FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Improving Multi Word Term Detection in the Patent Domain with Deep Learning

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Business Informatics

eingereicht von

BSc. Tobias Fink
Matrikelnummer 00925109

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof.Dr. Allan Hanbury
Mitwirkung: MSC Linda Andersson
Prof.Dr. Akiko Aizawa

Wien, 4. Oktober 2018

Tobias Fink

Allan Hanbury

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Improving Multi Word Term Detection in the Patent Domain with Deep Learning

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

BSc. Tobias Fink
Registration Number 00925109

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof.Dr. Allan Hanbury
Assistance: MSC Linda Andersson
Prof.Dr. Akiko Aizawa

Vienna, 4th October, 2018

Tobias Fink

Allan Hanbury

Erklärung zur Verfassung der Arbeit

BSc. Tobias Fink
Address

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 4. Oktober 2018

Tobias Fink

Kurzfassung

Im Bereich der Patentsuche spielen technische Begriffe eine wichtige Rolle, wenn es darum geht herauszufinden, welche Patentdokumente einem bestimmten Patentdokument am ähnlichsten sind. Im Englischen bestehen technische Begriffe zumeist aus mehreren Wörtern und formen weiters Substanivgruppen (noun phrases). Letzteres wird von Methoden, die solche Mehrwortbegriffe (multi word terms) erkennen sollen, genutzt. Da aber Patenttexte sich oftmals durch eine Schreibstil kennzeichnen, der von anderen englischen Texten abweicht und Sätze sehr lang werden können, sind übliche Methoden zur Erkennung von linguistischen Informationen weniger effektiv. Weiters werden manche Mehrwortbegriffe eher selten in Patenten verwendet, was Methoden, die die Erkennung von Substantivgruppen sowie Information über die Vorkommenshäufigkeit der Begriffe benötigen, nicht unproblematisch macht. In dieser Arbeit präsentieren wir eine Methode zur Erkennung von Mehrwortbegriffen jeglicher Vorkommenshäufigkeit, die nicht die vorherige Erkennung von Substantivgruppen benötigt. Mithilfe von überwachtem maschinellem Lernen und einem künstlichen neuronalen Netz, das durch Methoden der Eigennamenerkennung (named entity recognition) und Schlüsselphrasenerkennung inspiriert wurde, trainieren wir Modelle auf Sätzen von 22 Patent, deren Mehrwortbegriffe beschriftet wurden, bestehend aus Wort-Tokens und Buchstaben-Tokens. Durch Verwendung von 'word embeddings', die mit dem CLEF-IP Patentdatensatz erstellt wurden, erreicht unser bestes Modell eine höhere Leistung als unser bestes, auf linguistischen Informationen basierendes Richtlinienmodell, im Bezug auf Genauigkeit (von 0.70 auf 0.85), Trefferquote (von 0.74 auf 0.84) und F-Maß (von 0.72 auf 0.84).

Abstract

In patent document information retrieval, the technical terms that are used in a particular patent document are an important factor in determining what the most relevant related documents are. In English, technical terms often consist of multiple words. Further, the fact that they are mostly noun phrases (NP) is utilized by methods detecting such multi word terms (MWT). However, due to the special nature of the patent domain, such as a special writing style and high maximum sentence length, common methods for extracting linguistic information are less effective. Further, some MWTs can occur very infrequently in patents, which makes the use of methods relying on NP extraction and frequency based information problematic. In this thesis we present a method for detecting even rare MWTs in patent texts that does not require the prior detection of NPs. Using supervised machine learning and an artificial neural network inspired by named entity extraction and keyphrase detection methods, we train models on sentences with annotated MWTs from 22 patents consisting of word tokens and character tokens. With the help of word embeddings trained on the CLEF-IP patent dataset, our best model outperforms our best linguistic baseline with regards to precision (from 0.70 to 0.85), recall (from 0.74 to 0.84) and F1 score (from 0.72 to 0.84).

Contents

CHAPTER 1

Introduction

The purpose of patent document information retrieval (IR) is to discover what patent documents or paragraphs (passages) of patents could be considered similar to other patents, which can be used to discover prior art. This is of great value for research and development and needs to be done by patent offices to ensure the novelty of new applications. Patent texts have a few special characteristics that need to be considered in patent IR. For one, patent texts cover a large number of different domains, such as computing, chemistry, engineering, and many others. Further, use cases in patent IR often center around an innovation and how already existing patents relate to this innovation. Since patents document time-limited monopolies granted by a state, these use cases usually also involve legal and administrative considerations. This also affects the language of patents, making them similar to other legal documents such as contracts, as well as the content in the form of patent claims, which describe exactly what parts of an invention are protected. Most important for the application of new patents, the inventions protected by a patent must be novel, which means they have not been described or used before anywhere else. For patent IR this often means, all relevant patents that can be found need to be found. To know which patents are relevant requires domain specific knowledge and legal expertise. For instance, if the author of a patent misses a relevant patent, this relevant patent could be found by a competitor or a patent examiner during their search and used to invalidate the patent or deny the patent application. The Prior Art search conducted by the patent examiner is an iterative process, during which multiple Boolean search queries are entered, the returned patents filtered for potentially relevant documents and a search report created. According to Andersson et al. [ALP+16], between 100 and 200 retrieved documents are carefully examined during such a search. [LHo13]

One of the characteristics that makes patent texts different from other domains and that can be exploited in patent IR is the extensive use of domain terminology and

technical language. In patent IR, technical terms, such as “blood cell count”, are often key to whether a document is considered relevant or not relevant for a particular query. Nonetheless, patents do not have a token coverage that is significantly different from general English in most domains [VDOK10], which means that most of the relevant technical terms introduced in patents use the same words as are used in general English and thus, are in fact compositions of commonly used words - **Multi Word Terms** (MWT). For example, “nickel plating solution bath” is a MWT that describes a concept different from the individual words ’nickel’, ’plating’, ’solution’ and ’bath’, but all of these words are used in general English. Whether a phrase should be considered a MWT depends on the termhoodness of that phrase, which is a measure that represents to which degree a phrase should be considered a term. Depending on the method that is used, the termhoodness of a phrase can depend on many different things. One thing MWTs in patents and in scientific papers have in common, is that they are usually considered to be **noun phrases** (NP). For example, according to the description of the SemEval 2017 Task 10 dataset [ADR+17], which is a dataset of scientific papers, 93% of keyphrases are NPs.

Finding the NPs of a text is an important step in many established MWT extraction algorithms like the C-value method [FAM00] and, sometimes, even the MWT ground truth is based on the results of a NP detection method [AVP14]. However, requiring NP detection for the detection of MWTs leads to a few problems. First, NPs can be nested and therefore the longest NP may not match the desired MWT boundaries. For instance, in the nested NP ’(NP (NP The method) (PP of) (NP infrared radiation drying))’ we are only interested in the technical term *infrared radiation drying*, which is a method to dry food (PP stands for preposition). This problem usually occurs when patent authors add subjective adjectives like ’good’ (“good heat resistance”) or ’high’ (“high crosslinking rate”) to MWTs. Especially for words like ’high’ it can be very difficult to determine — even for a non-expert human — whether it is essential to the MWT or redundant. Furthermore, because of longer sentences, a more complex sentence structure and writing style different from other English texts, Part-of-Speech (PoS) tagging for patent texts is less reliable [ALP+16]. This problem is amplified by the fact that many PoS-taggers are trained on general English texts, such as newspaper texts, and not on technical texts.

The second issue with methods like the C-value method is their reliance on frequency based measures to determine the termhoodness of a phrase. Especially in the patent domain, NPs that occur very infrequently could often still be considered MWTs. New MWTs are frequently introduced and thus only occur in a single patent. Authors oftentimes are somewhat inconsistent in what words are used to form a MWT and sometimes omit parts of a MWT, but still describe the same concept. For this reason, the frequency of a NP should not be the primary factor to determine its termhoodness.

To avoid being dependent on PoS tagged patent text and be able to detect even rare

MWTs (i.e. MWTs that occur infrequently in our dataset), we propose a method inspired by state-of-the-art **named entity recognition** (NER) and keyphrase detection methods for creating models capable of detecting the boundaries of MWTs in patent texts without the need of detecting NPs first. These MWT models detect MWTs in patent sentences by processing the sentence as a sequence of word and character tokens. To create these models we use supervised machine learning with **artificial neural networks** (ANN), also known as deep learning, which requires an annotated dataset for both training and evaluation. The network consists of a series of components, each of which designed to handle a specific task, such as representing words based on their distributional semantics or character composition or their context in the sentence. For the process of creating these models, the word representation component of our ANN architecture requires a dictionary of word representations (interchangeably called word embeddings and word vectors) that have been trained with an unsupervised method on a larger corpus. For the task of supervised machine learning, we create an annotated dataset consisting of 22 patent texts with word level MWT boundary annotations as follows: we read through all of the 22 patents and manually mark any sequence of words that fulfill our termhoodness criteria as MWTs. MWTs discovered this way are considered the ground truth MWTs for all patents. This is different to methodologies described in other papers, such as [AVP14], where the ground truth is based on candidate phrases, which are extracted via NP detection.

As our contribution to the state-of-the-art we analyze the impact of a number of model creation method variations on the performance of the resulting MWT models. We investigate

- the impact of regularization techniques that can be applied to most deep learning models,
- the impact of a custom character convolutional neural network component,
- the impact of hyperparameter changes on different parts of the network,
- the impact of additional PoS tag features extracted by the Python natural language processing library spaCy[1],
- the difference in performance between general purpose word embeddings, such as those created with a dataset of Wikipedia articles, and specialized patent word embeddings created with the CLEF-IP patent document collection[2],
- to what degree different training data set sizes affect the model performance,
- to what extent rare MWTs, i.e. MWTs that occur infrequently in our dataset, can be detect,

[1] `https://spacy.io/`

[2] http://www.ifs.tuwien.ac.at/~clef-ip/download-central.shtml

- common prediction errors of our models and their possible causes.

To be able to judge how well our method performs, we compare it to a baseline method, based on a certain set of performance metrics. The baseline method extracts MWTs from a sentence with a pattern matching approach that detects certain patterns of PoS tags in a sentence. To gain a better understanding of how the MWT models created by our method behave, we analyze our best models by investigating common prediction errors, i.e. which MWTs are missed and which phrases are falsely predicted to be MWTs, and what the cause for these errors could be.

Chapter 2 describes the fundamentals of deep learning as well as related work to this thesis, such as papers in the patent domain or recent developments in NER. Chapter 3 describes our approach, model architecture and other parts of the methodology in detail. Chapter 4 presents the evaluation results as mentioned above. The final chapter, Chapter 5, summarizes our findings and concludes the thesis.

CHAPTER 2

Fundamentals & Related Work

In this chapter we present basic definitions for MWTs and related work for MWT detection in patents. Further, we introduce machine learning and deep learning concepts that are used by our method. These concepts include the basics about working with neural networks and a few advanced architectures and regularization techniques. Most of the following explanations about deep learning and machine learning are summarizations of the book "Deep Learning" [GBC16]. To tie everything together, we present a few papers that use methods making use of the presented machine learning and deep learning techniques to detect and classify named entities, which is a task that is very similar to detecting MWTs. In Chapter 3 we will use these methods to detect MWTs.

2.1 Multiword Terms in the Patent Domain

"Definition 2.2 (Term). *A terminology is a specialised lexicon corresponding to the set of words that characterise a specialised language of a domain. A term is the basic lexical unit of a terminology.*"
"Definition 2.3 (Multiword term). *A multiword term is a specialised lexical unit composed of two or more lexemes, and whose properties cannot be directly inferred by a non-expert from its parts because they depend on the specialised domain and on the concept it describes.*"
–Ramisch, Carlos. "Multiword Expressions Acquisition." Page 33 [Ram14]

The definitions given by Ramish for the multiword term "Multiword Term" (MWT) suggest that MWTs consist of two or more words, describe concepts of a specific domain and that it is necessary to have access to domain specific expert knowledge to understand the properties of the concepts that a MWT describes. However, only a minimum of expert knowledge might be required to successfully identify MWTs in patents, since MWTs in a technical domain, such as the patent domain, often follow certain linguistic rules.

Justeson and Katz [JK95] suggest that technical MWTs primarily are NPs that consist of nouns (NOUN), adjectives (ADJ) and, less often, prepositions (part of ADP in the universal PoS tags), which means they could be extracted using a linguistic pattern (also called linguistic filter) that selects phrases based on their PoS tags. The C-value method [FAM00] extracts MWT candidates this way and additionally calculates a frequency dependent score for these phrases to rank them based on their termhoodness, i.e. the degree to which a phrase should be considered a term. The longer and more frequent a phrase is, the higher it is ranked, discounting the frequency of the phrase being nested in even longer phrases.

In the patent domain, sentences are often densely filled with technical terms and one can expect new terms to be introduced frequently. These new terms are often MWTs that consist of words also used in general English, but when put together describe a new concept. Furthermore, new MWTs often do not describe a new concept but instead paraphrase a concept already mentioned previously, i.e. describe a concept more abstractly to widen the scope of a claim [NKT+09]. It is thus expected that a human annotator who is capable of correctly identifying nouns and adjectives in a patent text and has an understanding of the meaning of the individual words can use this information to correctly identify a large number of MWTs. Even if lacking the required domain specific expert knowledge to understand the meaning of each MWT completely, it should still be possible to identify them and conjecture enough of their meaning to be able to detect the correct boundaries. While it is possible to use PoS tag information to automatically detect MWTs in the patent domain, Andersson et al. [ALP+16] showed that PoS taggers provide worse results in the patent domain on average when compared to other domains and thus should not be relied on primarily. Furthermore, not all adjectives are equally good for MWT detection, as there are some adjectives that merely describe a subjective quality of a term and should not be considered part of the term itself. For example, while the phrase “good productivity” would be tagged as <ADJ NOUN>, one should not consider it a term that describes a concept very specific to a domain. In the case of “high crosslinking rate” with the tags <ADJ NOUN NOUN>, it might be unclear whether the preferred MWT is “high crosslinking rate”, just “crosslinking rate” or both are different MWTs, although for a human it is possible to determine the subjectiveness of the word “high” from the context. If “crosslinking rate” appears about as often as “high crosslinking rate”, the C-Value method would rank the longer version higher, although it might be obvious from the context that <high> is only subjective. If either version would occur only a few times in the corpus (for example, as a new phrase or paraphrase), both versions would receive a low score, and possibly be discarded. However, in the patent domain, a low-frequency phrase might still be of importance and should be detected. Instead of relying on a single metric, Andersson et al. combine a number of features by training a random forest classifier on 4400 preselected NPs of which 2700 are labeled MWTs and 1700 are labeled not MWTs and report an F1 score of 0.85 and an accuracy of 0.79. The features that were used are phrase length, document frequency of a phrase

and Pointwise Mutual Information (PMI) of a phrase. The PMI for a phrase is defined as

$$pmi = \log_2 \frac{c(w_1^n)}{\mathbb{E}(w_1^n)}, \tag{2.1}$$

where $c(w_1^n)$ is the observed frequency of the phrase w_1^n of phrase length n and $\mathbb{E}(w_1^n)$ is the expected frequency of the phrase, based on the overall frequency of the individual words $w_1, w_2, ..., w_n$ [Ram14]. Additionally, they use features based on the International Patent Classification (IPC) frequency, which is a system that classifies patents based on their topics:

$$freq(p_i, ipc_j) = |\{d \in D \,|\, p_i \in d \wedge d \in ipc_j\}| \tag{2.2}$$

In this equation, p_i is the ith sample phrase, ipc_j is a set of documents assigned to the jth IPC code and D are all available documents. This results in a $|P| \times |IPC|$ matrix with $|P|$ being the number of phrases and $|IPC|$ being the number of IPC codes. From this matrix the number of unique IPC frequencies (freq > 0), sum, median, mean and variance for each phrase were used as a feature. One downside of these statistical features is that they require the phrases to be distributed across the dataset. This means, if phrases occur in too few documents in the dataset, the IPC based features can not be used to differentiate between phrases. Additionally, features dependent on PoS tags were also considered, but not used due to the mentioned limitations of PoS tagging in patent texts.

A similar approach for extracting technical terms from patent texts using supervised machine learning is described by Anick et al. [AVP14]. They also rely on extracting PoS features for patent sentences and use these features to identify NPs in the text. These extracted NP phrases are then treated as term candidates, which they manually evaluate to create their ground truth phrases. For a term candidate to be accepted as a technical term they require it to reference either an **artifact** (object), **process** or **field** (of research). Further, the whole term candidate is either accepted or rejected and no sub-phrases are extracted from the term candidates. The features they used include contextual features, such as word n-grams to the left and right of the phrase, rule based dependency relationships (i.e. grammatical structure of the context) and position in the document, as well as phrase internal features such as token number, first word, last word and suffixes. Using a maximum entropy classifier, their English patent model trained on 490 English patents (3808 positive instances, 40589 negative instances) achieved a precision of 0.63, a recall of 0.57 and an F1 score of 0.60 on their test set of 10 patents. Using a larger model trained on a larger dataset (10000 patent documents), they managed to increase their recall to 0.77 while only slightly decreasing their precision to 0.57. Further, they find that internal features contribute significantly to the models ability to predict technical terms correctly.

2.2 Deep Learning

Deep Learning refers to a family of machine learning methods that are commonly based on **artificial neural networks** (ANN) which use **hidden layers** to solve problems that are beyond the limitations of linear models.

Given a set of examples that consist of various features, which are quantifiable representations of various example properties, the purpose of a machine learning model is to solve a specific task, by predicting output values. For this thesis we will mostly discuss the prediction of values that represent class labels. The machine learning model is supposed to improve its performance by tuning its model parameters according to the examples it receives, with an optimization algorithm. A model can only model reality accurately if the provided examples overall reflect reality. For linear models to be able to correctly predict non-linear data, the input needs to be transformed by a non-linear transformation function or mapping function ϕ so that it becomes linear. For example, **Support Vector Machines** (SVM) often use a **Kernel function** as a ϕ function. However, this limits SVMs, since these functions are only very generic hyperparameters that make it difficult to enhance an SVM model with additional information known about the problem. Another way of transforming the input data are hidden layers utilized by ANNs, which do not only make it possible to encode additional information in the hidden layer's architecture, but also allow the network to learn the transformation function ϕ from the data.

