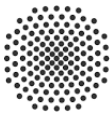

University of Stuttgart

Institute for Natural Language Processing
University of Stuttgart
Pfaffenwaldring 5b
70569 Stuttgart

Master thesis
Generating TEI-based XML
for
literary texts
Nidhi Sihag

Study program:	M.Sc. Infotech
Examiner:	Prof. Dr. Jonas Kuhn
Advisor:	Janis Pagel
start date:	01.12.2021
end date:	01.07.2022

Acknowledgement

I want to express my sincere gratitude towards my Supervisor Janis Pagel. He guided me in the right direction and provided me with necessary support from time to time.

I would also like to thank Prof. Dr. Jonas Kuhn for his valuable input during the entire duration of my thesis.

Abstract

Generating TEI-based XML files for literary texts is a long-standing problem in the Natural Language Processing. It is a task that requires developing a system to encode the text in their relevant TEI tags. We address the challenge of enriching the plain text with the learned XML elements. We are going to deal with the theatre plays (i.e. dramatic texts) and letters. These are encoded in XML. For now, we have these XML files for a few hundred plays and letters, but, as we can probably imagine , creating this kind of annotation manually is a lot of work. And since when new plays are digitized, they are initially only available as (OCRd) plain text. So we tried to build an automatic process for this. So that if these XML elements are recognized as an annotation, we could predict them essentially as a sequence labeling task.

This thesis takes its starting point from the recent advances in Natural Language Processing being developed upon the Transformer model. One of the significant developments recently was the release of a deep bidirectional encoder called BERT that broke several state-of-the-art results at its release. BERT utilises Transfer Learning to improve modelling language dependencies in texts. BERT is used for several different Natural Language Processing tasks, this thesis looks at Named Entity Recognition, sometimes referred to as sequence classification.

The purpose of this thesis is to investigate whether Bidirectional Encoder Representations from Transformers (BERT) is suitable for the automatic annotation of plain text. Therefore, we follow a deep learning approach for the extraction of plain text along with its tags from XML files. We use a neural network architecture based on BERT, a deep language representation model that has significantly increased performance on many natural language processing tasks. We experiment with different BERT models and input formats. The experiments are evaluated on a challenging dataset that contains letters in English and plays in multi-languages.

Contents

1 Introduction **7**

1.1 Research Question 10

1.2 Research Methodology 10

1.3 Thesis Outline 10

2 Background **11**

2.1 Natural Language Processing 11

2.1.1 Data Pre-processing 11

2.2 Multi-layer perceptrons 12

2.3 Recurrent neural networks 15

2.3.1 General form 15

2.3.2 LSTMs 16

2.4 Transfer Learning 17

2.5 Transformers 19

2.6 BERT 24

2.6.1 Architecture and Interface 26

2.6.2 Pre-training 27

2.6.3 Fine-tuning 27

2.7 Conditional Random Fields 28

2.7.1 General form 29

2.8 Named entity recognition 30

2.8.1 Standard Named Entity Recognition Model 31

2.8.2 Custom Named Entity Recognition Model 31

2.8.3 NER Tools 32

2.8.4 Evaluation 33

3 Related Work **33**

4 Proposed Method **35**

4.1 Model Architecture 35

4.2 Choice of BERT Model 37

4.3 Input Format 38

4.4 Implementation 39

5 Data **40**

5.1 Data Annotations 40

5.1.1 Annotation of Dramas 41

5.1.2 Annotation of Letters 41

5.2 Data Preparation 42

6 Evaluation **45**

6.1 Evaluation Procedure 45

6.1.1 Calculation of the Results 46

6.1.2 Experimental Setup 46

6.2 Experiments 47

6.2.1 Experiments on letters 47

6.2.2 Experiments on German Drama 50

6.2.3 Experiments on French Drama 51

6.2.4 Experiments on Russian Drama 52

7 Discussion **53**

7.1 Interpretation of the Results 53

7.2 Error Analysis 56

7.3 Limitations And Future Work 60

8 Conclusion 61

List of acronyms

CNN	Convolution Neural Network
LSTM	Long Short-Term Memory
RNN	Recurrent Neural Network
MLP	Multi Layer Perceptron
NLP	Natural Language Processing
NER	Named Entity Recognition
POS	Part of speech tagging
MLM	Masked Language Modelling
TL	Transfer Learning
NSP	Next Sentence Prediction
BERT	Bidirectional Encoder Representations from Transformers
CRF	Conditional Random Fields

1 Introduction

There are many tasks related to making machines understand language. This subfield of machine learning is called Natural Language Processing (NLP), with the task that is explored in this thesis called Named Entity Recognition (NER) [Li et al. (2018)]. NER can be used to understand both grammar and concepts in texts, for example, organizations, persons, or locations. An example could be the text [ORGANISATION U.N.] official [PERSON Ekeus], heads for [LOCATION Baghdad]. We want the computer to understand that a word may be a name using the context it exists in and not only the specific word. Techniques that perform NER well can handle poor spelling or words that have not been observed prior by the machine.

