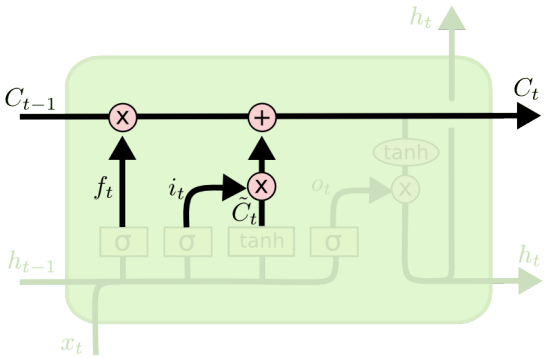



Figure 3.14: XXXXX

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.17}$$

### Output

The output is made using an output gate, which is also a sigmoid function. The cell state is passed through a tanh function and then multiplied by the result of the sigmoid function. The tanh function converts the values to a range of $-1$ to 1, which is the desired range of artificial neural network outputs. With the multiplication we make sure to only output the parts we decided to.

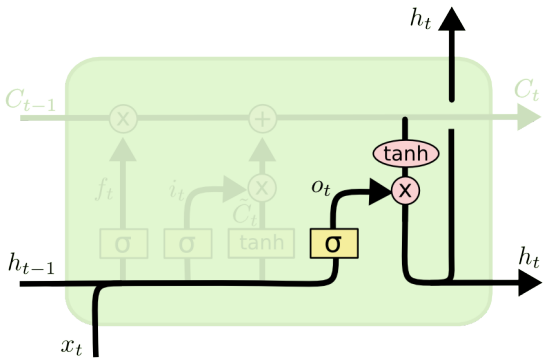



Figure 3.15: XXXXX

$$O_t = \sigma(W_o\ [h_{t-1},\ x_t]\ +\ b_o) \tag{3.18}$$

$$h_t = o_t * \tanh(Ct) \tag{3.19}$$

**Summary**

A memory location consists of a cell state that acts as a memory and three gates that protect and control the cell state. Each gate uses a sigmoid function that outputs a range of values between 0 and 1, which determines the intensity of the action. The forget gate is responsible for deleting unwanted information. The input gate takes over the re-memorizing action by storing new information in the cell state. The output gate determines the information that is output.

### 3.5.2 Bidirectional LSTM

While simple LSTMs may cover a wide context space, they are not entirely perfect. An important point is that context is only learned on the left side. Timing can only look at their past, but not at the future. To get around this problem Bidirectional LSTMs (BiLSTMs) have been proposed [18, 9, 51]. This procedure requires two LSTMs: $LSTM_1$ and $LSTM_2$. $LSTM_1$ is trained normally on the training data. For $LSTM_2$ the entire training body is reversed, that is, viewed from the back to the front. Thus $LSTM_2$

learns the right-sided context and $LSTM_1$ the left-sided. If you give these LSTMs a sentence of length $n$ as input, this is put into $LSTM_1$ and reversed into $LSTM_2$. The results of an input are concatenated and serve as output: $ht = h1_t \oplus h2_{n-t}$.

Typically, BiLSTMs perform slightly better on NLP problems than LSTMs. After all, in language it is never clear whether relevant context is before or after a word [11].

## 3.6 Training Neural Networks

The characteristic feature of neural networks is the ability to systematically learn from patterns in the input data. Once a network has learned the connections between input and output by generalizing a solution for arbitrary input values should also be produced or approximated. This chapter - like the whole thesis - focuses on supervised learning for neural networks.

Learning happens by changing the weights and thresholds of the neurons during training. The most commonly used method is the adaptation of the connection weights, because in addition the construction and dismantling of connections can be modeled. If all neurons are initially connected to each other, unnecessary connections can be weighted with zero during a learning process and thus removed. Depending on the network type and application, different learning methods are available, which are explained in this subchapter.

For supervised learning techniques of neural networks, there are various processes related to the timing at which the weights are adjusted. The first option is to calculate the output of the network and the error for each individual input vector and to change the connection weights. This approach is referred to as **online learning** [29, p. 23]. However, P. Werbos additionally distinguishes between *real-time learning*, where the training data is trained only once, and *pattern learning*, in which there are several passes through the record [55].

**Batch / offline learning** differs from the online methods, because a weight change is performed only after the processing of the entire training data set [29, p. 24].

Another online method is *stochastic learning*, which randomly selects an example from the data set at each iteration.

### 3.6.1 Hebb's rule

The Hebb Rule is one of the simplest learning rules. It models the aspect of the natural neural networks that the synapses become stronger the more voltage pulses they transmit (see section 3.1.1).

In his book *The Organization of Behavior* Donald Olding Hebb published the following assumption [20]:

> Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability. [...] When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

The Hebb Rule is often simplified by the saying "What fires together, wires together" [42]. This means that weights between two neurons are only changed if both neurons are active at the same time. The weights are increased depending on the learning parameter. Since for the classical Hebb rule the neurons can only have zero or a positive value, this learning rule has a few disadvantages: Weights can only ever be increased, but never lowered. Additionally, due to the limited value range, there is only a small range of possible states. However, these issues can be addressed by various adjustments, such as lowering the weights for neurons that are not active at the same time. The Hebb Rule can be used for supervised, unsupervised and reinforcement learning.

### 3.6.2 Delta rule

The Delta rule, also known as Least Mean Square (LMS) method, is a gradient descent learn rule for supervised learning [42]. This method is only suitable for the training of two-layered non-recurrent networks.

The procedure is that the existing output signal of a neuron is compared with the desired output signal and changes are made according to the deviation. This process is reminiscent of the basic principle of backpropagation and is believed to be its predecessor.

### 3.6.3 Backpropagation

Error backpropagation is a gradient-based learning method for neural networks [32]. It was introduced in 1974 by Paul Werbos, but became known mainly through a publication by Rumelhart, Hinton and Williams [54, 45].

The procedure is explained in this section using a multilayer perceptron with a supervised learning process. The aim of this learning method is to adjust the connection weights between the neurons so that the output of the neural network coincides with the desired output with sufficient accuracy. In addition to the input values, the training data therefore also contains the desired output values. The backpropagation method includes the following steps.

**1. Calculate 1st output of the network**

At the beginning it is necessary to calculate for the input $x$ the output $o$ of the neural network. For the example network in Figure FFFFF, the calculation is the same as for the multilayer perceptron. The activation function is the logistics function $\sigma$ (see section 3.1.3) and the connection weights were initialized at random. As can be seen in the figure, the network outputs the value $x = \{1, 1\}$ at $o = \{1\}$.

**2. Calculating errors**

In order to demonstrate the error feedback, the network in Figure XXXXX is to be trained by example data so that it maps the XOR function. If $0 = false$ and $1 = true$, the setpoint is defined as:[JORM]

$$y_1 = (x_1 \vee \neg x_2) \vee (\neg x_1 \wedge x_2) \tag{3.20}$$

The choice of error function depends on the problem / task. It indicates the deviation of the setpoint from the actual output and in the simplest case is the difference between these two values. For several output values, however,

the errors could be canceled, which is why in these cases e.g. the mean square error (Equation XXXXX) is more appropriate. The factor before the sum serves to simplify the derivation.

$$E = \frac{1}{2}\sum_{i=1}^{N}(y_i - o_i)^2 \tag{3.21}$$

Where $y_i$ is the setpoint and $o_i$ is the actual output of a neuron. For the example in Figure XXXXX the error given by this formula is $E = \frac{1}{2}(y_1 - 0_1)^2 = 0.5$. For the network to approximate the desired function, this error must be minimized.