2.2.1 Basic Components

Simple ANNs are feedforward neural networks / multilayer perceptrons (MLP) that consist only of an input layer and an output layer as well as some hidden layers in between. It is called feedforward as there is no feedback mechanism, which would reuse the output of the model also as model input (see Section 2.3.2).

The basic building blocks of ANNs are vector-to-scalar transformations called **neurons**, which multiply an input vector with a trainable or pre-trained weight vector, add a bias and pass the resulting scalar through an often times non-linear activation function. If the weights (or biases) are trainable, they are considered to be part of the model parameters, which need to learned to improve the prediction. Generally, multiple neurons are grouped together to form a simple layer capable of performing vector-to-vector transformations, with every element of the input being connected to each neuron of the layer. This can be expressed with the function

$$f(input) = a(input \times W + b) \tag{2.3}$$

This function consists of an input vector $input$, a weight matrix W and a bias vector b and an activation function a. Since the layers are functions, they can be chained to form

networks of arbitrary size.

One of the simplest layer architectures is the **linear layer**, which just uses the identity function as activation function a. If a layer should consist of N neurons and be able to process an input vector of size M, the weight matrix variable must have the shape $(M \times N)$. Only using linear layers limits the model capacity (i.e. the relationships a model can model) to linear functions, making it a linear model. However, a linear model is only able to handle non-linear data, if the data is first linearized with a transformation function. In an ANN, this transformation is achieved with hidden layers that have non-linear activation functions. In some deep learning libraries, such as Tensorflow[1] or Keras[2], the general case of a simple layer that uses either a linear or a non-linear activation function is called **Dense Layer**. Which activation functions are used generally depends on their intended use in the architecture. If a probability needs to be expressed, the logistic sigmoid function $\sigma(x)$ (Figure 2.1a) may be used since its output lies between 0 and 1.

$$\sigma(x) = \frac{1}{(1 + \exp(-x))} \tag{2.4}$$

Another commonly used function is the hyperbolic tangent function $tanh(x)$ (Figure 2.1b) which has an output range of (-1,1).

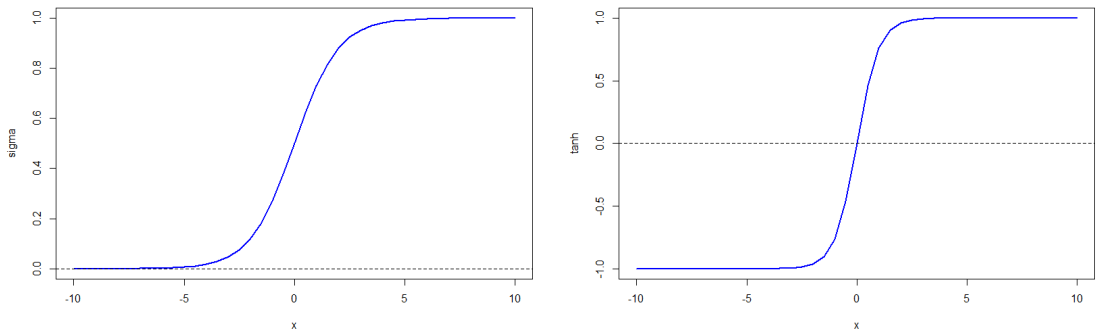
$$\tanh(x) = \frac{1 - \exp(-2x)}{(1 + \exp(-2x))} \tag{2.5}$$

In recent years, the rectified linear unit (ReLU) function (Figure 2.2) gained popularity as non-linear activation function, since it allows for larger network architectures featuring more layers.

$$relu(x) = max(0, x) \tag{2.6}$$

This is because it shares many properties that linear functions have, making models that use it easier to optimize with gradient descent. Many more functions can be used as activation functions, but they are usually limited to differentiable functions, due to the way ANNs are usually trained.

[1] https://www.tensorflow.org/
[2] https://keras.io/



(a) sigmoid (b) tanh

Figure 2.1: Comparison of the σ and *tanh* functions.

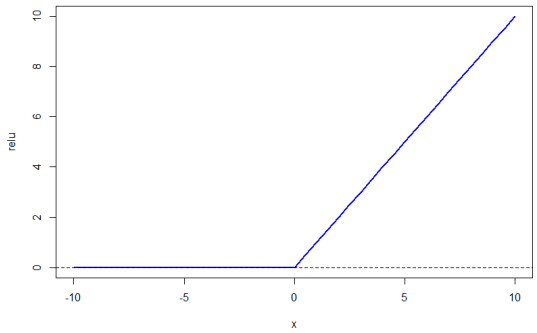


Figure 2.2: ReLU functions.

2.2.2 Input Units and Preprocessing

The input layer of an ANN is supplied with examples / samples $x_1, ..., x_M$ from a dataset of size M, which follows an unknown data-generating distribution $X(x)$, and consist of a number of features. The samples are fed into the network often in the shape of a so called **Design Matrix**, which contains one sample per row and one feature per column. Depending on the available resources and data, a single sample, a number of samples (a mini-batch) or the whole dataset (a batch) may be processed in a single step. While features can be categorical, numerical or of any other type, some sort of numerical representation must be passed to the network. Of course, the input of the ANN is not always in the shape of a matrix. If a sample only consists of a single feature, a vector of size *batchsize* can represent the data, with *batchsize* specifying the number of samples that are processed at the same time. If each sample consists of F features without any other known structure of these features, the design matrix of shape (*batchsize*, F)

will be used. If some additional structure of the data is known and the network can make use of it, tensors of higher rank can also be used. For example, if the dataset consists of samples of 2D images, the input could come in form of a tensor with shape (*batchsize*, *width*, *height*), with *width* being the maximum width for each image and *height* being the maximum height for each image.

A good way of representing the possible values of a categorical feature is one-hot encoding. One-hot encoding transforms a categorical feature that has N possible values into N binary features. For example, given a dictionary containing one million words, a categorical feature that should represent a word of this dictionary needs to have one million possible categories, with each word being a category. One-hot encoding would convert this feature into one million binary features, resulting in a vector of size $N = 1000000$.

Another way of handling categorical features are abstract representations. Instead of having a binary feature for each value, a value can also be represented by a combination of D numerical values. For example, if the feature 'weather' can take the values 'SUNNY' and 'RAINY' and $D = 3$, 'SUNNY' might be represented by the vector $(1, 2, 4)$ while 'RAINY' is represented by the vector $(0, -1, 3)$. The individual values of these vectors do not have to be interpretable by humans. With N possible values, a vector representation for each value of the feature can be stored in an $N \times D$ lookup matrix. The contents of this matrix can either be learned during training or supplied to the model pre-trained on some other data. Compared to the method of one-hot encoding, abstract representations of features reduce the dimensionality of the vector that is passed through the network (if $D < N$) without losing important information (if D is large enough). To go back to the dictionary example from above: the words of the dictionary of size $N = 1000000$ could be represented in an abstract fashion by vectors of size $D = 300$ (D chosen arbitrarily or through experimentation), which are then used for further calculations in the next layer. With one-hot encoding, a vector of size $N = 1000000$ that contains largely zeros, would have to be used instead. This topic is covered in more detail in Section 2.4.

2.2.3 Output Units and Cost Function

The output layer of an ANN depends on the task the ANN is supposed to handle. Usually a distinction between **supervised learning** and **unsupervised learning** tasks is made, although the category of a particular model oftentimes falls somewhere in between those two. Broadly speaking, unsupervised learning does not require a dataset annotated by a human supervisor and is used to determine structural or distributional properties of a dataset. An example method would be clustering, where samples are grouped together based on their similarities (according to some similarity measure). Another example are dimensionality reduction algorithms that calculate lower dimensional dense data representations for high dimensional, sparse data. In such cases, the output of the ANN might represent how well the data representation has been trained on the

training data, while the learned data representations are a subset of the model parameters.

On the other hand, supervised learning models require an annotated dataset (oftentimes created by human annotators) and directly compare the calculated results of the prediction with the known ground truth (also known as label or target value of the prediction), which follows the empirical distribution $Y(x)$. For regression tasks, the predicted value $\hat{y}$ and target value y are real numbers. For binary classification tasks, the target value is either one (true) or zero (false). Should the model predict the probability $p(y|x)$ - meaning the probability of target value y being produced by Y given the input observation x - a sigmoid activation function can be used for the output layer, as it produces values between zero and one and is monotonically increasing. If a choice between N possible labels with $N > 2$ needs to be made and a probability for each of these labels $l_1, .., l_N$ should be predicted, the output layer should be configured to predict a score / unnormalized probability $score_1, .., score_N$ for each possible label. By using the softmax function, the score for each label l_i can be transformed into label probabilities that sum up to 1, which estimate the true probabilities for a given sample.

$$softmax(score)_{l_i} = \frac{\exp(score_{l_i})}{\sum_{j=1}^{N} \exp(score_{l_j})} \tag{2.7}$$

When the output of the model estimates the true probabilities, the distribution of these outputs is called the model distribution $M(x;\theta)$ with θ being the model parameters. To measure the performance of an ANN in the context of supervised learning, a cost function that expresses how different the current prediction is from the ground truth must be used. The cost function is also called loss function, error function or objective function. One way of computing the distance between prediction and target value is to calculate the **Cross-Entropy** of their distributions.

$$H(Y, M) = -\mathbb{E}_{x \sim Y} \log M(x;\theta) = -\sum_{i} y_i \log(\hat{y}_i) \tag{2.8}$$

To calculate the cross entropy in the case of multi-label prediction, the model distribution $M(x)$ is compared to the empirical distribution $Y(x)$. By using softmax, $\hat{y}$ already expresses the predicted label probability. By converting the ground truth label y into probabilities with one-hot encoding, the probabilities between prediction $\hat{y}$ and ground truth label y can be compared. One-hot encoding transforms y into a vector of size N (with N being the number of possible labels) containing elements, which are 1 at the index of the true class and 0 everywhere else. When the cross-entropy is small, the model predictions will be similar to the given labels of the training set.

Further, common metrics for measuring the performance of a classification model include accuracy (2.9) as well as precision (2.10), recall (2.11) and F1 score (2.12).

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.9}$$

$$precision = \frac{TP}{TP + FP} \tag{2.10}$$

$$recall = \frac{TP}{TP + FN} \tag{2.11}$$

$$F1 = \frac{2 * TP}{2 * TP + FP + FN} \tag{2.12}$$

They are dependent on the true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) produced by the model. While the loss is generally used as a performance measure by the algorithm training the model, precision, recall and F1 score provide a further method of evaluation, that is easier to interpret when comparing different classification models that solve a similar task.

2.2.4 Model Training

When the model is initialized, the weights and biases will be initialized according to some initialization strategy. These strategies are often based on some heuristics and have the goal of achieving some useful properties upon network initialization. For example, symmetrical states in neurons that belong to the same layer that have the same input and activation function should be prevented, due to them being updated the same way with deterministic optimization algorithms. A simple strategy that helps with this is to initialize the weights with some small random values generated by a normal distribution with a mean of zero and a standard deviation of one. Further, too small initial weights can make a network more difficult to optimize, but too big initial weights can make the network oversensitive to small changes. To prevent this, all elements of a layer should be considered when initializing the weights of a layer, so that the resulting outputs are neither too big nor too small. For example, if the layer is not too large, instead of a standard deviation of one, a standard deviation of $\frac{1}{\sqrt{m}}$ could be used, with m being the number of inputs for a given layer. If the layers are too large so that the standard deviation becomes too small, a different strategy should be employed. In most cases the biases can just be initialized to zero.

Directly after initialization, it is very likely that the difference between the model distribution and the empirical distribution of the labels is large, leading to a high loss. To move the model distribution towards the empirical distribution, the model parameters need to be trained. In an ANN this is done via **gradient descent**, an iterative optimization algorithm to move towards a local minimum by taking small steps in the direction of the negative derivative / gradient of a function. In an ANN, the gradient contains the partial derivatives for each of the model parameters that influence the output (i.e. the loss)

of the model. The size of these steps generally depends on the learning rate, a model hyperparameter. For deep learning, finding a local minimum that is close to the global minimum usually provides a reasonable result. Since the objective function requires all input samples to compute the loss, the computational cost increases as the number of samples rises. Since the prediction achieves better results with more samples, a bigger dataset is generally preferable.

To combat the problem of ever increasing computational cost, **stochastic gradient descent** (SGD) was developed. Instead of using the whole dataset (batch) as input, only a random mini-batch of samples is taken from the training set and used to compute the gradient, providing a significant speedup at the cost of calculating less optimal gradients. To produce a random mini-batch, often the training set sample order is randomized and then split into same sized mini-batches. This is repeated at the start of each epoch, which is one iteration over the training set. Since the gradient steps are not as optimal as opposed to using the whole training set for its computation, more steps might be needed to descend, but this is more than compensated by the faster computation of the gradient. Generally, up to a few hundred samples are part of a single mini-batch. There are many optimization algorithms, such as Adam [KB14] or RMSProp [TH12], that further improve on certain aspects of SGD, such as utilizing the so called momentum of the gradient or adapting the learning rate during training.

The gradient needed for these optimizers is obtained by the back-propagation algorithm. While the gradient can also be computed by just using the chain rule of differentiation, this would lead to exploding computational costs, as with more network layers more and more terms would have to be calculated multiple times. The back-propagation algorithm instead stores the results of already calculated terms in memory. To do so, the back-propagation algorithm makes use of the computational graph defined by the network architecture. In the computational graph, each node represents a basic computation, such as a matrix multiplication or a max function application, and each edge represents a data flow from or to a node. During forward-propagation, starting at the input nodes, computations are performed forward through the network until the loss can be calculated. During back-propagation, the graph is traversed in reverse and starting at the loss V_n, for each node V_i and each edge $(V_i \leftarrow V_j)$ with $V_j \in Parents(V_i)$, the local derivatives $\partial V_i / \partial V_j$ are calculated and stored for said edge $(V_i \leftarrow V_j)$. After the loss node is set to $\partial V_n / V_n = 1$, the graph is again traversed in reverse and $\partial V_n / \partial V_i$ for each node V_i is calculated:

$$\begin{aligned} \frac{\partial V_n}{\partial V_i} &= \sum_{V_k \in Children(V_i)} V_k \cdot (V_k \leftarrow V_i) \\ &= \sum_{V_k \in Children(V_i)} V_k \cdot \frac{\partial V_k}{\partial V_i} \end{aligned} \tag{2.13}$$

Figure 2.3 and Figure 2.4 show a simple example of the algorithm by calculating $\partial f / \partial a$,

$\partial f/\partial b$ and $\partial f/\partial c$ for the equation $f(a,b,c) = max(a,b) * (b+c)$ with the input $a = 1, b = 2, c = 3$, an equation that could i.e. compute an output value for an example network. First, the graph is traversed forward to calculate f, then the local derivatives $\partial f/\partial d$, $\partial d/\partial a$, ... are calculated (Figure 2.3). Finally, the partial derivatives $\partial f/\partial d$, $\partial f/\partial a$, ... are calculated and the results shown in the nodes (Figure 2.4).

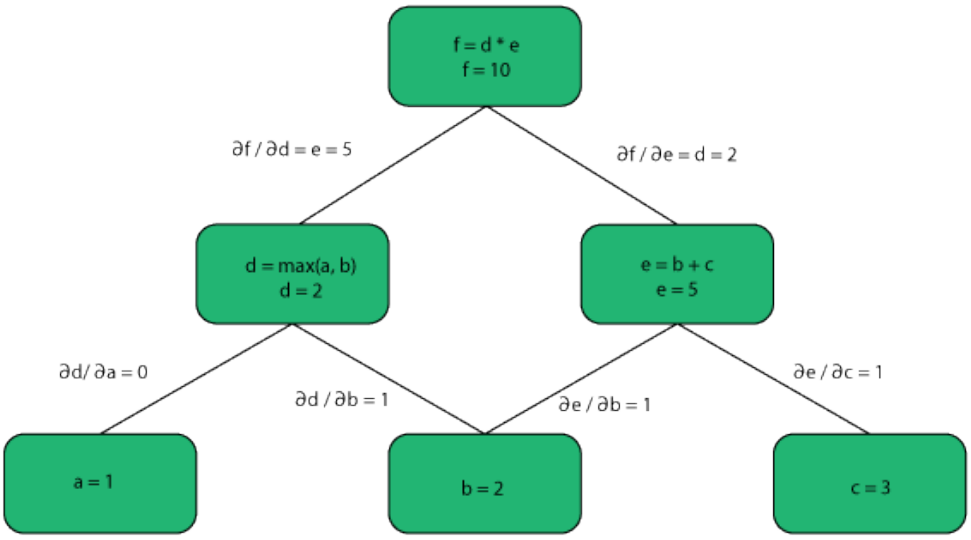


Figure 2.3: Forward-propagation with $f(a,b,c) = max(a,b) * (b+c)$

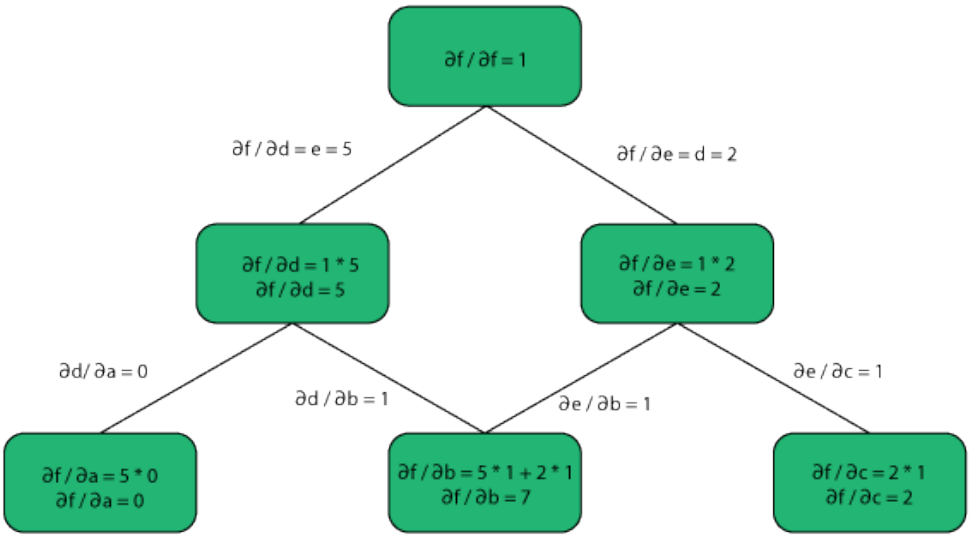


Figure 2.4: Backward-propagation with $f(a,b,c) = max(a,b) * (b+c)$

2.2.5 Regularization

One of the central goals of machine learning is teaching a model to not only perform well on the training data, but also generalize a problem in such a way that the model performs well on new, previously unseen data. One issue with creating a machine learning model is, the model needs to have enough capacity (i.e. be large enough to detect more complex connections between input features and output classes) to be able to achieve good results on the training data. Otherwise, the model will perhaps be too simple to effectively solve the task. However, a big model can lead to a different problem: instead of learning to solve the problem generally, the model could just overfit on the training set. I.e. the model could 'remember' the training set and be influenced too much by random noise in the training set - parts of the training set that contain no information relevant for the task that should be solved. This would lead to good results during training but poor results when working with new data. To reduce the difference between test error and training error, there are multiple regularization techniques that try to tackle different aspects of the problem.