NLP has several general applications, like summarising texts [Gaikwad and Mahender (2016)] or understanding sentiment in reviews [Mäntylä et al. (2018)]. The purposes of NER are varied, such as tagging a document for search algorithms or anonymising personal information. When datasets contain sensitive information, such as name, location, or email addresses, an algorithm can look at the context and remove the sensitive words and replace them with a token. Text information might be interesting to researchers and companies but is considered too sensitive to work on. Anonymisation by machine learning is a cost-effective way to remove sensitive information and has become even more important as more governmental and medical information is stored in digital records. Additionally, reports and documents are produced but may not be easily accessible due to poor search algorithms. Allowing the machine to highlight interesting topics in documents by finding for example locations or companies can greatly improve information retrieval by helping the search engines find the most relevant documents based on the context in the search query.

Lately, algorithms called Neural Networks have shown great success in many areas of the field of machine learning, including NLP. Neural Networks are modeled after biological brains and neurons. They are small and simple operations that imitate how the individual brain cells work, but a large number of these artificial neurons may solve complex tasks. Operation for each neuron is a simple combination of weights that takes the input and enhances or suppresses the input signal to find interesting features that best

describe the data. These models are then shown training data and learn to output the probability for each class. When we want to improve the model we update the weights with a technique called Backpropagation [Dreyfus (2018)], where the weights are iteratively changed to provide better results. Backpropagation goes through the model backward and suppresses the weights that do not contribute to a correct classification, and reinforces the neurons that do.

There have been significant developments in the field of NLP in recent years. During 2018, several new models broke the state-of-the-art barrier on multiple occasions [Howard and Ruder (2018)] [Peters (2018)]. One of the reasons for this is that increased computational capabilities have allowed the field to create large models that have been historically infeasible.

Another concept that has helped the field is Transfer Learning. An issue in the field of machine learning is the lack of structured data. Many tasks and models require large quantities of data to provide good results for a specific task. It is often expensive to create large datasets, since humans may need to look at and annotate millions of images or texts. Transfer Learning is based on the idea that we can train our models on large datasets that may not require expensive annotation and then use that understanding in other related tasks where some limited amount of structured data may be available. By using Transfer Learning we can limit our dependency on structured data and get better results on the limited available data. Several of the improvements shown in [Howard and Ruder (2018)] and [Peters (2018)] are based on Language Modelling, when we want the model to create a general understanding of language. This is done by showing the model a partial sentence and asking it to guess what the next word in the sentence is. A model can then be fine-tuned for a specific task, with results being improved by the Language Model. The analogy for humans is that if we want to learn a highly specific task, like understanding medical texts, we want to have a good general language understanding first. Historically Transfer Learning has not had success within NLP, but first found success in Computer Vision tasks [Deng et al. (2009)]. Within Computer Vision success has been achieved by letting a model train on many pictures, training it to understand shapes and colors in general. These models with their general understanding were then adapted, or fine-tuned, to specific tasks as needed.

One of the biggest and most successful of the models that implemented Transfer Learning in NLP is called BERT [Devlin (2018)]. It was developed by Devlin et al. and established a new state-of-the-art results within many areas of NLP, including NER. Part of what makes it so successful is its size and deep architecture. The fact that BERT is deep means that it is more capable at finding complex relationships between words. BERT was trained on Wikipedia and BookCorpus [Zhu et al. (2015)], parsing a total of 3.3 billion words in order to build its language model. As the techniques used to achieve these results are new further exploration of ways to improve the results seems like an interesting avenue for research.

For NER applications it is important for the model to observe the entire sequence of words in order to capture context. Older Neural Network models were only capable at classifying each word independently of its neighbors, which increased the risk of unpermitted or unwanted predictions. A classic example of this is grammatical structures, where some combinations of words are grammatically forbidden. There are several different algorithms for NER and sequence prediction, but one of the most successful utilizes Conditional Random Fields (CRF) [Li et al. (2018)].

One of the possible uses of NER is anonymisation of sensitive information in texts. The demand for good anonymisation techniques has increased thanks to new regulations increasing the expectations on corporations to safeguard personal information, or even blocking some data to be used prior to anonymisation. The background for this thesis is that the principal is investigating several techniques for NER applications within their product stack and BERT's breakthroughs in NLP may offer a possibly compelling technical solution with its performance.