**3. Form partial derivatives and calculate changes**

To minimize the error, the connection weights of the network should be adapted. For this, the partial derivative of the error is calculated according to the edge weights. This gives the amount and direction of the change. A prerequisite for the use of backpropagation is therefore the use of differentiable activation functions. Since only the error at the output neurons is known at first, error recovery is also started in this layer. The adjustment of the weights between hidden layer $j$ and output layer $k$ is calculated according to the following formula:

$$\triangle w_{jk} = -\alpha\frac{\delta E}{\delta w_{jk}} \tag{3.22}$$

The learning rate alpha determines the influence of the change on the edge weight and the negative sign causes the weight to be adjusted opposite to the curve increase (towards minimum). Since the error depends on the output value $o$ and this is determined by the input values and weights $w$, the following formula applies:[JORM]

$$\frac{\delta E}{\delta w_{jk}} = \frac{\delta E}{\delta o_k}\cdot\frac{\delta o_k}{\delta w_{jk}} = \frac{\delta}{\delta o_k}\,\frac{1}{2}\sum_k (y_k - o_k)^2 \cdot \frac{\delta}{\delta w_{jk}}\,\sigma\left(\sum_j w_{jk}\cdot o_j\right) \tag{3.23}$$

Deriving the error and the output yields for the last layer:[JORM]

$$\frac{\delta E}{\delta w_{jk}} = \delta_k \cdot o_j = (y_k - o_k) \cdot o_k(1 - o_k) \cdot o_j \tag{3.24}$$

From the penultimate layer, first of all the existing error must be calculated. For this purpose, the error calculated above (Equation XXXXX) of all outgoing connections of a neuron is added up and then divided into the incoming connections:[JORM]

$$\frac{\delta E}{\delta w_{ij}} = \delta_j \cdot o_i = \sum_k (\delta_k \cdot w_{jk}) \cdot o_j(1 - o_j) \cdot o_i \tag{3.25}$$

The intermediate steps and a complete derivation can be taken, for example, from the book Artificial Intelligence by Lämmel and Cleve. [JORM]

By applying the learning rate according to Equation 3.22 to the last two equations (3.24 and 3.25), the sought weight change ($\Delta w_{jk}$ or $\Delta w_{ij}$) can be calculated.

As with *forward propagation*, the use of matrices and vectors makes it possible to calculate the weight changes for all weights of a compound layer with one expression. For the example above, the following weight changes result:[JORM]

$$\begin{aligned} \Delta w_{jk} &= \alpha \cdot (0 - 0.64) \cdot 0.64 \cdot (1 - 0.64) \cdot \begin{pmatrix} 0 \\ 0.57 \end{pmatrix} \\ &= \alpha \cdot -0.15 \cdot \begin{pmatrix} 0 \\ 0.57 \end{pmatrix} \\ &= \alpha \cdot \begin{pmatrix} 0 \\ -0.08 \end{pmatrix} \end{aligned} \tag{3.26}$$

**4. Adjusting the connection weights**

The weights are then adjusted for each layer by adding the calculated change:[JORM]

$$w_{jk} = w_{jk} + \Delta w_{jk} \tag{3.27}$$

This process with the four steps presented is repeated until the weights converge or the error is sufficiently small. As mentioned in the delta rule, the weights can also be adjusted at different times (online / batch). Which of these learning methods is suitable depends on the task.

Backpropagation is a very popular learning method for multilayer perceptrons, but cannot easily be used for networks such as e.g. recurrent networks. However, the *Backpropagation Through Time Algorithm* described below is capable of doing so.

### 3.6.4 Backpropagation through time

Vanishing Training recurrent networks has long been considered difficult. Especially because of the difficulty of learning from far-reaching dependencies, as described by Bengio et al. [4] and expanded by Hochreiter et al. [23]. The difficulty stems from the problems described in the previous sections (see subsection 3.4.1).

Since with recurrent networks the result and thus the error does not only depend on the current time step, backpropagation has to be extended in order to work meaningfully. Backpropagation through time (BPTT), introduced in 1990 by Paul Werbos, extends the normal backpropagation by a factor of time [55]. This makes it possible that the influence on the error can be determined of the weights of previous steps. BPTT is needed to train recurrent neural networks.

The basic idea is that a recurrent network can be interpreted as a set of multilayer perceptrons that are interconnected. Each time step is represented as a multilayer perceptron. Consequently, this chaining of networks can be considered as a large, multilayered network where training with backpropagation is possible. It should be noted, however, that all weights occur several times, but at different times.

Ways to merge these are, among others, the average and the weighted average, where changes are less and less influential the longer ago they took place.

The training data for BPTT is an ordered sequence of $k$ input / output pairs. Next, the RNN must be unfolded (figure 3.16). The unfolded network then contains $k$ inputs and $k$ outputs. This happens by expanding the net-

work over time. Each input signal, hidden neuron and each output signal is duplicated and timestamped. Thus, the network is completely duplicated for each required time step. If you have a network with four hidden nodes and two time steps you get a network with eight hidden nodes. In the example in Figure 3.16, a network with a hidden node (left side) is expanded an arbitrary number of times (right side). The resulting network behaves like a feedforward network and can use the normal backpropagation algorithm.

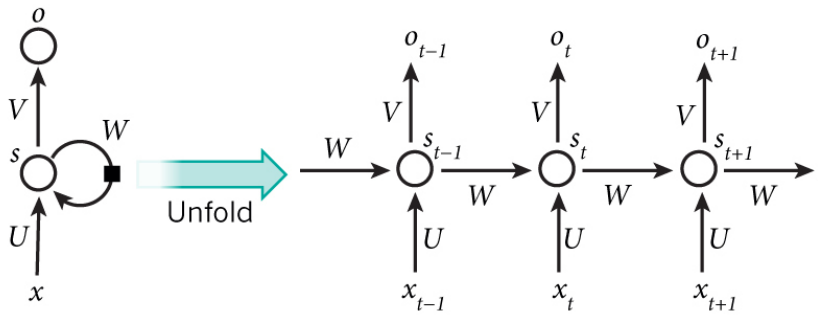



Figure 3.16: Backpropagation Through Time: Unfolding of an RNN over time.

Of course, this method requires more memory since all previous states and data must be stored for a certain number of time steps.

Since BPTT becomes very complex for long training sequences, there are various possibilities of approximation. One of these is Truncated backpropagation through time (TBPTT). It is proposed as a solution to the exploding gradient problem for continuously moving networks [Williams and Zipser, 1989]. TBPTT specifies a maximum number of time steps along which errors can be transmitted. While TBPTT with a small cutoff can be used to alleviate the exploding gradient problem, sacrificing the ability to learn long-range dependencies requires it. The LSTM architecture described below uses carefully constructed nodes with recurring fixed-weight edges as a solution to the vanishing gradient problem.