To be able to evaluate how well a model is able to generalize on unseen data, a part of the available data is usually reserved for testing and not used during training. Especially with small datasets the measured testing performance of a model can substantially depend on which part of the dataset is used for training and which part is used for testing. In such a scenario, a more accurate estimate of the model performance can be obtained by repeating model training and testing on different dataset splits. A common method of doing so is the k-fold **Cross Validation** method, that repeats the splitting process k times and ensures that each test set of these k splits does not overlap with a test set of any other split. The average performance of these k models can then be interpreted as the overall performance of the model. However, this method requires training a model k times, which is not always feasible if a model takes a long time to train.

A common method of regularization is **weight decay**: by adding a small error value to the objective function that depends on the value of the model's weights, the model can be trained to avoid weight vectors that only perform well on the training set. One such error value is the L2 regularization term $\lambda||W||_2^2$, with λ being a hyperparameter that affects the strength of the regularization. Another commonly used method to prevent the model from overfitting is **early stopping**: during the training process, the performance of the model on the test set is evaluated in regular intervals, such as a fixed number of training steps or training epochs. If the model performance has not increased after an arbitrary number of performance checks, training is terminated and the model state at the time of best performance restored. Since the model also indirectly "sees" the test set due to the repeated performance checks on it during training, this might also cause overfitting on the test set. So instead of checking the performance on the test set, it is recommended to use a development set instead and only use the test set for the final performance check with the best model. The next method can be seen as a way of training multiple models at

once or rather an approximation thereof. **Dropout** [SHK+14] trains the network to not only rely on specific individual neurons by setting the output of randomly chosen neurons to zero, thus completely removing their influence on the network. Dropout is applied as a layer or wrapper around a layer and each affected neuron has a predetermined probability of being chosen for a training step. The probability of keeping a neuron (i.e. not setting it to zero) is a hyperparameter that is often fixed to 0.5 during training and always 1.0 during evaluation.

2.3 Specialized Architectures

Sometimes there is additional information about the features of samples available that is not easily expressible as an additional feature. This could include, for instance, the grid-like relationship between pixels in a picture or the sequential relationship of words in a sentence. For these problems, it is possible to encode such additional information directly into an ANN architecture. Common problems like spatial or sequential relationships are active fields of research and many different approaches exist to make use of these relationships.

In this section and beyond we will use the Numpy notation[3] to describe tensors, which uses the 'x:y' to indicate tensor slices. Omitting x or y means the index starts or ends at the lowest or highest possible index value. For example, the whole ith row vector of a matrix M is referred to as $M[i,:]$. The vector of the jth through kth elements of the ith row vector are referred to as $M[i, j:k]$. This helps with emphasizing the shape of a tensor and keeps longer descriptions of index calculations more readable.

2.3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) consist of a variety of layers that make use of sparse connectivity and parameter sharing, which stands in contrast to typical dense layers, where every neuron has its own set of weights that are connected to all inputs. Parameter sharing in this context means that, if represented as a dense layer, multiple neurons in the dense layer would reference and reuse the same weight variables. Convolutional layers (CNN layer) can be depicted as having a kernel of size Sp that moves over the input data and produces an output of depth Out for each step it takes. How big the size of these steps are is defined by the stride of the kernel, which can be different for each of the kernel's movement directions. In how many directions the kernel needs to move depends on the data that is used. For example, the elements (e.g. words or characters) in each sentence can be traversed using one dimensional movements (i.e. from left to right in the sentence), while the pixels in an image can be traversed using two dimensional movement (i.e. from left to right and from top to bottom). Using the sentence example, the sentence representation $Sent$ of shape $L \times In$, consists of L sentence elements, each of which are

[3] https://docs.scipy.org/doc/numpy-1.13.0/reference/arrays.indexing.html

represented by a vector of size In. A CNN layer that processes this sentence uses a kernel K, which is a tensor of shape $Sp \times In \times Out$, where Sp is the size of the kernel or the span of sentence elements that influence the output value, In is the input depth and Out is the output depth. Further, a stride St needs to be defined that determines how many words the kernel moves after each kernel application. The result is an output matrix CNN with shape $L/St \times Out$, showing that a stride of 2 or higher results in a shorter output tensor in the direction of the sentence length. The output size Out can be chosen arbitrarily. Each ith output vector[4] $CNN[i,:]$ of the output matrix CNN can now be calculated with the following formula:

$$CNN[i,:] = \sum_{s=0}^{Sp-1} Sent[i * St - \lfloor Sp/2 \rfloor + s,:]\, K[s,:,:] \tag{2.14}$$

The key difference to dense or linear layers is, the weights that the kernel consists of are only applied to a window of the input. Further, the same weights are used for all possible windows of the layer. Figure 2.5 shows an example CNN for processing a sentence. A $3 \times 2 \times 4$ kernel moves over the sentence with a stride of $St = 1$ and a span $Sp = 3$. The sentence consists of sentence elements that are represented by input vectors of size $In = 2$. Each kernel step produces an output vector of size $Out = 4$ by applying the kernel to the sentence elements of the current observation window. In the example, the current observation window consists of the three vectors $In[0,:]$, $In[1,:]$ and $In[2,:]$. Since the span $Sp = 3$, the kernel also consists of the three matrices $K[0,:,:]$, $K[1,:,:]$ and $K[2,:,:]$, one for each sentence element in the observation window. To apply the kernel, the first element of the window is multiplied with the first kernel matrix i.e. $In[0,:]\, K[0,:,:]$, resulting in a vector of shape 1×4. This is repeated for the second and third element respectively. These three resulting vectors are then summed up using element-wise addition, which results in the output vector $CNN[1,:]$.

If in the example from above the vector $CNN[0,:]$ should be calculated, the observation window would need to start one index location earlier, at element $In[-1,:]$. If we assume that $In[0,:]$ is already the first sentence element in the sentence, then $In[-1,:]$ is not defined. This means that around the edges of a sentence (or image) there is an area of undefined inputs that need to be handled when calculating the convolutions. Typically, this is handled in two ways: the values are not calculated and the image is shrunk accordingly with each convolution or the values are calculated by assuming undefined inputs are all zero.

Another very similar layer that is often used in conjunction with CNN layers is the pooling layer. The pooling layer follows the same principles as the CNN layer, only the kernel matrix multiplication is replaced with a simple function, such as the max function

[4] Numpy notation

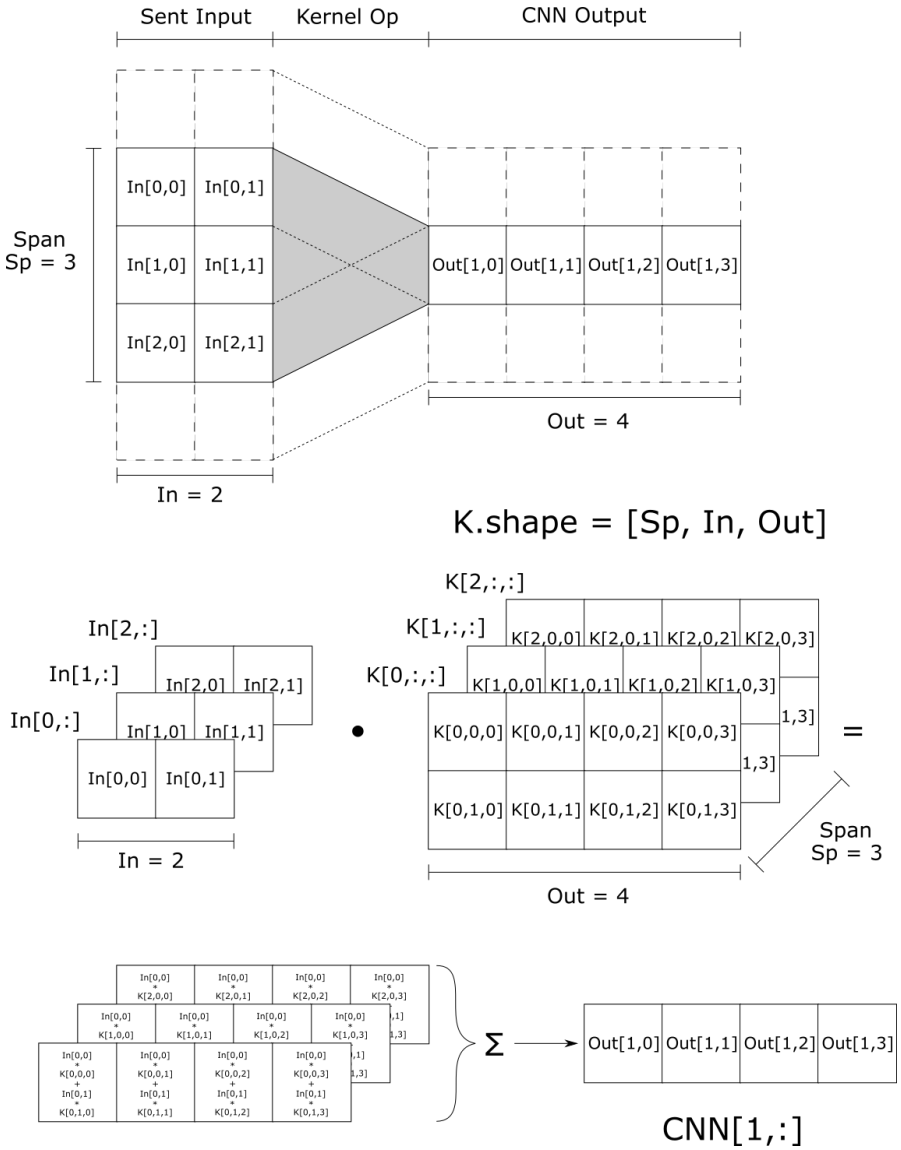


Figure 2.5: A single kernel application in a CNN Example for a sentence. It shows which parts of the input matrix are needed to calculate a row in the output matrix and which parts of the kernel tensor are multiplied with which vector of the input matrix.

(max pooling) or average function (average pooling), that is applied to the whole window.

Architecture designs for CNNs are still an active field of research and many layer composition variations exist, although a common theme is the stacking of multiple CNN or pooling layers on top of each other. Zhang et al. [ZZL15] show that an architecture alternating between character level CNN layers and pooling layers with ReLU activation

functions is effective for text classification. He et al. [HZRS16] show that for the task of image classification a high accuracy can be achieved with very deep, but relatively low complexity neural networks by fitting a residual mapping $F(x) = H(x) - x$ instead of the underlying mapping $H(x)$. In their network, $H(x)$ is formulated as $F(x) + x$ with $F(x)$ being a network building block. A two layered $F(x)$ block would have the form $CNN(relu(CNN(x))$ with CNN denoting a CNN layer.

2.3.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are special types of networks that are well-suited to handle sequential data, such as of a time series or sentence, and are able to learn if the expected output depends on previous information in the sequence. In contrast to other layer types, the output state h^t of a RNN layer at the sequence position / time t not only depends on its input x^t and the model parameters, but also the previous state h^{t-1}.

$$h^t = a(Wx^t + Uh^{t-1} + b) \tag{2.15}$$

In this equation a refers to the activation function, W and U are weight matrices, h^{t-1} refers to the previous state in the sequence and b refers to a bias vector. Since the length of a sequence might not be known beforehand and it could be possible that the same information appears at different points in the sequence (such as phrases in a sentence), parameters between the different points in time are usually shared. For this reason, a RNN layer consists of a single RNN cell having some weight parameters that loops its output state to itself. Since in practice all outputs are needed to produce a gradient via back-propagation, all inputs need to be fed into a layer simultaneously, which is achieved by "unrolling" the RNN layer to a fixed maximum length. This means the function representation of the layer is transformed from taking only the previous state h^{t-1} and x^t as input to taking x^0 to $x^{maxLength}$ as input and producing output states h^1 to $h^{maxLength}$, whereas x^0 is a predetermined initial state. If it is not only necessary to learn based on information from the past, but it is expected that the whole sequence affects the current state (as is the case in sentences), a bidirectional RNN can be used. This is a RNN architecture that consist of two sub-RNNs, one that moves forward and one that moves backward in time, which produce two output vectors that are combined (e.g. concatenated).

One problem with RNNs working on long sequences is that the gradient might become very high (exploding gradient) or very low (vanishing gradient), both scenarios make it difficult to train the model. The exploding gradient problem occurs rarely and can easily be fixed by clipping the gradient if it becomes too large. The more frequent vanishing gradient problem requires a more elaborate method. In the case of a simple RNN cell the vanishing gradient in long sequences is caused by the fact that the same RNN cell is reused, thus leading to a high number of multiplications of identical weight matrices, which makes it difficult for information further back in the sequence to affect the present. Some architectures extend the RNN cell in a way that counters this effect to some degree,

such as the Long Short Term Memory (LSTM) cell [HS97].

The LSTM cell architecture introduces trainable hidden units inside the cell that modify the received input and previous states in a gate-like fashion, making it possible for the cell to pass on only certain data by multiplying it with values between 0 and 1. This enables the cell to remember information over many time steps and discard it when it is not needed anymore. An LSTM cell uses a forget gate $f^t = \sigma(W_f x^t + U_f h^{t-1} + b_f)$, input gate $in^t = \sigma(W_{in} x^t + U_{in} h^{t-1} + b_{in})$ and output gate $out^t = \sigma(W_{out} x^t + U_{out} h^{t-1} + b_{out})$ and, in addition to the output state $h^t = \tanh(s^t) * out^t$, also passes on an internal state $s^t = f^t s^{t-1} + in^t \sigma(W x^t + U h^{t-1} + b)$. Again, in these equations, the Ws and Us refer to weight matrices and the bs refer to bias vectors.

Today, the LSTM cell is not the only RNN architecture that uses gates and various architectures can be considered modifications of the original architecture, in some cases changing connections inside the cell or changing the number of gates, such as the Gated Recurrent Unit architecture [CGCB14].

2.4 Representation Learning and Word2Vec

In an ANN classifier, the last layer before the cost function is usually a linear layer that produces a score for each possible label. This means each hidden layer earlier in the network has the purpose of producing a representation of the data that makes it easier for the next layer to work with, which translates to transforming the data into a linearly separable representation (for each class compared to all other classes) for the final layer. While there is usually a large amount of unlabeled data available for unsupervised learning, the labeled data needed for supervised tasks like classification is usually not as plentiful. However, unlabeled data can sometimes be used for learning representations that can be inserted into classifiers as a form of pre-training (Semi-supervised learning). The information that can be obtained via pre-training generally depends on the type of data and influences the architecture of the representation that is to be learned. For example, for creating representations of words in a dictionary, the $N \times D$ hidden layer weight matrix of a network with a single hidden layer can be sufficient, with N being the size of the dictionary and D being an arbitrarily chosen word vector size. In this case each of the N vectors with size D of the $N \times D$ weight matrix represent a word, making the vectors **vector representations** of individual words in the dictionary (also called **word vectors** or **word embeddings**).

One frequently used group of methods of creating word vectors is word2vec by Mikolov et al. [MSC+13, MCCD13], which consists of the Continuous Bag of Words (CBOW) model and the Skip-gram model, two similar models for predicting word-context pairs. The CBOW model tries to predict the word in the center of a context window based on the other neighboring words in the context window. For each word w_t in vocabulary V in the

training corpus T, the CBOW model uses the dictionary indices of the N neighbors w_{t+i} and w_{t-i} with $i \in 1..N/2$ as input. The then one-hot encoded input is passed through a shared **projection layer** which transforms the one-hot inputs into N (continuous) word vectors that then get averaged into a single vector. This process is also sometimes called embedding lookup and the matrix of the **projection layer** is sometimes called lookup table. The output of this **projection layer** is passed through a hidden layer with V output units to calculate the probability of each word. If the softmax function were to be used for this, the computational complexity would depend on $|V|$, which can be in the millions. To prevent this problem, a hierarchical version of the softmax function [MB05] is used, which causes the complexity to only depend on $log_2(V)$.

