universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Knowledge Graph Construction From Streaming Social Media Data“

verfasst von / submitted by

Sebastian Komposch BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2022 / Vienna, 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066921
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Informatik UG2002
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Inform.Univ. Dr. Claudia Plant

Acknowledgements

I want to thank my supervisor Univ.-Prof. Dipl.-Inform.Univ. Dr. Claudia Plant for supervising this thesis and my Ylli Sadikaj M.Sc. and Andreas Stephan B.Sc. B.Sc. M.Sc. for their continuous support and their valuable inputs and guidance throughout the creation of this thesis. Furthermore I want to thank the team of Cortecs, who not only provided me with resources so I could carry out my experiments, but also offered valuable ideas and guidance. I also want to give special thanks to my girlfriend Nina, without whose support and encouragement this thesis would not be finished yet. Finally I want to thank my family and my friends who have supported me throughout my studies.

Abstract

Knowledge graphs have recently encountered a surge in research interest. When constructing such a graph, it is necessary to extract entities and relations between them from the information source. One such source of particular interest is social media. As social media data is very short-lived, it needs to be processed quickly, preferably in real-time. This thesis proposes a joint entity recognition and relation classification model based on Long Short-Term Memory Networks to construct knowledge graphs from tweets. Furthermore a dataset of tweets is labelled to measure the model's performance. The model outperforms all baselines in terms of relation classification. Additionally, experiments with filtering samples are conducted, which show that removing low-confidence predictions from relation loss calculation can improve performance. Finally, the model's usefulness and limitations are discussed in a qualitative analysis.

Kurzfassung

In den letzen Jahren stieg das Interesse an Knowledge Graphen in der Forschung an. Um einen Knowledge Graphen zu konstruieren, ist es nötig, Entitäten und deren Relationen untereinander aus Informationsquellen zu extrahieren. Eine besonders relevante Quelle solcher Informationen sind soziale Netzwerke. Informationen, die in sozialen Netzwerken veröffentlicht werden, sind meist nur für kurze Zeit relevant. Daher ist es nötig, sie schnell zu verarbeiten, am besten in Echtzeit. In dieser Arbeit präsentieren wir ein Modell basierend auf Long Short-Term Memory Netzwerken, um Entitäten und deren Relationen aus Tweets zu extrahieren. Um unser Modell zu evaluieren, annotieren wir einen eigenen Datensatz bestehend aus Tweets. Wir zeigen, dass unser Modell bessere Ergebnisse als Vergleichsmodelle liefert. Weiters zeigen wir, dass das Filtern der Vorhersagen für die "Loss"-Berechnung die Ergebnisse weiter verbessern kann. Abschließend evaluieren wir die Nützlichkeit der Vorhersagen sowie die Grenzen des Modells in einer qualitativen Analyse.

Contents

List of Tables

List of Figures

List of Algorithms

Listings

1 Introduction

With the increase of publicly accessible data on the internet, the desire to infer knowledge from it and present it in a structured way rises. One way to present knowledge in such a structured, machine-understandable way is knowledge graphs. Knowledge graphs have been surging in popularity since the introduction of Google's knowledge graph[7] in 2012. Researchers and companies alike have started to research construction techniques as well as applications for knowledge graphs [8, 9, 10, 11]. Generally speaking, a knowledge graph is a graph-like structure, which represents facts in some way. These facts usually consist of various entities, such as objects, concepts, events or anything else which could describe a given fact. These entities act as nodes and are connected via edges. Edges represent some kind of relation between these entities. Additionally, nodes and edges can be annotated with further semantic information[12]. Facts from knowledge graphs can usually be described using a triplet in the form *(subject, predicate, object)*. *Subject* and *object* would be nodes (entities) and the predicate is the edge (relation) between them. An example of such a fact would be (*White House, locatedIn, Washington D.C.*). Figure 1.1 shows this fact integrated into a simple knowledge graph.

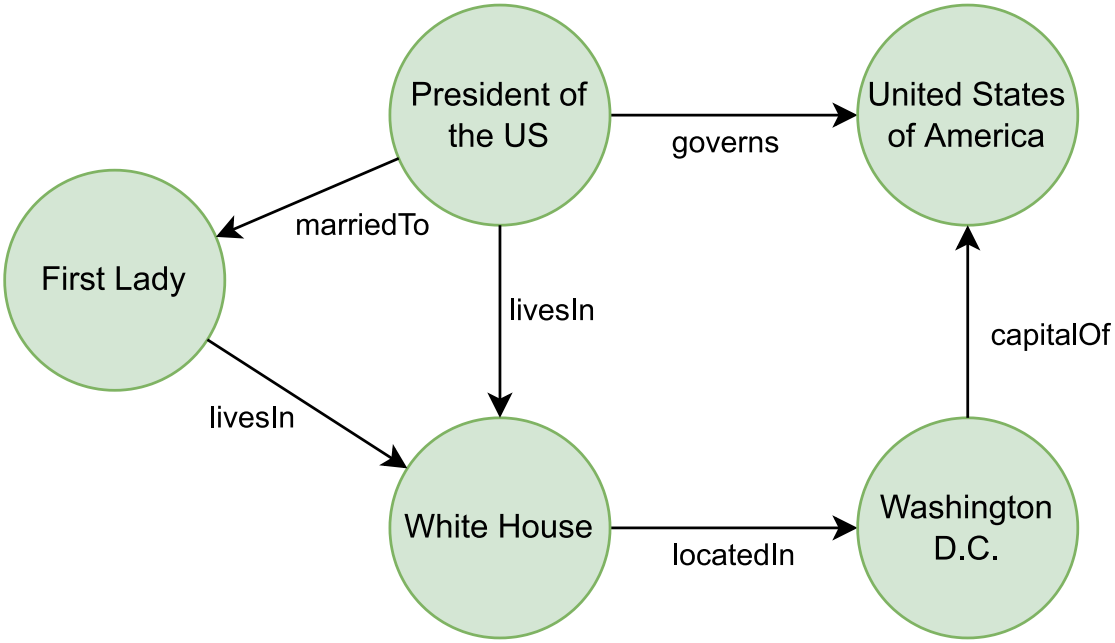

Figure 1.1: Illustration of a simple knowledge graph. Nodes have directed edges describing the relations between them.

Real-world knowledge graphs are of course much larger and contain a greater variety of knowledge. Google for example uses their knowledge graph to provide semantic information to certain searches in the form of text boxes, which show up above or next to the search results. DBPedia[13] extracts structural information from Wikipedia articles such as infoboxes, tables and similar facts and transforms them into triplets in the Resource Description Framework (RDF)[14]. These triplets form a knowledge graph which is provided in various ways to the public. Further examples include YAGO[15], another knowledge graph based on Wikipedia data, or WordNet[16], a lexical knowledge graph, connecting English words by lexical relations and annotating them with their definitions. There are also domain-specific knowledge graphs, which only cover a certain area of expertise. For example, the STRING database[17] contains predicted and observed functional and physical associations between proteins. Another example is the DrugBank database[18] which provides information about various drugs. It connects drugs with each other based on their interactions.

Most of these knowledge graphs derive their knowledge from some kind of text, usually natural language. In order to build knowledge graphs from this natural language source, it is necessary to process it either by hand or using natural language processing (NLP) techniques. NLP has its roots in the 1950s when it was attempted to process text using hand-crafted rules, which however did not succeed due to natural language's ambiguity and often ungrammatical use. In the 1980s a change of direction occurred, focusing more on simpler probabilistic machine learning models which are trained on large text corpora[19]. NLP introduced many methods to solve subtasks such as tokenization, part-of-speech tagging, named entity recognition or coreference-resolution which are subtasks often used for knowledge graph construction. In the course of this thesis, we will take advantage of some of these techniques, such as part-of-speech tagging or dependency parsing.

The underlying data of a knowledge graph can come from almost any source. One such source is social media. Looking for example at Twitter, there is a vast amount of knowledge published daily in the form of tweets. This is however hidden between equally vast amounts of irrelevant content. Automatically extracting the relevant knowledge from tweets could form knowledge graphs with a wide variety of use cases [20, 21]. This thesis focuses on the construction of knowledge graphs from social media data. Because social media is a very short-lived medium and knowledge might be irrelevant after a very short time, it would be advantageous to be able to process the data in real-time. We therefore pose the additional restriction on our solution to be able to run in a streaming setting, receiving data as it is published.

1.1 Motivation

As seen above, there are many different knowledge graphs already present, both general and domain-specific. This raises the question as to why it would be interesting to construct new knowledge graphs and which additional value they provide. There are three ways in

which a knowledge graph can differ from already existing ones. First, the domain that the knowledge graph is built from could be new. While there are many different domains, not all have a dedicated universal knowledge graph and new domains are constantly appearing. Second, the underlying data could be different. Knowledge graphs are built from real-world data. This data needs to be acquired and processed. Creating a new knowledge graph allows restricting the time window from which data can come, providing for example only the most recent knowledge in a domain, discarding outdated information. It would also be possible to use different data sources which have not been investigated yet. Social media is an example of such a source. Finally, the types of nodes and edges in a knowledge graph are usually limited. Certain use cases might require different types than existing knowledge graphs provide in order to represent different knowledge.
Social media data has been a popular target for various NLP techniques, mainly to extract data about events[22] or sentiments[23]. Knowledge extraction from social media to construct knowledge graphs is a less active field of research. Using this data could however yield interesting insights. Getting the impression of what a large group of individuals considers as facts in nearly real-time could be beneficial for various tasks. Attempting to derive new facts from these existing graphs might allow predicting what conclusions a certain group of people might come to. Also, correlating facts published on social media with real-world events or facts from other sources might show the influence that certain authors of such facts have, and using these influential people's newer publications might allow predicting different real-world events. An example of this is, when influential stockholders such as Warren Buffet or Elon Musk publish their opinion about certain stocks thus influencing the general price of these stocks. No matter the actual use case, it is always necessary to construct a knowledge graph containing facts of sufficient quality and size first. We therefore try to apply knowledge graph construction techniques to social media data and attempt to extract high-quality facts to form a custom domain-specific knowledge graph.

1.2 Problem Definition

The goal of this thesis is to develop an application using machine learning techniques that is able to automatically extract facts from social media microblog texts. Twitter is chosen as the source of the microblogs due to its very liberal and well documented API[1]. In order to build a knowledge graph, we need information that is represented by nodes and information that is represented by edges. A fixed predefined set of entities is used as nodes. These entities could be names of people, companies or other organizations or completely different things like numbers, locations or even fruits. The set is defined based on the domain that the task is applied to. The entities need to be detected in the text and their type (eg. person, company, fruit) needs to be classified. Edges in the graph should describe semantic relations between the different entities. Given two entity nodes e_1 and e_2, the directed edge r_1 from e_1 to e_2 represents a semantic relation of a certain type between these two entities. An example for this would be that given the nodes *Elon*

[1] https://developer.twitter.com/en/docs/twitter-api

Musk and *Tesla*, the relation *isOwnerOf* holds in the direction from *Elon Musk* to *Tesla*. The primary problem in building the graph is to extract the entities as nodes and relations as edges from the microblog text. Therefore we focus on two subtasks: named entity recognition, which is the process of finding entities and classifying their type, and relation classification, which is finding a relation given an input text and two entities. The result of these two subtasks is a set of entities and the potential relations between them. Building a graph structure from this is only a matter of connecting the entities via relations which is of negligible complexity and therefore not included in the proposed solution. Further enhancements such as deduplication of nodes and knowledge graph completion[24] are out of scope for this thesis and require further research on their own. For this reason the task we try to solve consists of named entity recognition and relation classification. Both subtasks are further described in chapter 2.
We postulate that using a joint approach in solving both tasks is advantageous due to inter-dependencies between the two tasks. We therefore propose a joint model and evaluate it against disjoint baselines, which are state of the art on their respective subtasks or are structurally similar to our model.

1.3 Summary

Our focus lies on three different aspects. We propose a joint machine learning model to perform the tasks of named entity recognition and relation classification in order to extract facts to form a knowledge graph. The model is based on an already existing model which is then modified to suit our needs. It needs to be able to run in a live-streaming setting, receiving tweets as a continuous stream. The developed model is then evaluated on social media data extracted from Twitter. For this purpose, a new dataset of domain-specific tweets is labelled and used for performance evaluation. The proposed model is evaluated against two baselines. Both baselines are disjoint approaches. We use them to show that using a joint model is beneficial over disjoint ones. The models are evaluated on the labelled tweet dataset, as well as another common dataset for this task. To summarize, our key contributions are:

- A new machine learning model based on an existing one to jointly extract named entities and classify relations.
- A new labelled dataset of tweets to evaluate the tasks of named entity recognition and relation classification.
- A performance evaluation showing that our proposed model outperforms disjoint baseline models in this task and can work in a live-streaming setting.

This thesis is organized in the following chapters: First, chapter 2 provides a short explanation of key concepts needed to understand the task and our solution. Chapter 3 looks at previous research which has been conducted in this field. In chapters 4 and 5 we present our model and its implementation. Chapter 6 presents the performed experiments

and discusses the results. Finally, in chapter 7 we conclude our findings and give an outlook on future work.

2 Background

Now that we have introduced the topic and described the problem, we need to introduce some key concepts which are necessary in order to develop our solution.

2.1 Knowledge Graph

The term "knowledge graph" is already used in the previous chapter; however, up until now, no precise definition has been given. One reason for this is that there is not just one definition for knowledge graphs. There is however existing research trying to find a suitable definition for knowledge graphs. Ehrlinger et Wöß[25] propose a definition which requires a knowledge graph to have an ontology, some kind of knowledge acquisition as well as a reasoner to derive new information. Paulheim[26] proposes a different definition which poses four requirements for a knowledge graph. He states in [26] that a knowledge graph should mainly describe real-world entities and their interrelations. The entities need to be of certain classes which are defined in a schema together with possible relations. Arbitrary entities should have relations with each other. Finally, he requires the graph to cover multiple different domains.
As our task is domain-specific, it is not possible to conform to the last requirement, but only the others before. There is another definition that suits our problem even better. Wang[27] defines a knowledge graph as follows:

> "A knowledge graph is a multi-relational graph composed of entities (nodes) and relations (different types of edges). Each edge is represented as a triple of the form (*head entity, relation, tail entity*), also called a fact, indicating that two entities are connected by a specific relation [...]"

So in the context of this paper, a knowledge graph is a graph of fact triplets, where the head and tail are entities and they have a relation between them. Head and tail form the nodes in the graph and the relation is the edge. Both entities and relations need to have predefined classes. The fact triplets are also defined formally in [28] by Zhang et al.. These triplets represent edges in the graph in the form *(head entity, relation, tail entity)* or denoted in short as *(h, r, t)*. There are different formal approaches which denote facts that use such a triple relation and might even extend it, for example, the Resource Description Framework (RDF)[14].

Figure 2.1 shows the knowledge graph example from figure 1.1 again, this time with the fact triplets that it consists of.

(**President of the US**, *marriedTo*, ***First Lady***)
(**First Lady,** *livesIn*, ***White House)***
(**President of the US**, *livesIn*, ***White House***)
(**President of the US**, *governs*, ***United States of America***)
(**White House**, *locatedIn*, ***Washington D.C.***)
(**Washington D.C.**, *capitalOf*, ***United States of America***)

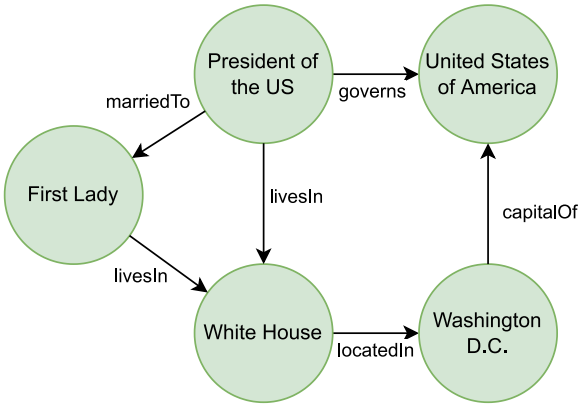

Figure 2.1: Illustration of a simple knowledge graph and the fact triplets that it consists of. Each fact consists of a head and tail entity (bold text) which represent the nodes in the graph and a relation (italic text) which represents the edge.

Building such a knowledge graph requires us to extract the fact triplets from the knowledge source. In order to extract these entities and relations we use the techniques of named entity recognition and relation classification.

2.2 Named Entity Recognition

One of the first occurrences of the term "named entity" comes from Grishman et Sundheim[29]. They refer to a named entity recognition task as "identifying the names of all the people, organizations and geographic locations in a text." This means that given the words of a text, we annotate each word with a label describing an entity or a special label that denotes the word as not belonging to an entity. We do not restrict ourselves to people, organizations and geographic locations, but choose different entity classes depending on the data that our solution is applied to.