## 3.7 Word Embeddings

A word embedding is a vector representation of a word. This vector is a unique numerical representation of that word and has any number of dimen-

sions. The goal of word embedding is that words with a similar meaning have a small distance in the vector space. This way, the computer system can later easily find similar words based on the word embedding with trivial vector arithmetic. The better these word embeddings are, the better the computer can work with this representation.[HUON]

In the past several methods to train word embeddings from unlabelled data have been proposed. Mikolov pretty much revolutionized the field of word embeddings when he proposed his algorithm word2vec [37]. One of the best algorithms that emerged in the wake of this breakthrough is Stanford University's Global Vectors for Word Representation (GloVe) [39]. Word2vec and GloVe have been shown to be the most successful in intrinsic tasks (Schnabel et. al) and are most commonly used.[MAI]

Which methods makes the most sense, of course, depends on the task. Here are some common methods that have been proven to work with neural networks.[JAAI]

### 3.7.1 Bag-Of-Words (BOW)

The simplest approach to word embeddings is the bag-of-words (BOW) model. The first reference to this model is from the 1954 article *Distributional Structure* by Zellig Harris [19].

Bag-of-word models work like this: After an initial cleanup of the entire text corpus of special characters, a dictionary of all occurring words is created. Usually this dictionary is sorted in descending order by the number of occurrences and each word is assigned a number. The number zero is not used because it is later used for words that are not in the dictionary (Out-Of-Vocabulary, OOV). That is, the number 1 in the BOW approach represents the word that is most common, the number 2 the second most common, and so on. With this dictionary every existing word in the text corpus is assigned a number.

This may already be sufficient for certain simple tasks. However, the bag-of-words approach has the following disadvantage: The size of the number is unrelated to the meaning of the word. As input for a neural network, therefore, only one one-hot encoded vector can be used per word, whose length corresponds to the number of words in the dictionary. This quickly creates vectors with a length of over 100,000, each containing only one "1"

and otherwise only zeros. These so-called *sparse vectors* make learning very difficult for neural networks and cause a high complexity.

### 3.7.2 Unsupervised Word Vectors – Word2Vec, GloVe

Ideally, the mathematical representation of a word should also represent the meaning of that word in the context of other words.

A much-used approach to achieving this is to have an algorithm arrange all the words in the text corpus in a multidimensional space based on their occurrence and surrounding words so that words that often appear in similar contexts also have a similar vector. Such algorithms use unsupervised learning (see section 2.4.3), i.e. they independently seek the representation model without external constraints, minimizing the differences between related words.

The result is a vector representation for each word contained in the text corpus. Depending on the model, this vector has between 100 and 300 dimensions and is significantly less complex than e.g. a one-hot-encoded BOW vector with 100,000 dimensions. Most importantly, each word vector also represents, at least to some extent, the semantic meaning of the word in context.[JAAI]

In the embedding space those words are close to each other that often occur in similar contexts. This frequently leads to words with similar meanings being neighbors. Figure 3.17 zooms in onto certain areas of a learned word embedding space to show how semantically similar words map to representations that are close together [17].

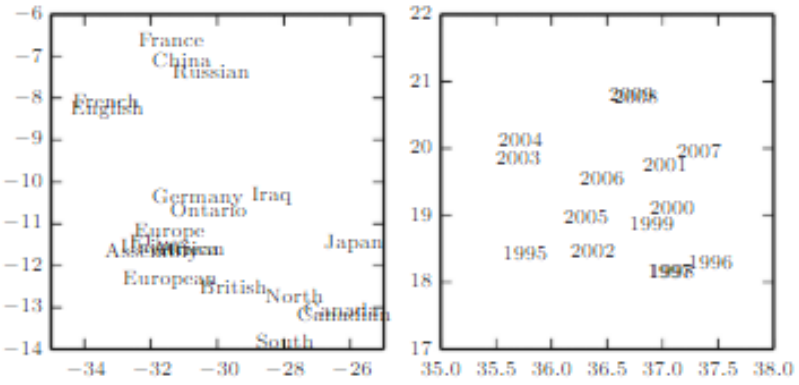



Figure 3.17: Two-dimensional visualizations of word embeddings

One of the benefits of Word2Vec and GloVe is that they can be pre-trained on a huge corpus of documents. Using models that have been pre-trained on all Wikipedia articles of a certain language has become common in NLP tasks. Just like with other machine learning tasks the rule of thumb is that word embeddings become more precise the bigger the corpus is that they were produced from. Such a pre-trained model can be stored and plugged into say a named entity recognition model. Also it is possible to do arithmetic with words using pre-trained Word2Vec or GloVe models:

The vector for the word KING minus the vector for the word MAN plus the vector for the word WOMAN results in the vector for QUEEN. Of course this is not exact, but the vector for QUEEN is the closest vector to the result and can therefore be uniquely determined as a result.

**KING - MAN + WOMAN = QUEEN**

The models actually contain a relatively useful representation of the meaning of words. Of course, this meaning is not comprehensive and refers only to individual words and not to word rolls or even entire passages. However, the practical vector format can be well handled by neural networks of all types, and Word2Vec, GloVe and other pre-trained models are therefore the basis for many current applications.

The disadvantage of Unsupervised Word Vectors is that you need a very very large and as "clean" text corpus as possible to generate a meaningful vector representation. In addition, the algorithms are very computationally intensive. For individual projects, this is usually not feasible because it

simply lacks the data volume required for meaningful training.

When using pre-trained models like Word2Vec and GloVe, there is the problem that even these models only know words from their corpus - all words that were not present during training receive a null vector. This can be particularly difficult for special areas with their own jargon. In this thesis we are looking at the German-speaking legal environment. If a pre-trained model is used that does not know legal words such as "Gebarung" and "Gutachten" it will assign zero vectors and then even the best neural network architecture will not be able to properly learn.

Another disadvantage is the storage capacity in RAM required by pre-trained models: In order to have quick access to the word vectors, the complete dictionary including the associated vectors must be present in memory. This quickly consumes 1-2 GB of RAM just for keeping the model. Some models, such as spaCy have therefore begun to let the Word Vectors learn from a CNN downstream - then only the CNN model needs to be loaded and this then creates the desired vector for each word upon request. An alternative to Word2Vec and Co are Supervised Word Vectors.

# Chapter 4

# Approach

In this chapter, the theoretical constituents of our approach are discussed. Although it refrains from including implementational details, some of them are mentioned if they impose constraints that influenced more significant design decisions.

## 4.1 Procurement of training data

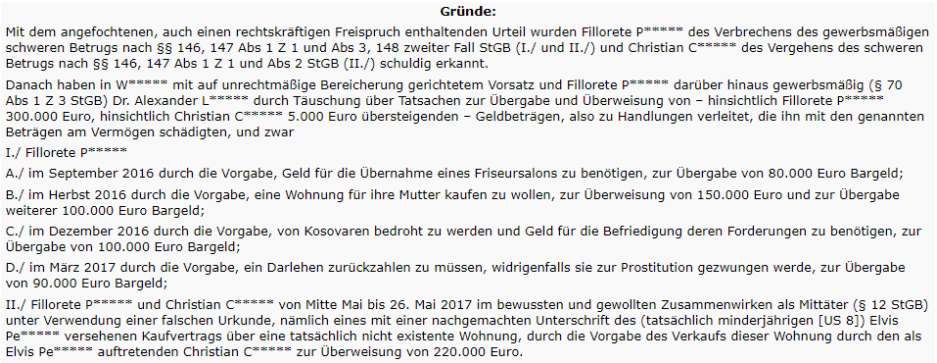
We need to teach our system to handle German language legal texts. As a first step towards that goal it makes sense to teach it to handle "normal" German. For that purpose the annotated data provided by the GermEval shared task is very useful.