The Skip-gram model can be seen as similar to a reverse CBOW model in that instead of predicting the current word from a number of context words, it predicts a context word from the current word. The context words of a training sequence $w_1, ..., w_T$ are again chosen by taking neighboring words w_{i+j} in a certain window around in the current word w_i. To account for the fact that words further away from the current word are generally less related, the neighbor window size R is randomly chosen between 1 and a maximum size C. For each context word w_{i+j} with $-R \leq j \leq R, j \neq 0$, a sample word-context pair (w_i, w_{i+j}) is created with its current word w_i. The objective function of the Skip-gram model is given as

$$\frac{1}{T} \sum_{i=1}^{T} \sum_{-R \leq j \leq R, j \neq 0} \log\, p(w_{i+j}|w_i) \tag{2.16}$$

In the architecture of the Skip-gram model v_w is the one-hot encoded input vector of a word and v_c is the one-hot encoded target vector of a context word. The vector v_w is multiplied with a projection matrix, the $D \times |V|$ word vector matrix WV, to receive the vector representation $emb_w(w_i)$ of word w_i. The vector representations of the contexts are vectors of a different matrix, the $D \times |V|$ context vector matrix CV, which contains the corresponding vector representation $emb_c(w_i)$ of the one-hot vector $v_c(w_i)$ of word w_i. Similar to CBOW, the probability $p(w_{i+j}|w_i)$ could be calculated with the softmax function

$$p(w_{i+j}|w_i) = \frac{\exp(emb_w(w_i)^\top\, emb_c(w_{i+j}))}{\sum_{w_c \in CV} \exp(emb_w(w_i)^\top\, emb_c(w_c))} \tag{2.17}$$

but since $|V|$ is generally big, a different approach is taken. Instead, the assumption is made, that with a high probability any random word w_x has a different context than w_i. During training the actually occurring pair (w_i, w_{i+j}) is contrasted to K random other pairs (w_i, w_x) in such a way that in the vector space the vectors $emb_w(w_i)$ and $emb_c(w_{i+j})$ move closer together and $emb_w(w_i)$ is pushed further away from $emb_c(w_x)$. This is done by replacing $\log p(w_{i+j}|w_i)$ in the objective function of the Skip-gram model with

$$\log\, \sigma(emb_w(w_i)^\top\, emb_c(w_{i+j})) - \sum_{k=1}^{K} \log\, \sigma(emb_w(w_i)^\top\, emb_c(w_k)) \tag{2.18}$$

whereas $emb_c(w_k)$ is the embedding vector of the kth randomly chosen word. Furthermore, to reduce the impact of frequently occurring words, each word w_i in the corpus is removed with the probability of $P(w_i) = 1 - \sqrt{t/f(w_i)}$, with t being a threshold around 10^{-5} and $f(w_i)$ being the frequency of word w_i.

The word vectors of the matrix WV created by the Skip-gram model were evaluated based on semantic and syntactic relationship properties. By using the measure of the cosine similarity $similarity(A, B) = (A \cdot B)/(||A||\,||B||)$ between vectors, words that are expected to share similar contexts (like "beer" and "alcohol") should have word vectors that are also similar. These semantic and syntactic properties were also tested for relationships between multiple word-pairs. For example, it was tested whether the relationship between WV("big") to WV("biggest") is the same as WV("small") to WV("smallest"). Mikolov showed that with simple vector addition and subtraction around 55% of the word-pair comparisons in their semantic test set (i.e. capital-country relationships) and around 59% of the word-pair comparisons in their syntactic test set (i.e. word form relationships) are expected to yield related results. For example, by taking vectors that have been trained on news data, WV("Madrid") - WV("Spain") + WV("France") results in a vector that is most similar to WV("Paris"), or WV("Germany") + WV("capital") is most similar to WV("Berlin"). This shows that the vectors of some words encode some semantic and syntactic word properties. What and how well these properties are preserved generally depends on the data the word vectors were trained on.

2.5 Named Entity Recognition

Similar tasks to MWT extraction that sometimes make use of word representations include named entity recognition (NER) and keyphrase extraction, both of which need to recognize if multiple words form a unit [ASC13]. The task of NER not only includes finding special phrases in a text document, but also assigning certain class labels to them, such as **city**, **person** or **organisation**, whereas in keyphrase extraction, the phrases that are selected also need to be representative of the document they have been extracted from. This is different from the process of MWT extraction in which identified terms are not assigned labels and do not need to be representative of their parent document. Nonetheless, some methods from the NER domain and keyphrase extraction domain can be expected to be similar enough in their approach, in a sense that they can also be applied to extract MWTs after some modifications.

The SemEval 2017 semantic evaluation [ADR+17] consists of a series of challenges, each focusing on a different area of study. Task 10, which is titled "Extracting Keyphrases and Relations from Scientific Publications" [ADR+17], combines aspects from both keyphrase extraction and NER. Given a dataset of 500 documents, each of which consists of one paragraph from scientific publications in the domains of computer science, physics and material science, three subtasks had to be performed. The first was extraction

of keyphrases from the document text, the second was classifying these phrases as either **Material**, **Process** or **Task** and the third one was identifying **hypernym** and **synonym** relationships between these keyphrases. The overall best performance was achieved by Ammar et al. [APPB17] with an F1 score of 0.55 for the entity extraction task, 0.44 for the entity classification task and 0.28 for the relationship extraction task. They use two ANN models for these three tasks, whereas their **entity model** is used for extracting and classifying entities and their **relation model** predicts the relationship between two mentioned entities. The **entity model**, which is more relevant for the task of MWT extraction, is a sequence to sequence model and predicts entities and their classes from the text sequence using BILOU encoding. The BILOU encoding and its variations are used to encode entities as label sequences (B = beginning, I = inside, L = last, O = outside, U = unit). For example, a 6 word token sentence 'The extended hypertext markup language:' consisting of two irrelevant word tokens ('The' and ':') and a 4 word entity ('extended hypertext markup language') that starts with the second word of the sentence would be encoded <O,B,I,I,L,O>. A single word of interest would be tagged with the label U.

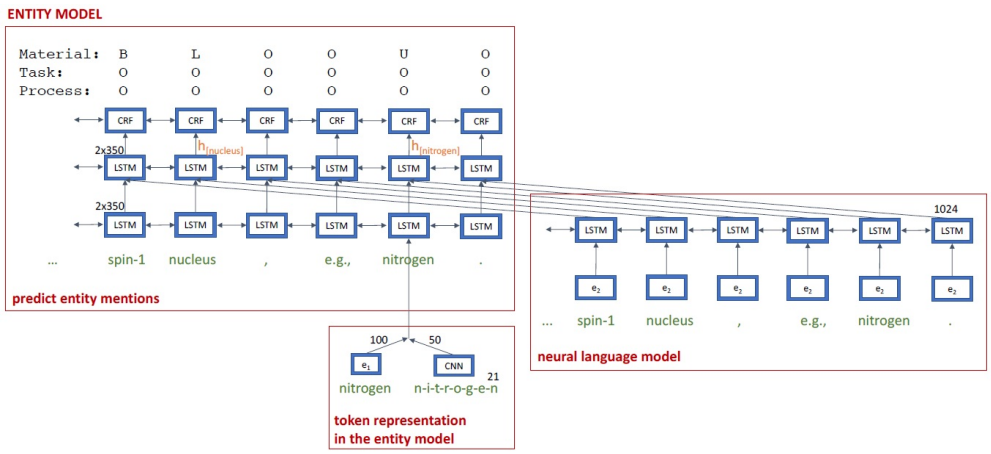


Figure 2.6: Entity model by Ammar et al. Image from Ammar et al. [APPB17]. *CNN* refers to a CNN component, *LSTM* to unrolled cells of an LSTM-layer and e_i to word embeddings.

The following enumeration describes the components of the **Entity Model** as seen in Figure 2.6:

1. The core of the model requires a sequence of word tokens as input and converts these tokens into word representations (Figure 2.6 token representation in the entity model). To convert the word tokens into word representations two methods which each produce their own word vector are combined by concatenating the results. The larger part of the final word representations with a dimensionality of 100 consists

of pre-trained GloVe word embeddings [PSM14], which are just looked up. The smaller part with a dimensionality of 50 consists of a CNN with a filter width of 3 that takes the characters of each word token as input.

2. The word representations are then passed through two bi-directional LSTM layers whose output is then used to calculate a label score for each word and each label with a linear layer (Figure 2.6 predict entity mentions). These scores are evaluated by a **conditional random field** (CRF) layer to produce a loss and the most likely sequence of labels.

3. Further, the output of a pre-trained RNN language model, which consists of a sequence of context dependent word representations, gets concatenating to the output of the first LSTM layer (Figure 2.6 neural language model).

4. Finally, gazetteer features, which are additional features based on a technical dictionary, get added to the model by representing the gazetteers as a sequence of binary feature vectors, resulting in a $N \times G$ feature matrix with N being the number of tokens in an input sequence and G being the number of gazetteers. These features are passed through a dense layer with a *tanh* activation function whose output is then concatenated to the output of the second LSTM layer (not shown in Figure 2.6).

2.6 Conditional Random Fields

Sequence to Sequence models that make use of a combination of word representations, LSTM layers and CRF, such as the just mentioned Entity Model, have been employed in the field of NER to great success and some architecture variations exist (Neural architectures for named entity recognition [LBS+16], Bidirectional LSTM-CRF models for sequence tagging [HXY15], Natural language processing (almost) from scratch [CWB+11]). A recurring element is the use of linear chain CRFs to predict the label sequence. A linear chain CRF is an undirected graphical model with a random variable $X = (X_1, X_2, ..., X_T)$ over an observation sequence of length t and a random variable $Y = (Y_1, Y_2, ..., Y_T)$ over a label sequence of the same length using a label alphabet $L = l_1, ..., l_{|L|}$ that also contains a special label for the start and end state. For example, if X models sentences, then $(X_1, X_2, ..., X_T)$ are random variables for word 1 through T in the sentence, that describe the occurrence probability of each word for each position in the sentence. Such a CRF is used to model the conditional distribution p(Y|X) without having to model the distribution of the observations $p(X)$. This discriminative modeling approach stands in contrast to the generative modeling approach of modeling the joint distribution $p(Y, X)$. A linear chain CRF (that can be used when deciding i.e. on sentence labels) uses a graph G with edges $E = (i, i + 1)$ and vertices $V = (1, 2, ..., t)$ and has the following properties:

"**Definition**. Let $G = (V, E)$ be a graph such that $Y = (Y_v)_v \in V$, so that Y is indexed by the vertices of G. Then (X, Y) is a conditional random field in case, when conditioned on X, the random variables Y_v obey the Markov property with respect to the graph: $p(Y_v|X, Y_w, w \neq v) = p(Y_v|X, Y_w, w \sim v)$, where $w \sim v$ means that w and v are neighbors in G."
–Lafferty, John. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data." Section 3 [LMP01]

In a sequence to sequence neural network a realization x of X that is used as an input for the CRF is in the form of a potential score (=unnormalized probability) matrix PS, due to some previous layer in the model with trainable parameters, such as a linear output layer. The potential score matrix PS consists of vectors[5] $PS[x_i, :] = (PS[x_i, l_1], ..., PS[x_i, l_{|L|}])$ for each element x_i in the realization x, with l_j being the jth label in the label alphabet L. This means $PS[x_i, l_j]$ refers to the label score for label l_j that the ANN has calculated for sequence element x_i. For example, if in a sentence the elements x_i are words and the labels are PoS tags, each word receives a different score for each PoS tag depending on what features the ANN has previously calculated for this word. E.g. a simple network could perhaps just look at the characters at the end of a word, and the suffix is 'ing', it assigns a score of 10 for the label 'VERB', a score of 5 for the label 'NOUN' and a score of 1 for the label 'ADJ'.

Further, a $|L| \times |L|$ transition matrix A that contains transition scores $A[from, to]$ for transitioning from label $from \in L$ to label $to \in L$ is used. For example, if there are two labels, 'ADJ' and 'NOUN', the transition matrix needs to have scores for the transitions of 'ADJ' to 'ADJ', 'ADJ' to 'NOUN', 'NOUN' to 'ADJ' and 'NOUN' to 'NOUN'. If a transition from label $from$ to label to is observed very frequently in the training data, the transition score $A[from, to]$ will be higher. Together, the matrix A and a realization $x = [x_1, ..., x_T]$ in form of the matrix PS can be used to calculate a score for a particular label sequence $y = [y_1, ..., y_T]$ with the scoring function s:

$$s(x, y) = \sum_{t=1}^{T} A[y_{t-1}, y_t] + PS[x_t, y_t] \, with \, y_t \in L \tag{2.19}$$

This score is part of the function to compute the conditional probability $p(y|x)$ of label sequence y given observation sequence x:

$$p(y|x) = \frac{\exp(s(x, y))}{Z(x)} \tag{2.20}$$

The partition function $Z(x)$ computes a sum over all possible sequences of y:

$$Z(x) = \sum_{y} \exp(s(x, y)) \tag{2.21}$$

[5]Numpy notation.

However, $Z(x)$ is not needed to predict the most likely sequence since it does not depend on y:

$$\underset{y}{\operatorname{argmax}}\, p(y|x) = \underset{y}{\operatorname{argmax}}\, s(x, y) \tag{2.22}$$

The best scoring sequence can then be calculated recursively with the Viterbi algorithm [For73] (Algorithm 2.1). This means first, the best scoring sub sequence of length 1 ending in label $l_j \in L$ is computed for each label (which is a sequence of length 1, i.e. just the label l_j itself). Then the resulting best sequence for each label in the previous time step is used to calculate the best scoring subsequence of length 2 ending in label l_k for each label $l_k \in L$. This process is repeated until the best scoring sequence of length T can be computed for each label. Of these, the one with the highest score is the end result. [LMP01, Wal04]

Algorithm 2.1: Viterbi for CRF

Input: A transition matrix **A** and a potential scores matrix **PS**.
Output: The sequence of labels with the highest score.

```
1  best_sequences ← empty list of lists;
2  scores ← empty list;
3  for label ∈ L do
4  |  best_sequences[label] ← [label];
5  |  scores[label] ← s([1], best_sequences[label], A, PS);
6  end
7  for step ← 2 to t do
8  |  for label ∈ L do
9  |  |  cur_label_scores ← empty list;
10 |  |  for last_label ∈ L do
11 |  |  |  cur_label_scores[last_label] ← s([1 : step],
   |  |  |   best_sequences[last_label] concat label, A, PS);
12 |  |  end
13 |  |  best_last_label ← argmax(cur_label_scores);
14 |  |  new_best_sequences[label] ← best_sequences[best_last_label] concat
   |  |   label;
15 |  |  scores[label] ← cur_label_scores[best_last_label];
16 |  end
17 |  best_sequences ← new_best_sequences;
18 end
19 best_score_final_label ← argmax(scores);
20 return best_sequences[best_score_final_label];
```

Since calculating all possible sequences is exponential with respect to the sequence length, $Z(x)$, which is needed for computing the CRF loss (i.e. the cross-entropy) during training, is calculated differently. Instead, the transition matrix A and potential score matrix PS

is used to produce the matrix elements $M_t[a, b]$ of a matrix M_t specific to observation x_t as follows:

$$M_t[a, b] = exp(A[a, b] + PS[x_t, b]) \tag{2.23}$$

The matrix M_t has the same size as matrix A, with $a, b \in L$. To calculate $Z(x)$, all T of these matrices have to be multiplied with each other and the sum of all elements of the resulting matrix taken:

$$Z(x) = \sum_{a,b \in L} (\prod_{t=1}^{T} M_t) = \sum_{a,b \in L} (M_1 \, M_2 \, ... \, M_{T-1} \, M_T) \tag{2.24}$$

This allows for calculating the probabilities $P(y|x)$ without having the calculate the score for each possible permutation of labels. [LMP01, Wal04]

CRFs are often used as the final component in a larger ANN to output a valid sequence of labels. For example, in the $BILOU$ encoding the label I can never appear directly before or directly after the label O. Given enough training, the model can learn this relationship and will never predict an impossible sequence.

2.7 Summary

In this chapter we discussed MWTs in patents and presented the basics of deep learning. By building an ANN using the described deep learning methods and specialized components and architectures, such as CNNs, RNNs using LSTMs, word embeddings and CRF, models can be trained to detect entities, keyphrases and MWTs. Deep learning frameworks such as Tensorflow already provide implementations for these basics and specialized components and architectures. There are also a few libraries, such as the gensim library[6], that provide a word2vec model implementation and a corresponding interface for training word embeddings. In the next chapter, we will present how we use the discussed components to create an ANN architecture for creating models capable of extracting MWTs from patent texts.

[6] https://radimrehurek.com/gensim/

CHAPTER 3

Methodology

In this chapter we discuss the methodology that is used to automatically detect MWTs in patent texts. Due to the decreased efficiency of PoS taggers in the patent domain, our approach needs to not primarily rely PoS tag information. Further, our method should also be capable of detecting rare patent MWTs, i.e. MWTs that appear only a few times in a dataset of patent texts, due to only small amounts of data being available to us. Because of the relative closeness of MWT detection, NER and keyphrase extraction, we look into successful NER and keyphrase extraction methodologies and apply them to MWT detection in the patent domain. Since the state-of-the-art algorithms capable of NER are deep recurrent neural networks trained on labeled text data, we create a similar ANN architecture consisting of the core components of the mentioned NER models and tune it for the patent domain. To be capable of training and evaluating our models, we create a MWT detection dataset in the patent domain. To to measure the effectiveness of our model, we also implement a baseline method and compare the results.

3.1 Dataset Creation

Due to lack of annotated data for our task, we create our own dataset by labeling the occurrences of technical MWTs in 22 randomly chosen patents from the CLEF-IP collection in plain text format by hand. The selected patents belong to a variety of domains, such as computing, textile production or chemistry, as is indicated by their IPC codes. Our decision as to what phrase constitutes a technical term is based on the linguistic properties of that phrase and our understanding of the meaning of the phrase and its surrounding context. We focused on the longest noun phrases and judged the termhoodness of a phrase and its sub-phrases based on their relevance in the sub-domain, leaving out sub-phrases that are possibly subjective / structural additions of the author. Examples of this include leaving out the word *high* in *high crosslinking rate* or entirely skipping phrases like *first component*. We still marked sub-phrases as MWTs that seemed

to describe a concept possibly relevant for the identification of the patent. While the overlying task of detecting keyphrases is not part of this work, we expect that some other algorithm further filters the detected MWTs for keyphrases. Under this consideration and due to a lack of expert knowledge we were more lenient with our termhoodness evaluation, meaning we favored a high recall over a high precision during our manual annotation. For example, in a patent regarding the production of *glazed baking products* we would also consider the phrase *shiny appearance* a MWT, and in a patent regarding the production of *feed additives* we also considered *grape juice* a MWT, despite the common usage of these phrases and words. While Andersson et al. [ALP+16], Frantzi et al. [FAM00] and Anick et al. [AVP14] mention that MWT candidates often consist of adjectives (ADJ) and nouns (NOUN), the first two authors write that prepositions (PREP) are "sometimes" or "occasionally" included and Anick et al. do not include them at all in their description of patterns capable of detecting MWT candidates. For this reason, if the entire NP includes a PREP, we annotated sub phrases before and after the preposition more often as MWTs than the entire NP. All unique MWTs found during the annotation are gathered in a list for further processing. To save time, we only added the first occurrence of a MWT to the list and marked all further occurrences in the text — as determined by a string search — as already annotated. The downside of this are annotation inaccuracies, meaning if a NP is a MWT at some point in the text, it will be marked as such even if the same token sequence is not a MWT at a different part of the text. For example, the MWT "time stamp" also occurs as a verb phrase in the text, i.e. not a MWT.