2.3 Relation Classification

Relation classification is the task of finding a semantic relation between two entities in a text[30]. For this task, the entities in the text need to be known. The set of possible types that the relation can have needs to be defined in advance. Relations can be uni- or bi-directional. A simple uni-directional relation would be (*Elon Musk, ownerOf, Tesla*), because Elon Musk owns Tesla, but Tesla does not own Elon Musk (which would semantically make no sense). There is however often a corresponding contrary relation in the other direction. In this case, this could be (*Tesla, ownedBy, Elon Musk*) which is again uni-directional but describes the respective other direction to the first relation. A bi-directional relation is a relation that would hold in both directions, for example when looking at drug-drug-interactions and the relation (*Drug A, interactsWith, Drug B*) holds,

then the other direction (*Drug B, interactsWith, Drug A*) also holds. This bi-directional relation would be described in the knowledge graph by an undirected edge between the two entities. There can be more than two entities in a sentence and all entities could form relations with each other, therefore each entity can have multiple relations with multiple other entities. This requires several relation classifications between each pair to find all relations [31]. Another issue with relation classification is that, depending on the possible relations, the information which hints at the correct relation is often very nuanced and cannot be directly derived from the entity-words, but from certain words in the remaining sentence. Following the example above, take for example the following two sentences:

> Tesla CEO Elon Musk offers to buy Twitter.[1]
>
> Republicans ask Elon Musk to reinstate Trump's Twitter account after Tesla founder becomes largest shareholder.[2]

In order to find the relation (*Elon Musk, ownerOf, Tesla*) in the first sentence, there is not much context needed. The relevant word "CEO" is directly between "Elon Musk" and "Tesla" and no other context except for understanding the term "CEO" is needed in order to find this relation. In the second sentence however, the two entities are further apart and the relevant term "founder" is far away from "Elon Musk". There are also ambiguities which could lead to confusion since the meaning could for example also be interpreted in the way that Trump is the Tesla founder, who becomes the largest shareholder of something. To understand the second sentence, the knowledge that Elon Musk and not Trump is the founder of Tesla is assumed. Such ambiguities make relation classification a challenging task.
Relation classification can be performed in isolation or as an intermediate task. In the scenario of knowledge graph construction, first, named entity recognition is performed. On top of the entity recognition, relations are predicted between all pairs of entities.

2.4 Long Short-Term Memory Networks

To perform the above mentioned tasks, long short-term memory recurrent neural networks (LSTM) are employed. A LSTM is a type of recurrent neural network proposed by Hochreiter et Schmidhuber[32]. Recurrent neural networks are applied to sequential data and incorporate information from previous entries in the sequence into the calculation of the current entry. This allows them to understand the context of previous time steps. LSTMs are an extension to this model which tackles the problem that the inputs from the very beginning of a sequence only have a marginal effect on the outputs of later steps. They use a cell architecture with different gates to prevent irrelevant inputs from causing a loss of relevant long-term memory. This allows LSTMs to learn dependencies over longer sequences. A simple illustration showing a cell and the different gates can be

[1] `https://mainichi.jp/english/articles/20220414/p2g/00m/0bu/057000c` (Accessed 2022-04-22)
[2] `https://www.independent.co.uk/news/world/americas/us-politics/elon-musk-twitter-donald-trump-b2050703.html` (Accessed 22-04-22)

seen in figure 2.2 The architecture makes the model a good choice for natural language processing tasks, where dependencies between words in a text provide relevant context for various tasks.

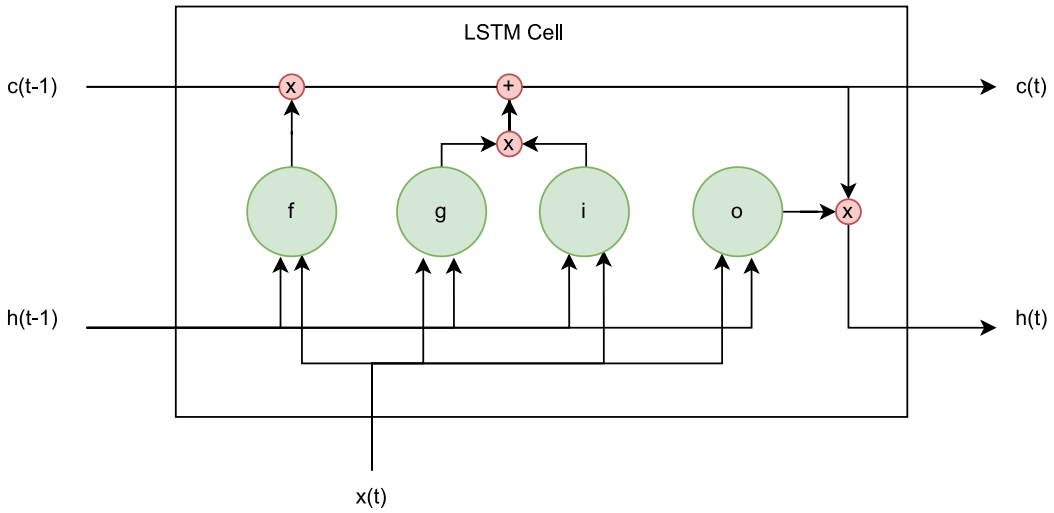

Figure 2.2: Illustration of an LSTM cell. The cell takes the input x at timestep t as well as the hidden state h and cell state c at timestep $t-1$ and computes the new cell- and hidden states. Figure adapted from [2].

2.5 Word2Vec

Neural networks such as the previously described LSTMs require numerical input, usually in the form of vectors. The input for named entity recognition and relation classification however is usually a sequence of words. In order to use word sequences as input, an additional layer is required which transforms words into vector representations. A commonly used technique is Word2Vec by Mikolov et al.[3]. They propose two neural networks which are trained on one-hot-encoded words to learn their representation by either predicting a word given its surrounding words in a sentence or predicting other words in a sentence given an input word. Figure 2.3 shows the general architecture of these two networks.

The trained hidden states of the model are used as vector representations for the words, and the authors show that the syntactic and semantic relations between words correspond to algebraic operations in their vector representations. An example using the relation between two such word pairs is shown in figure 2.4.

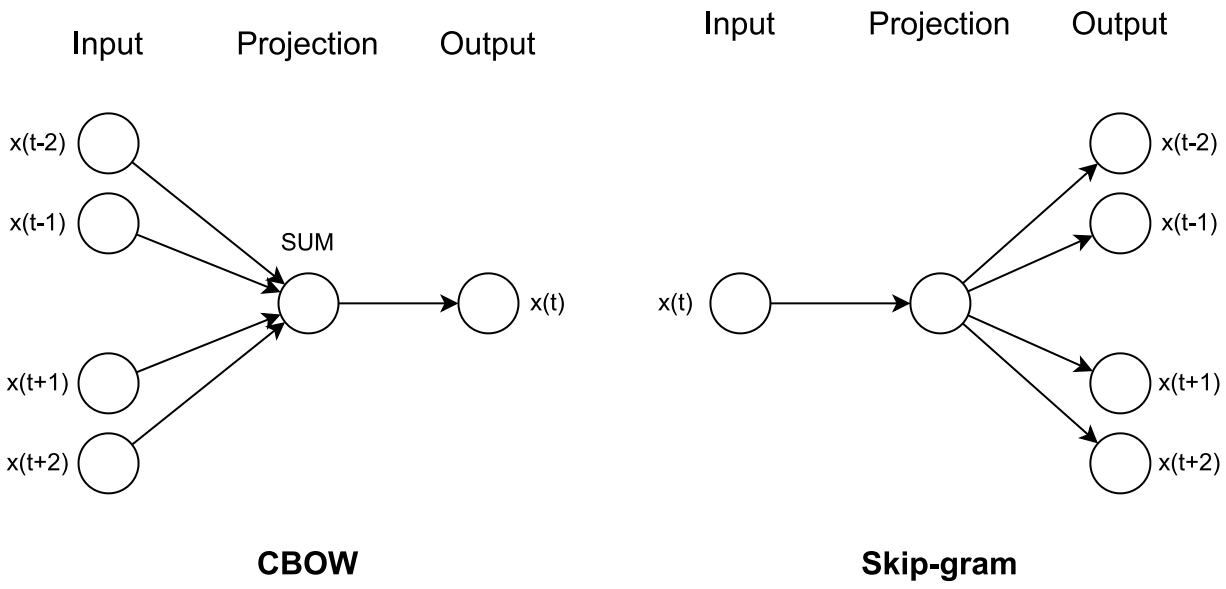

Figure 2.3: Model architecture of the two Word2Vec neural networks. The Continuous-Bag-Of-Words network takes the surrounding words as context and predicts the current word $x(t)$, and the Skip-gram network takes the current word and predicts surrounding ones. Figure adapted from [3].

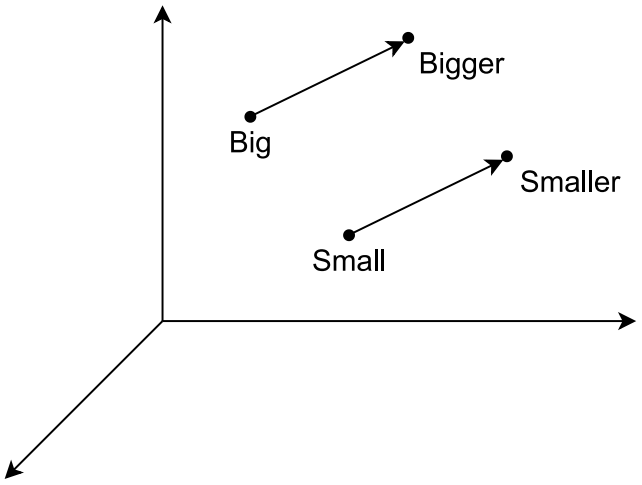

Figure 2.4: Illustration of the syntactic relationship between two word pairs in the Word2Vec model. The vector from "Big" to "Bigger" is equal to the vector from "Small" to "Smaller", suggesting that the relation between both word pairs (comparative degree) is the same. Figure adapted from [4].

2.6 Transformer

In order to evaluate the performance of our work, a baseline to compare against is necessary. The current state-of-the-art models in this field are mainly based on the Transformer architecture by Vaswani et al.[5]. The purpose of a transformer is to take a sequence, usually text, as input and produce another sequence as output. To achieve this a transformer consists of two parts, an encoder and a decoder. The encoder is actually a stack of encoders, usually six which are processed sequentially. Initially, the sentence is embedded and a positional encoding is appended, then it is passed to the first encoder. The output of this encoder is passed to the second one and so on. The final output is passed to the decoder to transform it into an output sequence. Both encoder and decoder use the concept of multi-head attention to achieve their performance. The architecture is shown in figure 2.5 and is further explained in the following paragraphs.

Encoder The encoder takes an embedding of a sequence concatenated with a positional encoding for each word. This input is first passed to a multi-head self-attention which calculates attention scores between each word. The attention is then combined with a residual connection and passed to a layer normalization. Then it is passed to a feed-forward layer and finally again into a layer normalization together with a residual connection. After processing this stack six times sequentially the output is used in the decoder.

Decoder The decoder is similar to the encoder, with the difference that additionally to the self-attention layer, it has another attention layer which incorporates the output of the encoder. The initial input of the decoder varies between training and inference. During training, the input is the full target sequence (which is the expected output from the decoder) and during inference, it is all previously inferred words. In the first step of inference, it only consists of the start token. This target sequence is again embedded and concatenated with positional encoding and then passed to a multi-head self-attention layer. This self-attention layer masks all tokens which are at a later timestep than the one which is predicted. This is done in order to prevent the model from "looking into the future". This is only necessary during training as during inference future tokens have not been computed yet. The output is then passed to a layer normalization together with a residual connection, similar to the encoder. It is then passed to another attention layer which, contrary to the self-attention layers, also uses the output from the encoder. This layer calculates the attention between the encoder outputs (which come from the initial sequence) and the decoder inputs (which come from the target sequence). This attention is then again fed to the layer normalization with a residual connection and passed to a feed-forward layer, before again being processed by a layer normalization. The final output is then passed to a linear layer and a softmax activation to get output probabilities for the sequence words.

Attention The attention layer actually requires three different inputs: a query Q, a key K and a value V. In the case of self-attention, all three values are actually the same. In

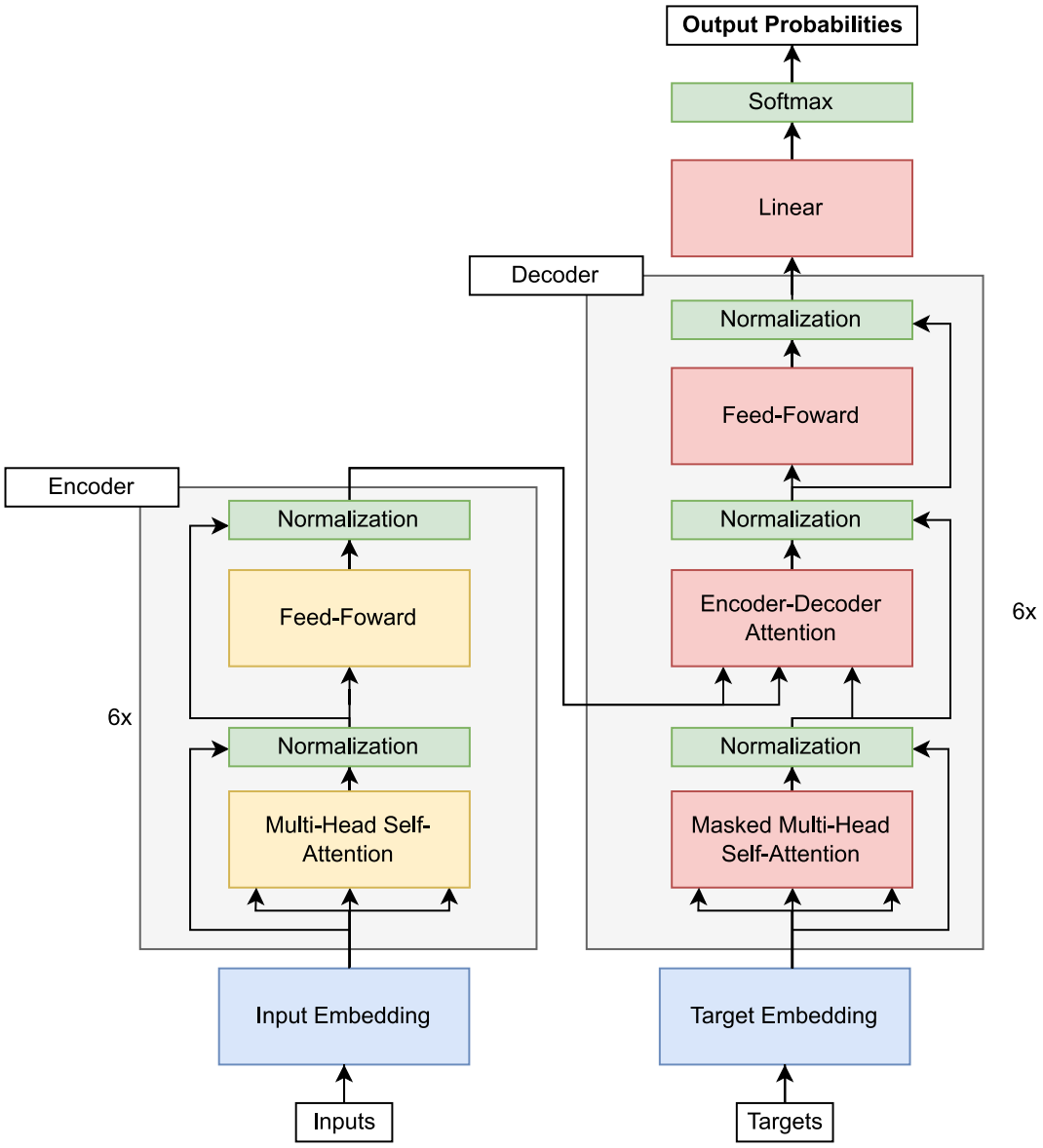

Figure 2.5: Architecture of a transformer. Inputs are embedded and processed by the encoder through self-attention and a feed-forward layer. Target outputs are processed similarly in the decoder, however the encoder input is used in a second attention layer to compute attention between input and target values. Finally decoder outputs are processed by a linear layer and a softmax activation to compute output probabilities. Figure adapted from [5]

the case of the encoder-decoder attention in the decoder, the key and value come from the encoder output, while the query comes from the decoder. Figure 2.6 shows the attention layer. First all three inputs Q, K and V are passed through a linear layer. Then equation 2.1 is applied to calculate attention scores, where d_k is the embedding dimension size of the inputs Q, K and V.

$$softmax(\frac{QK^T}{\sqrt{d_k}}) * V \tag{2.1}$$

Additionally, a masking is applied to ignore words which should not be computed. In the case of self-attention, these are padding tokens and in the case of encoder-decoder-attention during training all future words are masked. The computed attention allows the model to learn dependencies between all words. In order to learn multiple features from the embedding representations, the concept of multi-head attention is introduced. This means that the query, key and values are split into eight parts and then attention is computed in parallel on each of them before the result is concatenated again. This gives the transformer the possibility to learn different information from different parts of the embedding.