However we need to also provide test data to train the system to handle legal text. For that purpose the only way to go is to build it from scratch. For this purpose the court rulings provided by the Rechtsinformationssystem des Bundes (RIS) and the European Court of Justice (CURIA) can be used. These are provided in the HTML format (as well as .doc and .pdf). The judgments provided by RIS come with sensual data (plaintiff and defender, not judges and lawyers) already removed, i.e. the job we want to do already performed. New CURIA files are also come with sensual data removed but older files where no anonymization has been performed can still be accessed.

To make these court judgments usable we have to ...

1. Turn them into plain text
2. Group the text by sentence
3. Turn them into the CoNLL file type (one word per line)
4. Undo the anonymization (in the case of RIS)
5. Tag all words with the correct named entity tags

As a point of reference here is part of court judgment as found on RIS:



**Gründe:**

Mit dem angefochtenen, auch einen rechtskräftigen Freispruch enthaltenden Urteil wurden Fillorete P***** des Verbrechens des gewerbsmäßigen schweren Betrugs nach §§ 146, 147 Abs 1 Z 1 und Abs 3, 148 zweiter Fall StGB (I./ und II./) und Christian C***** des Vergehens des schweren Betrugs nach §§ 146, 147 Abs 1 Z 1 und Abs 2 StGB (II./) schuldig erkannt.

Danach haben in W***** mit auf unrechtmäßige Bereicherung gerichtetem Vorsatz und Fillorete P***** darüber hinaus gewerbsmäßig (§ 70 Abs 1 Z 3 StGB) Dr. Alexander L***** durch Täuschung über Tatsachen zur Übergabe und Überweisung von – hinsichtlich Fillorete P***** 300.000 Euro, hinsichtlich Christian C***** 5.000 Euro übersteigenden – Geldbeträgen, also zu Handlungen verleitet, die ihn mit den genannten Beträgen am Vermögen schädigten, und zwar

I./ Fillorete P*****

A./ im September 2016 durch die Vorgabe, Geld für die Übernahme eines Friseursalons zu benötigen, zur Übergabe von 80.000 Euro Bargeld;

B./ im Herbst 2016 durch die Vorgabe, eine Wohnung für ihre Mutter kaufen zu wollen, zur Überweisung von 150.000 Euro und zur Übergabe weiterer 100.000 Euro Bargeld;

C./ im Dezember 2016 durch die Vorgabe, von Kosovaren bedroht zu werden und Geld für die Befriedigung deren Forderungen zu benötigen, zur Übergabe von 100.000 Euro Bargeld;

D./ im März 2017 durch die Vorgabe, ein Darlehen zurückzahlen zu müssen, widrigenfalls sie zur Prostitution gezwungen werde, zur Übergabe von 90.000 Euro Bargeld;

II./ Fillorete P***** und Christian C***** von Mitte Mai bis 26. Mai 2017 im bewussten und gewollten Zusammenwirken als Mittäter (§ 12 StGB) unter Verwendung einer falschen Urkunde, nämlich eines mit einer nachgemachten Unterschrift des (tatsächlich minderjährigen [US 8]) Elvis Pe***** versehenen Kaufvertrags über eine tatsächlich nicht existente Wohnung, durch die Vorgabe des Verkaufs dieser Wohnung durch den als Elvis Pe***** auftretenden Christian C***** zur Überweisung von 220.000 Euro.

Figure 4.1: Part of court judgment 12Os108/18a

### 4.1.1 Turn them into plain text

Since the RIS court judgments come in the form of html these files need to be turned into plain text. There are tools for converting a large number of html files to txt while somewhat keeping the old formatting intact. One obstacle here is that the text has line breaks added at arbitrary positions. These line breaks need to be looked at and removed if they appear in the middle of a sentence.

Furthermore certain letters, e.g. the hyphen, come encoded for the web. These also have to be taken care of ('&#45 ' → -).

### 4.1.2 Group the text by sentence

The text of court judgments at RIS and CURIA is grouped in paragraphs that on first glance seem to correspond to sentences. The difficulty comes when these blocks need to be checked to be broken up in two or more sentences. This is necessary since a large portion of these "sentences" are actually more than one sentence.

Few people realize how hard it can be to split text into sentences. This problem in natural language processing is known as *Sentence boundary disambiguation (SBD)*. It is not enough to split the text at every point ('.') since it is possible that points appear at arbitrary position, having a different purpose. In German semicolons and colons can denote the end of a sentence; it causes a stronger separation than the comma, but weaker than the point. New sentences that start with a semicolon are usually not fully grammatically correct when looked at without the preceding sentence, which might affect named entity recognition results.

German texts, and the texts of court judgments, are littered with points were they do not mark the end of a sentences. Various titles (e.g. Mag., Dr., Ing.) are used, law text is quoted (e.g. "8 Ob 122/65 = MietSlg 17.500; 3 Ob 141/57 = RZ 1957, 168") and technical terms are used (e.g. "Plan Beilage ./E").

All these things make it increasingly hard to properly detect sentence boundaries in German language texts. Just like named entity recognition sentence segmentation is a topic that would be best handled by using machine learning.

Apart from sentence splitting a few other things have to be taken care of: The reasoning in a court judgment is oftentimes written in the form of a (numbered) list. It is not uncommon that 20 lines of text are a single sentence only separated by semicolons. If the list is numbered it makes most sense to remove the numbers to restore a coherent sentence. In figure 4.1 that would be 'I./', 'B./', 'C./', 'D./' and 'II./'

### 4.1.3 Turn them into the CoNLL file type

The purpose of machine learning model for named entity recognition is to determine the type for each word in a sentence. The sentence that is to be checked can not, however, be transmitted in an arbitrary format. The simplest and most common form is to have one token per line and a blank line to denote the end of sentence. Most formats that work like this are derived from the famous format of the CoNLL-2003 shared task.

The columns (i.e. elements of a line separated by tab stops) in a text file of the CoNLL task have four columns:

- first column: a word
- second column: a part-of-speech (POS) tag
- third column: a syntactic chunk tag
- fourth column: the named entity tag, see 2.3.4

Most relevant is of course the first column that holds the word and the fourth column that holds the named entity tag. In this thesis the data format of the GermEval 2014 NER Shared Task is used which is based on the CoNLL-2003. It is similar to the CoNLL format and adds the token index within a sentence in the first column (effectively shifting everything to the right by one). Additionally a second named entity tag column is added to allow to mark words that are part of more than one named entity.

If arbitrarily formatted text is supposed to be analyzed preliminary code is needed that brings the text into the desired format for NER.

### 4.1.4 Undo the anonymization

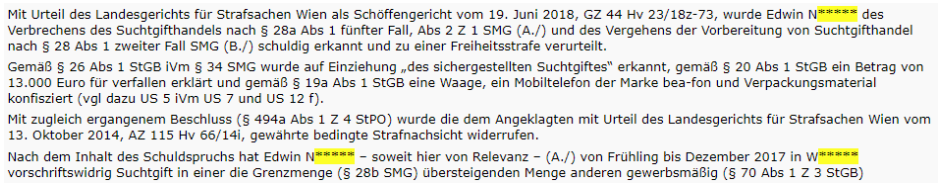
The RIS court judgments come pre-anonymized. What has to be done is to put (faux) sensual data in place of words with *-placeholders.