3.2 Preprocessing

To convert the dataset into a usable format, we employ a custom preprocessing pipeline coded in Python capable of converting raw text into samples (Figure 3.1). To help with some of the preprocessing steps, we also make use of some functionalities of either the **Natural Language Toolkit** [1] or **spaCy**[2] natural language processing libraries. The original dataset consists of one file of raw text for each of the 22 patents and one file containing all unique MWTs. Our goal is to convert the text into samples consisting of one sentence each that consist of word / label pairs following the BILO encoding scheme (see Section 2.1). Further, we ensured that we have a good match between the dictionary of the our word embedding resources and the resulting tokens of our preprocessing pipeline.

First, the files are read from disk and split into sentences, then any disrupting characters or character sequences are substituted: any characters not in our alphabet are replaced with a special character, any remaining new line feeds are replaced with white spaces, any brackets are replaced with round brackets and any numbers (as detected by a RegEx pattern) are replaced by a number symbol. After the sentence has been converted to lower

[1] `https://www.nltk.org/`
[2] `https://spacy.io/`

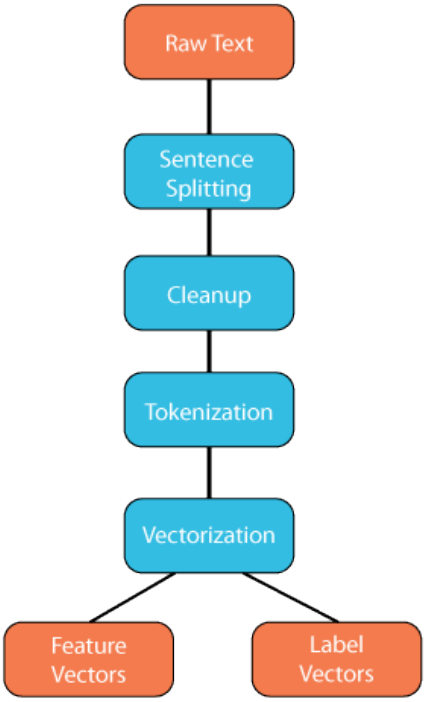


Figure 3.1: Visual representation of the preprocessing pipeline.

case it is passed to spaCy for **tokenization** and lemmatization (only of plural nouns), producing a list of tokens, with each token being an individual word or punctuation symbol. In the case of the baseline method, we also compare the performance of spaCy and NLTK in this regard. In this step we also extract PoS tag features for each token for the experiments that make use of this information. Furthermore, to ensure compatibility between the word embedding resources and the preprocessed tokens, we remove all dashes from the dataset. This also means that words separated by dashes instead of white spaces are treated as multiple tokens, meaning phrases like '5-vinyl-2-norbornene' are treated as MWTs. Since sentences need to be converted into sentence tensors in the next step, all sentences that exceed a maximum sentence length $SMAX$ are naively split into multiple samples, by taking the first $SMAX$ tokens of a sentence as one sample, the second $SMAX$ tokens as the next sample and so on.

During **vectorization**, the tokenized sentence samples are used to generate sentence level feature tensors for word level and character level features as well as sentence level label tensors. The word level sentence tensor is a padded vector of size $SMAX$ with each token in the sentence being represented by the index integer of that token in the dictionary. Special values are reserved for the padding word and the unknown word, as well as a list of additional tokens we added to the dictionary but for which we do not have any word embeddings, such as the special token that represents the concept of numbers in

the text. The character level sentence tensor is a padded matrix of size $SMAX \times CMAX$ with each token in the sentence being represented by a padded character vector of size $CMAX$. In the padded character vector each character in the character sequence of the token is represented by the index integer of that character in the alphabet or a special value for the padding character. To create the sentence level label tensor, we start with finding the boundaries of all the longest MWTs in the sentence. For the first word in a MWT, we use the label **B**, for all but the last word in a MWT we use the label **I**, for the last word in a MWT we use the label **L** and for all other words in the sentence we use the label **O**. The tensor for this label sentence is created the same way as the word level tensor of the sentence, only with the dictionary being exchanged for a label dictionary.

3.3 Baseline

Our baseline method is a pattern matching based method that uses the the PoS based linguistic patterns described in Frantzi et al., but does not include the statistical scoring of the C-Value method. We apply the preprocessing pipeline to our dataset until the **tokenization** step, extract token and PoS features using either the NLTK[3] or spaCy library and annotate the ground truth MWT boundaries for each sentence in form of start token index and end token index for each MWT. Then we extract the candidate MWT boundaries — also in the form of start and end token indices — with patter matching using the extracted tokens and PoS features and one of the linguistic patterns from Frantzi et al. [FAM00]. For the extraction we experiment with different linguistic filter and library combinations.

A more complex method might now further filter the candidate MWTs using machine learning based on some additional features, such as the occurrence frequency of a phrase in the corpus or its document frequency, and produce a set of predicted MWTs that is a subset of the candidate MWTs. For this naive method we skip the machine learning phase and consider all candidate MWTs to be predicted MWTs and compare the predicted MWT boundaries to the ground truth MWT boundaries. Only exact matches are counted as TPs, incorrect predictions (i.e. predictions with at least one incorrect start or end token index) are counted as FPs and ground truth MWT boundaries that are not detected by the pattern matcher are counted as FNs. For example, the sentence “The blood cell count.” consists of the token sequence [’the’, ’blood’, ’cell’, ’count’, ’.’]. The ground truth MWT “blood cell count” has the boundary $(start = 1, end = 3)$ in this sentence. If the baseline now returns “blood cell” in form of the boundary $(start = 1, end = 2)$ for this sentence, this equates to one FP (“blood cell” is not in the ground truth) and one FN (“blood cell count” was not found) for this sentence. In our experiments we compare the effectiveness of different linguistic pattern combined with different PoS taggers.

[3] https://www.nltk.org/

Instead of a PoS based linguistic pattern it would also be possible to use a NP chunker, which also makes use of linguistic information. However, upon inspecting the performance of chunking tools, such as the spaCy chunker which uses dependency parsing to analyze a sentence, we found that they were unsuitable since they tend to split long NPs that belong together into two or more NPs when working with patent data.

Due to the small size of our dataset, about 80% of unique MWTs are rare MWTs (occur less than 5 times in the dataset), which makes frequency based features, such as IPC based features (see Section 2.1), unsuitable for differentiating between MWTs and not-MWTs, even with machine learning. This inability of frequency based features to function with low frequency phrases is something we also discovered in previous experiments on the SemEval 2017 Task 10 dataset, where statistical features, such as the statistical part of the C-Value method, combined with machine learning did not produce better results than just classifying all extracted NPs as entities. A potential solution to this problem could be calculating frequency based features with the help of a larger dataset were MWTs appear more often. This would make it is possible to calculate the required statistics for either the C-Value method or the method described by Andersson et al. [ALP+16] for each candidate MWT in the smaller dataset. However, this only avoids the problem of rare MWTs insofar as the number of MWTs that are considered rare is reduced. MWTs that are rare MWTs even in the context of a larger dataset, such as MWTs that refer to a new concept or invention, will still not be detectable, since the features are not usable. Due to the additional requirement of extracting various features from a larger patent corpus for these more complex state-of-the-art MWT detection methods, we consider using these methods as a baseline to be outside the scope of this master thesis.

3.4 Deep Learning Model Architecture

When designing our model solution for the task of detecting MWTs in patent texts, we first investigated recent methods of NER and keyphrase extraction in technical and scientific texts, such as the SemEval 2017 Task 10 challenge [ADR+17], because of the similarity between the task of NER and the task of MWT detection as well as the similarity between the scientific papers domain and the patent domain. We decided that for our task a deep learning solution similar to the solution found by Ammar et al. [APPB17] for the SemEval 2017 Task 10 would likely perform well, due to it being the overall best performing solution in that particular challenge. This method does not require PoS tags and is not dependent on phrase frequencies, making it well suited to detect MWTs in the patent domain, even if they occur infrequently. Since we do not have a large amount of computational resources, we build a smaller model by only implementing the core parts of the original model architecture, using the open-source Tensorflow[4] library by Google. A recurring combination of components in publications

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

dealing with NER and natural language processing are the use of LSTMs combined with CRFs, as well as word representations combined with LSTMs (see Neural Architectures for Named Entity Recognition [LBS+16], Bidirectional LSTM-CRF models for sequence tagging [HXY15], Natural language processing (almost) from scratch [CWB+11]). We consider these components to be the core parts of the model. Other model parts, such as the neural language model described by Ammar et al., are skipped to make training the model faster. Furthermore, we add the PoS tags of the input words as features in some of our experiments. The resulting architecture creates a sequence to sequence model that predicts a BILO-encoded sentence-level sequence of word to label mappings. (Figure 3.2). The architecture consists of word embedding, character CNN, Bi-directional LSTM, scoring and CRF components and is shown in Figure 3.4.

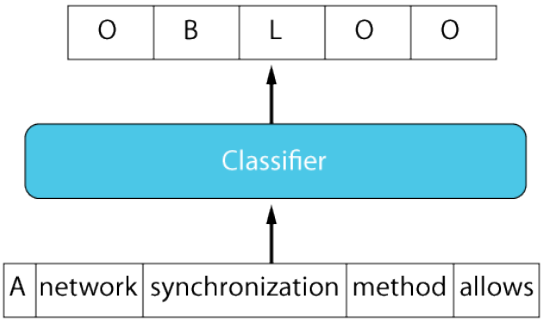


Figure 3.2: A black box view of the MWT classifier.

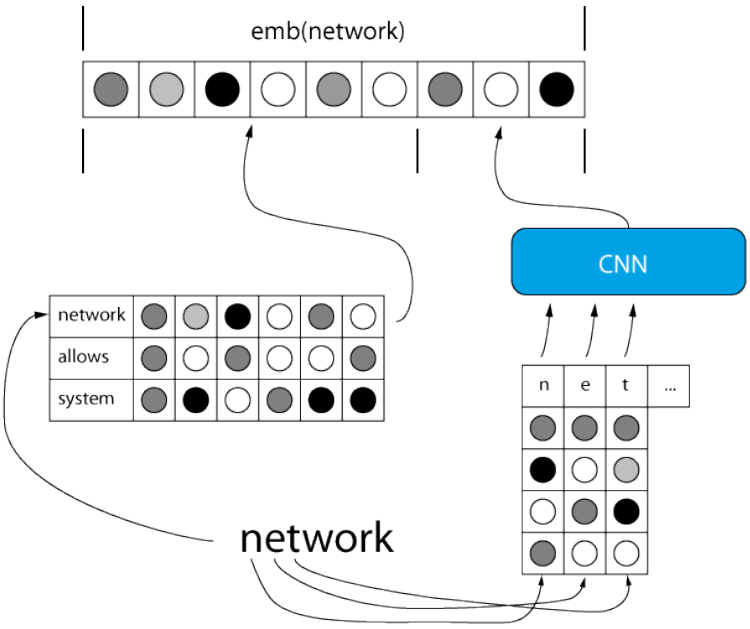


Figure 3.3: The word embedding component. Pre-trained word vectors and character representations are combined to a token vector representation.

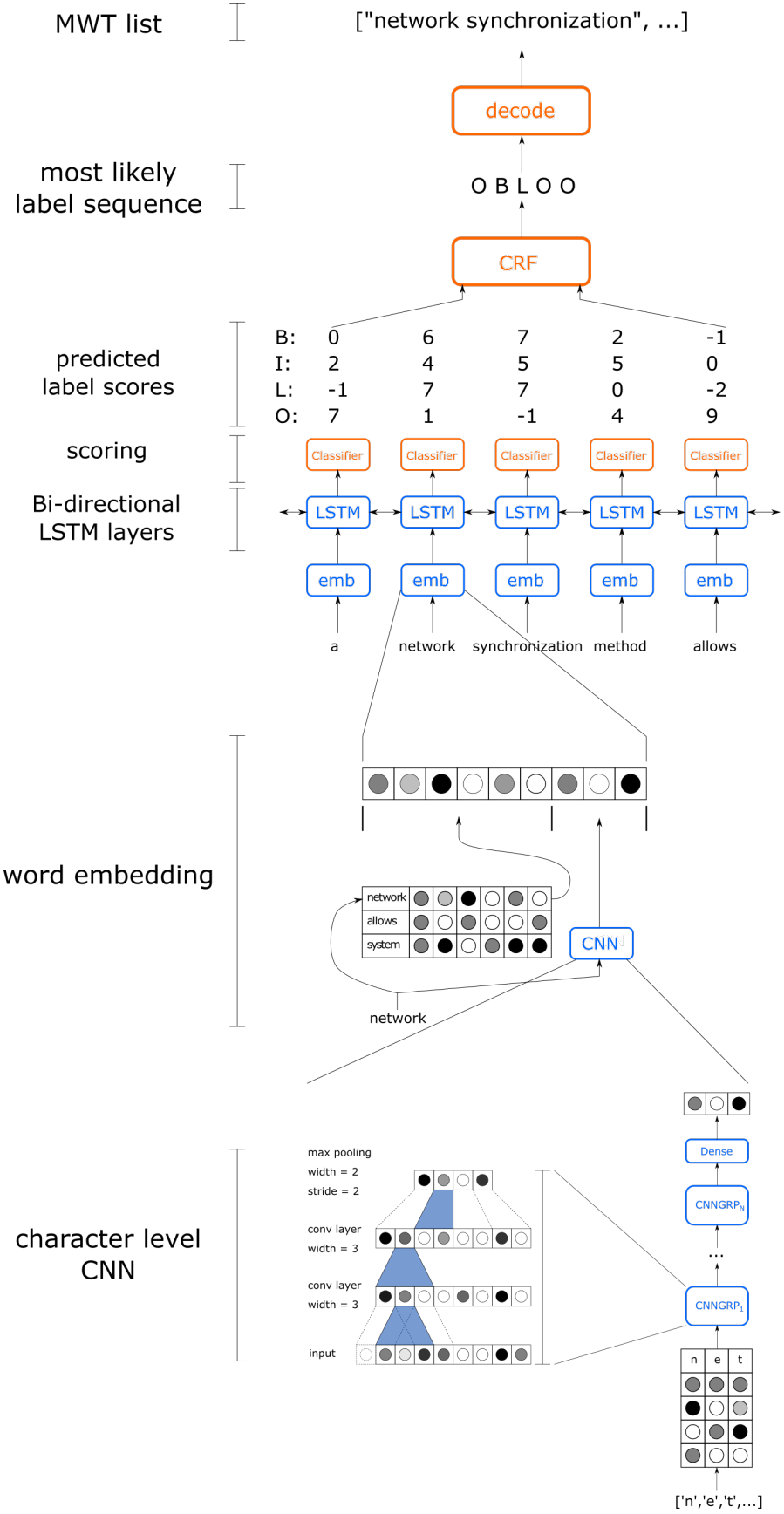


Figure 3.4: The complete architecture of the MWT model.

The first part of the model (Figure 3.3) consists of a word embedding and a character embedding component and converts the padded input sequence $tokens = token_1$,..., $token_{SMAX}$ with maximum length $SMAX$ to a sequence of token vector representations $emb(tokens) = emb(token_1)$, ..., $emb(token_{SMAX})$. Each $token_i$ consisting of a token index $tokenIDX$ and padded character index sequence $charSEQ = charIDX_1$, ..., $charIDX_{CMAX}$ with maximum length $CMAX$. A token vector representation $emb(token_i)$ is a concatenation of a word vector representation part $emb(word_i)$ and a character vector representation part $emb(charSEQ_i)$. The word representation part $emb(word_i)$ is the pre-trained word vector of size D_{WEMB} of $token_i$ if there exists a word vector for that token in the word vector matrix WV. If the token is a token for which we do not have a word vector pre-trained, but the token is in a manually created list of tokens that might still be of interest, the word vector is learned during training of the MWT model. An example for such tokens would be punctuation tokens, such as ',' or '(', which are usually not included in word embeddings but indicate clear boundaries in a sentence that often act as MWT separators. If the token is an unknown token, the <UNK> word vector is used, which is also learned during training of the MWT model. If the token is a padding token, the word vector is set to zero.

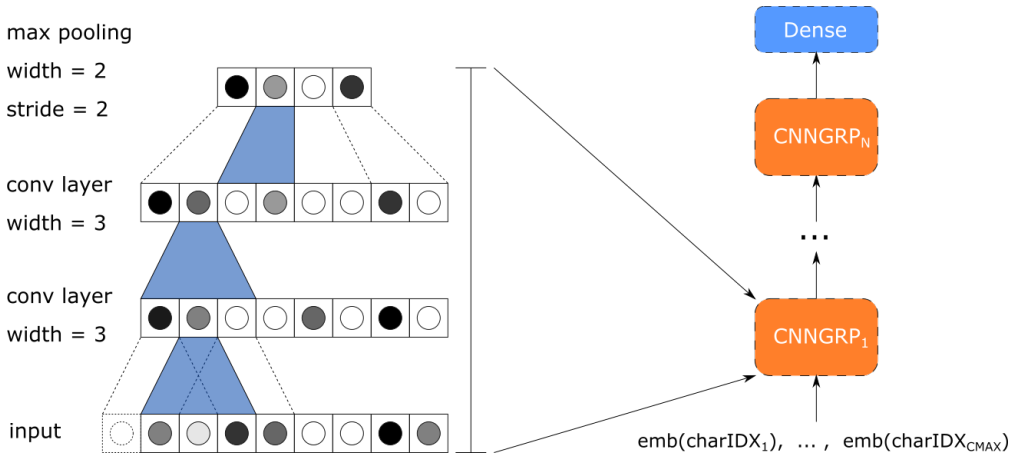


Figure 3.5: Character CNN component to calculate $emb(charSEQ)$ from $charSEQ$.