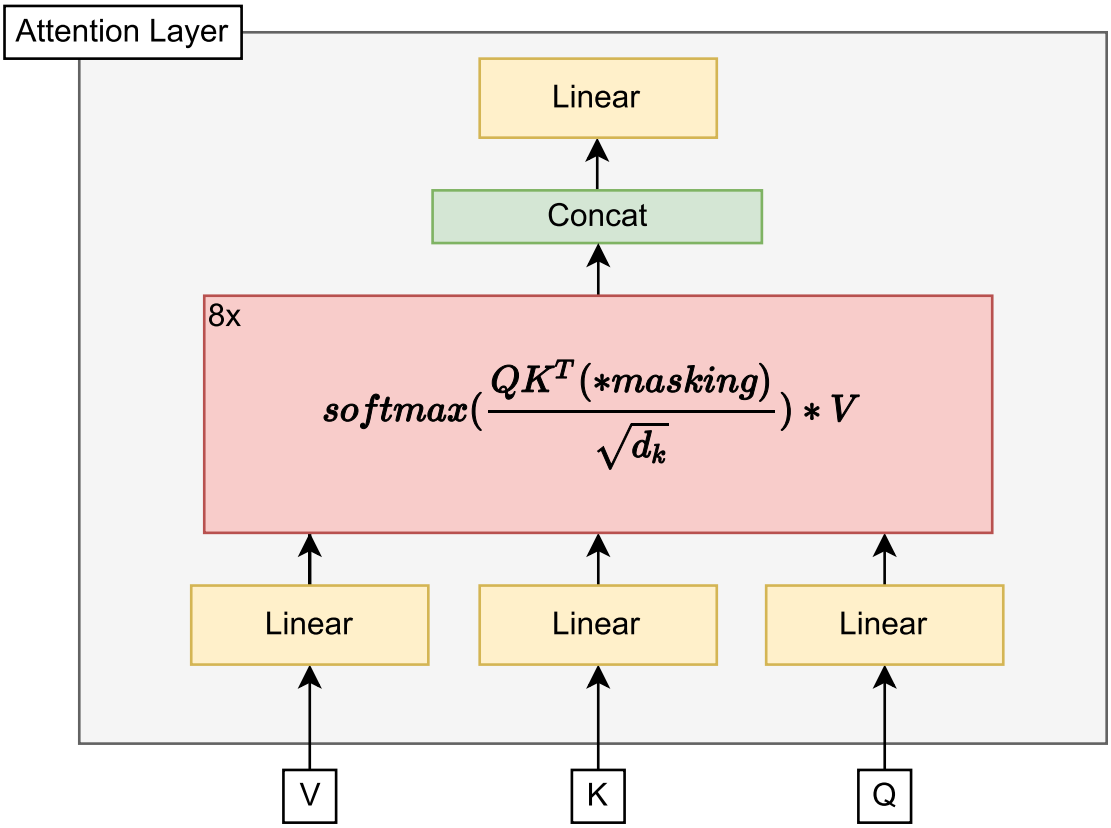

Figure 2.6: Attention layer of a transformer model. The input values V, K and Q are passed through a linear layer. The input vectors are split into eight parts along the embedding dimension and then processed in parallel before being reconcatenated and passed through another linear layer. Figure adapted from [5]

3 Related Work

In this chapter related work in the field is examined. We first focus on general knowledge graph construction. Then we examine it in the context of social media knowledge extraction, and finally, we review the tasks of named entity recognition and relation classification in particular.

3.1 Knowledge Graph Construction

Knowledge Graph Construction refers to the process of creating fact triplets (h, r, t), where h and t are head and tail-entities, while r is a relation between them. An example of such a triplet would be (*White House*, *locatedIn*, *Washington D.C.*). The entities h and t are nodes of the graph, while the relation r is an edge. The facts can come from arbitrary sources and can be extracted or created in arbitrary ways. One common way is to extract the needed information from semi-structured or unstructured text using natural language processing techniques.

Automatic knowledge extraction from documents In 2012 Fan et al. [8] developed a system to extract knowledge for IBM Watson. The software is called PRISMATIC and focuses on applications such as question answering or link prediction for clue terms and candidates in the US quiz show Jeopardy[1]. To build the necessary knowledge to solve such problems, shallow knowledge is extracted from large, redundant document collections. Then, additional knowledge is inferred using aggregate statistics of the shallow knowledge. This is done in three steps. The first step applies a dependency parser and a relation detector to identify binary relations in the corpus of the data. The parser is an English Slot Grammar Parser [33]. Additionally, to correctly identify entities in sentences, a co-reference resolution component is employed. Finally, the found entities are annotated with additional type information using a rule-based named entity recognizer (NER). In the second step sets of entities and their relations, which are also referred to as "Frames", are extracted from the parse tree created in step 1. For frame creation, the depth of the considered parse tree is restricted, so each frame stays focused on a predicate and fewer errors are introduced by complex parse trees. In the third step, additional knowledge is inferred using aggregate statistics. Frames are projected to certain relation types which are considered as facts if certain statistical scores, such as the frequency, conditional probability or normalized point-wise mutual information, are high enough. Examples for projections are "subject-verb-object" or "noun-preposition-object type". The authors evaluate their application by two metrics. They measure the coverage by counting the

[1] Quiz-show where the candidate is given an answer and has to find the corresponding question.

number of frames produced per sentence of their input data, as well as how many of the entities recognized by the NER are part of extracted frames, and they reason about the usability of PRISMATIC on IBM Watson in assisting with Jeopardy Tasks. PRISMATIC produces 1.3 frames per sentence and 94% of the recognized entities appear as part of one or more frames. These numbers sound very promising, however, the authors do not give any information about the correctness of the inferred facts and entities. In terms of practical usage for Jeopardy, PRISMATIC is used for type coercion, type inference, link prediction and answer candidate generation. The authors suggest that their system is superior to most other IBM Watson solutions in these use cases.

CN-DBpedia: A Never-Ending Chinese Knowledge Extraction System In 2015 Xu et al. [9] created a large scale knowledge graph as a Chinese version of DBpedia. In 2017 they published the knowledge extraction system that they used for it. Their goal was the reduction of the human cost in knowledge graph creation, as well as keeping it up-to-date at as low cost as possible. To remove the human factor from the extraction, they decided to reuse the taxonomy provided by the English DBpedia. The Chinese unlabelled entity should be mapped to the English correspondent in DBpedia. For this purpose, they developed a cross-lingual type inference system. To fix issues with the training data quality, such as incomplete or wrong types, the DBpedia entries are first completed and then a noisy filter step is deployed to remove wrong samples. Another attempt to reduce human involvement is infobox creation and completion. Instead of manually creating infoboxes, a Long Short-Term Memory Recurrent Neural Network tries to infer the content that should be put into the infoboxes. Training data is generated based on the idea that all relevant information in the infoboxes is also contained in the content of the article.

The authors solve their second problem, keeping update costs low by employing an active update strategy. They monitor recent news and popular search keywords for upcoming topics that might create new entities. The news articles are then greedily searched for any word combinations that could be entities. All created word combinations are matched against encyclopedia entries to find actual valid entities and update the knowledge base with them. Unfortunately, no metrics that show the accuracy or completeness of the system are published. The only available information is the extent of the knowledge graph as of 2016. The graph contained over 41 million infoboxes and 19 million tags. The shown metrics unfortunately do not give away anything about the quality of the content. The fact that the system had over 164 million API calls in its first year seems to suggest however that the data is at least in some ways useful or partially correct. Further analysis of the content would be beneficial and interesting to assess the quality of the extraction system.

OpenIE-based approach for Knowledge Graph construction from text Martinez-Rodriguez et al.[10] follow a more complex approach to extract knowledge in 2018. They try to construct RDF triplets [34] from plain English text without focus on a specific domain. Their approach is based on OpenIE, an information extraction paradigm to

extract data from a text corpus without human involvement [35]. OpenIE techniques are incorporated in a multi-step approach beginning with document acquisition, in which plain text is extracted from various sources using different extraction and cleaning techniques. For their evaluation they extract news article text from IT news websites. The text is split into sentences, words are tagged with their grammatical types, and a constituency-based parse tree is generated. Very long or short sentences, as well as sentences without a suitable grammatical structure, are omitted. In the next step entities are annotated using multiple Entity Annotation Services and the results are merged. When conflicts arise, the longest match or the match that is found by the majority of services is taken. The entities are then matched with identified nominal phrases in the parse tree. Using OpenIE-techniques, binary relations are extracted from the text. Semantic Role Labelling is applied to find arguments associated with the predicate in the sentence such as the agent, which is the active performer of an action, and the patient, which is the passive bearer of the action. These arguments are then matched with the nominal phrases. The predicates found by the relation extraction are labelled by querying them on existing Knowledge Graphs. Finally, the RDF triplets are created and organized in named graphs to represent n-ary relations.

The authors evaluated the approach using the standard measures precision, recall and F-measure as well as human reviewers to judge the quality and correctness of the extracted facts. They collected content from 605 IT-news web pages and applied their system to this data. They concluded that the system was worse than similar systems which were manually trimmed to domain-specific content. However comparing it to a baseline system without noun phrase and entity association, it was revealed to be superior. Another restriction to the system is that even though it might not be limited in its domain, the input text must be coherent and correct regarding its ideas, so it is not suitable for content such as microblogs.

A General Framework for Information Extraction using Dynamic Span Graphs Another approach to the task of information extraction is presented by Luan et al. [36]. They develop a general framework that couples multiple information extraction tasks through shared span representations with the goal to predict entities, relations and entity coreferences from texts. They derive word sequence spans from the input text. For each span, vector space representations are created. Using a dynamic span graph, global information is incorporated into the span representations. Bi-Directional Long Short-Term Memory Recurrent Neural Networks are used to identify spans that represent entities. These are then set as nodes in the graph. Coreference and relation links are predicted by the same neural network and are used to further refine the span representations with this new contextual information. Finally, entity and relation labels are predicted by Feed-Forward Neural Networks and the coreference scores are computed from a coreference graph. The authors compared their framework against several recent state-of-the-art techniques on four different datasets covering several domains. They used the F1-score as their performance metric. Using this measure, they managed to outperform almost all other techniques in their respective datasets. While there are no other metrics directly

available in the paper, the authors are confident that their system could be useful and high-performing in various domains.

Language Models are open knowledge graphs Chenguang et al.[11] took quite a different turn on knowledge extraction and knowledge graph creation. They state that knowledge graphs can be constructed from pre-trained language models such as BERT[37] instead of text alone. The text corpus that should be extracted is matched with a pre-trained language model to produce fact triplets. The triplets are then mapped to an open knowledge graph. The graph has a fixed schema such as the Wikidata schema. Additional facts that cannot be mapped to the schema are added to an open schemaless part of the graph. This is conducted in two steps. In the first step, the match-phase, a beam search is conducted over the attention matrix of the language model for each head-tail entity pair. In each search step the entry with the highest attention score is added to the fact. This is repeated until the tail entity is reached. The facts are then further filtered by total attention score, distinct frequency and additionally, only relations which are contiguous sequences are kept. In the second step, the map-phase, the facts are mapped to the knowledge graph schema. Entities are linked using an unsupervised entity linker based on a mention-to-entity dictionary. Additionally, word embeddings of the context of an entity are used. Relations are mapped by an offline relation map and manually checked by one of the authors. Facts that are unmapped or only partially mapped are added to a schemaless open knowledge graph. To test the performance of their system, the authors compared it to two OpenIE solutions, Stanford OpenIE and OpenIE 5.1. The system outperformed them both in terms of precision, recall and F1-score of the mapped facts. Unfortunately, the authors did not compare their system to newer methods, such as the ones mentioned above. Unmapped facts are manually verified by humans. They found that only 35.3% of unmapped facts were correct. The errors in the unmapped facts were traced back to the noun chunk extraction library used. While the low percentage of correct facts in the open schemaless part of the knowledge graph seems to be discouraging, it could prove to be very useful to use language models as fact sources anyway. In combination with state-of-the-art entity and relation identification approaches, this could lead to interesting results.

3.2 Social Media knowledge extraction

When focusing on social media content and microblogs, additional considerations apply to the task of knowledge extraction. Microblogs are short and often lack context to correctly classify information. They are often unstructured and focus on events and sentiments rather than facts. Nonetheless, there exist methods to extract knowledge from such microblogs.[38].

Adding semantics to microblog posts One of the older publications in this direction comes from Meij et al.[38]. They attempt to enhance tweets with additional semantics by annotating concepts to them. A concept is represented by a Wikipedia article. To achieve

this, they use a two-step approach: First, they generate a list of candidate concepts for each n-gram in that tweet. Then they apply supervised machine learning algorithms to classify each candidate concept as actually relevant or not. To get the candidates, all possible n-grams from a tweet are created which are then used to find candidates using different algorithms. Some of them are lexical matching of the n-gram with the Wikipedia title, search-engine-like searches of the n-gram or whole tweet and a set of other methods from related literature, such as DBpedia Spotlight[2] or TagMe[3]. For each of the returned n-gram-candidate combinations, a relevance score is derived using a broad set of features from the n-gram, the Wikipedia article, the tweet or a combination of all. Some examples are n-gram length and information gain, Wikipedia article title, content and anchors. These features are used by several supervised machine learning algorithms, such as Naive Bayes, Support Vector Machines, a C4.5 decision tree classifier, random forests or gradient-boosted regression trees. Evaluating the different algorithms shows that solely matching parts of the tweets with the Wikipedia titles performs badly. Matching the n-grams to the content and anchor of the articles significantly improves the recall. Other baselines achieve comparable results to this, with TagMe, which is designed especially for short texts, being even slightly better. The best result however is achieved simply by using the commonness metric that measures the probability of a candidate being the target of a link with the n-gram as anchor text. Applying the machine learning techniques on the baselines improves almost all metrics in almost all cases significantly. The best performing metric is the commonness-based one, enhanced by random forests. The authors also add an analysis of the relevance of each feature they used for the overall performance of the algorithms. While conceptually similar, this approach is not directly suitable for knowledge graph construction, as no triplets can be formed from the annotated concepts.

TwitIE: An Open-Source Information Extraction Pipeline for Microblog Text Bontcheva et al.[39] build a pipeline to extract information from Twitter microblogs. They use several existing algorithms[40, 41, 42, 43] which are optimized for and trained in social media content in a pipeline to extract information. The implementation is based on GATE[4] and is a customization of ANNIE[41]. It uses its gazetteer lists and sentence splitters without modification. The pipeline starts by importing the tweets and converting them to GATE-specific documents. Then the language is inferred using TextCat[40] and only English tweets are kept. The authors used Ritter's tokenising[42] which keeps abbreviations and URLs as one token but splits # from hashtags and @ from user-mentions which makes recognizing them as entities easier. There is further support for emoticons and normalization. Normalization is applied using a generic spelling-correction dictionary and an additional social-media-specific dictionary which contains common words and abbreviations used in tweets. The part-of-speech tagger they use is trained on a set of hand-annotated tweets, the NPS IRC corpus[5] and news texts. Named Entity Recognition

[2]https://www.dbpedia-spotlight.org/
[3]https://tagme.d4science.org/tagme/
[4]https://gate.ac.uk/
[5]http://faculty.nps.edu/cmartell/NPSChat.htm

performed better than the algorithms it was compared against. The authors attribute this to the specialisations on social media content during the former steps. They claim a precision of 77% and a recall of 83% for their test data. These values should serve as a baseline for future information extraction tasks of social media content. This approach yields named entities which could form a basis for knowledge graphs but no relations between them are extracted. Furthermore, this model relies mainly on grammar-based tools, while the approach in this thesis uses grammatical information only in the first step and focuses then on using recurrent neural networks.

Microblog semantic context retrieval system based on linked open data and graph-based theory Kalloubi et al.[44] try to serve user queries on tweets by constructing graphs of DBpedia entities. Their aim is to construct a graph of contextualized and weighted DBpedia concepts from tweets and link them based on Wikipedia links between the concepts. They perform shallow parsing on tweets to find phrases that are common named entities. The parsed phrases are searched for surface forms which are added to a collection. If no match is found, noun-phrase chunking is performed and noun phrases are searched in DBpedia for matching concepts. Hashtags are processed using the TagDef API. The found entities are added as nodes to the graphs and Wikipedia links between them are set as edges. Furthermore, each node gets a centrality factor assigned. The authors also present a similarity search algorithm to find tweets with concepts similar to a user's search query. The similarity is calculated based on the common predecessor and successor rate between the query's concept and tweet-graph's concept, also respecting their weights. The system was tested with 50 manually selected queries on a corpus of 50 million tweets. The results show that long user queries perform better than short queries in terms of precision and recall. This is because a longer query containing more relevant entities can better transport a user's interest and need, and narrow down the context that a user is interested in. Comparing the algorithm with the degree centrality algorithm also showed significantly better results in terms of precision of the top 30 result tweets.

Sentiment analysis for Chinese microblog based on deep neural networks with convolutional extension features Another approach to information extraction is to not get semantic information, but sentiments, from a corpus. This is done by Sun et al.[45] who perform sentiment analysis on Chinese microblog posts. Their algorithm first performs word segmentation and part-of-speech tagging. Then they use a special emotional dictionary which marks emotional words and assigns them a degree score. Additional modifying words that might emphasize or negate an emotional word are also considered and included in the score calculation. A difference from previous methods is, that the authors filter comments for direct replies and also add them to the sentiment analysis. The post and its replies are embedded and the feature vectors are combined using convolutional neural networks to a feature vector representing the whole conversation. Then a deep neural network consisting of restricted Boltzmann machines is trained to classify sentiments of the input vectors. Evaluation is performed on three distinct datasets. Several experiments were conducted to find optimal parameters. Sun et al. show that incorporating the

context in the form of replies improves the performance of their method. When compared to naive Bayes classifier and support vector machines, their approach shows a better overall performance of sentiment classification.

Lithium NLP: A System for Rich Information Extraction from Noisy User Generated Text on Social Media Bhargava et al.[46] follow a similar approach as Bontcheva et al.[39] do with TwitIE and implement a NLP system for noisy social media data. Their system extracts several types of information from tweets, including entities, topics, hashtags and sentiments. Extracted entities are based on a subset of Freebase entities chosen specifically for social media content. The basis of their approach is formed by several dictionaries. A mention-entity co-occurrence dictionary based on Wikipedia pages is used to map mentions to entities. An entity importance dictionary is created from linear regression models. Features for the regression come from Wikipedia links and their page rank. Finally, the authors create a topic-parent dictionary and topic-hashtag dictionary based on a manually curated ontology. The pipeline starts by detecting the language of the tweet. An open source language detector that employs a naive Bayesian filter achieves this. Then the text is normalized and sentence breaking is performed. Using the Lucene Standard Tokenizer [6] and, for Chinese, the Lucene Smart Chinese Analyzer, a tokenization step is performed. In the next step, entity extraction is performed. The authors use the entity mention dictionary on sliding n-gram windows to extract the longest matching mentions and generate candidate entities. Using context-independent and context-dependent features from the generated dictionaries as features for decision trees and logistic regression models, the best entity among the candidates is chosen. The authors claim an F1-score of 73% for this task on an unpublished dataset. Then the entities are mapped to important topics that they fit into. The topic choices are based on a weighted ensemble of semi-supervised models that the authors do not describe in more detail. The chosen topics are annotated with relevant hashtags using the formerly generated dictionary. Finally, sentiment analysis is performed using several lexicons and term counting with negation handling, and the entity is decorated with metadata. The authors report an F1-score of 46% on the sentiment analysis task, again on the same unpublished dataset. There is no overall performance comparison to other similar approaches regarding precision, recall or F-score. The authors could not perform these comparisons due to limited resources at that time on their side. Instead, they performed a runtime comparison to AIDA[47] in which they claim superior runtime performance due to AIDA's slower Stanford NER. Furthermore, they compare their information extraction feature set to several other tools and state that they support a more diverse information extraction than most other tools.