The process is straight-forward when done by hand:

1. Search for occurrences of '*****'

2. For each occurrence figure out what type of sensual data was removed (name, place, organization, etc.)

3. Replace the '*****' with working replacement data for that particular case: If it is a name, replace with a name, etc. Do that for all occurrences of that particular anonymized phrase (e.g. 'St*****') that appear in the dataset (so that one piece of sensual data is consistent over the dataset). For an illustration of this process see figures 4.2 and 4.3 below.

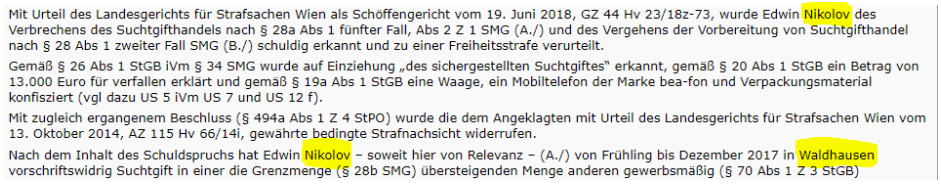

Mit Urteil des Landesgerichts für Strafsachen Wien als Schöffengericht vom 19. Juni 2018, GZ 44 Hv 23/18z-73, wurde Edwin N***** des Verbrechens des Suchtgifthandels nach § 28a Abs 1 fünfter Fall, Abs 2 Z 1 SMG (A./) und des Vergehens der Vorbereitung von Suchtgifthandel nach § 28 Abs 1 zweiter Fall SMG (B./) schuldig erkannt und zu einer Freiheitsstrafe verurteilt.

Gemäß § 26 Abs 1 StGB iVm § 34 SMG wurde auf Einziehung „des sichergestellten Suchtgiftes" erkannt, gemäß § 20 Abs 1 StGB ein Betrag von 13.000 Euro für verfallen erklärt und gemäß § 19a Abs 1 StGB eine Waage, ein Mobiltelefon der Marke bea-fon und Verpackungsmaterial konfisziert (vgl dazu US 5 iVm US 7 und US 12 f).

Mit zugleich ergangenem Beschluss (§ 494a Abs 1 Z 4 StPO) wurde die dem Angeklagten mit Urteil des Landesgerichts für Strafsachen Wien vom 13. Oktober 2014, AZ 115 Hv 66/14i, gewährte bedingte Strafnachsicht widerrufen.

Nach dem Inhalt des Schuldspruchs hat Edwin N***** – soweit hier von Relevanz – (A./) von Frühling bis Dezember 2017 in W***** vorschriftswidrig Suchtgift in einer die Grenzmenge (§ 28b SMG) übersteigenden Menge anderen gewerbsmäßig (§ 70 Abs 1 Z 3 StGB)

Figure 4.2: Before de-anonymization


Mit Urteil des Landesgerichts für Strafsachen Wien als Schöffengericht vom 19. Juni 2018, GZ 44 Hv 23/18z-73, wurde Edwin Nikolov des Verbrechens des Suchtgifthandels nach § 28a Abs 1 fünfter Fall, Abs 2 Z 1 SMG (A./) und des Vergehens der Vorbereitung von Suchtgifthandel nach § 28 Abs 1 zweiter Fall SMG (B./) schuldig erkannt und zu einer Freiheitsstrafe verurteilt.

Gemäß § 26 Abs 1 StGB iVm § 34 SMG wurde auf Einziehung „des sichergestellten Suchtgiftes" erkannt, gemäß § 20 Abs 1 StGB ein Betrag von 13.000 Euro für verfallen erklärt und gemäß § 19a Abs 1 StGB eine Waage, ein Mobiltelefon der Marke bea-fon und Verpackungsmaterial konfisziert (vgl dazu US 5 iVm US 7 und US 12 f).

Mit zugleich ergangenem Beschluss (§ 494a Abs 1 Z 4 StPO) wurde die dem Angeklagten mit Urteil des Landesgerichts für Strafsachen Wien vom 13. Oktober 2014, AZ 115 Hv 66/14i, gewährte bedingte Strafnachsicht widerrufen.

Nach dem Inhalt des Schuldspruchs hat Edwin Nikolov – soweit hier von Relevanz – (A./) von Frühling bis Dezember 2017 in Waldhausen vorschriftswidrig Suchtgift in einer die Grenzmenge (§ 28b SMG) übersteigenden Menge anderen gewerbsmäßig (§ 70 Abs 1 Z 3 StGB)

Figure 4.3: After de-anonymization

Computerized this is harder: We would actually need the finalized NER tagger to identify the type of anonymized data. Yet we only have it after the test data is procured and run through a model.

### 4.1.5 Tag all words with the correct named entity tags

To train a model a lot of annotated data is needed. What we have until here is data. For the RIS data we vaguely know the minimum of words that need to be tagged as named entities: The ones that were anonymized and replaced with stars. We do not, however, know what named entity type a word has. No information like that exists for CURIA.

The process for making the test data is effectively the same as for the whole thesis project: Looking at sentences from left to right and noting down for each word what type of word it is.

Big, existing corpora normally heavily rely on manual tagging by linguists, supported by tools. In the end it comes down to a lot of time consuming manual tagging and correcting of files.

### 4.1.6 Splitting into train, dev and test sets

Lastly we have to merge everything and the split it into three sets of test data:

- Training set: data used for training the model [train]
- Validation set: data used for tuning the model's parameters [dev]
- Test set: data used to assess the performance of the final model [test]

There are a few guidelines for doing this split:

- The split train / dev / test should always be the same across experiments
  - otherwise models are evaluated on different conditions
  - a reproducible script should exist to perform that split (making it possible to re-create the sets from a corpus of test data)
- It needs to be tested whether or not the dev and test sets shome from the same distribution

## 4.2 Neural Network Architecture

In this section, we describe the components (layers) of our neural network architecture. We introduce the neural layers in our neural network one-by-one from bottom to top.[XUEZ]

### 4.2.1 CNN for Character-level Representation

Previous studies (Santos and Zadrozny, 2014; Chiu and Nichols, 2015) have shown that CNN is an effective approach to extract morphological information (like the prefix or suffix of a word) from characters of words and encode it into neural representations. Figure 4.4 shows the CNN we use to extract character-level representation of a given word. The CNN is similar to the one in Chiu and Nichols (2015), except that we use only character embeddings as the inputs to CNN, without character type features. A dropout layer (Srivastava et al., 2014) is applied before character embeddings are input to CNN.[XUEZ]

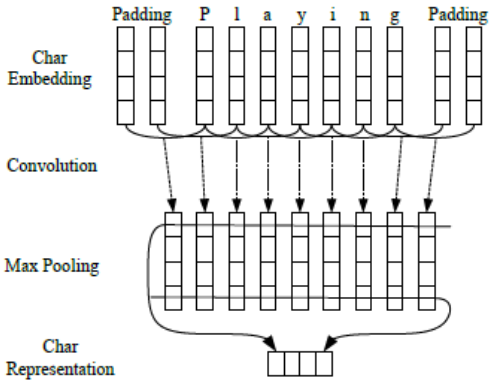



Figure 4.4: The convolution neural network for extracting character-level representations of words. Dashed arrows indicate a dropout layer applied before character embeddings are input to CNN.[XUEZ]

### 4.2.2 Bi-directional LSTM

#### LSTM Unit

Since for named entity recognition context is very important we have to use a recurrent network (see section 3.4). A feedforward network simply would not do: Information about previous (and subsequent) words is very important for a task like this. RNNs are plagued by the vanishing/exploding problems problems, though.