This thesis sets out to compare the model architecture proposed by Devlin et al. [Devlin (2018)] with a model that utilises a more popular classifier technique for NER. We are going to use BertForTokenClassification which is included in the Transformers library by HuggingFace. This model has BERT as its base architecture, with a token classification head on top, allowing it to make predictions at the token level, rather than the sequence level. Named entity recognition is typically treated as a token classification problem, so that's what we are going to use it for. The thesis investigates the performance across the model settings, called hyperparameters. The goal is to improve the creation of the XML

files with the predicted tags and compare the difference between the original XML file and the file generated with predicted tags.

1.1 Research Question

The main research question this thesis aims to investigate and answer is the following:

"To what extent can active learning techniques accelerate the model training of state-of-the-art language models and hence minimize the manual annotation effort of XML files in the context of Named Entity Recognition?"

To answer this research question, we examine further sub-research questions:

- Is BERT a suitable choice for the task of labeling of TEI tags?
- Which BERT model and input format achieve the best performance on the task?
- What is the best approach for training a given BERT-based model on the task?

1.2 Research Methodology

The research question is investigated by conducting experiments with a meticulous structure. Due to the inherent variation of Transfer Learning strategy performance for different problem contexts, experiments need to be done with several configurations. This is done by testing each strategy with varying datasets, pre-trained models, and acquisition batch sizes. Furthermore, each model is trained multiple times to measure statistical uncertainties. Results are analyzed visually by a wide variety of plots that expose different behavior aspects of the strategies. To conclude the comparisons, aggregations of all experiments will be done to achieve a scalar score for each strategy. This results in a simple way of comparing the strategies' overall experiment performances.

1.3 Thesis Outline

Each chapter of the thesis is briefly described as follows:

- Chapter 2 will provide a background of the various sub-fields that this thesis builds upon. The background aims to be sufficient for the reader to understand the next chapters.
- Chapter 3 outlines related work.
- Chapter 4 contains the approach and architecture used in our thesis.
- Chapter 5 shows examples from the datasets and detailed descriptions of the prediction heads
- Chapter 6 contains all the detailed experiments and their results.
- Chapter 7 interprets results and has limitations of our thesis and what we can do in the future.
- Chapter 8 concludes by summarizing our work and results.

2 Background

2.1 Natural Language Processing

The field of NLP aims to have computers analyze, manipulate or generate natural language. The recent rapid progress has evolved NLP from limited statistical methods that require careful pre-processing to deep neural networks, yielding near human-level understanding and generation of text. The language model architecture we use today is the result of an iterative process, where one architecture often was built with inspiration from its predecessor. To grasp the technology used today, it is often helpful to analyze the path which led us here. The following subsection explores the recent evolution of language models.

2.1.1 Data Pre-processing

The data used by language models come in the form of text. Before tackling the particular NLP tasks, such as sentiment analysis, NER and question answering, one often attempts

to reduce the problem complexity by pre-processing the data. This is not unique to NLP and is done in most sub-fields of machine learning. However, pre-processing differs for textual and numerical data.

For statistical approaches, a common first step in pre-processing is that of punctuation removal. Another important step is tokenization, a process of turning a string of text into a sequence of tokens. This is often done by splitting spaces to get a sequence of words. Some methods also prune the data of stopwords (e.g. ”the”, ”is”, ”at”) as they carry little information and can dominate frequencies in statistical methods.

Furthermore, it is often desirable to reduce the size of the vocabulary that a language model needs to understand. Two scenarios of this are Stemming and Lemmatization. Stemming is the process of breaking down words to their word stem. As an example, fishing, fished and fisher would all be reduced to their stem fish. The stem is not necessarily a proper word, e.g. stemming of argued to argue. On the contrary, lemmatizing aims to find the root word using a dictionary-based approach and analyzing the context. The data preprocessing, however, varies for different problems and architectures.

Before the deep learning era, the traditional NLP relied heavily on preprocessing for frequency methods such as Bag of Words. Of course, there were sophisticated probability-based methods as well, using e.g. Hidden Markov Models.

2.2 Multi-layer perceptrons

The beginning of today’s tools and algorithms of deep learning started with the attempt to model neurons in the brain with mathematics during the fifties. These early Perceptron learners, introduced by Rosenblatt [10], could only model simple decision boundaries (see Figure [1]). The use of only a single hidden layer limits the kind of solutions the algorithm can produce [Minsky and Papert (1969)]. Later developments have introduced more complex capabilities enabled by the Multi-Layered Perceptron (MLP) [White and Rosenblatt. (1963)]. The extension of the simple Perceptron to a multi-layered variant allowed for more complex problem-solving abilities (see Figure [2]), as the multiple hidden layers enable non-linear decision boundaries to be modelled.