Mining Event-Oriented Topics in Microblog Stream with Unsupervised Multi-View Hierarchical Embedding One of the most recent approaches in social media information extraction comes from Peng et al.[48]. Their goal is to generate topics in high accordance

[6] https://lucene.apache.org/core/4_5_0/analyzers-common/org/apache/lucene/analysis/standard/StandardTokenizer.html

with real-world events in a streaming microblog setting. They try to extract such event-oriented topics from microblog batches. First, Latent Dirichlet Allocation (LDA) is used to extract topics from feeds of tweets. This yields two views on the data, a feed-topic matrix and a topic-word matrix. To get similarity between two topics, a Kullbach-Leibler-divergence-based similarity function is used on the topic-word view and a Jaccard similarity metric is used on the feed-topic view. A Bayesian rose tree is constructed from the topics. In the beginning each topic is a separate tree. Then trees are either joined, absorbed or collapsed together to form a new parent topic. The combination of trees happens in a way to maximize a probability function which incorporates both feed and word views. Because of this, the authors call their tree Multi-view Bayesian rose tree. Furthermore, to optimize construction speed, instead of considering all pairs of trees for merging, only the K-nearest neighbours are considered, according to a custom K-d tree and Best Bin First based KNN algorithm. TransR[49] is used to create embeddings of topics and their relations from the Bayesian rose tree. Finally, the embeddings are used in spectral clustering to find clusters which represent event topics.

The authors used three datasets in their evaluation, one from Sina Weibo[7] and two from Twitter, where one is a tweet stream. Their framework is compared to eight baseline approaches. In this comparison it performed best for event detection in terms of F1-score and consistency. It also was best in most cases for topic coherence based on both human annotation and Pointwise Mutual Information. Finally, case studies were performed on the hierarchical structure and the stream dataset's performance.

3.3 Entity-Relation Extraction

The chosen way to obtain triplets to form knowledge graphs is to apply named entity recognition to the input corpus and then apply relation extraction with the found entities as head and tail tokens, and use these relations as edges in the graph. This section focuses on papers that present ways to apply both named entity recognition and relation extraction.

End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures
A neural end-to-end entity and relation extraction model was developed by Miwa et Bansal[50] in 2016. They stack bidirectional tree-LSTMs for relation extraction onto sequential LSTMs for entity extraction. In the first step, named entities are detected from embedded input words and their part-of-speech tags via the sequential LSTM and a linear classifier on top of it. The predicted entities are then used together with the input words, as well as word dependency embeddings derived from a neural dependency parser. A reduced version of the dependency tree defines the evaluation order in the tree-LSTM. The tree that only contains entries that form the shortest path between two target entities produces the best results. As further improvements, entity pretraining and scheduled sampling are employed. The model is evaluated on the ACE04 and ACE05[51]

[7] https://weibo.com/

datasets as well as SemEval 2010 Task 8[52]. The model has superior scores over the feature-based baseline that the authors compared to. A comparison study is also provided that shows how the model performs without several components or improvements. It shows that scheduled sampling and especially entity pretraining have positive influence on the performance of the model.

GraphRel: Modeling Text as Relational Graphs for Joint Entity and Relation Extraction Fu et al.[53] develop GraphRel, an end-to-end relation extraction model to jointly learn entities and relations. The model works in a 2-phase process to jointly extract entities and relations. The first phase consists of a bi-directional LSTM-RNN and a bi-directional Graph Convolutional Network (GCN). The LSTM-RNN is fed with word and part-of-speech embeddings and its output hidden states are used as nodes for the GCN. The graph structure is derived from the sentence's dependency tree. The GCN then calculates hidden states from neighbouring nodes and predicts named entities for each word and relations for each word pair. In the second phase, a relation-weighted bi-directional GCN is built from the predictions of phase one. This second GCN does a second entity- and relation detection which can now consider further interaction between entities and relations as well as more implicit features. The authors tested their model on the NYT dataset[54] and the WebNLG dataset[55]. They report improvements of 3.2% on NYT and 5.8% on WebNLG over their best baseline. An important focus in the development of this model was to acknowledge overlapping relations which share one or two entities. The model predicts such relations better, however this is not described in further detail regarding the experimental results. One reason for this might be that, according to the authors, the NYT dataset contains few such relations. This would make it difficult to draw conclusions regarding the impact of considering overlapping relations on real-world data.

Joint entity recognition and relation extraction as a multi-head selection problem Bekoulis et al.[56] remove the need for external NLP tools or handcrafted features with their model. They use word- and character-embeddings as input and feed them to bi-directional LSTMs. The outputs from this are then passed to conditional random fields (CRF) to detect entities and relations. This architecture also allows identifying entities and corresponding relations for these entities at once. A notable property of the model is its ability to find multiple heads and therefore multiple relations for each discovered entity. The authors evaluate the model on four different datasets, having different corpora and different languages: ACE04[51], ADE[57], DREC[58] and CoNLL04[59]. They report improvements over the respective baselines on all datasets. This shows potential for the model to work in various settings, which can be ascribed to the independence from handcrafted features and grammar parsing.

Adversarial training for multi-context joint entity and relation extraction In a further step, Bekoulis et al.[60] attempt to further improve their model by using adversarial training. They generate adversarial examples by perturbing input embeddings to maximize

the loss using approximations by Goodfellow et al[61]. They evaluate this approach on the same four datasets as above. An improvement of less than one percent is visible in all datasets using adversarial training. A correlation between the effectiveness of adversarial training and the size of the dataset is suggested but needs further investigation.

Span-based Joint Entity and Relation Extraction with Transformer Pre-training Eberts et Ulges[62] use a transformer-based model for prediction. Contrary to most former methods, they try to classify entities on spans instead of using a token-labelling scheme like BIO or BILOU (see Ratinov et Roth[63]). This gives them the possibility to find overlapping entities. The input sentence is first embedded by a BERT based model. The output for each possible span in conjunction with span width embeddings and a sentence context embedding is then passed to a softmax classifier. In the next step, all non-entity spans are filtered, and for the remaining spans, each possible pair is used for relation classification. The input to the relation classifier is the BERT output of both spans including span width and a context embedding of the spans between the two entity spans. As a further improvement during training, negative sampling is performed, where random non-entities, as well as entities without relations are used as negative training samples. The authors report large performance differences with F1 scores ranging from 10% up to 70% for relations with and without this negative sampling technique. The model was evaluated on CoNLL04, ADE and SciERC[64]. The model outperforms previous state-of-the-art (among others also [56] and [60]) on all datasets.

LUKE: Deep Contextualized Entity Representations with Entity-aware Self-attention LUKE by Yamada et al.[65] is technically not a joint model, but a transformer-based model that is capable of doing both named entity recognition and relation classification. It is a pretrained contextualized representation based on BERT. The pretraining is based on masked language models. A main difference from other language models is its entity awareness. Attention scores are calculated differently depending on if the relevant tokens are entities or not. The model input is a token-, position-, and entity-type embedding. For training, entities are randomly replaced with a special "[MASK]" entity and the model tries to predict them. Depending on the final task that the model is used for, different special input entities are defined and different classifiers are set on top of the model.

The model is trained on datasets for entity typing, relation classification, named entity recognition, cloze-style question answering and extractive question answering. The model sets a new state-of-the-art for all tasks compared to other transformers and task-specific baselines.

4 Method

In this chapter the detailed architecture of the model that is proposed as our solution is described. The foundation for our proposed solution is the model proposed by Miwa et Bansal in [50]. First we describe the model input, then we describe the model design and the adaptions made to it, and finally we describe the training process.

The architecture of the model can be seen in figure 4.1. It consists of three general parts. First, part-of-speech tags and dependencies are parsed from the input sentence and its words are embedded using Word2Vec[3] embeddings. Then the inputs are fed to the entity model to produce entity predictions. These predictions are then embedded and together with the word embeddings and dependency embeddings fed to the relation model, which then produces relation predictions. In the following sections the various model parts are described in more detail.

4.1 Model Input

The model processes sentences to extract entities and relations from them. This requires however that certain input processings are applied first and further grammatical features are extracted. The sentence is embedded and part-of-speech as well as dependency information is extracted.

Word2Vec In order to pass the sentence to the model, it needs to be embedded in a vector. One such approach is Word2Vec embeddings[3]. The intuition is that the vector representations of similar words should be close in distance, while non-related words are far apart. To create such vectors, a neural network is trained to predict a word given its context in the corpus. The trained dictionary is then exported and used as the embedding layer. Words are trained on a Continuous Bag-of-Words Model. We train the model on a full English Wikipedia dump as well as all sentences of the respective dataset that our model will also be trained on. The final dictionary is then trimmed down to contain only words used in the training and evaluation dataset in order to save memory.

POS Tagging Part-of-Speech Tagging (POS-Tagging) refers to the task of annotating words with labels indicating the word's part-of-speech as well as further grammatical categories. We use Stanford Natural Language Processing Group's natural language package Stanza[66] to get the POS-tags. Stanza defines 17 different universal POS tags. Each word is annotated with a single tag.

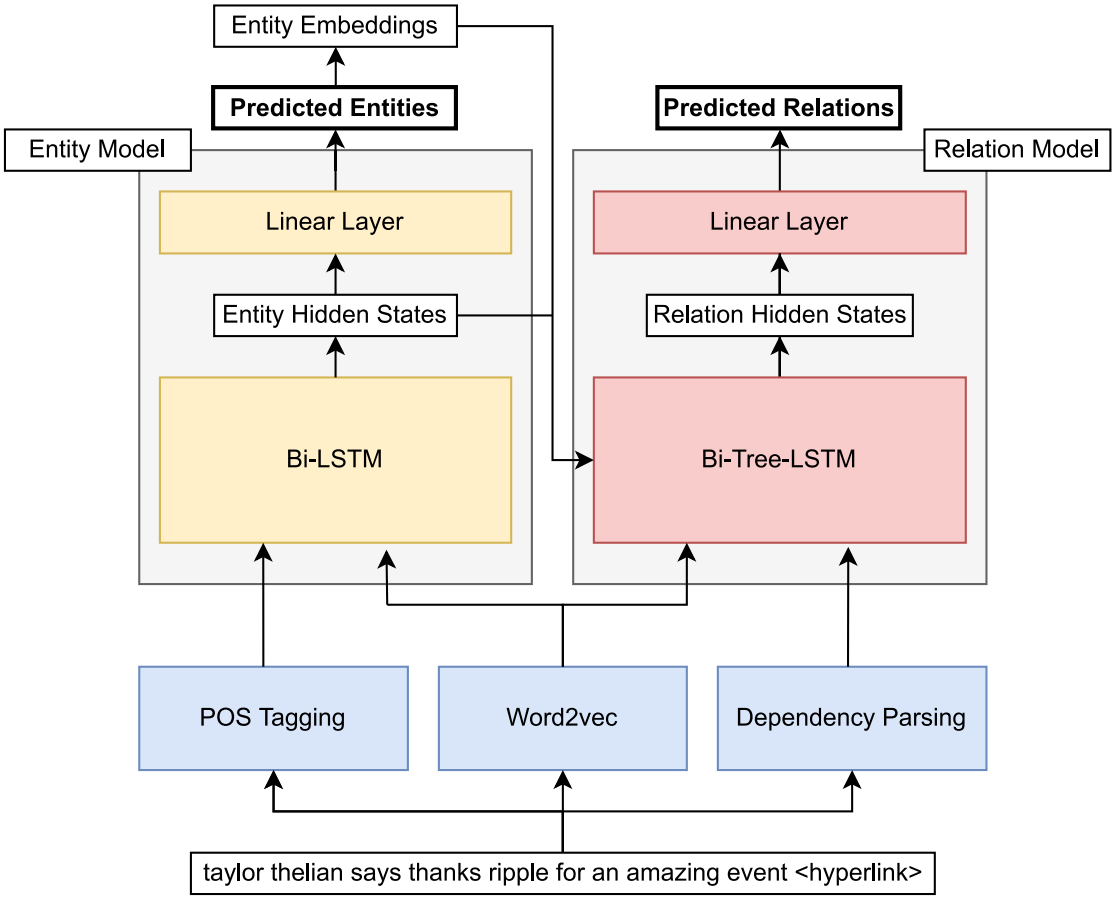

Figure 4.1: Architecture of the model. Part-of-speech tags, dependencies and word embeddings (blue) are extracted from the input sentence (bottom). The embeddings and POS-tags are passed to the entity model, which uses a bidirectional LSTM to calculate hidden states from which entities are predicted. The entities' hidden states, and the word embeddings and dependency tags are passed to the relation model, which calculates hidden states using a bidirectional tree-LSTM and then predicts relations on top of these hidden states.

Dependency Parsing Dependency parsing is a task similar to POS Tagging where instead of POS-tags the dependency relations between tokens in a sentence are parsed. Dependency relations give context about which words refer to which others, which is especially useful in languages where word order is not fixed. For example, given the phrase "The rainy day", both "The" and "rainy" are dependents of the word "day", creating the dependency graph in figure 4.2. Stanza is used again, which uses 66 different dependency types including sub-types. The parsing produces a tree structure which describes the dependency structure of the words in the sentence. Also, a dependency type is given to each token. The types are embedded and used as additional input to the relation model. The dependency tree is used to define the structure of the tree-LSTM which is used in the relation model.

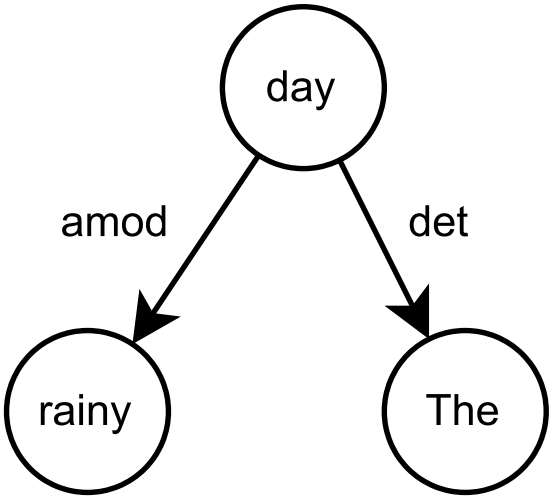

Figure 4.2: Example of the dependency tree of the phrase "The rainy day". "Rainy" is an adjective modifier of the word "day" and "the" is a determiner of "day". This creates a view of the word relations which is decoupled from the word order in the phrase.

4.2 Model Definition

The model consists of two overall parts. After processing the inputs, the first step is the entity model, which recognizes named entities from the input. Then a relation model is stacked on top, which detects relations on top of the entities.

4.2.1 Entity Model

The entity model is directly adapted from Miwa et Bansal's[50] *Sequence Layer* and *Entity Detection Layer*. It performs the task of named entity recognition and consists of two steps: First a bidirectional LSTM is used to generate hidden states from the input

embeddings. These are then fed to a two-layered neural network to predict entities for each token.

Entity LSTM

The word embedding vectors $v^{(w)}$ and POS-embedding vectors $v^{(p)}$ are concatenated to an input vector x, where the t-th word of x is defined as $x_t = [v_t^{(w)}, v_t^{(p)}]$. The input vector x is then passed to a bidirectional LSTM network. The LSTM-cell producing the hidden state of the t-th word is defined by the equations 4.1 to 4.6. In these equations i_t, f_t, o_t, u_t are the input gate, forget gate, output gate and cell gate for word t. Each of these gates has two weight matrices W and U as well as a bias vector b. The term σ denotes the logistic function and $\odot$ denotes the element-wise multiplication operation.

$$i_t = \sigma(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}), \tag{4.1}$$

This equation defines the input gate. Input and previous hidden state are weighted and a logistic function transforms the values into the range between 0 and 1. This function gates which input values are used for calculation. If the values are zero they are ignored, if they are close to one they are maintained completely.

$$f_t = \sigma(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}) \tag{4.2}$$

This equation defines the forget gate. Similar to the input gate, it also transforms input and hidden state to a value between 0 and 1. This value is however used to gate the previous cell state. Simply spoken, it decides based on input and previous hidden state how important the previous cell state is and if it can be forgotten.

$$o_t = \sigma(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)}) \tag{4.3}$$

This equation defines the output gate. This equation is again identical to the two above, however it is used to restrict the final hidden state that is calculated. Given a calculated new cell state, it gates what part of the cell state is transformed to the new hidden state.

$$u_t = tanh(W^{(u)}x_t + U^{(u)}h_{t-1} + b^{(u)}) \tag{4.4}$$

This equation is the second part of the input gate. Input and previous hidden state are transformed by a tanh-function which transforms them to values between -1 and 1. The transformed values are the new input to the cell.

$$c_t = i_t \odot u_t + f_t \odot c_{t-1} \tag{4.5}$$

c_t is the new cell state. It is calculated by point-wise multiplication of the input gates, regulating the input. The result of this calculation is added to result of the point-wise multiplication of the previous cell state and the forget gate. The forget gate's values regulate how much of the previous input is relevant and if it should be forgotten.

$$h_t = o_t \odot tanh(c_t) \tag{4.6}$$

Finally, to calculate the new hidden-state h_t, the cell state is again passed through a tanh-function and then gated again by the output gate. This hidden state is the relevant value which is used for further calculations.