Using an LSTM as the major building block of our system is the logical step (see section 3.5). LSTMs are designed to cope with the vanishing/exploding

problems problems and have been shown to perform well in natural language processing tasks [57, 16].

#### BLSTM

For a sequence labelling task such as named entity recognition it is useful to both have the past (left) and future (right) contexts of a word that is looked at. A bidirectional LSTM, as described in section 3.5.2 is perfect for that. Two separate hidden states are used. Each sequence is fed forwards to the first hidden state and also backwards to the second hidden state. Finally the two hidden states are concatenated to create the final output. BLSTMs have been successfully used for natural language processing tasks in general and named entity recognition in particular [34, 16, 26, 11, 35].

### 4.2.3 CRF

Another measure that can be considered to improve the accuracy of the named entity recognition model is the correlation between labels near each other. For example, it is not possible that a word labelled with I-ORG comes directly after a word that is labelled with B-PER. Therefore we use a conditional random field layer (see section 2.5) to jointly decode the best chain of labels rather than decoding each label independently.

For CRF training, we use the maximum conditional likelihood estimation. For a sequence CRF model (only interactions between two successive labels are considered), training and decoding can be solved efficiently by adopting the Viterbi algorithm.[XUEZ]

### 4.2.4 BLSTM-CNNs-CRF

Finally, we construct our neural network model by feeding the output vectors of BLSTM into a CRF layer. For each word, the character-level representation is computed by the CNN with character embeddings as inputs. Then the character-level representation vector is concatenated with the word embedding vector to feed into the BLSTM network. Finally, the output vectors of BLSTM are fed to the CRF layer to jointly decode the best label se-

quence. Dropout layers are applied on both the input and output vectors of BLSTM. Experimental results show that using dropout significantly improve the performance of our model.[XUEZ]

# Chapter 5

# Implementation

## 5.1 Used software and libraries

### 5.1.1 Python

For machine learning Python is the most popular programming language. It is reportedly used by 57% of programmers that work with machine learning [53]. The reason for this is that there are very well made libraries for machine learning in Python. Ones that were ported only to other languages or exist exclusively for Python. All the tools used in this thesis are built on top of Python.

### 5.1.2 TensorFlow

Tensorflow is a library for dataflow programming. Google's AI team developed TensorFlow and later decided to make it open source. They released TensorFlow 1.0 in 2015, and as of the time of this writing, the current version is 1.12.0. It's provided under the Apache 2.0 open source license, which means you're free to use it, modify it, and distribute your modifications.

Google is investing a lot into TensorFlow since it wants TensorFlow to be

the lingua franca of machine learning researchers and developers [1]. It is used for a wide variety of tasks in Google products such as Google Maps and Gmail.

Tensorflow supports being executed on GPUs as well as CPUs. Additionally Tensorflow applications can be run on the Google Cloud Platform (GCP). GPU and cloud are the preferred variants since CPU-only is very slow in comparison.

### 5.1.3 Keras


Keras is an open source neural network library that is natively written in Python. It is a wrapper that allows you to use the TensorFlow, the Theano or the Microsoft Cognitive Toolkit back-end. Keras offers a higher-level set of abstractions for the computational back-end that is used. Keras offers a big degree of simplicity thanks to its small API and intuitive set of functions. François Chollet implemented Keras with the idea to enable faster experimentation. Because of that many TensorFlow developers prefer to code their neural networks using Keras. Chollet released Keras under the MIT License. Google later incorporated its interface into the API of TensorFlow.

## 5.2 Procurement of training data

In the previous chapter a theoretical overview of what has to be done to create training data out of court judgments available online at RIS and CURIA has been given (see 4.1). As mentioned before they are offered for download as html, pdf and word, of which html is the most convenient.

In this section the process is explained in detail and some code is shown. All in all 300 court judgments from RIS and CURIA were annotated and turned into test data. Some statistical information about the first batch of 100 court judgments is given for each step and after that an overview of the full 300 judgments.

### 5.2.1 Turn them into plain text & Group the text by sentence

For the first two steps we read all downloaded HTML files and make use of Python's *HTMLParser* to extract the sentences. The parts where stars are used to anonymize names are transferred as-is and not yet taken care of. Text chunks are merged when they are part of the same sentence. If text chunks are merged numbering at the start of a text chunk are dropped by means of a number of regexes:

```
text = re.sub(r"^[a-z]\)\s+", "", text)
text = re.sub(r"^\d\.(\d\.)?\s?", "", text)
text = re.sub(r"^[A-Z]\./\s?", "", text)
text = re.sub(r"^[IVX]{1,3}\./\s?", "", text)
text = re.sub(r"^[a-z]\./\s?", "", text)
text = re.sub(r"^\d\./\s?", "", text)
```

This removes different types of numbering that can occur alone or in combination. Since it is not clear which if any actually are applicable for a given line of text trailing whitespace is always removed (with the \s?).

Merging sentences is easy enough. Splitting them is harder as detailed in subsection 4.1.2.

Natural Language Toolkit (NLTK) is a widely used standard library for all things natural language processing. One approach is to use their bundled sentence tokenizer that comes with a pre-trained model for handling German text that is based on the *PunktSentenceTokenizer*.

This does, however, yield, rather bad results for my example paragraph:

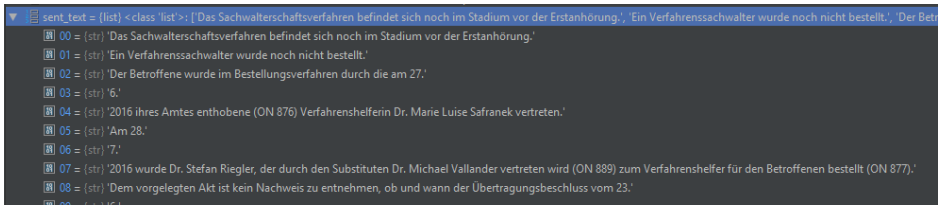



Figure 5.1: Mediocre results of sentence splitting

What was used, finally, was a simple custom implementation that checks

the context of a point and only splits when a set of conditions are fulfilled. The results are much better than the ones shown in figure 5.1 but could be improved by training a model just for sentence splitting. For the purpose of creating test data the custom implementation is, however, sufficient.

To get an overview on how effective this process is, here is some information about the first 100 court judgments from RIS that were worked at:

- Number of files: 100
- total number of the redaction string '******': 1134
- Average redactions: 11.34 per file
- Total number of sentences: 1744
- Average sentences: 17.44 per file
- Maximum number of sentences: 48

### 5.2.2 Turn them into the right format

When the sentences are extracted the next step is easy enough. For each sentence:

1. Split it into words by whitespace
2. For each each word:
   (a) Check if there are both word characters and special characters
   (b) If so: split in further parts (e.g. 'kitchen:' becomes 'kitchen' and ':')
3. For each each word: Write it to the output file
4. Prefix with an index that counts up within a sentence
5. Put 'O' and 'O' after each word as the named entity tags

As for statistics:

- Total number of sentences: 1744
- Total number of tokens: 141,898
- Average number of tokens per file: 1419
- Average number of tokens per sentence: 81

### 5.2.3 Undo the anonymization

This step is only needed for the RIS court judgments since only these come pre-anonymized. For this step a nifty little Python library, *Faker*, was used. It is simply a library that creates fake data of many sorts for the user. In *Pro Python Best Practices: Debugging, Testing and Maintenance* it is mentioned as best practice for creating random data for testing [44]. *Faker* has functions to create what we need, mostly names for people, locations and organizations. It comes localized to most major languages, among them German. Similar libraries for other programming languages exist, such as PHP Faker, Perl Faker and Ruby Faker.