The character representation part of the token is calculated with a character level CNN. While Ammar et al. [APPB17] also use a character CNN, the description of their CNN is not very detailed, we create our own character CNN architecture based on concepts from Zhang et al. [ZZL15]. Similar to the word representations, an embedding lookup layer converts the character sequence $charSEQ$ into a sequence of vector representations of characters $emb(charIDX_1), ..., emb(charIDX_{CMAX})$, with each vector $emb(charIDX_i)$ having a size D_{CHAR} and each padding character being set to the zero vector of the same size. In contrast to the pre-trained word embeddings, these character embeddings are learned during training. To convert this sequence of individual character embeddings into the character representation part of the token $emb(charSEQ)$ we use the CNN

architecture depicted in Figure 3.5. The CNN consists of multiple stacked layer groups $CNNGRP_1, ..., CNNGRP_N$, each of which consists of two convolutional layers[5] with ReLU activations and a max-pooling layer after the two convolutional layers. While the convolutional layers use a stride of 1 and a width of 3, the pooling layer uses a stride of 2 and a width of 2, resulting in a reduction of the sequence length by half after each layer group $CNNGRP_i$. We employ as many groups as are needed to reduce the sequence to length 1, after which we employ a dense layer with ReLU activation to produce the character representation part of the token $emb(charSEQ)$ of size D_{CEMB}.

The LSTM component (Figure 3.6 BI-directional LSTM layers) converts the individual token representations $emb(token_i)$ from the first part into token representations $lstm_N^{BI}(token_i)$ that are influenced by the past and future tokens of the token embedding sequence $emb(tokens)$. It consists of N Bi-directional LSTM layers, that produce a concatenated output $lstm_j^{BI}(token_i) = [lstm_j^{FW}(token_i); lstm_j^{BW}(token_i)]$ of the forward output vector
$lstm_j^{FW}(token_i)$ of size D_{FW_j} and backward output vector $lstm_j^{BW}(token_i)$ of size D_{BW_j}. The first LSTM layer $lstm_1^{BI}(tokens)$ uses $emb(tokens)$ as input. Some of our experiments also use the PoS tag of a word as additional feature. In that case the one-hot encoded tag vector tag_i of the PoS tag sequence $tags = tag_1, ..., tag_{SMAX}$ is concatenated to the output vector $lstm_1^{BI}(token_i)$.

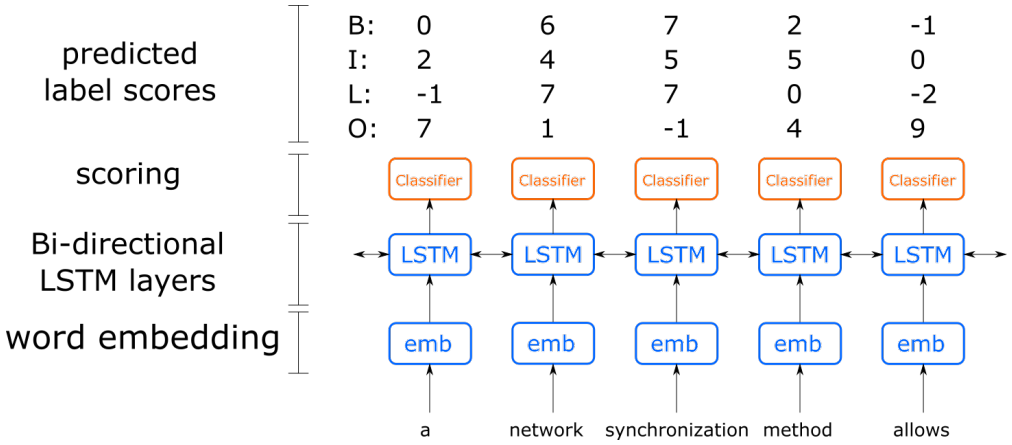


Figure 3.6: Token vector representations are passed through multiple LSTM layers and then produce a score for each label.

The scoring component (Figure 3.6 scoring) converts the LSTM output $lstm_N^{BI}(tokens)$ into a sequence of label score vectors (Figure 3.6 predicted label scores). Each label score vector in the sequence represents a token in the original sentence and contains a score for each label. To calculate the label score vectors, we use a linear output layer preceded by

[5]The use of more than one convolutional layer is common in the image classification domain.

optional dense hidden layers with ReLu activations.

The CRF component (Figure 3.7) takes the sequence of label score vectors, predicts the label sequence and calculates the loss with a cost function (i.e. the prediction error added to a regularization term) during training. This is done with the Tensorflow CRF module, which includes a CRF log-likelihood cost function to compute the loss, trainable parameters and the Viterbi decoding algorithm to compute the most likely label sequence. The predicted label sequence is converted to a prediction of MWT boundaries, which are then compared to the ground truth MWT boundaries.

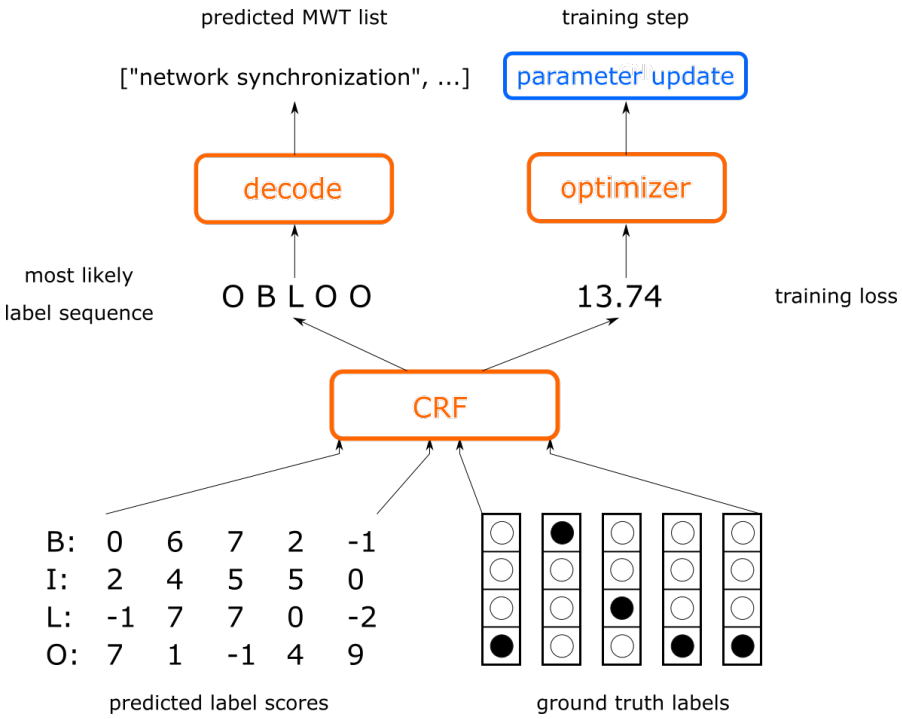


Figure 3.7: Predicted label scores are used to predict the most likely label sequence with the CRF. During training, they are compared to the ground truth labels to calculate the loss. The true label for word is indicated by the black dot. With the loss, the optimizer calculates parameter updates during each training step.

To update the model parameters, such as weights and biases, during training, the loss is passed to the optimizer, which calculates the gradient. The gradient is then further used by the optimizer to calculate the the exact values by which the model parameters should be modified. To combat the exploding gradient problem, we clip the gradient. Once the weights have been updated, the next batch of samples may be processed.

3.5 Experimentation Overview

Since we want to know what impact hyperparameter changes have and how different parts of the model and selection of resources affect the model performance, we run various experiments with different setups. We split our dataset into training, validation and test sets that each contain approximately 80%, 10% and 10% of patent documents, respectively, which results in 18 patents for training, 2 for validation and 2 for testing. Unfortunately we cannot use k-fold cross validation as our validation strategy because of the long time it takes to train the model on our hardware. Instead, we perform splitting our dataset two times, with the resulting splits further being referenced as **A-Split** and **B-Split**. Although the number of sentence samples per patent has a high variance, we choose to split based on patent documents and not individual samples to make sure the model is validated on previously not known MWTs. All experiments that are performed on the A-split are also reproduced on the B-split and vice versa. With our experiments we aim to gain insights into the impact of

- standard regularization techniques that can be applied to most deep learning models,
- the custom character CNN component,
- different hyperparameter choices for the custom character CNN component and scoring component,
- additional PoS tag features extracted by spaCy,
- using Wikipedia or patent data word embeddings,
- different training set sizes,

We use the same method of evaluation as in the baseline method, i.e. an exact match of start and end indices of the predicted MWT is counted as TP, a predicted MWT boundary where either the start or the end index does not exactly match the respective index of the ground truth MWT counterpart is counted as FP and any ground truth MWT boundaries that have no predicted MWT counterpart are counted as FN. The accuracy of the model is computed based on the number of correct label predictions, while precision, recall and F1 score are calculated based on the number of correctly predicted boundaries MWTs. The performance of the deep learning models as well as the baseline models is measured based on precision, recall and F1 score.

Further, we take a deeper look at the best performing models (based on their F1 score) concerning common prediction errors, such as not detecting certain MWTs or falsely predicting certain non-MWT phrases to be MWTs. Further, we also investigate their ability to detect rare MWTs compared to the baseline method, by comparing the rare MWT recall between them and the baseline method on the test and validation patents.

The rare MWT recall is defined as the recall of the MWTs that appear less than 5 times in the entire dataset.

3.6 Summary

We presented a method inspired by methods for NER and keyphrase detection that is capable of detecting MWTs in patent texts that does not rely on PoS tags or MWT occurrence frequencies, which allows models created by this method to extract even rare patent MWTs despite a worse performance of PoS taggers in the patent domain. We discussed how we create a dataset for training these models and according to which criteria we annotate phrases as MWTs. We presented how our baseline method extracts MWTs and shortly discussed the problems of frequency based features concerning our dataset. Further, we presented the ANN architecture of our method and described word embedding, character CNN, Bi-directional LSTM, scoring and CRF components as well as what preprocessing steps are performed to convert patent texts and and ground truth MWT annotations into labels and input vectors. Finally, we presented what experiments are performed, how we evaluate the performance of these experiments when compared to our baseline method.

CHAPTER 4

Results & Discussion

In this chapter we present in-depth descriptions of our experiments, explain our hyper-parameter choices and discuss the results and implications of our experiments. While training and validation datasets are used during model creation, the final result is based on the model's performance on the test set. Since a test set only consists of two patents and thus the impact of a single patent on the model performance is very high, we create two different splits of data used during model creation and testing to get a better understanding of the model's overall performance and consistency. Furthermore, we take the best models produced by the method that scores the highest F1 score on the test set and analyze the results of their validation and test set predictions in greater detail.

4.1 Created Dataset

Annotating the MWTs of 22 patents took around 4 weeks. While most of the patents were either short or of medium length, a few patents were fairly long, with the longest patent having a length of 70 pages in the original PDF document. The total dataset consists of around 10000 sentences with average length of 22.45 tokens but also a huge standard deviation of 21.84 tokens (Table 4.1). In total about 19000 MWT instances consisting of around 5000 unique MWTs were identified (not including nested occurrences). Almost 80% of unique MWTs occur less than 5 times in all patents combined. On average a MWT would occur only 5.12 times (including nested occurrences). Compared to that the MWT occurrence standard deviation of 12.84 is very high. Among all unique MWTs, the average token length is 2.70 tokens with a standard deviation of 1.12 tokens. As mentioned before, the dataset is split into training, test and validation sets in two different ways. Both the **A-Split** and the **B-Split** consist of 18 patents for training, 2 patents for validation and 2 patents for testing and their sentence distribution is shown in Table 4.2.

Table 4.1: Dataset Statistics

Number of Tokens	232065	MWTs occurring exactly once	2353
Number of Sentences	10337	MWTs occurring at least 5 times	1049
Sentence Length Mean	22.45	MWT Occurrence Mean	5.12
Sentence Length Std. Dev.	21.84	MWT Occurrence Std. Dev.	12.84
Number of MWTs	19465	Unique MWT Length Mean	2.70
Number of Unique MWTs	5099	Unique MWT Length Std. Dev.	1.12

Table 4.2: Dataset Split Sentence Count

	A-Split	B-Split
Training Set	8996	9441
Validation Set	950	313
Test Set	391	583

Table 4.3: MWT Detection Baseline Results

Models	A-Split			B-Split		
	Precision	Recall	F1	Precision	Recall	F1
AdjNoun+NLTK	0.60	0.64	0.62	0.41	0.64	0.50
AdjNoun+spaCy	**0.70**	0.74	**0.72**	**0.53**	**0.70**	**0.60**
Prep+NLTK	0.29	0.63	0.40	0.22	0.61	0.33
Prep+spaCy	0.35	**0.75**	0.47	0.26	0.66	0.37

4.2 Baseline Models

To create our linguistic baseline model, we need to pick values for two hyperparameters: the linguistic pattern, which needs to be capable of extracting specific NPs that have boundaries that match the desired MWT boundaries, and the PoS tagger / tokenizer, which needs to provide the correct PoS tags and token splits. For our task, we assume that the two linguistic patterns $(ADJ|NOUN) + NOUN$ and $((ADJ|NOUN) + |((ADJ|NOUN) * (NOUN\ PREP)?)(ADJ|NOUN)*)NOUN$ (see [FAM00]) are able to extract most important MWTs from the text. The first pattern just considers combinations of adjectives and nouns, ending with nouns as predicted MWTs and will further be called **AdjNoun**. The second pattern also allows for prepositions to be part of the predicted MWTs and will further be called **Prep**. To obtain the necessary PoS tags, we use the default PoS tagger of the NLTK library `nltk.pos_tag` and the `en_core_web_md` model[1] of the spaCy library. Table 4.3 shows the results on the A-Split and B-Split test sets for all hyperparameter combinations. The best performance overall is achieved with the **AdjNoun+spaCy** combination. Since this algorithm is very simple and just filters the text for MWTs based on PoS tags without machine

[1] `https://spacy.io/models/en#en_core_web_md`

learning, we expected a high recall and a low precision. However, the precision of the **AdjNoun+spaCy** baseline was higher than expected, going up to 0.70 on the A-Split test set. With a recall of 0.74 on the A-Split, the baseline detects around 3 out of 4 MWTs. Using the **Prep** pattern, the A (B) Split precision is reduced to 0.35 (0.26), while the recall changes to 0.75 (0.66). Since PREP were rarely included during our MWT annotation of the dataset, it is not surprising that using **Prep** pattern results in a worse precision. However, an equal or worse recall can only be explained if we assume that the pattern not only does not detect any MWTs containing PREPs but also misses some MWTs compared to the **AdjNoun** pattern, due to it extracting longer phrases and our baseline requiring exact matches of start and end indices of the predicted MWT with the ground truth MWT to be considered a TP.

While we expected the recall of the best baseline to be higher, the precision is surprisingly high. The high precision could indicate that the MWT annotation of the dataset is perhaps biased towards the **AdjNoun** pattern. Due to a lack of expert knowledge and IR in the patent domain being more recall oriented, we set the termhoodness threshold needed for a particular NP to be considered a MWT low, as we considered it more important to assure a high number of MWTs in our dataset so that as few as possible MWTs are missed during patent IR.

Another noteworthy aspect is that spaCy consistently outperforms NLTK across all experiments. The used spaCy model is trained on **OntoNotes**[2] and **Common Crawl**[3], covering a broad spectrum of domains which could explain why it performs better on our data than the default NLTK tagger[4].

4.3 Deep Learning Experiments

The choice of hyperparameters and output vector sizes for our deep learning architecture (see Section 3.4) is based on common hyperparameter settings from papers that describe methods that our method is based on, such as Ammar et al. [APPB17], as well the requirements of our own data. We use spaCy for sentence splitting, tokenization and PoS tagging (for some experiments) in our preprocessing pipeline. While the mean sentence length is only about 22 tokens, the standard deviation of 21 is large in comparison and a deeper look into our data reveals that some sentences, especially in the abstract and claim sections of the patents, are well over 50 tokens long and contain many important MWTs that we would like our models to detect. While it is possible to split sentences, the current solution of splitting them at a certain maximum length into two or more samples is suboptimal, since with the current algorithm no information can be transferred between the split samples, making it harder to detect MWTs, especially in the area

[2] `https://www.ldc.upenn.edu/`
[3] `http://commoncrawl.org/`
[4] Implementation details: `https://explosion.ai/blog/part-of-speech-pos-tagger-in-python`

surrounding the split. Further, if the sentence is split in the middle of a MWT, the MWT is removed from the samples, which is also not preferred. For these reasons and to cover most sentences in full length, we set the maximum sentence length $SMAX$ to 75. The maximum number of characters of a word $CMAX$ is set to 32, which causes the character CNN component to use a stack of 5 layer groups that each half the sequence length of the character representation. A $CMAX$ of 16 was too short for many tokens and while there are still a few tokens consisting of more than 32 characters, we considered an incomplete character representation of these tokens to be acceptable, due to their rarity. We use a character vector size $D_{CHAR} = 21$, which was also used by Ammar et al. [APPB17]. The output vector sizes of $CNNGRP_1, ..., CNNGRP_5$ are $[32, 64, 128, 256, 256]$, meaning they double with every layer group (the last group is kept the same as the previous group because of memory limitations). The dense layer providing the embedding output for the character CNN component is first set to an output size $D_{CEMB} = 50$.

The word embedding component uses pre-trained word embeddings of size $D_{WEMB} = 300$. These word embeddings are Skipgram word embeddings that have been trained on the CLEF-IP patent dataset with **gensim**[5]. CLEF-IP[6] consists of around 2.5 million patent documents (more than 1 million patents). The word embedding dictionary contains around 700,000 words and each word is represented by a word vector of size 300.

The number of Bi-directional LSTM layers is set to 2 and the output vector size of the forward and backward sub-layers is set to 350. We only use a linear scoring layer to calculate the output scores for each class from the last LSTM layer output $lstm_2^{BI}(tokens)$. The gradient is clipped if its norm exceeds 5.0. The learning rate (LR) at the start of training is set 0.01 and is decayed by 0.96 every 200 training steps, each step processing a mini-batch of 32 samples. Further we employ early stopping during training by checking the model performance on the validation set based on the F1 score after every epoch and stop training if no improvement was detected for 20 epochs. All biases are initialized to a zero tensor and all weights are initialized with the random normal initializer that generates a tensor with a normal distribution with a mean of zero and a standard deviation of one.