We use these equations to compute the cell state c and the hidden state h. As the LSTM is bidirectional, these equations are applied to each word from both directions, creating the hidden states $\overrightarrow{h}$ and $\overleftarrow{h}$ which are concatenated to a new vector $s = [\overrightarrow{h}, \overleftarrow{h}]$ and used as input for further layers.

Entity Neural Network

To detect entities from the hidden states of the entity LSTM, a two-layered neural network is employed. It takes the concatenated hidden states as well as the embedding of the previously detected entity in the sentence to predict the next entity. Equation 4.7 describes the network's architecture, where W is a weight matrix, b is a bias vector and $v_{t-1}^{(e)}$ is the embedding of the previously detected entity:

$$\begin{aligned} h_t^{(e)} &= tanh(W^{(e_h)}[s_t; v_{t-1}^{(e)}] + b^{(e_h)}) \\ y_t &= softmax(W^{(e_y)} h_t^{(e)} + b^{(e_y)}) \end{aligned} \tag{4.7}$$

The intuition behind using the previously detected entity is that there are dependencies regarding what label can follow after another one. This has to do with the way that datasets label their entities, especially entities spanning multiple words. We use two labelling formats for our experiments, BILOU and BIO[63]. These labelling formats define separate labels for the different tokens that an entity spans. BIO defines "B" for the beginning of an entity, "I" for all other tokens in an entity and "O" for tokens not being part of an entity. BILOU extends this schema by adding two additional labels: "L" for the last token of a multi-token entity and "U" for single-token entities. Further details on the format and which dataset uses which format are given in chapter 6. These formats impose certain restrictions on which label can follow after another. For example, in the BILOU format, a "B"-label can only be followed by an "I" or an "L" and an "I"-label can only be followed by another "I" or an "L". Passing the previously detected entity label allows the model to potentially learn such inter-dependencies and improve its performance.
In the early steps of training, this part of the model might detect many wrong entities and by using wrong previous entity types might also learn incorrect dependencies between the entity labels. In order to alleviate this problem, a technique called scheduled sampling by Bengio[67] is applied. During training, each time an entity is predicted with a certain probability ϵ_i, the ground truth entity label of the previous token is used instead of the predicted label. The probability ϵ_i is computed using an inverse sigmoid decay function given in equation 4.8:

$$\epsilon_i = \frac{k}{k + e^{\frac{i}{k}}} \tag{4.8}$$

Here i is the training epoch and k is a hyperparameter adjusting the probability of using the ground truth labels. The higher k, the higher is the initial probability for the first epoch and the more slowly it converges to zero. Visualizations for different k are given in figure 4.3 depicting this behavior. This technique allows us to use ground truth labels very often at the beginning of training and replace them step-by-step with the predicted labels during further training epochs.

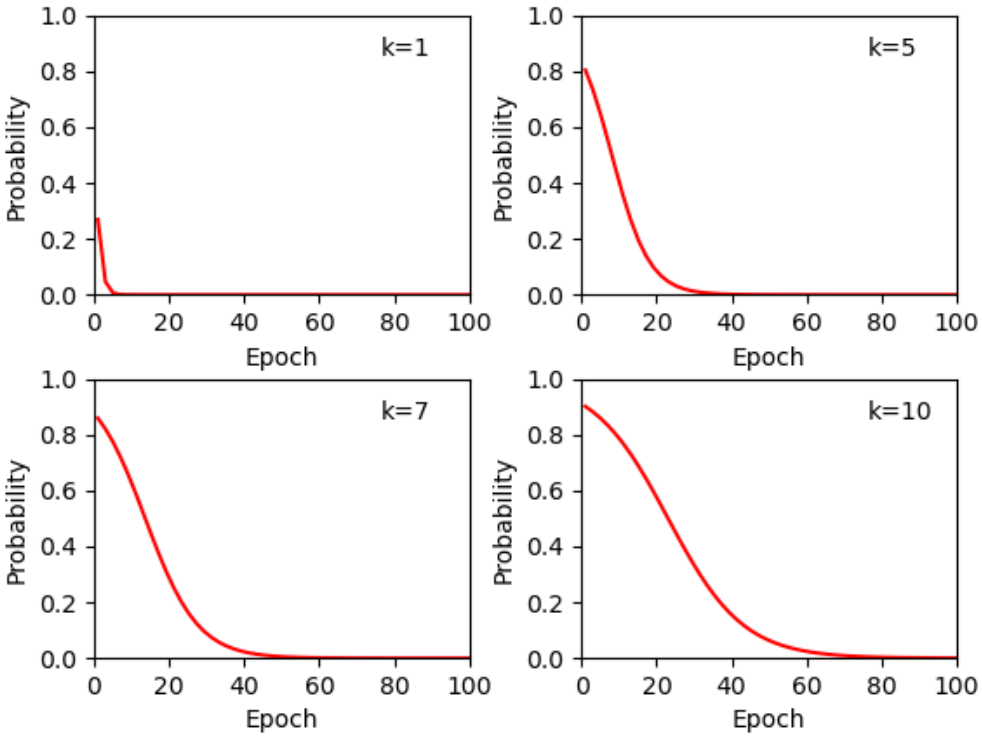

Figure 4.3: Distributions of the probability to apply scheduled sampling to a sample for various values of the hyperparameter k. The graph is plotted over 100 epochs.

4.2.2 Relation Model

The relation model which performs the relation classification is again derived from Miwa et Bansal[50], however, some adaptions are made to the LSTM's structure. The model takes a concatenated vector of word embedding, dependency-tag embedding and embedding of the predicted entities: $x_t = [v_t^{(w)}, v_t^{(d)}, v_t^{(e)}]$. This is already the first difference to the source model, which instead of the word embedding takes the output of the two hidden states of the entity-LSTM. The reason for this change is the assumption that relation classification focuses on different hidden features of the input words than named

entity recognition. For this reason, the word is processed again in conjunction with the dependency- and entity embedding. Hidden states of the entity model are used again in a later step in accordance with the original model. The input is passed to a bidirectional tree-LSTM which computes hidden states, which are in turn passed to another two-layered neural network to predict relations.

Relation Tree LSTM

The tree LSTM used in the original model is an adaption of the *Child-Sum Tree-LSTM* proposed by Tai et al.[68]. The tree structure is built from the dependency relations parsed in the dependency parsing step. The tree nodes are then assigned one of two types: either they are part of the shortest path between the two target entities for which a relation is classified, or they are not part of it. Depending on the type, a different weight matrix is used. The calculation of the hidden states and cell states is similar to normal LSTMs, with the difference that instead of propagating through time from left-to-right and right-to-left, the sum of the parent states or sum of child states (depending on the direction) is propagated. Miwa et Bansal[50] propose three different types of trees: The *SPTree*, which only consists of the shortest dependency path between the two target entities, the *SubTree*, which consists of all nodes below the lowest common ancestor of the two entities, and the *FullTree*, which consists of all nodes in the dependency tree. Because our model's focus is on classifying tweets which have a character limit and therefore also a very limited context, we decided to use the *FullTree*. Furthermore, our model is supposed to be used in a real-time setting, which imposes runtime constraints. For this reason, we also discard the typing of nodes, as identifying whether or not a node is part of the shortest path is a very time-consuming task because it requires us to build the shortest path tree. The resulting formula for the Bi-Tree-LSTM can be seen in equations 4.9 to 4.14 and closely resembles the formula defined by Tai et al.[68]:

$$i_t = \sigma(W^{(i)}x_t + \sum_{l \in C(t)} U^{(i)}h_{tl} + b^{(i)}) \tag{4.9}$$

This equation defines the input gate. The input and the hidden states of all child nodes in the tree structure are weighted and then summed together with a bias. Then a logistic function is applied to transform the values into the range between 0 and 1. Similar to the entity model's LSTM definition this gate gates the input values used for calculation.

$$f_{tl} = \sigma(W^{(f)}x_t + U^{(f)}h_{tl} + b^{(f)}) \tag{4.10}$$

This equation defines the forget gate. Contrary to the input gate, it is defined for each child separately and does not sum up the hidden states, but just uses the respective child node's hidden state and weights it. Apart from this, it is identical to the other gates, taking and weighting the input and summing all parts with a bias before applying a logistic function. It gates the importance of the previous cell state.

$$o_t = \sigma(W^{(o)}x_t + \sum_{l \in C(t)} U^{(o)}h_{tl} + b^{(o)}) \tag{4.11}$$

The output gate is defined by the equation above. It is identical to the first equation but gates how the new cell state is transformed to the new hidden state.

$$u_t = tanh(W^{(u)}x_t + \sum_{l \in C(t)} U^{(u)}h_{tl} + b^{(u)}) \tag{4.12}$$

This equation is the second part of the input gate. Values are transformed between -1 and 1 using the tanh-function and used as input to the cell.

$$c_t = i_t \odot u_t + \sum_{l \in C(t)} f_{tl} \odot c_l \tag{4.13}$$

The new cell state c_t is calculated by point-wise multiplication of i_t and u_t and summation of all point-wise multiplications of the forget gates with the respective child states of the child nodes.

$$h_t = o_t \odot tanh(c_t) \tag{4.14}$$

Finally, to calculate the new hidden state h_t, a tanh-function is applied to the cell state and is gated by the output gate. The hidden state is then used in further steps.

The equations are very similar to the ones defined for the entity LSTM in equations 4.1 to 4.6. The main difference is that the four gate equations 4.9 to 4.12 do not use the hidden state of the previous time-step, but instead use the state of all child nodes of the current node. The states are multiplied with the weight matrix U and then summed. This way the tree structure and thus subsequently the dependency structure from which the tree is built is considered.

Relation Neural Network

From the hidden states produced by the relation tree LSTM we take the hidden states of the lowest common ancestor node h_{pA} and the two entity nodes h_{p1} and h_{p2} and concatenate them to a new vector $d_p = [\uparrow h_{pA}; \downarrow h_{p1} \downarrow h_{p2}]$. In order to also incorporate the features of the entities which currently are only considered by their embedding in the input to the relation tree LSTM, the average hidden states of all words that form an entity are appended to d_p, resulting in equation 4.15:

$$d'_p = \left[d_p; \frac{1}{|I_{p1}|} \sum_{i \in I_{p1}} s_i; \frac{1}{|I_{p2}|} \sum_{i \in I_{p2}} s_i \right] \tag{4.15}$$

Such inputs are computed for each pair of entities. As an entity pair can have multiple words and we do not want to predict an entity pair multiple times by using all of its words, we focus on the last word of the entity. In the BILOU, format this simply means using all words that have an “L” or “U” label as input. In the BIO format we have to consider both “B”- and “I” labels because we do not have explicit labels for entity endings

and single-word entities. The inputs for all possible entity pairs are then fed to another two-layered neural network defined in equation 4.16:

$$\begin{aligned} h_p^{(r)} &= tanh(W^{(r_h)} d_p + b^{(r_h)}) \\ y_p &= softmax(W^{(r_y)} h_t^{(r)} + b^{(r_y)}) \end{aligned} \tag{4.16}$$

This network outputs the final relation labels.

4.3 Training Process

Model training uses Back-Propagation-Through-Time[69] and ADAM[70] as the optimization algorithm. Furthermore gradient clipping[71], weight decay[72] as well as dropout[73] on the embedding and hidden layers is applied. The entity model is trained in isolation beforehand for as many epochs as regular training runs afterwards. This should prevent bad entity predictions that might propagate through the model and impede the training's success.

Loss We use Cross-Entropy-Loss[74] as loss function for training. The entity loss function is weighted with class weights based on the number of class occurrences in the respective training and evaluation dataset. The relation loss is unweighted.

Sample Filtering As a further regularization technique, samples are filtered before the computation of the relation loss is carried out. Experiments are conducted using three different filters on the loss of relation predictions. The initial loss is computed using all possible pairs of words in a sentence. Words which do not correspond to entities are predicted as having no relation. This allows to include all false negative and false positive predictions into the loss computation but also distorts the actual mean loss that is used for back-propagation by the sheer number of negative samples from non-entity words. In order to reduce the distortion, as a second variant, all samples where the prediction as well as the ground truth label, have no relation are filtered out before the loss computation. This filter allows keeping the false negatives and false positives while greatly reducing the impact of pairs containing words which do not represent an entity on the relation loss.
Finally, another filter is applied which removes samples based on their prediction confidence. The intuition behind this filtering is that manually labelled datasets, especially on such noisy and biased data as tweets, always contain incorrect labels. Learning such labels would hurt a model's generalization capabilities. Therefore, they should be excluded during loss computation. The general concept for this approach is taken from Pleiss et al.[75] who discard samples based on their margin, which is the difference between the logit of the assigned label and the largest other logit. This difference is positive and high if the sample is confidently classified correctly and negative if the label is classified incorrectly. The authors claim that a mislabelled sample will have a smaller margin than a correctly labelled sample and use this to filter all samples below a certain threshold for

their margins. We apply a similar filter, however, instead of calculating the logit margin we simply use the confidence of the predicted label. Assuming that the model generalizes a prediction from former training data, it should result in high confidence. However, if the data opposes previously learned features, prediction confidence should be low. We therefore apply a hyperparameter t which acts as a threshold to filter samples based on their prediction confidence.
Given a set of samples S, a set of classes C and a function to get the prediction confidence of a sample for a class $p(s, c)$, the subset S' which is used to compute the relation loss is defined in equation 4.17:

$$S' = \{x \mid x \in S, max_{c \in C}\, p(x, c) > t\} \tag{4.17}$$

We apply this filtering on top of the previous one and compute the loss on the remaining samples in S'.

5 Implementation

This chapter covers the implementation of the model and the accompanying procedures such as training and hyperparameter tuning. First, the POS-tagging and dependency tagging are introduced. Then the model implementation, the training process and finally hyperparameter tuning are described. The whole implementation is written in Python 3. Pytorch[76] is chosen as the general framework for the implementation. Pytorch is a framework for tensor computation and machine learning offering a high-level API and GPU computation capabilities in an imperative style. It is mainly used to develop the actual model implementations and training code.

5.1 Model Input

POS- and Dependency Tagging The proposed model requires part-of-speech tags and dependency tags as well as a dependency tree. These features are extracted from the input sentence using the natural language processing library Stanza[66] developed by the Stanford NLP group. It provides various features for text processing such as tokenization, lemmatization, POS-tagging or dependency parsing. We use both POS-tagging as well as dependency parsing to extract our input features from the sentences.

Word2Vec In order to process data in neural networks, the data needs to be represented as a vector. In order to map the text input to the model proposed in chapter 4, the Word2Vec algorithm[3] is used. The topic modelling library Gensim[77] has algorithms built in to train and use the Word2Vec algorithm, so it was a natural choice to use it here. Using Gensim, a training script is developed to parse a full dump of all articles of the English Wikipedia[1] and then train a *gensim.models.Word2Vec* model on these sentences to produce 200-dimensional word embeddings. The model is then further fine-tuned on the specific dataset that is used by training on the sentences of the dataset. The word-to-index dictionary, as well as the embeddings, are then extracted from the Word2Vec model and used as embeddings in the proposed named entity recognition and relation classification model.

5.2 Model

The model described in chapter 4 is implemented using the Pytorch framework. It is segmented into several self-contained classes of type *torch.nn.Module* which are then

[1] https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2

stacked together to form the complete model. Pytorch models require that each element in an input batch needs to have the same dimensions. The same applies to the output batches. Otherwise a batch could not be stored in a single tensor. As the input length depends on the length of the underlying sentence, it may vary. To counter this problem, the input is padded to the maximum sentence length. For the output, the problem is slightly more complicated. In the entity model, the produced output is padded as well to the maximum sentence length. In the relation model, the output size might however vary depending on the number of predicted entities, as each pair could possibly have a relation. In order to have a constant output size, the relation model outputs a relation for each possible pair of word tokens, including padding tokens. This makes the relation output size independent of the input size or the number of classified entities. Given a sentence s (including the padding words), the size of the relation output is defined as $|s| * (|s| - 1)$. This fixes the output size of the model and allows for implementation using PyTorch's *torch.nn.Module*.
In the following subsections, the separate model parts are described, beginning with the entity model and its parts and then continuing to the relation model.

5.2.1 Entity Model

The entity model consists of two major sub-components: The entity LSTM network and the decoder, which is a two-layered neural network which classifies the entity based on the hidden state of the LSTM. It also contains layers for dropout and embeddings. The model takes the word indices and the part-of-speech-tag indices as inputs, then embeds them, feeds them to the LSTM and finally predicts the actual entities and embeds them again. Algorithm 1 shows the pseudocode for this process.

Data: Indices for words w and part-of-speech tags p in their respective dictionaries.
Result: Entities e and their embedding representations e_{emb}.
1 Embed w and apply dropout;
2 Embed part-of-speech tags p and apply dropout;
3 Apply LSTM onto embeddings to get hidden and cell states h and c;
4 Apply dropout to h;
5 $e \leftarrow \{\}$;
6 **for** h_t *in* h **do**
7 | Apply neural network to h_t and e_{t-1} to get entity predictions e_t;
8 | $e \leftarrow e + e_t$;
9 **end**
10 Embed entity predictions e to get e_{emb};

Algorithm 1: Entity Detection logic

The loop in lines 6 to 9 takes the hidden state and predicts entities based on the hidden state and the previous, detected entity using a neural network. Here, scheduled sampling is applied to swap out the previous entity with the actual ground truth entity with a given probability. The results from the entity model are then returned and passed to the relation model to classify relations.