Exchanging an anonymized name with a proper, unique German surname is as easy as follows:

```
fakeLastName = Faker('de_DE').last_name()
line = line.replace(lastName, fakeLastName)
```

Every time the function is called a new name is generated. Since it potentially has to be used more than once generated names are saved in a dictionary.

Faker offers methods for all the relevant tasks:

- person →first_name_male, first_name_female, last_name, etc.
- location →city_name (also city), street_address, etc.
- organization →company
- other →nothing for this but that was to be expected

A problem that arises here is that we do not know what kind of data (i.e. person, location, organization, other) we have to substitute. This is closely related to the next step. Here we do not care about all words in our test data, just the ones that have '*****' in them.

Luckily enough we are building a model for named entity recognition. A viable option is to use this model in its current state (where it is only trained on general-purpose German text) and analyze the data with that. What has to be kept in mind is that more names will be found that were redacted initially. Names of court staff (e.g. judges) and other non-redacted people will be identified as personal names additionally.

As for statistics:

- Initial number of redaction strings '******': 1134
- Total Number of named entity tags assigned: 5792
- Redaction strings without a named entity tag: 350 (i.e. false negatives)
- Detailed numbers of named entity tags assigned in the table below

| | **normal** | **deriv** | **part** |
|---|---|---|---|
| **B-PER** | 1394 | | |
| **I-PER** | 1117 | | |
| **B-LOC** | 562 | 70 | |
| **I-LOC** | 57 | | 4 |
| **B-ORG** | 611 | | |
| **I-ORG** | 869 | | |
| **B-OTH** | 431 | 36 | 1 |
| **I-OTH** | 634 | | |
| | **5792** | **106** | **5** |

68 per cent of the redaction strings were recognized. This is an okay number since for many of these even for a human it is hard to figure out what kind of information has been redacted.

The next step is to use *faker* to go through the files and look at all words with '*****' and a named entity tag assigned and replace it with fake data. Then check the result, fix the input files and run the faker script again.

| BEFORE | | | |
|---|---|---|---|
| 65 | Mag | O | O |
| 66 | . | O | O |
| 67 | Ing | O | O |
| 68 | . | O | O |
| 69 | Walter | B-PER | O |
| 70 | S***** | I-PER | O |
| 71 | , | O | O |
| 72 | vertreten | O | O |
| 73 | durch | O | O |
| 74 | Dr | O | O |
| 75 | . | O | O |
| 76 | Herbert | B-PER | O |
| 77 | Gartner | I-PER | O |

| AFTER | | | |
|---|---|---|---|
| Mag | O | O | O |
| 66 | . | O | O |
| 67 | Ing | O | O |
| 68 | . | O | O |
| 69 | Walter | B-PER | O |
| 70 | Hein | I-PER | O |
| 71 | , | O | O |
| 72 | vertreten | O | O |
| 73 | durch | O | O |
| 74 | Dr | O | O |
| 75 | . | O | O |
| 76 | Herbert | B-PER | O |
| 77 | Gartner | I-PER | O |

After a few re-runs of this there are 191 occurrences of '*****' left in the text. This is lower than the 357 false positives reported above since within a file all occurrences of the same redaction string were replaced even if they did not have a named entity tag assigned (e.g. all tokens that are 'M*****' were replaced by 'Mayer'). These last 182 '*****' will not be resolved until the last step.

### 5.2.4 Tag all words with the correct named entity tags and merge the files

In the third step we wrote the testdata in the correct format and replaced most of the redacted words. What remains is the most time consuming task: Looking through all the files and manually making sure that everything is correct. Since this data is going to be used to train the model is has to be absolutely correct and this step cannot be skipped or fully automated.

A common error that the model at this current state produces is that it recognizes titles (e.g. Dr., Mag.) as part of a person's name, which it should not. Another common thing was that the second retraction in 'der klagenden Partei F***** GmbH, *****, vertreten durch' would not be recognized as a place name. Which makes sense since there is no real indication there that this is a place. The model has to learn this detail from correctly annotated training data.

After this has been done the 100 files can be merged and split into the three sets that are needed for training: *train*, *dev* and *test*.

### 5.2.5 Conclusion

For this thesis a total of 300 court judgment were annotated. Table XXX shows the final corpora statistics. Worth noting is that the average CURIA court judgment is much larger than the average Austrian. Despite 67% of the court judgments being Austrian ones the CURIA judgments have twice as many total tokens.

## 5.3 Neural Network Architecture

### 5.3.1 Structure of the project

All of the source code used is bundled together in a project called *LegalTexts*. On the top level there is the file **Train_NER.py** that is compiling a model according to a set of parameters. *ProcessSingleFile.py* is meant to be used from the command line. It takes a file, parses it and creates a second, NER labelled file as its output.

**preparecustomdataset** contains the utility scripts to generate the test data. It is split up in five steps and there are both parsers for RIS and for Curia.

**data** contains the test data that is used for creating models. That is the custom created test data from the previous section as well as the GermEval data for creating the very first iteration of the model.

**wordembeddings** contains the used word embeddings package from Reimers et. al.

The file can be publicly downloaded from: `https://public.ukp.informatik.tu-darmstadt.de/reimers/2014_german_embeddings/2014_tudarmstadt_german_50mincount.vocab.gz`

**pkl** contains pickle files. When *Train_NER.py* is run the word embed-

dings are parsed and adapted for the format of the test data. By saving the result of this process in a pickle file time is saved when re-running the model generation.

**models** contains the models generated by running *Train_ NER.py*. During the model generation process several models are potentially written. For each epoch it is checked whether the model has been improved and if so it is exported to the folder. After a given number (5) of epochs without an improvement the process is stopped.

**util** contains various utility functions, among them parsers for RIS and CURIA court judgments and tools for evaluating the F1 score of a model.

**neuralnets** contains the most vital parts of the system: Firstly the implementation of the bi-directional LSTM, the core building block of our neural network. Secondly Philipp Gross' implementation of ChainCRF, that is used for the final layer of our neural network. CRFs are a very well researched area of machine learning and Gross' implementation is widely used and tested.

## 5.4 Network Training

In this section, we provide details about training the neural network. We implement the neural network using Keras and Tensorflow. The computations for a single model are run on a NVIDIA GeForce GTX 1060 GPU with 6 GB of RAM. Using the settings discussed in this section it takes about five hours to train the model.

### 5.4.1 Parameter Initialization

**Word Embeddings.** We use the publicly available embeddings by Reimers et al [41], trained on 116 Million German sentences. It is advised to used word embeddings trained on a huge corpus that is unrelated to the test data and Reimers fits this descriptions perfectly.

**Character Embeddings.** Character embeddings are initialized with uniform samples.

**Weight Matrices and Bias Vectors.** Matrix parameters are randomly

initialized with uniform samples.

### 5.4.2 Optimization Algorithm

For optimization the Adam algorithim by Kingma and Ba was used [27].