Using this setup, training on the A (B) split resulted in a MWT detection performance shown in Table 4.6 **NoReg**, with one epoch of training taking around 600 seconds on our hardware (Table 4.5). However, the experiments had to be repeated a few times with different initializations since in some cases the performance on the validation and even the training set would not exceed an F1 of zero. With this setup, training seems to be very dependent on its starting parameters and the model tends to overfit immensely (precision, recall and F1 go up to 1.0) on the training data if the evaluation measures manage to go above 0.0. Figure 4.1 depicts the course of training over time (measured in training epochs) with regards to F1 score. Although training stops after 20 epochs of no

[5]We thank Navid Rekabsaz for providing us with this data. Also see [RLHZ16].

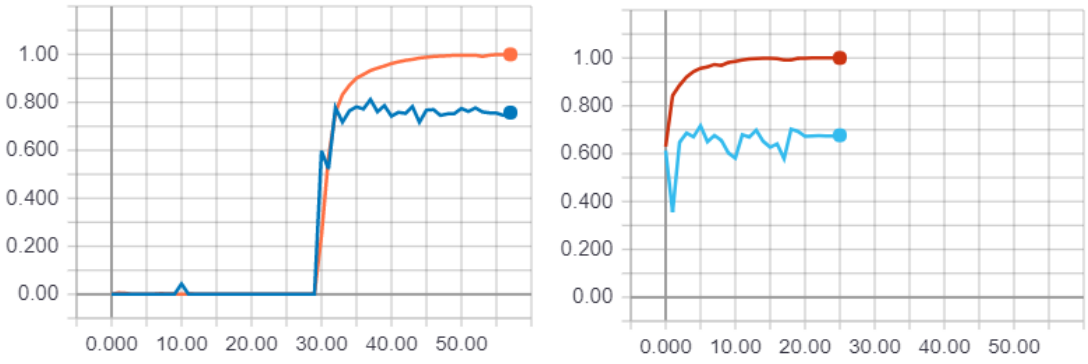
[6]`http://www.ifs.tuwien.ac.at/~clef-ip/download-central.shtml`

Table 4.4: Hyperparameter Settings for the Reg-Experiment and Changes

Hyperparameter	Experiment				
	Reg	No-Reg	PoS	CNN-100	HiddenLayer
$SMAX$	75				
$CMAX$	32				
D_{CHAR}	21				
$CNNGRP_1$	32				
$CNNGRP_2$	64				
$CNNGRP_3$	128				
$CNNGRP_4$	256				
$CNNGRP_5$	256				
D_{CEMB}	50			100	
D_{WEMB}	300				
LSTM layer count	2				
LSTM output size	350				
scoring hidden layer	False				True
scoring hidden layer size					300
Gradient Norm clipping	5				
initial LR	0.01				
LR decay	0.96				
LR decay steps	200				
L2-coefficient	0.01	0.00			
Mini-batch size	32				
Early stopping epochs	20				
PoS Features	False		True		

Table 4.5: Used Hardware

Lenovo(TM) ideapad(TM) 310	
CPU	Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz, 2 Core(s)
RAM	12.0 GB
GPU	NVIDIA GeForce 920MX
VRAM	2.0 GB



(a) A-Split (b) B-Split

Figure 4.1: Training (top/red lines) and Validation (bottom/blue lines) F1 scores during training. The abscissa shows the number of training epochs, whereas the ordinate shows the resulting F1 score at that epoch.

improvement, the model trained on A-Split manages to exceed this limit on Figure 4.1a while remaining at a F1 score performance of zero due to a random performance spike at epoch 10. Actual improvements could only be measured onwards from epoch 30.

To reduce the influence of the initialization and prevent the model from overfitting, we add L2 regularization with a L2-coefficient of 0.01 to the loss. Further we apply a dropout of 0.5 during training to the output vectors of both LSTM layers. With these new settings, the experiment was repeated four times without the performance of any model instance ever staying at zero for extended periods of time. The performance of these four experiment repetitions is shown in Table 4.6 **Reg 1-4**, sorted by F1 score. As for the A-Split, the F1 score reaches around 0.79 to 0.80 for 3 of the repetitions, but the best repetition goes up to 0.84. As for the B-Split, the F1 score ranges from 0.65 to 0.71, but appears to be more evenly distributed between these two values compared to the A-Split.

One hypothesis we wanted to test was whether adding PoS tags as features would affect the performance in any way, even if the PoS tagger providing these tags was not trained on the patent domain. For this reason, we also used spaCy to extract PoS features for each token of the sample sentences and converted these tokens to binary features using one-hot encoding. These PoS features were token-wise concatenated to the output of the first LSTM layer $lstm_1^{BI}$ to be input into the second LSTM layer $lstm_2^{BI}$. One way to add the PoS features to the model would be adding them directly to the input of the scoring component, allowing the PoS tag of a token to directly influence the BILOU label scores of that token. We decided against that and added them to the input of an LSTM layer because we wanted to make it possible for the model to learn potential

relations between PoS tags in a sentence. The PoS models achieved the highest recall of all models on the A-Split, but not the highest F1 score and did perform worse on the B-Split (see Table 4.6 **PoS**). However, the Reg experiments have shown that the F1 score of an experiment can vary by up to 0.06 and the F1 score of this experiment is not higher or lower than the best or worst Reg models, meaning the use of PoS features probably does not substantially affect the model performance.

Since the character CNN component used by Ammar et al. in [APPB17] was not closer specified in the paper, we used a different, custom character CNN component. As such, it was not certain if the D_{CEMB} size of 50 as used by Ammar et al. was still best to use. We hypothesized that the character representation of a token, as learned by the character CNN component, needs a different size to correctly represent the token in a way so it is possible for the network to differentiate between MWTs and non-MWTs. To test if an increased capacity had an effect on the performance of the model, we changed D_{CEMB} to 100. As can be seen in Table 4.6 **CNN-100**, the models created by the experiment perform about as well as the other "Hyperparameter Changes" and "Regularization" models, meaning the increased capacity did not have an effect.

One consideration regarding the scoring component was that it might lack the capacity to differentiate between the output labels seeing as it only consists of a linear layer. We hypothesized that with a higher capacity scoring component the model might be able to learn what labels correlate with what types of tokens more accurately, giving these labels a higher score. For this reason, we added a hidden dense layer with ReLU activation and an output size of 300 (chosen arbitrarily) between the last LSTM layer output $lstm_2^{BI}(tokens)$ and the linear scoring layer. Table 4.6 **HiddenLayer** shows that the model performance did not change substantially compared to the previous experiments.

Another set of experiments shown in Table 4.6 "Limited Resources" had the purpose of observing the negative impact of leaving out or substituting key resources and components. While there are many tools and resources available for general NLP, only few come into contact with patent data during their creation. One such resource are word vectors, which are often trained on non-technical texts, such as news articles or semi-technical texts, such as Wikipedia articles. In the **Wikipedia** experiment we train a model based on our regularized architecture after substituting the CLEF-IP patent word vectors with Wikipedia word vectors[7]. For the A-Split model precision, recall and F1 score all lie considerably below the previous A-Split experiments. As for the B-Split model, the recall is only 0.59 compared to the previous worst recall of 0.65, but with 0.68 compared to 0.63 the precision is slightly higher than the worst B-Split precision. Compared to the results of this experiment, the **Reg 1** A-Split F1 score is 17% higher and the **Reg 1** B-Split F1 score is 13% higher. Overall, the results are only on about the same level as

[7]We again thank Navid Rekabsaz for providing us with this data.

Table 4.6: Performances of all experiments

Models	A-Split			B-Split		
	Precision	Recall	F1	Precision	Recall	F1
Baseline						
AdjNoun+NLTK	0.60	0.64	0.62	0.41	0.64	0.50
AdjNoun+spaCy	0.70	0.74	0.72	0.53	0.70	0.60
Prep+NLTK	0.29	0.63	0.40	0.22	0.61	0.33
Prep+spaCy	0.35	0.75	0.47	0.26	0.66	0.37
Regularization						
NoReg	0.82	0.79	0.81	0.69	0.70	0.69
Reg 1	**0.85**	0.84	**0.84**	**0.73**	0.68	**0.71**
Reg 2	0.81	0.80	0.80	0.68	0.70	0.69
Reg 3	0.79	0.80	0.79	0.68	0.65	0.67
Reg 4	0.79	0.78	0.79	0.63	0.67	0.65
Hyperparameter Changes						
PoS	0.75	**0.88**	0.81	0.65	0.65	0.65
CNN-100	0.84	0.73	0.78	0.70	**0.72**	**0.71**
HiddenLayer	0.81	0.85	0.83	0.68	0.69	0.69
Limited Resources						
Wikipedia	0.73	0.71	0.72	0.68	0.59	0.63
No-CNN	0.73	0.72	0.72	0.61	0.57	0.59
Half-TS	0.67	0.59	0.63	0.65	0.45	0.53

the pattern-matching baseline method.

Further, we dedicated the **No-CNN** experiment to test the performance of the custom character CNN component by again taking the regularized model architecture and removing the custom character CNN component. This means that only the pre-trained (patent) word embeddings are used as model input. The results are similar to the **Wikipedia** experiment, only that in this case no performance measure is better than in the previous experiments. Compared to the results of this experiment, the **Reg 1** A-Split F1 score is 17% higher and the **Reg 1** B-Split F1 score is 20% higher. It seems that without learning a character representation, the model is missing important information when representing the meaning of an input token, even if word embeddings are available. Again, the results are only on about the same level as the pattern-matching baseline method.

While using more training data generally improves the performance of machine learning models (as long as it is representative of the data source), the effectiveness of additional training data is often non-linear, i.e. the more data is already available for training the more data needs to be added to increase the model performance further. Goodfellow et al. [GBC16] suggest experimenting with training set sizes on a logarithmic scale, such

as doubling the number of examples between experiments, to measure the impact that different training set sizes have on the model performance with respect to generalization. To test the impact of changes to the available training data for the task of detecting MWTs in patents, we train our model with 9 patents instead of the available 18. We hypothesize that due to the small size of our dataset, differences in the training set size outweigh other changes to hyperparameters. Since in this scenario one training epoch consists of only half as many steps we tried to compensate the reduced training time by increasing the early stopping limit to 40 epochs from 20 epochs. Table 4.6 **Half-TS** shows the results of this experiment. The F1 scores of this experiment are similar to the F1 scores of the **AdjNoun+NLTK** baseline experiment, with precision being slightly higher and recall being slightly lower. In contrast, the performance of the model is substantially higher on the training data. During training the A (B) Split model achieves an F1 score of 0.84 (0.92), a precision of 0.83 (0.92) and a recall of 0.85 (0.92), which is very similar to the training set performance of all other models. For example, the **Reg-1** model achieves a similar performance during training with an F1 score of 0.89 (0.92), a precision of 0.88 (0.92), and a recall of 0.89 (0.92) after 48 (62) epochs. The difference in F1 score between training set and test set for the A (B) Split model of the **Half-TS** experiment is 0.20 (0.39) compared to 0.05 (0.21) of the **Reg-1** experiment. Due to this large difference in generalization performance based on dataset size and the ineffectiveness of hyperparameter changes during our other experiments, we assume that doubling the training set size to 36 patents would improve test set performance more than further hyperparameter tuning.

4.4 Analysis of Model Behavior

So far we have only looked at general performance metrics to evaluate and understand our method, but to gain the necessary information needed to further improve the method, we need to investigate how exactly our trained models behave. While the model parameters of the trained model store all the data needed by the model to make its predictions, this data consists of only high dimensional tensors, whose meanings are difficult to interpret. Although interpreting a tensor is not impossible and informative visualizations via projection into a lower dimensional space have been made for network data such as word vectors (see [MSC+13]), bigger networks consisting of multiple layers and a multitude of tensors are more difficult to interpret in such a way, because of the more complex tensor interactions and abstract meaning of tensors near the output of the model. Instead we take the two best models (**Reg-Best-F1** experiment), inspect the output on their respective validation plus test set patents (i.e. all patents the model is not directly trained on) and discuss their prediction quality as well as possible causes for this behavior on the basis of a number of representative examples from the prediction that highlight prediction errors. Further, we take a look at the ability of these two models to detect rare MWTs by calculating their rare MWT recall and compare their performance to the baseline rare MWT recall on the same patents.

Table 4.7: Example Sentences A-Split

ID	Sentence
1	A distributed system with a timing signal path (22, 90-94, 220) for increased precision in time synchronization among distributed system clocks.
2	[0003] Distributed systems commonly benefit from precise control of the timing at the distributed nodes.
3	8. A system as in claim 7, wherein the continuous frequency signal includes a distinguished pattern which is aligned to the time event.
4	The nodes that contain the master clock 10 and the slave clocks 12-14 may include hardware/software elements that perform application-specific functions associated with the distributed system 30 as well as hardware/elements for clock signal generation and time synchronization.
5	A timing packet recognizer (TPR) 124 and a time-stamp latch 126 are used to detect and time stamp timing packets received via the network 20.
6	Of these, 5-vinyl-2-norbornene, 5-methylene-2-norbornene, 5-(2-propenyl)-2-norbornene, 5-(3-butenyl)-2-norbornene, 5-(4-pentenyl)-2-norbornene, 5-(5-hexenyl)-2-norbornene, 5-(6-heptenyl)-2-norbornene and 5-(7-octenyl)-2-norbornene are preferred.
7	The cam sleeve and the knock sleeve are linked to each other through a recess elsewhere than the through hole in the engaging wall.

4.4.1 Reg-Best-F1 A-Split

During training and testing the A-Split model was evaluated on four patents: EP 1324520 A2 and EP 1669402 A1 for validation and EP 1295985 A1 and EP 1600304 A2 for testing. The first validation set patent EP 1324520 A2 has the title “Synchronization in a distributed system” and, according to its IPC codes, belongs to the fields ’Horology’, ’Computing / Calculating / Counting’ and ’Electric Communication Technique’. It consists of 142 sentences with an average length of 26.35 word tokens per sentence. For this patent, the recall for rare MWTs (occur less than 5 times in the entire dataset) of this model is 0.70. In comparison, the baseline (**AdjNoun+spaCy**) rare MWT recall for this patent is 0.63. In the first sentence of the abstract (Table 4.7/1) the ground truth consists of the MWTs ’distributed system’, ’timing signal path’, ’time synchronization’ and ’distributed system clocks’. However, while ’system clocks’ - a MWT but not part of the ground truth of this sentence - is incorrectly identified, ’distributed system’ and ’distributed system clocks’ are not identified by the model. The other two MWTs are correctly identified. The word ’distributed’ also appears in other MWTs, such as ’distributed nodes’ in sentence 4.7/2, which is also not detected by the model. With ’precise control’ the model instead produces a FP in the sentence, which is a phrase we consider to be too broad and unspecific to be considered a MWT in our context. Although there is a second patent that belongs to the fields ’Horology’, ’Computing / Calculating / Counting’ and ’Electric Communication Technique’, the word ’distributed’ is not used in that patent and,

while used as an adjective in a few sentences, is not used as part of MWTs in any other patent in the collection either. The word vector of 'distributed' does not appear to be similar to word vectors that the model associates with MWTs and is most similar (cosine similarity) to the word vectors of 'distribute' (0.6789), 'evenly' (0.6781) and 'distributing' (0.6452). While 'evenly' most likely co-occurred with 'distributed' in the word embedding training data in the form of 'evenly distributed', 'distribute' and 'distributing' are most likely only similar to 'distributed' because these tokens often substitute 'distributed'. As such, it is likely that the model only learned that 'distributed' is not associated with MWTs from the few examples that it had, thus leading 'distributed' to be excluded from any of the model's predictions.

Another interesting example is sentence 4.7/3, for which we originally concluded that the phrases 'frequency signal' and 'time event' are MWTs. The model identified 'continuous frequency signal' and 'distinguished pattern' as MWTs. While 'distinguished pattern' is a NP, we also considered it too broad to be usable for identifying patents. However, after revision we concluded that 'continuous frequency signal' is also a MWT and should be preferred over 'frequency signal' in this sentence. The MWT 'time event', which we consider borderline with regards to broadness and termhoodness, was not detected. In sentence 4.7/4 we encounter the phrase 'hardware/elements' that we assume to be an oversight by the author and that should instead read 'hardware/software elements', which is the reason why we did not mark this phrase as a MWT. Nonetheless, the model detects 'hardware/elements' as a MWT, seemingly ignoring the '/' token in the middle of the phrase. Further, the phrase 'hardware/software elements', which is also spelled out earlier in the sentence, was difficult to assess, since it actually consists of two distinct terms, 'hardware elements' and 'software elements', with 'hardware elements' being implied in the text, but not actually occurring as a token sequence in the text. For this reason we did not annotate 'hardware/software elements' as MWT and only annotated 'software elements', which was correctly identified. Having both terms 'hardware elements' and 'software elements' would be preferable, but is unfortunately not possible with our algorithm alone. An extra preprocessing step would be needed to handle such **merged MWTs** and explicitly convert the text 'hardware/software elements' into a text containing both MWTs, such as 'hardware elements / software elements' or 'hardware elements and software elements'. The current annotation approach of marking the MWT token sequence in the text would also need to be expanded to allow for adding non-consecutive phrases to the ground truth. This would also make it possible to handle **split MWTs** contained in phrases, such as 'low impedance (target only) load' (not part of this patent), were the correct MWT would be 'low impedance load'.

Sentence 4.7/5 highlights an issue regarding detection of verbs that has its roots in our chosen methodology. In this sentence, the phrase 'time stamp' is used as a verb phrase 'to time stamp' and should not be considered a MWT in this sentence. However, 'time stamp' is also used as a MWT in different sentences and since our script that converts

our annotation into labeled sentences has no PoS information available when generating the ground truth from the text, 'time stamp' is incorrectly annotated as ground truth in this sentence. The correct ground truth would be 'timing packets'. The model does not differentiate between the verb phrase form and noun phrase form of 'time stamp' and incorrectly identifies 'time stamp timing packets' as MWT.