5.2.2 Relation Model

The relation model contains two sub-components, the tree-LSTM and the two-layered neural network which classifies the relation. The model takes the words, entities and dependency tags as input, then embeds and concatenates them to an input tensor. Then, given the batch size and sequence length, the input is repeated to form an input for all possible combinations of words which will be predicted.
For each combination, the two predicted entities are examined. If any of them is not in the set of labels which could possibly form a relation ("L" and "U" labels for the BILOU format, "B" and "I" labels for the BIO format) the entry is discarded and no relation is calculated. In a later step, the output of these entries is hard-coded to the label that stands for "no relation".
The remaining input samples are then batched together and passed to the bidirectional tree-LSTM to compute the hidden states for each word. The tree structure for each sentence is encoded as an adjacency list of the tree nodes. From this adjacency list, an evaluation order for the nodes and edges of the tree is derived. The cell- and hidden states are then iteratively computed bottom-up and top-down. The implementation of the tree-LSTM including the adjacency structure is based on a Github Repository[2] which implements the *Child-Sum Tree-LSTM* by Tai et al. [68] and is adapted to suit our proposed model.

For each word pair, the corresponding hidden states, as well as the hidden state of the lowest common ancestor word from the tree-LSTM, are extracted and concatenated. Additionally, the average hidden state from the entity-model for each entity is concatenated as described in equation 4.15. The concatenated tensor is then passed to a two-layered neural network to classify the relation for each pair. The classified pairs are inserted into the result tensor, which already contains the hard-coded results for the pairs which cannot have a relation due to not having the correct entity labels.

5.3 Training

Training logic is also implemented directly in Python. Individual training scripts are developed for each dataset, however they are all similar apart from dataset-specific details, such as the entity labelling format. The training data is loaded from the dataset, which is implemented in a separate class that inherits from *torch.utils.data.Dataset*. In order to save time during training, the POS-tags, dependency tags and adjacency lists as well as some further structures derived from the dependency list are pre-computed when initializing the dataset. Usually, in a real setting, these values would be computed by the model when needed, but as the training is repeated over several epochs it would require the model to recompute these values for all samples each epoch, which is redundant.
The general setup for a training script is to first load all configurations and the dataset as well as the dataset-specific Word2Vec embeddings. Then the model, the loss function and the optimizer are initialized. After the initialization, the entity model is pre-trained

[2] https://github.com/unbounce/pytorch-tree-lstm

for several epochs before the actual training begins.

Data: Dataset, model.
Result: Trained model.
1 **for** *(sample, labels) in dataset* **do**
2 | Update correct labels in model with *labels*;
3 | Predict entities and relations for *sample* with *model*;
4 | Calculate entity and relation loss from entity and relation predictions, and labels;
5 | Reset gradient on optimizer;
6 | Back-propagate the loss;
7 | Clip gradient norm;
8 | Perform optimization step;
9 **end**

Algorithm 2: Training loop

Line 4 takes the predictions, labels and loss function definitions and calculates the loss separately for the named entity recognition and relation classification. For relation classification, the additional filters are implemented before loss calculation occurs. The filter implementation and loss calculation for the relation classification are shown in algorithm 3.

Data: relation predictions r, relation labels l, filter index f, loss threshold t, loss function $loss\_fn$
Result: Filtered loss $loss$
1 Remove all predictions of value f from r ;
2 Remove all predictions of value f from l;
3 Remove all values with max probability $< t$ from r;
4 Remove all values with max probability $< t$ from l;
5 Apply $loss\_fn$ to r and l ;
6 Apply mean to loss;
7 **return** loss;

Algorithm 3: Relation loss calculation

This loss is summed with the loss of the entity predictions and is used for backpropagation-through-time in the training loop (algorithm 2).

Hyperparameter Tuning To find optimal configurations for the model, we perform hyperparameter tuning using the framework Optuna[78]. Optuna offers a simple API to run optimization studies and allows to conduct multi-objective studies. The framework also offers various types of parameter samplers. Due to limited resources in computing power, it was decided to use a multi-objective tree-structured Parzen estimator[79] as the sampling method. For limited resources, this is favorable over simpler approaches such as grid search or random sampling. The tree-structured Parzen estimator favors configurations that it estimates close to the optimal solution and for this reason needs to try fewer different configurations to converge to an optimal one.

For each model that is evaluated in chapter 6, a study is conducted on each dataset. Optuna requires an objective function which returns a score to optimize, an optimization direction and a number of trials. As objective functions, the training script logic for each model and dataset combination is wrapped into a callable and edited to return the performance metric. Finally, the calls to load the hyperparameters are replaced with methods by Optuna to set them from a predefined search space. An example of loading such parameters in the context of a trial is given in listing 5.1.

```
...
learning_rate = trial.suggest_categorical(
    "learning_rate",
    [5e-3, 2e-3, 1e-3, 5e-4, 2e-4, 1e-4]
)
weight_decay = trial.suggest_categorical(
    "weight_decay",
    [1e-4, 1e-5, 1e-6, 1e-7]
)
optimizer = torch.optim.Adam(
        model.parameters(),
        lr=learning_rate,
        weight_decay=weight_decay
    )
epochs = trial.suggest_int("epochs", 5, 25)
...

```

Listing 5.1: Setting parameters with Optuna. The parameters after the variable name define the range from which parameter values are chosen.

The results of the trials are saved in a MySQL database and the top configurations are extracted and used for the experiments and final evaluation.

6 Experiments and Results

In this chapter the previously described model is evaluated. We first describe the datasets which we use to evaluate our model and the hyperparameter tuning process. Then we show the baselines against which the model is compared. Finally, the results of the experiments are presented.

6.1 Datasets

To evaluate the proposed model, it is tested on multiple datasets. The main focus lies on the performance when applying it to domain-specific tweets, but to make it more comparable and show how the model components perform in isolation, we tested it on three common datasets as well.

6.1.1 Tweet Dataset

The goal is to develop a model which can successfully extract entities and their relations from a stream of tweets. As applications of such a model would be very domain-specific, the tweets would most likely be domain-specific as well. As we could not find a suitable dataset to fulfill these requirements, a new dataset is labelled. The dataset is restricted to the domain of cryptocurrencies and contains 1185 tweets that were written between July 31, 2020, and August 15, 2020. The tweets are taken from a MongoDB database provided by Cortecs. The database is constructed from the raw input stream of tweets fetched from the Twitter API[1] with Tweepy[2].

The fetched tweets are filtered by a spam-detection system based on Random Forest classifiers trained on a supervised dataset. The original design is based on Twitter Weighted Logistic Regression and Support Vector Machines by Hinterndorfer[80]. Then a story-detection system is applied to find tweets which belong to a news story or similar event. The story-detection is an adapted version of Petrovic et al.[81], where the locality-sensitive hashing is replaced by the IVF_FLAT algorithm of the Milvus project [82].

Next, the text is extracted from the tweet and preprocessed. It is lowercased, numbers and hyperlinks are masked and special characters are removed. Finally, the username of the tweet author is added to the beginning of the text in the form 'User says'. This allows for tweets which focus on a single entity and lack a second entity to form a relation with the author. An artificial example to show this is given in figure 6.1.

[1] https://developer.twitter.com/en
[2] https://www.tweepy.org/

Labels

After examining the content of the tweets and identifying potential entities and relations, a set of labels for the annotation is defined. We find six labels for entities and two for relations. Entity labels are inspired by the CoNLL 2003 dataset[83], which is further described in 6.1.3, and extended by some more entities which fit the financial context of the tweets. The entity labels are *person*, *group*, *company*, *exchange*, *geopolitical-entity* and *asset*.
Person is any natural person. This can be an individual person identified by their name, their profession, their title or any other description making clear that it is a natural person.
Group is any group of natural persons or institutions. This can for example describe a population, a profession or any other aggregation of entities.
Company is any company, either described by name, field of operation or other description making clear that it is a single company. Non-profit organizations also fall under this category.
Exchange is a specific type of company. It describes an institution which exchanges assets. This is mostly used for cryptocurrency exchanges but can also mean any other company which provides trades, such as a stock exchange.
Geopolitical-entity is any political structure which is associated with some geographical area. Examples of this would be cities and their administration, countries and similar governmental structures.
Asset is anything which can be invested in and possesses financial value. Examples are cryptocurrencies, cash, stocks or natural resources such as oil or gold.
The corpus contains a wide variety of complex as well as very unspecific relations. This is why only two very broad labels were chosen for relations, namely *supports* and *opposes*. *Supports* signifies a positive effect, sentiment or action from the head entity to the tail entity, while *opposes* signifies a negative one.

Annotation

The sentences are labelled using the BRAT rapid annotation tool [6]. BRAT is a self-hostable, web-based annotation tool to create datasets for various NLP tasks. The focus lies on high configurability and simplicity of use to reduce annotation time. An example of the tool can be seen in figure 6.2. The labelling is conducted by three annotators with various degrees of domain knowledge, one of which is the author of this thesis. The author's annotations are taken as a baseline and further refined through comparison with the other annotators' labels. The tokens are labelled in the BILOU format[63]. As explained in chapter 4, this format defines five different possible tags for a token: "B", which is the first token in a multi-token-entity, "I", which is a token inside a multi-token-entity, "L", which is the last token in a multi-token entity, "U", which is a single-token entity, and "O", means that the token does not belong to an entity at all. [63]

The statistics of the final dataset can be seen in tables 6.1 and 6.2.

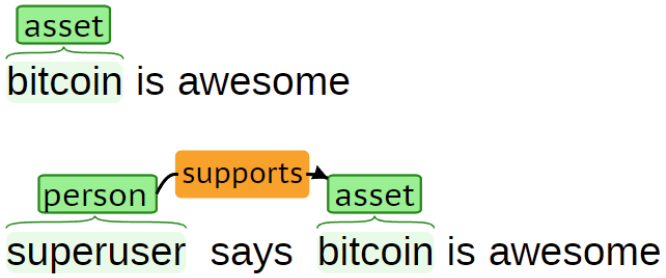

Figure 6.1: Example of how prepending the author to the tweet text can create relations containing additional information which would otherwise be lost. The first sentence is the original text containing a single entity. The second sentence is the same text prepended by the author. This creates a second entity, which allows forming a relation between these two entities.

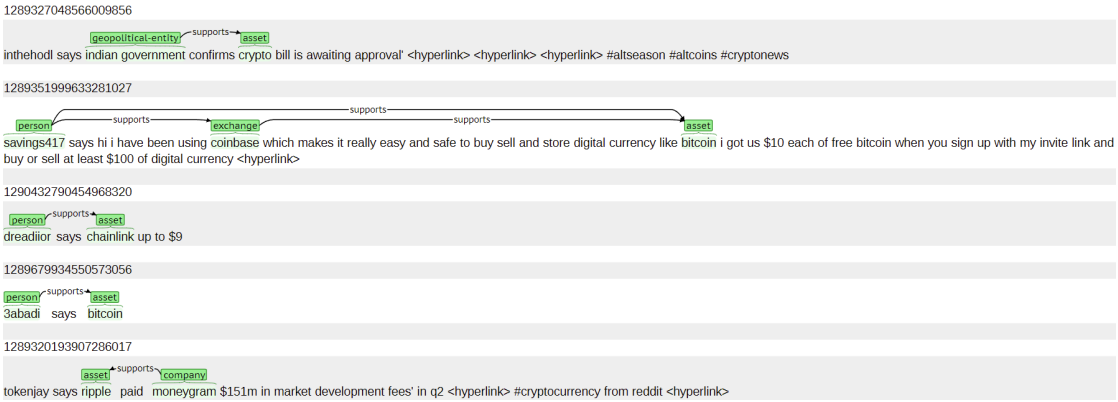

Figure 6.2: Example of the user interface of the BRAT rapid annotation tool[6].

6.1.2 CoNLL 2004

The CoNLL04 dataset[1] is a dataset that is based on articles from various news sources, such as Wall Street Journal or Associated Press. It defines the entity labels *persons*, *locations*, *organizations* and *others* as well as the relation labels *located_in*, *work_for*, *orgBased_in*, *live_in*, *kill* and *none*. The entities are provided in the BIO format [84], which is also described in chapter 4. This format is similar to the BILOU format, but has only three types of tags: "B", which stands for the starting token of an entity, "I", which stands for a non-starting token of an entity, and "O", which stands for a token that does not belong to an entity. The dataset consists of 1437 sentences with 5336 entities and 19048 possible relations, most of which belong to the class *none*. The exact numbers can be found in table 6.3.

Sentences	1185
Tokens	24004
Maximum sentence length	55
Mean sentence length	20.26
Entities	2511
Mean entities per sentence	2.12
Relations	1535
Mean relations per sentence	1.30

Table 6.1: Statistics of the tweet dataset.

Labels	Absolute occurrences	Relative occurrences
Person	346	13.78%
Group	195	7.77%
Company	541	21.55%
Exchange	108	4.30%
Geopolitical-Entity	150	5.97%
Asset	1171	46.63%
Supports	1219	79.41%
Opposes	316	20.59%

Table 6.2: Distribution of entities and relations of the tweet dataset.

6.1.3 CoNLL 2003

CoNLL 2003[83] describes a shared task of named entity recognition on two datasets, one in German and one in English. For our evaluations, we use the English dataset, which contains 301418 tokens over 22137 sentences. The task defines four different entities: 10645 *locations*, 9323 *organizations*, 10059 *names of persons* and 5062 *miscellaneous* entities that belong to neither of the other types. Performance is evaluated using the F1-score.

6.1.4 SemEval 2010 Task 8

SemEval 2010 Task 8[52] is a relation classification task which provides a dataset of 10717 English sentences. For each sentence two tagged nominals are given for which a relation must be found, together with its direction. The possible relations, including their frequencies, are 1331 *Cause-Effect*, 1253 *Component-Whole*, 1137 *Entity-Destination*, 974 *Entity-Origin*, 948 *Product-Producer*, 923 *Member-Collection*, 895 *Message-Topic*, *732 Content-Container*, 660 *Instrument-Agency* and 1864 *Other*. Each relation, except for *Other*, is directional, meaning it can appear in both directions, which need to be classified correctly as well. The official metric of the task is the macro-average F1-score ignoring the *Other* relation but taking directionality into account.

Label	Absolute occurrences	Relative occurrences
`persons`	1685	31.58%
`locations`	1968	36.88%
`organizations`	978	18.33%
`others`	705	13.21%
`located_in`	406	2.13%
`work_for`	394	2.07%
`orgBased_in`	451	2.37%
`live_in`	521	2.74%
`kill`	268	1.41%
`none`	17007	89.28%

Table 6.3: Distribution of labels in the CoNLL04 dataset[1].

6.2 Hyperparameter Tuning

To find the optimal model configuration, hyperparameter tuning is performed. 8 hyperparameters are tuned on two objective values. The goal is to find ideal values in specific ranges for these 8 parameters such that both the performance of detected entities as well as detected relations are maximized. The 8 parameters are:

- *Dropout Rate*: The probability that a value is replaced with zeros
- *Epochs*: Number of training iterations
- *Gradient Clip*: The value at which the gradient is clipped
- *Hidden Nodes*: Size of the hidden features of both LSTMs
- *Learning Rate*: Step size of the gradient descent
- *Scheduled Sampling k*: The parameter k for scheduled sampling used in equation 4.8
- *Weight Decay*: The factor by which the model weights which are incorporated in the loss are reduced in each back-propagation step
- *Loss Confidence Threshold*: The minimum confidence for a sample to appear in the loss (see equation 4.17)

The tuning algorithm uses a multi-objective tree-structured Parzen estimator[79] to find the most optimal parameter configurations. A tree-structured Parzen estimator builds a probability model from previous parameter trials and uses it to suggest configurations for future trials. Given a threshold, the model builds two distributions, one from samples which are below the threshold and one from above. It then tries to maximize the expected improvement of the final score by drawing configurations from the favorable distribution and trying to find ones that maximize the difference between the favorable and unfavorable distribution. With a longer trial history, the distributions become more accurate and the

model is able to derive better parameters.

Hyperparameter tuning is performed individually for each dataset. Table 6.4 shows the parameters and their ranges.

Parameter	Range
`Dropout Rate`	[0.0 ,0.1, 0.2, 0.3, 0.4, 0.5]
`Epochs`	5-25
`Gradient Clip`	[5, 10, 50, 100]
`Hidden Nodes`	[50, 70, 100]
`Learning Rate`	[0.0001, 0.0002, 0.0005, 0.001, 0.002]
`Scheduled Sampling k`	[5, 10, 50, 100]
`Weight Decay`	[1e-4, 1e-5, 1e-6, 1e-7]
`Loss Confidence Threshold`	0.15-0.20

Table 6.4: Hyperparameter ranges for the proposed model.