**Early Stopping.** We use early stopping based on the improvement of the model. If the model is not improved in 5 epochs the process will stop. The model usually reached its final states between epochs 15 and 25.

### 5.4.3 Tuning Hyper-Parameters

Table 1 summarizes the chosen hyper-parameters.

# Chapter 6

# Evaluation

## 6.1 Evaluation Metrics

The performance of named entity recognition systems is usually determined by *Precision*, *Recall* and *F1 score*. Since NER is a multiclass classification, these performance criteria are calculated for each class individually and then averaged to obtain an overall rating of the program.

These measures can be calculated by making use of the numbers of true positives (TP), false positives (FP), false negatives (FN) and true negatives (TN). Hereby the number of properly detected named entities is referred to as TP. The annotations on entities that actually are not entities are called FP. Not detected named entities are being referred to as FN. Lastly TNs are tokens that have been correctly classified as non-entities.

| | | **Actual** | |
|---|---|---|---|
| | | Positive | Negative |
| **Predicted** | Positive | **True Positive** | **False Positive** |
| | Negative | **False Negative** | **True Negative** |

Table 6.1: False positives and false negatives

Precision (P) is the proportion of properly mapped entities divided by the number of found entities in a class:

$$P = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{6.1}$$

Recall (R) is the proportion of properly mapped (and classified) entities divided by the number of all entities in a class:

$$R = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{6.2}$$

F1 score is the harmonic average of the precision and recall:

$$F1 = \frac{2 * P * R}{P + R} \tag{6.3}$$

## 6.2 Test results

100 RIS judgments –> Dev-Score: 0.8532 Test-Score 0.7794 Test-Data: Prec: 0.804, Rec: 0.755, F1: 0.7788

200 RIS judgments –> Dev-Score: 0.8981 Test-Score 0.9080 Test-Data: Prec: 0.922, Rec: 0.905, F1: 0.9131

# Bibliography


[1] Sam Abrahams, Danijar Hafner, Erik Erwitt, and Ariel Scarpinelli. *TensorFlow for Machine Intelligence: A Hands-on Introduction to Learning Algorithms*. Bleeding Edge Press, 2016.

[2] John R Anderson and Joachim Funke. *Kognitive Psychologie*, volume 2. Spektrum Akademischer Verlag Heidelberg, 2001.

[3] Frederico AC Azevedo, Ludmila RB Carvalho, Lea T Grinberg, José Marcelo Farfel, Renata EL Ferretti, Renata EP Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513(5):532–541, 2009.

[4] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[5] Darina Benikova, Chris Biemann, Max Kisselew, and Sebastian Pado. Germeval 2014 named entity recognition shared task: companion paper. 2014.

[6] Daniel M Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. Nymble: a high-performance learning name-finder. *arXiv preprint cmp-lg/9803003*, 1998.

[7] Andrew Borthwick, John Sterling, Eugene Agichtein, and Ralph Grishman. Description of the mene named entity system as used in muc-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7), Fairfax, Virginia, April 29-May 1, 1998*, 1998.

[8] Nikhil Buduma and Nicholas Locascio. *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms*. " O'Reilly Media, Inc.", 2017.

[9] Jinmiao Chen and Narendra S Chaudhari. Protein secondary structure prediction with bidirectional lstm networks. In *International Joint Conference on Neural Networks: Post-Conference Workshop on Computational Intelligence Approaches for the Analysis of Bio-data (CIBIO)(August 2005)*, 2005.

[10] Laura Chiticariu, Yunyao Li, and Frederick R Reiss. Rule-based information extraction is dead! long live rule-based information extraction systems! In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 827–832, 2013.

[11] Jason PC Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *arXiv preprint arXiv:1511.08308*, 2015.

[12] Wolfgang Ertel. Grundkurs künstliche intelligenz. *Eine praxisorientierte Einführung*, 3, 2009.

[13] Manaal Faruqui, Sebastian Padó, and Maschinelle Sprachverarbeitung. Training and evaluating a german named entity recognizer with semantic generalization. In *KONVENS*, pages 129–133, 2010.

[14] Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named entity recognition through classifier combination. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 168–171. Association for Computational Linguistics, 2003.

[15] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[16] Shalini Ghosh, Oriol Vinyals, Brian Strope, Scott Roy, Tom Dean, and Larry Heck. Contextual lstm (clstm) models for large scale nlp tasks. *arXiv preprint arXiv:1602.06291*, 2016.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[18] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005.

[19] Zellig Harris. Distributional structure. *Word*, 10(23):146–162, 1954.

[20] Donald O Hebb. The organization of behavior. a neuropsychological theory. 1949.

[21] Geoffrey E Hinton. Connectionist learning procedures. In *Machine Learning, Volume III*, pages 555–610. Elsevier, 1990.

[22] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[23] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

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

[25] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[26] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] David Kriesel. A brief introduction on neural networks. 2007.

[29] Rudolf Kruse, Christian Borgelt, Christian Braune, Frank Klawonn, Christian Moewes, and Matthias Steinbrecher. Computational intelligence-eine methodische einführung in künstliche neuronale netze, evolutionäre algorithmen, fuzzy-systeme und bayes-netze. 1. *Auflage. Wiesbaden: Vieweg+ Teubner*, 2011.

[30] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[32] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[33] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Unsupervised learning of hierarchical representations with convolutional deep belief networks. *Communications of the ACM*, 54(10):95–103, 2011.

[34] Nut Limsopatham and Nigel Henry Collier. Bidirectional lstm for named entity recognition in twitter messages. 2016.

[35] Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*, 2016.

[36] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[37] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[38] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[39] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[40] George Psyllides. Top court bans publication of judgments over gdpr. `https://cyprus-mail.com/2018/06/07/top-court-bans-publication-of-judgments-over-gdpr`, 2018. Accessed: 2018-12-10.

[41] Nils Reimers, Judith Eckle-Kohler, Carsten Schnober, Jungi Kim, and Iryna Gurevych. Germeval-2014: Nested named entity recognition with neural networks. 2014.

[42] Günter Daniel Rey and Karl F Wender. *Neuronale Netze: Eine Einführung in die Grundlagen, Anwendungen und Datenauswertung*. Huber, 2010.

[43] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[44] Kristian Rother. Organizing test data. In *Pro Python Best Practices*, pages 117–128. Springer, 2017.

[45] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[46] Erik F Sang and Jorn Veenstra. Representing text chunks. In *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, pages 173–179. Association for Computational Linguistics, 1999.

[47] R Sathya and Annamma Abraham. Comparison of supervised and unsupervised learning algorithms for pattern classification. *International Journal of Advanced Research in Artificial Intelligence*, 2(2):34–38, 2013.

[48] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[49] Herbert Süße and Erik Rodner. *Bildverarbeitung und Objekterkennung*. Springer, 2014.

[50] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

[51] Trias Thireou and Martin Reczko. Bidirectional long short-term memory networks for predicting the subcellular localization of eukaryotic proteins. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 4(3):441–446, 2007.

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

[53] C Voskoglou. What is the best programming language for machine learning? *Retrieved April*, 11:2018, 2017.

[54] Paul J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

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

[56] Andreas Zell. *Simulation neuronaler netze*, volume 1. Addison-Wesley Bonn, 1994.

[57] Donghuo Zeng, Chengjie Sun, Lei Lin, and Bingquan Liu. Lstm-crf for drug-named entity recognition. *Entropy*, 19(6):283, 2017.

# Appendix A

# Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.