The second patent of the validation set is EP 1669402 A1 with the title "Rubber composition for cable connector seals", which belongs to the fields 'Organic Macromolecular Compounds', 'Dyes / Paints', 'Information Storage' and 'Basic Electric Elements'. It consists of 808 sentences with an average length of 23.88 word tokens per sentence. For this patent, the recall for rare MWTs of this model is 0.72. In comparison, the baseline rare MWT recall for this patent is 0.64. The patent is notable for containing long enumerations of chemical substances such as sentence 4.7/6. In this sentence the model only detects the MWTs '5-vinyl-2-norbornene' and '5-methylene-2-norbornene', while missing the remaining chemical substances in the sentence, such as 5-(2-propenyl)-2-norbornene. This is likely due to the use of brackets within the MWTs, while in most other patents brackets are not contained in MWTs, because they do not describe any chemical substances. To handle such substances correctly, the model needs to be trained on training data that contains more examples of chemical substances written this way. If multiple fields are supposed to be handled, we expect adding the field (or a representation of the field) as an additional feature could allow the model to better detect MWTs despite different writing styles in different fields.

The first patent of the test set is EP 1295985 A1 with the title "Method for making coated metallic cord", which belongs to the fields 'Mechanical Metal-Working', 'Vehicles in General', 'Coating Metallic Material' and 'Ropes / Cables other than Electric'. It consists of 228 sentences with an average length of 20.79 word tokens per sentence. For this patent, the recall for rare MWTs of this model is 0.74. In comparison, the baseline rare MWT recall for this patent is 0.81. Already in the title the model misses the MWT 'coated metallic cord' and instead suggests 'making coated metallic' to be more likely. Due to the fact that the model is unable to correctly identify any MWT containing the token 'cord' while correctly identifying MWTs such as 'metallic wire', we assume that the model was not able to classify 'cord' as sufficiently technical from the training data. Furthermore, the first token of 'making coated metallic' - 'making' - is a verb and the last token - 'metallic' - is an adjective, which disqualifies the phrase from a PoS point of view. However, an '-ing' suffix could also indicate that a token is a noun, which is generally more common in patent texts, which could explain the inclusion of 'making'. While the '-ic' suffix is common with adjectives, not all tokens with the '-ic' suffix are always used as adjectives with words such as 'dynamic', 'ceramic' or 'narcotic' being usable as both adjectives and nouns.

The second patent of the test set is EP 1600304 A2 with the title "Composite writing tool", which belongs to the field 'Writing or Drawing Implements'. It consists of 163

Table 4.8: Example Sentences B-Split

ID	Sentence
1	[0005] The present invention is intended to create an arrangement for activating and deactivating automatic noise cancellation when it is required.
2	[0052] The various thermal barrier coatings set forth herein may be characterized with a columnar structure.
3	[0002] Printed wiring boards are extensively used in the areas where any electric connection is needed including electronic devices.

sentences with an average length of 38.42 word tokens per sentence. For this patent, the recall for rare MWTs of this model is 0.80. In comparison, the baseline rare MWT recall for this patent is 0.76. Notable are the frequent use of MWTs that describe parts of an artifact based on their relative location, such as 'rear shaft' and 'intermediate shaft', which could be considered borderline non-MWTs. While the model did find most of these MWTs, it also included phrases such as 'rear part', which we did not include. In the sentence 4.7/7 the model misses the MWT 'through hole' which could be explained by the atypical usage of the word 'through'. Further, with 'recess elsewhere' it produces a FP, which could perhaps be explained by the rare usage of the word 'elsewhere' that does not occur in the training set at all. The most similar word vector to 'elsewhere' is 'wherever' with a cosine similarity of 0.60, making it a word that is fairly dissimilar to other words.

4.4.2 Reg-Best-F1 B-Split

During training and testing the B-Split model was evaluated on four patents: EP 1246167 A1 and EP 1400610 A1 for validation and EP 1237134 A2 and EP 1270193 A1 for testing. The first validation set patent EP 1246167 A1 has the title "Arrangement for de-activating automatic noise cancellation in a mobile station" and belongs to the field 'Musical Instruments / Acoustics'. It consists of 83 sentences with an average length of 31.90 word tokens per sentence. For this patent, the recall for rare MWTs of this model is 0.57. In comparison, the baseline rare MWT recall for this patent is 0.51. As in the previous model, verbs that are misinterpreted as adjectives cause the model to make mistakes. For instance, in sentence 4.8/1 'deactivating automatic noise cancellation' is predicted, while the correct MWT is 'automatic noise cancellation'. Other errors in the prediction suggest that the model predicted the correct phrase and that the annotation is incorrect, such as the correct prediction of 'receiver volume level control value', while the ground truth MWT is only 'volume level control'. Furthermore, the hard-to-evaluate word 'set' is used frequently in the patent, which causes some issues that the model is not fit to handle. While the MWT 'set noise level' should be detected, the phrase 'set criterion value' is a borderline case with regards to termhoodness and 'set value' by itself should definitely not be classified as MWT. The model classifies all these phrases as MWT.

The second validation set patent EP 1400610 A1 has the title “Thermal barrier coatings with low thermal conductivity comprising lanthanide sesquioxides” and belongs to the fields ’Inorganic Chemistry’, ’Dyes / Paints’, ’Coating Metallic Material’, ’Machines or Engines in General’, ’Combustion Engines’ and ’Combustion Apparatus’. It consists of 230 sentences with an average length of 34.60 word tokens per sentence. For this patent, the recall for rare MWTs of this model is 0.72. In comparison, the baseline rare MWT recall for this patent is 0.80. In this patent the MWTs ’first oxide’, ’second oxide’ and ’third oxide’ are used frequently, which are phrases this model does not recognize as MWTs. In most other contexts the words ’first’, ’second’ and ’third’ should not be included as part of the MWT, which is reflected in our annotations, but in the chemical context this is not the case. To solve this issue, the model would also have to be trained on some representations of the contexts or fields a sample sentence comes from. With the phrase ’set forth’, sentence 4.8/2 also highlights another usage of the word ’set’. Again, the model incorrectly includes ’set’ in the actual MWT ’thermal barrier coatings’, turning it into ’thermal barrier coatings set’. The model has apparently learned that the word set has a very high termhoodness factor, but is unable to differentiate between the different meanings of the word or take into account how a low termhoodness of other words could prevent a phrase from being classified as MWT.

The first test set patent EP 1237134 A2 has the title “Electronic transactions” and belongs to the fields ’Computing / Calculating / Counting’ and ’Checking-Devices’. It consists of 232 sentences with an average length of 29.54 word tokens per sentence. For this patent, the recall for rare MWTs of this model is 0.52. In comparison, the baseline rare MWT recall for this patent is 0.70. Examples of model errors regarding this patent include the model not recognizing MWTs containing the tokens ’identity’ or ’entity’ - tokens that can be found in the MWTs ’information entity’ or ’personal identity number’. As before, the problem stems from the fact that both of these tokens only appear in this particular patent in the dataset and that both of these words are unexpectedly unique, having a cosine similarity of around 0.5 at most for words that are not just different word forms (e.g. not ’entities’). We assume that for this reason the model was unable to learn the termhoodness of these words.

The second test set patent EP 1270193 A1 has the title ’Heat resistant cushioning material for press molding’ and belongs to the fields ’Working of Plastics’, ’Presses / Layered Products’, ’Braiding / Lace-Making’ and ’Electric Techniques not otherwise provided for’. It consists of 351 sentences with an average length of 29.65 word tokens per sentence. For this patent, the recall for rare MWTs of this model is 0.76. In comparison, the baseline rare MWT recall for this patent is 0.58. Noticeable is the inconsistent prediction behavior of the model regarding the MWT ’printed wiring board’ and variations thereof. While this MWT occurs quite frequently in the text, and most of the time it is identified correctly, in some occasions - such as in sentence 4.8/3 - only the phrase ’wiring board’

is detected for no apparent reason. In other locations in the text, the MWT 'multilayer printed wiring board' is used, which is also sometimes recognized by the model and sometimes only 'wiring board' is predicted. Furthermore, the MWT 'single sided printed wiring board' is never detected (the model detects 'printed wiring board' instead) and the MWT '4-layer printed wiring board' is always detected. It seems that the model does not always assign a high termhoodness to words ending with the '-ed' suffix and perhaps makes the termhoodness of such a word more dependent on its surroundings. If now a few words with the '-ed' suffix appear in sequence, all of them might be less likely to be selected as part of the MWT, but there are too few examples of such a constellation in our dataset to further inspect this assumption.

4.5 Summary

In this chapter we presented the MWT patent dataset we created, the results of our pattern matching baseline method as well as the results of various experiments conducted with our MWT detection method on that dataset and a more detailed discussion about the performance and common errors of our best models. While the baseline method is recall oriented method, by using the spaCy library and a pattern only including nouns and adjectives, the precision was higher than expected. Our MWT detection method outperforms the baseline with respect to precision, recall and F1 score if we use regularization, word embeddings trained on the patent domain and make use of our character CNN component. The improvement in generalization performance when using 18 instead of 9 patents is large while experiments with different hyperparameters did not seem to affect performance in any substantial way, which leads us to assume that adding more data is preferred over further hyperparameter tuning at the moment. However, since there is also a jump in performance when the character CNN component is added to the model, adding other complex components, such as a component that adds language model information to the model as seen in Ammar et al. [APPB17], could also further improve the results.

Of the two best models discussed in more detail, both models have problems with technical words if they appear only a few times in the dataset. Tokens with the '-ed' suffix (i.e. verb participles) are particularly error prone, most likely due to them being closely related to verbs, which are not parts of MWTs. Overall, it seems that the char CNN component is adding suffix information to the models in a useful way by pushing tokens towards termhoodness or away from it. Rare words with word vectors that are very dissimilar to other word vectors also generally cause identification errors. Tokens that that can occur in the context of MWTs but can also be used in non-MWT contexts, such as the token 'set', also frequently cause errors. For this reason, we suggest combining word representations with representations of technical or non-technical fields, so that a model can differentiate between the different usages of words. Furthermore, analyzing the model revealed a few flaws in our annotation approach that could be improved on in future

work on the topic, such as improving the normalization of MWTs during annotation, designing guidelines for term broadness and reviewing some of the annotations we did. Especially the imprecise definition of how domain specific a term must be to be considered a MWT is an issue at the moment, causing a number of prediction errors and inconsistencies.

In the last chapter we summarize the contributions of our work and further discuss its limitations and potential future work.

CHAPTER 5

Conclusion

In this thesis we presented a method for detecting even rare MWTs in patent texts that does not require the prior detection of PoS tags. This method can be used for creating a terminology for patent passages, patent documents or patent collections. We showed that our method performs better with regards to F1 score than a simple linguistic baseline under the condition that the word embeddings required by the ANN architecture are trained on data from the patent domain.

By using word representations based on the characters a word consists of, the distributional semantics of a word and the relationship to other words in the input sentence, the method is able to express whether a word in the input sentence is likely to be either the first word in a MWT, the last word, a word in the middle or not part of the MWT at all and assign label probabilities accordingly. The most likely sequence of word labels is then used to identify the MWTs in the input sentence. We used two different dataset splits (A-Split and B-Split) for model training and evaluation on which our best models achieved an F1 score of 0.84 and 0.71 respectively. While the performance between A and B Split was very similar during training, the B-Split models would always perform comparatively worse during testing.

What was the impact of regularization techniques that can be applied to most deep learning models on our MWT model? Our experiments showed that without L2 regularization model training is very dependent on the initial state of the model parameters. Without regularization it was highly random whether a model was trainable at all or would completely fail to detect any MWTs during and after training.

What was the impact of our custom character CNN component? Using the custom character CNN component resulted in an up to 20% higher F1 score than not using it.

Not using the component resulted in a model performance that was very close to our best pattern matching baseline.

What was the impact of hyperparameter changes on different parts of the network? We noticed that running the **Reg** experiment with the same hyperparameters multiple times would create models with F1 scores as high as 0.84 (0.71) and as low as 0.79 (0.65) when tested on the A-Split (B-Split) test set. While small variations are expected due to the random normal initialization of the weight and bias model parameters, we attributed these comparatively large variations to the low amount of available annotated data for training and testing. But since we did not experiment with different initialization strategies and using just random normal initialization does not take the size or the activation function of a tensor into account, different initialization strategies might still be able to improve the model performance or consistency thereof. Changing the size of the output of the character CNN component as well as adding hidden layers to the scoring component did not produce any F1 scores outside this observed range of variation.

What was the impact of additional PoS tag features extracted by the Python natural language processing library spaCy? Similar to experiments with different hyperparameters, adding PoS tag features to the Bi-directional LSTM component did not produce any F1 scores outside the observed range of F1 score variation.

What was the difference in performance between general purpose word embeddings, such as those created with a dataset of Wikipedia articles, and specialized patent word embeddings created with the CLEF-IP patent document collection? Using patent word embeddings improved F1 score by up to 17%.

To what degree did different training data set sizes affect the MWT model performance? While using half the training set for training did not affect the performance during training, the test set F1 score of the resulting models was substantially worse.

To what extent could rare MWTs be detect? We showed that the described approach is able to detect rare MWTs in 5 out of 8 analyzed patents with higher recall than the baseline method, with the A-Split model performing better than the B-Split model in this regard.

What were common prediction errors of our models and their possible causes? Analyzing the model output of 8 validation and test set patents has shown that rare word tokens can cause the models to make errors. These errors happen more frequently if the word in question is a noun that has not appeared during training or is a verb in participle form that has also not appeared during training and the word embedding of the word token is

dissimilar to other word embeddings.

To combat these issues, we suggest to create a larger dataset of annotated patent data to cover more frequently used word tokens for each field and also further reduce the influence of the model parameter initialization. The creation of such a dataset should be preceded by a more precise definition of MWTs and their termhoodness as well as the creation of precise annotation guidelines including borderline examples to combat the issue of borderline phrases that are either barely MWTs or barely not MWTs. Further, additional components, such as a language model component, could be added to the network architecture to allow the models to represent sentences in more meaningful ways.

However, there are also a few aspects of the approach that require larger changes to lead to improvements, such as word token ambiguity and text normalization. The different meanings of the same word in different contexts needs to be addressed, which could be achieved either by adding a way for the model to differentiate between contexts through a separate context representation or by including the context of a word in the word embedding and handling the task of context detection in the unsupervised learning part of the approach. Text normalization (i.e. handling merged MWTs and split MWTs) is another issue that needs to be handled in the preprocessing of the data and during the dataset creation process. The current annotation method of marking only consecutive sequences of tokens in the text as MWTs is flawed since it is not possible to add merged MWTs and split MWTs to the ground truth and should be adapted to allow for interruptions and word tokens being part of multiple MWTs. Consequently, an extra preprocessing step should be added to make the new annotation approach compatible with our method and normalize the input sentences accordingly, for example, by expanding the sentences in such a way that all individual MWTs of a merged MWT appear in the sentence as consecutive token sequence.

Finally, while we have shown that our method can be used to detect MWTs in the patent domain, its effectiveness when used in prior art search is not part of this thesis. Whether our approach can be used to improve query generation methods such as described in [ALP+16] should be explored and the impact of missing MWTs (recall too low) and incorrect MWTs (precision too low) compared.

List of Figures

List of Tables

List of Algorithms

Bibliography

[ADR+17] Isabelle Augenstein, Mrinal Das, Sebastian Riedel, Lakshmi Vikraman, and Andrew McCallum. SemEval 2017 Task 10: ScienceIE-Extracting Keyphrases and Relations from Scientific Publications. *arXiv preprint arXiv:1704.02853*, 2017.

[ALP+16] Linda Andersson, Mihai Lupu, João Palotti, Allan Hanbury, and Andreas Rauber. When is the Time Ripe for Natural Language Processing for Patent Passage Retrieval? In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1453–1462. ACM, 2016.

[APPB17] Waleed Ammar, Matthew Peters, Russell Power, and Chandra Bhagavatula. The AI2 system at SemEval-2017 Task 10 (ScienceIE): semi-supervised end-to-end entity and relation extraction. *nucleus*, 2(e2):e2, 2017.

[ASC13] Akiko Aizawa, Takeshi Sagara, and Panot Chimongkol. Technical Term Identification for Semantic Analysis of Scientific Papers. 2013.

[AVP14] Peter G. Anick, Marc Verhagen, and James Pustejovsky. Identification of Technology Terms in Patents. In *LREC*, pages 2008–2014, 2014.

[CGCB14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[CWB+11] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[FAM00] Katerina Frantzi, Sophia Ananiadou, and Hideki Mima. Automatic recognition of multi-word terms:. the c-value/nc-value method. *International Journal on Digital Libraries*, 3(2):115–130, 2000.

[For73] G. David Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HXY15] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[JK95] John S. Justeson and Slava M. Katz. Technical terminology: some linguistic properties and an algorithm for identification in text. *Natural language engineering*, 1(1):9–27, 1995.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[LBS+16] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[LHo13] Mihai Lupu, Allan Hanbury, and others. Patent Retrieval. *Foundations and Trends in Information Retrieval*, 7(1):1–97, 2013.

[LMP01] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[MB05] Frederic Morin and Yoshua Bengio. Hierarchical Probabilistic Neural Network Language Model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[NKT+09] Hidetsugu Nanba, Hideaki Kamaya, Toshiyuki Takezawa, Manabu Okumura, Akihiro Shinmori, and Hidekazu Tanigawa. Automatic translation of scholarly terms into patent terms. In *Proceedings of the 2nd international workshop on Patent information retrieval*, pages 21–24. ACM, 2009.

[PSM14] 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.

[Ram14] Carlos Ramisch. *Multiword Expressions Acquisition.* Springer, 2014.

[RLHZ16] Navid Rekabsaz, Mihai Lupu, Allan Hanbury, and Guido Zuccon. Generalizing translation models in the probabilistic relevance framework. In *Proceedings of the 25th ACM international on conference on information and knowledge management*, pages 711–720. ACM, 2016.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[TH12] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[VDOK10] Suzan Verberne, Eva D'hondt, Nelleke Oostdijk, and Cornelis Koster. Quantifying the challenges in parsing patent claims. In *Proceedings of the 1st International Workshop on Advances in Patent Information Retrieval (AsPIRe 2010)*, pages 14–21, 2010.

[Wal04] Hanna M. Wallach. Conditional random fields: An introduction. *Technical Reports (CIS)*, page 22, 2004.

[ZZL15] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.