Due to resource constraints, each parameter study consists of 32 trials at most. Most parameters were chosen similar to Miwa et Bansal[50], with the exception of the epochs, whose upper bound was drastically reduced from 100 to 25. The trials were performed on an AWS g4dn.12xlarge instance[3] which has 4 NVIDIA-T4-GPUs and 192GB of memory. In addition to the optimal parameters, Optuna also computes the importance of each parameter that was optimized using functional ANOVA[85]. This approach fits a random forest to predict the objective function value of the parameter optimization given the parameters. Then functional ANOVA is carried out on the random forest to quantize parameter interactions and importances. The importances for our model on the tweet dataset are shown in figure 6.3 and the importances for CoNLL04 are shown in figure 6.4. The tweet dataset is mainly sensitive to the learning rate and the dropout rate. This suggests that great care should be taken when choosing these two parameters. This makes sense when comparing them to the other parameters in the figure and looking at the dataset. Some tweets in the dataset are written by the same authors, which might make them quite similar. An unfortunate train-test-split might encourage overfitting due to having more similar samples in the training data, and dropout tackles this and prevents overfitting [73]. The least important parameter is gradient clipping, which suggests that exploding gradients are not a problem for this dataset. This becomes especially visible when compared to the CoNLL04 dataset in figure 6.4, where gradient clipping is the second most important parameter. The reason for this becomes evident when looking at the maximum sentence length of both datasets. The longest sentence in the tweet dataset has 55 tokens, while the longest in CoNLL04 has 119 tokens. More tokens in the sequence lead to longer back-propagation-through-time, which favors exploding gradients. This effect is tackled by gradient-clipping.[71] It should also be noted that the figure of the tweet dataset does not have a bar for the number of epochs. This is due to a change in

[3] https://aws.amazon.com/de/ec2/instance-types/g4/

the parameter range, which the framework cannot handle when computing importances. When looking at the importance of the epochs in the CoNLL04 dataset and the number of epochs in the paper by Miwa et Bansal[50] who use 100 as the upper bound, it becomes evident that epochs should also play an important role in the tweet dataset. We therefore assume that the obtained results could still be further refined with a larger number of training epochs, which unfortunately was not possible in our experiments due to resource constraints.

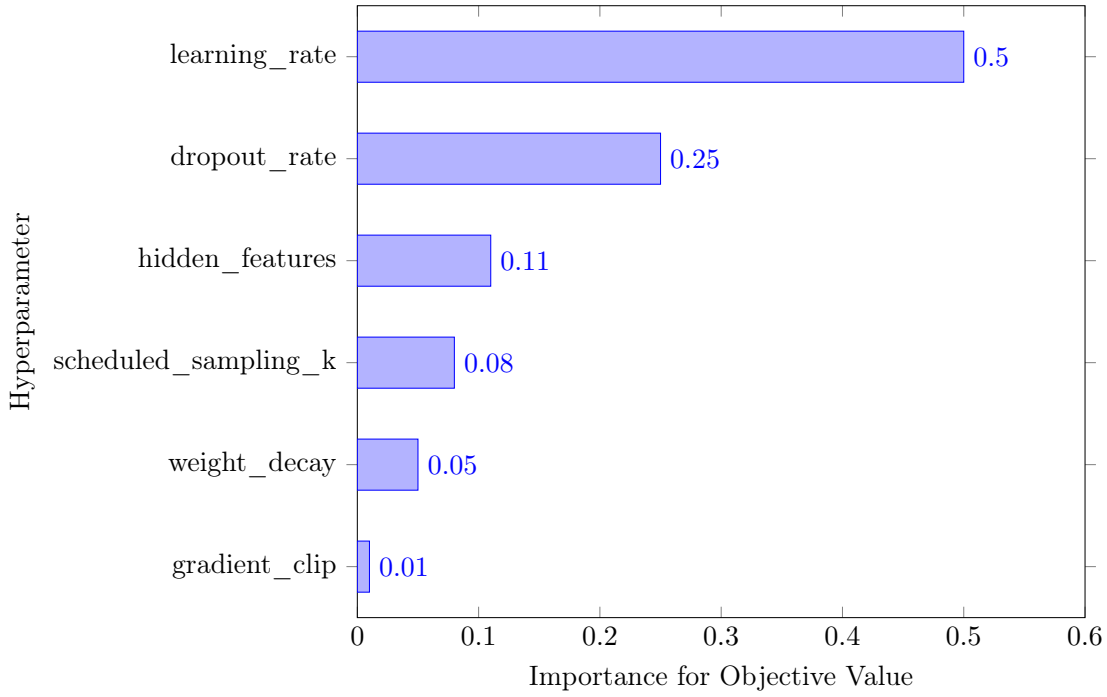

Figure 6.3: Parameter importance on the tweet dataset.

6.3 Baseline

When looking for a baseline to compare the model to, the following factors were considered. The baseline should not be a joint model because one of the goals is to find out if joint models have advantages over separate ones. The baseline should also provide good performance on the individual tasks of named entity recognition and relation classification. Lastly, the underlying architecture should be similar to the proposed model. When looking for state-of-the-art performance in the two relevant tasks, most benchmarks are led by transformer models[86]. To make implementation easier, the huggingface transformers[4] library is used, which provides pre-trained transformer models to download and fine-tune. For the task of named entity recognition, an uncased version of BERT[37]

[4]https://huggingface.co/docs/transformers/index

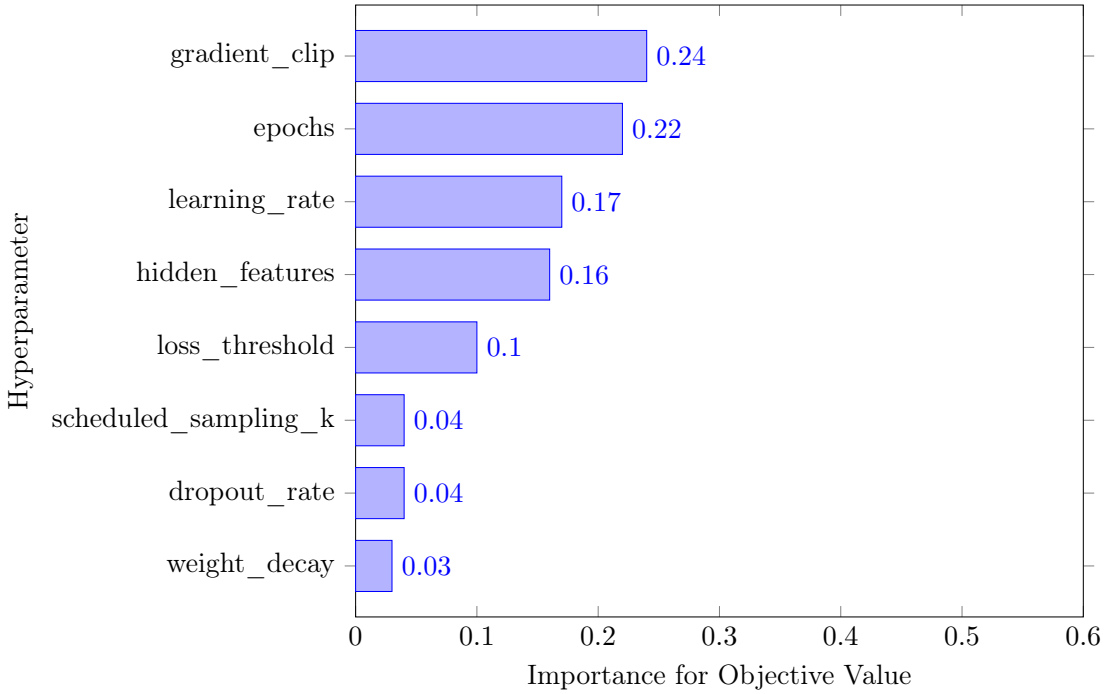

Figure 6.4: Parameter importance on the CoNLL04 dataset.

with an additional linear layer for token classification is used. For relation extraction, LUKE[65] with an additional linear layer for entity-pair-classification is used. These models are separate and have very good performances reported on their respective tasks. They are however based on a different architecture than the proposed model, which uses Long Short-Term Memory Networks. In order to compare different models of the same architecture and therefore show the potential improvements that the various design choices and optimizations of our bring, a simple model based on LSTMs is necessary. For this reason, another model based on bidirectional LSTMs is used as a second baseline. Named entity recognition is performed by a model consisting of an embedding layer, a bidirectional LSTM layer and a linear layer for classification. Dropout is used after the embedding and linear layer. The relation classifier is similar, but as additional input the relative position for the two input entities is used as well. The named entity recognition is implemented in Pytorch, while the relation classification is done in Keras.

In both cases, the two model parts are trained separately. First the named entity recognition model is trained and evaluated. The final predictions of the training and evaluation dataset from this are used as inputs for training and evaluation of the relation classifier. In order to handle class imbalances, the individual losses are weighted by the number of samples of each class in the ground truth labels. Performance metrics are calculated identically to the proposed model, where a relation is only classified as correct if both entities' last tokens are correct and the relation itself is correct.

6.4 Evaluation

The model is evaluated against the previously described baseline on both the tweet dataset as well as CoNLL04. To evaluate the performance on individual tasks, all models are also evaluated against CoNLL03 for named entity recognition and SemEval2010 Task-8 for relation classification. We first present the performance comparison of the model against the baseline on the evaluated datasets. Then we present the results of an ablation study using the different filters for the loss calculation. We also verify that the model's runtime is sufficient for real-time processing. Finally, we conduct a qualitative evaluation of the best model on unseen real-world data.

6.4.1 Performance

As performance metric, the macro-average F1-Score is chosen. The F1-Score is the harmonic mean of precision and recall. Precision is the number of true positive samples of all positives. Recall is the number of true positive samples of all true samples. Precision, recall and F1-score are defined in formulas 6.1, 6.2 and 6.3. The macro-average F1-score is the unweighted mean of the F1-scores of all positive classes.

$$precision = \frac{true\,positive}{true\,positive + false\,positive} \tag{6.1}$$

$$recall = \frac{true\,positive}{true\,positive + false\,negative} \tag{6.2}$$

$$F1\text{-}score = \frac{2 * precision * recall}{precision + recall} \tag{6.3}$$

The results can be found in table 6.5. Our proposed model outperforms both baselines in terms of relation score on both datasets while being inferior to the transformer in terms of entity score. On the tweet dataset, the relation classification score is better by 5.5 percentage points, while the transformer has 14.9 percentage points better score on entity recognition. On CoNLL04 the difference is even more pronounced, with our model outperforming transformers by 36.2 percentage points while being outperformed on entity recognition by 18,8 percentage points. The superior performance of the transformer on named entity recognition is not surprising, given that at this stage the models get the same error-free input. This means that the task of entity recognition can be considered in isolation, and for this task alone BERT is superior. The LSTM baseline performs slightly worse than our model. This can be explained by the optimizations implemented by Miwa et Bansal[50] that we take over into our model, such as scheduled sampling[67] or the greedy entity assignment which are both not implemented in the LSTM baseline.

These results are supported by the scores on CoNLL03 in table 6.6, where BERT outperforms the proposed model as well and the LSTM is outperformed by both models.

However, when looking at relation classification, our model outperforms transformers on both datasets, while being inferior when looking at the task in isolation. This can be seen in table 6.7, which shows results for the SemEval2010-Task 8 dataset. These results

suggest that the proposed model is able to benefit from the additional context given by the hidden states of the entity model when predicting the relation between these entities.

In the case of the LSTM baseline, both scores are inferior for both datasets. While the differences in entity scores are small, the relation classification scores differ by 13 percentage points and 49.2 percentage points respectively. When looking at the scores on CoNLL03 in table 6.6, we can see that the proposed model also outperforms the LSTM baseline when doing named entity recognition in isolation. This suggests that our model's entity scores suffer from joint training, while relation scores benefit from it.

Furthermore, both baselines do not distinguish entity types in their inputs. This information is also an advantage that our model has which can be especially critical, as most relations have certain semantic limitations in terms of which entities can occur as head or tail. For example, when looking at the relation "kill" from CoNLL04, it is not possible to kill a location, hence this entity can never occur as a tail entity for the relation "kill". Such interactions between entity types can only be found by the transformer model when processing the semantics of the actual sentence, but not structurally by the type of the passed entity.

Model	Tweet Dataset		CoNLL04 dataset	
	Entity F1	Relation F1	Entity F1	Relation F1
Transformer	**47.6%**	15.5%	**86.3%**	15.4%
LSTM	32.5%	8.0%	66.8%	2.4%
Proposed Model	32.7%	**21.0%**	67.5%	**51.6%**

Table 6.5: Macro F1-scores of our proposed model compared to the baselines.

Model	F1-Score
BERT*[37]	**92.4%**
LSTM	74.4%
Proposed Model	84.4%

Table 6.6: F1-Scores on the CoNLL03 dataset. * means that the result was taken from the original paper.

Model	F1-Score
LUKE[65]	**87.9%**
LSTM	56.4%
Proposed Model	71.6%

Table 6.7: F1-Scores on the SemEval2010 Task 8 dataset.

6.4.2 Ablation Study

In order to see the impact of the changes performed on the loss calculation, an ablation study is performed comparing three different versions of the proposed model on the tweet dataset and CoNLL04. The first version is the proposed model, where the samples which are considered for loss calculation are filtered out if both the prediction and the actual label are the negative relation. Simply spoken this means that a sample is discarded if the sample is correctly predicted as not being a relation at all. Additionally, all samples whose confidence is below a certain threshold t are discarded as well. The threshold t is a hyperparameter which is tuned separately. The second version discards the threshold filter and only filters the negative samples. The third version does not filter any samples at all. The results are presented in table 6.8.
Looking at the performances on the relation score for both datasets, we can observe performance improvements of the proposed model compared to the other versions. On the tweet dataset, it outperforms the best model without a threshold by 2.1 percentage points. The version where all losses are kept and nothing is filtered out is even outperformed by 14.5 percentage points. On the CoNLL04 dataset, where scores are generally higher, the model without a threshold is outperformed by 2.8 percentage points and the model that does not filter loss is outperformed by 49.3 percentage points. One thing that stands out when comparing the proposed model with the version that does not use sample thresholds for the loss is that while the relation classification score improves, the entity recognition score decreases.

One possible explanation for this could be that the loss of the relation model that gets propagated back through to the entity model (via the predicted entities and the passed hidden state of the entity model) is reduced, as low confidence relations are filtered out. Low confidence samples might occur when incorrectly predicted entities contradict other input features, which prevents the model from making a confident prediction. Such low confidence predictions would have an impact on the entity predictions, however as the proposed model filters out low confidence samples below a certain threshold, these effects are weakened. Such effects can however not be observed on the CoNLL04 dataset.

Model	Tweet Dataset		CoNLL04 dataset	
	Entity F1	Relation F1	Entity F1	Relation F1
Proposed Model	32.7%	**21.0%**	67.5%	**51.6%**
No threshold (best entity)	**44.3%**	15,5%	67.4%	48.8%
No threshold (best relation)	34.4%	18.9%	NA	NA
No Loss filtering	39.3%	6.5%	**68.3%**	2.3%

Table 6.8: Macro F1-scores of the ablation study. During hyperparameter tuning for the tweet dataset, two optimal configurations were found, one with a better entity score and one with a better relation score. For the sake of completeness, both scores are reported here. For the CoNLL04 dataset, one trial had the best scores for both metrics, therefore the second entry is filled with NA.

6.4.3 Runtime Analysis

Applying our model in the social media domain requires us to process data quickly, as it is usually outdated after a short time. To ensure that our model works in a streaming setting processing it in real-time, we measured its runtime during inference over 50 random tweets. We take the average of 100 runs. The model was tested on a Desktop PC with 16GB of RAM and an Nvidia RTX 2080 GPU. It takes the model on average 21.7 seconds to process these 50 tweets. This means that we process on average 2.3 tweets per second. While this is only a fraction of the tweets that are published daily[5], if we apply spam filtering, use multi-gpu machines and only restrict ourselves to a sub-domain of tweets, the performance is most likely perfectly acceptable for real-time processing.

6.4.4 Qualitative Evaluation

In order to assess the model's usefulness in a real-world scenario, in addition to the quantitative approach of evaluating the F1-score, a qualitative analysis of the model is performed. To do so, the model is given unseen real-world sentences and its predictions will be judged by their quality and usefulness. The sentences are taken from the same database used to create the tweet dataset which is described in 6.1.1. We discuss some general findings in the predictions and showcase these on selected tweets.
Our first finding is that the final model is quite reserved in predicting any positive relations. 27 of the 50 sentences have no relations found in them, despite 47 of them having at least two entities. If a relation is found, it is correct most of the time if the entities are correct or at least close in meaning. For example, if a company and exchange or a company and a person are wrongly predicted, the overall meaning of the relation that exists between this entity and another can still be similar and correctly predicted, while for example mistaking a person for an asset would create a completely different meaning leading to a higher chance of wrong relations. A very common mistake is that the model misses entities completely. This can be attributed to the Word2Vec[3] embeddings. If a word is not already in the dictionary of known words, there exists no embedding for this word. So as a default case a random vector is added to the embedding for this word instead. Usually it is possible to further train this vector, but in this case only inference and no training happens, so the random vector is used as-is. The used sentences are new to the model and might contain such unknown words, which could explain that entities are overlooked.

Finally, we show what the model is capable of, as well as the mistakes it makes in four examples. The examples are given in figure 6.5.

The first sentence showcases already a lot of mistakes as well as some correct classifications. The model correctly inferred that the author *johmorganFL* is most probably a person. The second found entity *South Korean City* is classified as a group. This should be considered a geopolitical entity. The model might interpret the phrase as the group governing members of the city, however it is still incorrect. The third entity is

[5]Several pages estimate the number of tweets per second in the magnitude of multiple thousands, see for example here: https://www.internetlivestats.com/twitter-statistics/

Figure 6.5: Predictions of the model on example sentences.

#Blockchain-Based and this is already close to an edge case. The word is considered an asset and the term *#Blockchain-Based* does not define a specific asset or cryptocurrency. However the model is trained to find relations regarding crypto assets and the mentioned blockchain-based payments are the closest thing to such a cryptocurrency and overall most probably refer to an actual cryptocurrency used, so the label is not completely incorrect, though it might also add the word *Payments* as part of the entity. This becomes even more evident when looking at the relation between *South Korean City* and *#Blockchain-Based*, which is *supports*. This makes sense semantically as the city supports the blockchain-based approach (or the respective cryptocurrency built on top) by using it as a payment service. Finally, a possible entity that the model missed could be *The Elderly*, which would be a group. This would also allow a possible *supports* relation between *South Korean City* and *The Elderly*, because the city is actively pursuing an approach to help elder citizens with this payment system.

The second sentence contains fewer words. Here the model correctly identified three entities: The author *Alt_Cash*, *LBRY* and *Youtube*, all identified as companies. The author of this sentence is another edge case because it is not clear if this is a person or a company. Inspecting the twitter profile[6] and the affiliated website shows hints of it belonging to a person, such as the description term "Crypto Enthusiast", but it is not clear if this is a purely private channel. Furthermore, there is the following sentence on the website (visited 05.04.2022 10:28), which suggests that this is an organization run by multiple people: "We're always looking for more faucets and airdrops, submissions are welcome @Alt_Cash". As all this information is not available to the model, it is reasonable that mistakes happen when predicting the author. Both other entities are correctly classified as companies, and the relation *opposes* between them can also be considered correct and relevant because the tweet's content suggests that LBRY is a direct competition to Youtube with a business model based on the user being paid to watch

[6] https://twitter.com/alt_cash

videos.

The third sentence again contains three entities, which are all correctly classified. *Matt_ _Hussey*, the author is classified as a person, *Bitcoin* is classified as an asset and *China* is classified as a geopolitical entity. Here, another problem becomes evident. The sentence contains no hints that the relation *supports* between *China* and *Bitcoin* is correct. In general, the entity *Bitcoin* should rather have been *Bitcoin miners* and be a group instead of an asset. But even if labelled as such, no unambiguous conclusion can be drawn. It is not clear if the miners lost access to cheap electricity because of actions taken by the republic of China or if this situation is independent of the state and just happened there geographically. It is also unclear if China favors this situation, therefore potentially opposing the miners. Finally, it is also unclear if the author supports or opposes the fate of the miners in China or the country in general.

The last tweet consists of two sentences and is slightly longer. Two entities are found: The author is labelled as a company and *Bitcoin* is labelled as an asset. Both entity labels seem correct, even though it was not possible to verify that *tron_ age* is indeed a company, as at the time of writing the account did not exist anymore. Regarding the relation it becomes clear that while entities and direction are correct, the type of the relation is incorrect. The author does not support bitcoin but rather opposes it, suggesting that it would be better to buy cheap shares of the UK than to invest in bitcoin. This could show a potential bias that the model has. Most tweets in the crypto-domain which revolve around bitcoin and contain terms such as "investing" or "buying" are in support of the currency, and only very few oppose it. While there are tweets having negative relations involving bitcoin or other currencies, they are usually much fewer. This way the model has generally a more positive attitude towards bitcoin or other currencies. Adding more negative tweets to the dataset could reduce such bias. The second part of the sentence also contains the term *UK shares*, which can also be considered an asset. This entity would actually form a *support* relation with *tron_ age*.

The above examples show that the domain contains a lot of ambiguities, which are hard for the model to label. Further samples containing *oppose*-relations are also required to prevent the model from having a too positive bias towards cryptocurrencies. Even with these problems, the model is capable of correctly labelling tweets, and some errors that are made due to ambiguities are plausible.

7 Conclusion

In this thesis, a model to solve the problem of knowledge graph construction on social media content was proposed. We identified named entity recognition and relation classification as the two important tasks to create nodes and edges for a knowledge graph and proposed a joint model to perform these tasks. We created this model by basing it on an existing model and suiting it to our needs. Furthermore, we implemented different filters on our loss function to further improve the prediction performance of the model. We created a domain-specific dataset based on tweets from the cryptocurrency domain and evaluated our model on this dataset, as well as other datasets commonly used for these tasks. Our model was evaluated against disjoint baselines, and we showed that it outperforms such baselines even if it is inferior on the individual tasks. Furthermore, we discussed the runtime and quality of the predictions made. We saw that while the model is able to find entities and relations, the predictions are not always correct. Unfortunately, the current performance would probably not suffice to create knowledge graphs which are production-ready for real-world use cases without further processing on the graph. It is however again important to state that the training time was lower than comparable methods used [50]. Even with the multi-objective tree-structured Parzen estimator[79], it is also not clear if the optimal hyper-parameters were found, as the number of trials was limited as well. Expending additional resources for further training and hyper-parameter tuning might result in better performance, which could be sufficient for real-world scenarios.

7.1 Future Work

Examining the contents of this thesis, different possibilities for future work remain. As already stated above, additional experiments with longer training time and more hyper-parameter tuning runs are expected to improve results further. Another approach would be to examine different domains in the social media data. People discuss various recent topics on social media and have formed many sub-communities. It would be interesting to see if data quality and content are identical in different domains and if the model performs equally well in other domains.
A different approach would be the extension of the proposed model to create a full knowledge graph. As predicted entities in the text might be written differently (e.g. *bitcoin*, *#bitcoin* or *btc*), we need to perform some sort of entity deduplication. Then inferring further knowledge via knowledge graph completion techniques could also be investigated. Finally, when a knowledge graph is constructed, its practical use in the context of the domain needs to be evaluated. For the cryptocurrency domain, it could for example be interesting to relate interactions between stakeholders to the market price of

their respective assets and attempt to find interactions.

Bibliography

[1] Dan Roth and Wen-tau Yih. A linear programming formulation for global inference in natural language tasks. In *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004) at HLT-NAACL 2004*, pages 1–8, Boston, Massachusetts, USA, May 6 - May 7 2004. Association for Computational Linguistics.

[2] Savvas Varsamopoulos, Koen Bertels, and Carmen Garcia Almudever. Comparing neural network based decoders for the surface code. *IEEE Transactions on Computers*, 69(2):300–311, 2019. `https://arxiv.org/abs/1811.12456v1`.

[3] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[4] Nikhil Birajdar. Word2vec research paper explained. `https://towardsdatascience.com/word2vec-research-paper-explained-205cb7eecc30`, March 2021. Accessed: 2022-04-23.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[6] Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. brat: a web-based tool for NLP-assisted text annotation. In *Proceedings of the Demonstrations Session at EACL 2012*, Avignon, France, April 2012. Association for Computational Linguistics.

[7] Amit Singhal. Introducing the knowledge graph: things, not strings. `https://blog.google/products/search/introducing-knowledge-graph-things-not/`, May 2012. Accessed: 2022-04-12.

[8] J. Fan, A. Kalyanpur, D. C. Gondek, and D. A. Ferrucci. Automatic knowledge extraction from documents. *IBM Journal of Research and Development*, 56(3.4):5:1–5:10, 2012.

[9] Bo Xu, Yong Xu, Jiaqing Liang, Chenhao Xie, Bin Liang, Wanyun Cui, and Yanghua Xiao. Cn-dbpedia: A never-ending chinese knowledge extraction system. In Salem Benferhat, Karim Tabia, and Moonis Ali, editors, *Advances in Artificial Intelligence: From Theory to Practice*, pages 428–438, Cham, 2017. Springer International Publishing.

[10] Jose L. Martinez-Rodriguez, Ivan Lopez-Arevalo, and Ana B. Rios-Alvarado. Openie-based approach for knowledge graph construction from text. *Expert Systems with Applications*, 113:339–355, 2018.

[11] Chenguang Wang, Xiao Liu, and Dawn Song. Language models are open knowledge graphs. *CoRR*, abs/2010.11967, 2020.

[12] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A survey on knowledge graphs: Representation, acquisition and applications. *CoRR*, abs/2002.00388, 2020.

[13] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web*, pages 722–735, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[14] G Klyne and J. J. Carroll. Resource description framework (rdf): Concepts and abstract syntax. `https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`, February 2014. Accessed: 2022-04-21.

[15] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 697–706, New York, NY, USA, 2007. Association for Computing Machinery.

[16] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, nov 1995.

[17] Damian Szklarczyk, Annika L Gable, Katerina C Nastou, David Lyon, Rebecca Kirsch, Sampo Pyysalo, Nadezhda T Doncheva, Marc Legeay, Tao Fang, Peer Bork, Lars J Jensen, and Christian von Mering. The STRING database in 2021: customizable protein-protein networks, and functional characterization of user-uploaded gene/measurement sets. *Nucleic Acids Res.*, 49(D1):D605–D612, January 2021.

[18] David S Wishart, Yannick D Feunang, An C Guo, Elvis J Lo, Ana Marcu, Jason R Grant, Tanvir Sajed, Daniel Johnson, Carin Li, Zinat Sayeeda, Nazanin Assempour, Ithayavani Iynkkaran, Yifeng Liu, Adam Maciejewski, Nicola Gale, Alex Wilson, Lucy Chin, Ryan Cummings, Diana Le, Allison Pon, Craig Knox, and Michael Wilson. DrugBank 5.0: a major update to the DrugBank database for 2018. *Nucleic Acids Res.*, 46(D1):D1074–D1082, January 2018.

[19] Prakash M Nadkarni, Lucila Ohno-Machado, and Wendy W Chapman. Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, 18(5):544–551, 09 2011.

[20] Hao Lian, Zemin Qin, Tieke He, and Bin Luo. Knowledge graph construction based on judicial data with social media. In *2017 14th Web Information Systems and Applications Conference (WISA)*, pages 225–227, 2017.

[21] Lei Cao, Huijun Zhang, and Ling Feng. Building and using personal knowledge graph to improve suicidal ideation detection on social media. *IEEE Transactions on Multimedia*, 24:87–102, 2022.

[22] Jingsheng Deng, Fengcai Qiao, Hongying Li, Xin Zhang, and Hui Wang. An overview of event extraction from twitter. In *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pages 251–256, 2015.

[23] Anastasia Giachanou and Fabio Crestani. Like it or not: A survey of twitter sentiment analysis methods. *ACM Comput. Surv.*, 49(2), jun 2016.

[24] Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha Talukdar, and Yiming Yang. A re-evaluation of knowledge graph completion methods. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5516–5522, Online, July 2020. Association for Computational Linguistics.

[25] Lisa Ehrlinger and W. Wöß. Towards a definition of knowledge graphs. In *SEMANTiCS*, 2016.

[26] Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, 8:489–508, 2017.

[27] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[28] Yongqi Zhang, Quanming Yao, Yingxia Shao, and Lei Chen. Nscaching: Simple and efficient negative sampling for knowledge graph embedding. *CoRR*, abs/1812.06410, 2018.

[29] Ralph Grishman and Beth Sundheim. Message Understanding Conference- 6: A brief history. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, 1996.

[30] Ji Wen, Xu Sun, Xuancheng Ren, and Qi Su. Structure regularized neural network for entity relation classification for Chinese literature text. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 365–370, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[31] Sri Nath Dwivedi, Harish Karnick, and Renu Jain. Relation classification: How well do neural network approaches work? In Boris Villazón-Terrazas, Fernando Ortiz-Rodríguez, Sanju M. Tiwari, and Shishir K. Shandilya, editors, *Knowledge Graphs and Semantic Web*, pages 102–112, Cham, 2020. Springer International Publishing.

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

[33] M. McCord, J. William Murdock, and B. Boguraev. Deep parsing in watson. *IBM J. Res. Dev.*, 56:3, 2012.

[34] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`, 1999.

[35] Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S. Weld. Open information extraction from the web. *Commun. ACM*, 51(12):68–74, December 2008.

[36] Yi Luan, Dave Wadden, Luheng He, Amy Shah, Mari Ostendorf, and Hannaneh Hajishirzi. A general framework for information extraction using dynamic span graphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3036–3046, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[37] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[38] Edgar Meij, Wouter Weerkamp, and Maarten de Rijke. Adding semantics to microblog posts. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, page 563–572, New York, NY, USA, 2012. Association for Computing Machinery.

[39] Kalina Bontcheva, L. Derczynski, Adam Funk, M.A. Greenwood, Diana Maynard, and Niraj Aswani. Twitie: An open-source information extraction pipeline for microblog text. *International Conference Recent Advances in Natural Language Processing, RANLP*, pages 83–90, 01 2013.

[40] William Cavnar and John Trenkle. N-gram-based text categorization. *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval*, 05 2001.

[41] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, 2002.

[42] Alan Ritter, Sam Clark, Mausam, and Oren Etzioni. Named entity recognition in tweets: An experimental study. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1524–1534, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics.

[43] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, page 173–180, USA, 2003. Association for Computational Linguistics.

[44] Fahd Kalloubi, El Habib Nfaoui, and Omar El beqqali. Microblog semantic context retrieval system based on linked open data and graph-based theory. *Expert Systems with Applications*, 53:138–148, 2016.

[45] Xiao Sun, Chengcheng Li, and Fuji Ren. Sentiment analysis for chinese microblog based on deep neural networks with convolutional extension features. *Neurocomputing*, 210:227–236, 2016. SI:Behavior Analysis In SN.

[46] Preeti Bhargava, Nemanja Spasojevic, and Guoning Hu. Lithium NLP: A system for rich information extraction from noisy user generated text on social media. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 131–139, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[47] D. Nguyen, Johannes Hoffart, Martin Theobald, and G. Weikum. Aida-light: High-throughput named-entity disambiguation. In *LDOW*, 2014.

[48] Min Peng, Jiahui Zhu, Hua Wang, Xuhui Li, Yanchun Zhang, Xiuzhen Zhang, and Gang Tian. Mining event-oriented topics in microblog stream with unsupervised multi-view hierarchical embedding. *ACM Trans. Knowl. Discov. Data*, 12(3), April 2018.

[49] Yankai Lin, Zhiyuan Liu, M. Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, 2015.

[50] Makoto Miwa and Mohit Bansal. End-to-end relation extraction using lstms on sequences and tree structures. *CoRR*, abs/1601.00770, 2016.

[51] George Doddington, Alexis Mitchell, Mark Przybocki, Lance Ramshaw, Stephanie Strassel, and Ralph Weischedel. The automatic content extraction (ACE) program – tasks, data, and evaluation. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, Lisbon, Portugal, May 2004. European Language Resources Association (ELRA).

[52] Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. SemEval-2010 task 8: Multi-way classification of semantic relations

between pairs of nominals. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 33–38, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

[53] Tsu-Jui Fu, Peng-Hsuan Li, and Wei-Yun Ma. GraphRel: Modeling text as relational graphs for joint entity and relation extraction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1409–1418, Florence, Italy, July 2019. Association for Computational Linguistics.

[54] Evan Sandhaus. The new york times annotated corpus. *Linguistic Data Consortium, Philadelphia*, 6(12):e26752, 2008.

[55] Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. Creating training corpora for NLG micro-planners. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 179–188, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[56] Giannis Bekoulis, Johannes Deleu, Thomas Demeester, and Chris Develder. Joint entity recognition and relation extraction as a multi-head selection problem. *Expert Systems with Applications*, 114:34–45, Dec 2018.

[57] Harsha Gurulingappa, Abdul Mateen Rajput, Angus Roberts, Juliane Fluck, Martin Hofmann-Apitius, and Luca Toldo. Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports. *Journal of Biomedical Informatics*, 45(5):885–892, 2012. Text Mining and Natural Language Processing in Pharmacogenomics.

[58] Giannis Bekoulis, Johannes Deleu, Thomas Demeester, and Chris Develder. Reconstructing the house from the ad: Structured prediction on real estate classifieds. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 274–279, Valencia, Spain, April 2017. Association for Computational Linguistics.

[59] Dan Roth and Wen-tau Yih. A linear programming formulation for global inference in natural language tasks. In *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004) at HLT-NAACL 2004*, pages 1–8, Boston, Massachusetts, USA, May 6 - May 7 2004. Association for Computational Linguistics.

[60] Giannis Bekoulis, Johannes Deleu, Thomas Demeester, and Chris Develder. Adversarial training for multi-context joint entity and relation extraction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2830–2836, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[61] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2015.

[62] Markus Eberts and Adrian Ulges. Span-based joint entity and relation extraction with transformer pre-training. *CoRR*, abs/1909.07755, 2019.

[63] Lev Ratinov and Dan Roth. Design Challenges and Misconceptions in Named Entity Recognition. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*, 6 2009.

[64] Yi Luan, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3219–3232, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[65] Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. LUKE: deep contextualized entity representations with entity-aware self-attention. *CoRR*, abs/2010.01057, 2020.

[66] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2020.

[67] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *CoRR*, abs/1506.03099, 2015.

[68] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China, July 2015. Association for Computational Linguistics.

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

[70] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[71] Tomáš Mikolov. *STATISTICAL LANGUAGE MODELS BASED ON NEURAL NETWORKS*. Ph.d. thesis, Brno University of Technology, Faculty of Information Technology, 2012.

[72] Anders Krogh and John Hertz. A simple weight decay can improve generalization. In J. Moody, S. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann, 1991.

[73] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[74] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[75] Geoff Pleiss, Tianyi Zhang, Ethan R. Elenberg, and Kilian Q. Weinberger. Identifying mislabeled data using the area under the margin ranking. *CoRR*, abs/2001.10528, 2020.

[76] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[77] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[78] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[79] Yoshihiko Ozaki, Yuki Tanigaki, Shuhei Watanabe, and Masaki Onishi. Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, page 533–541, New York, NY, USA, 2020. Association for Computing Machinery.

[80] David Hinterndorfer. *Binary text classification using positive and unlabeled data in application to Twitter*. Diploma thesis, Technische Universität Wien, Wien, 2020.

[81] Saša Petrović, Miles Osborne, and Victor Lavrenko. Streaming first story detection with application to twitter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, page 181–189, USA, 2010. Association for Computational Linguistics.

[82] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. *Milvus: A Purpose-Built Vector Data Management System*, page 2614–2627. Association for Computing Machinery, New York, NY, USA, 2021.

[83] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147, 2003.

[84] Erik F. Tjong Kim Sang and Sabine Buchholz. Introduction to the CoNLL-2000 shared task chunking. In *Fourth Conference on Computational Natural Language Learning and the Second Learning Language in Logic Workshop*, 2000.

[85] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 754–762, Bejing, China, 22–24 Jun 2014. PMLR.

[86] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.