One of the big developments between the Perceptron and its multi-layered variant is the way it is trained. It is not possible to use a step function to train MLPs, thus new activation functions (see table [1]) needed to be introduced. The activation function is a non-linear function applied to a linear function, often denoted as $\sigma(\cdot)$ (see equation [1]). The reason for the new activation functions is that the weights needed to be updated iteratively. The weights are updated according to the gradient in an algorithm called Back-propagation developed first by [Dreyfus (2018)] and later rediscovered and popularised through the work of Rumelhart et al. [Rumelhart et al. (1988)]. The lack of available computational power at the time hindered the sort of rapid development in Neural Networks we see today.

$$\hat{y} = \sigma(Wx + b) \tag{1}$$

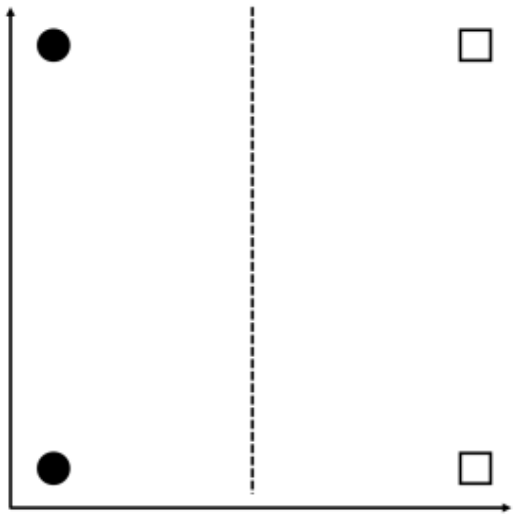

Figure 1: A problem solved easily by Perceptron, with a decision boundary (dotted line) drawn between the two classes.

Originally, popular functions were sigmoid or tanh, but today these have been largely replaced with other functions like the Rectified Linear Unit (ReLU) [Nair and Hinton

Sigmoid	Tanh	ReLU
$\frac{1}{1+e^{-z}}$	$\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	max(0, z)

Table 1: Three common activation functions.

(2010)], table [1], and its various variants. These non-linear functions play a crucial part in the ability of the machine to learn. Without activation, it would just be a linear mapping, and the algorithm would not be able to generalize. In equation [1] we see that the function is generally simple, with the input vector as x and the trainable weights denoted as W and b. b is called bias and is a special form of weight vector to help the $(W \cdot x)$-vector not get stuck around the origin.

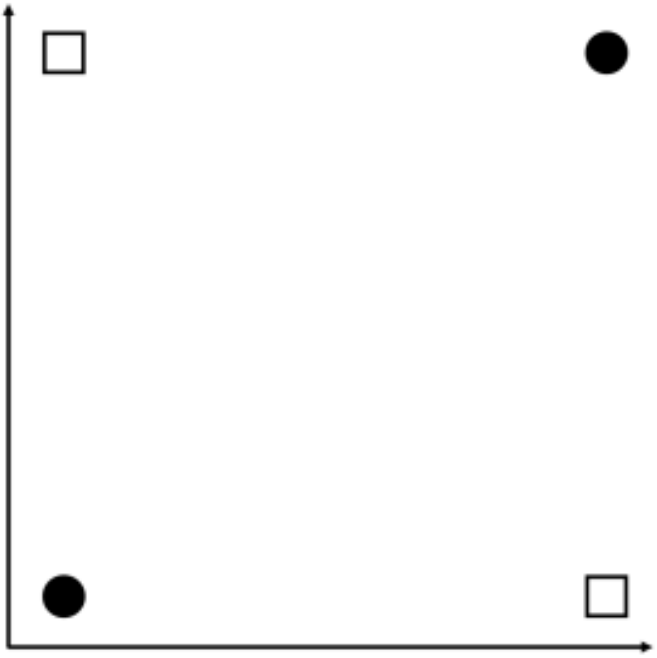

Figure 2: The XOR problem, that cannot be solved by a Perceptron. No decision boundary can be drawn for a single layered Perceptron, as no single straight line can separate the two different classes shown. The XOR problem can be solved with MLPs.

2.3 Recurrent neural networks

2.3.1 General form

Recurrent Neural Networks (RNN) are an extension of MLPs that solve some of their temporal problems. In NLP the handling of sequences of data causes particular problems for the MLPs, as it has no way of storing the previous state in a sequence. This is equivalent of trying to understand a written sentence but forgetting each word when reading. Additionally, text is rarely normalized in length and cannot be split up easily while keeping the original information intact. Hopfield [Hopfield (1982)] and Rumelhart et al. [Rumelhart et al. (1988)] solved this by feeding the previous hidden state (see Figure [2]) in the network back to the current state. This allows the network to keep a form of internal memory of what has been seen previously in the sequence. This solves two problems at once: Firstly, we can now model dependencies back in time as the model is influenced by what it has seen earlier. Secondly, we now have a structured way of handling different lengths of inputs, as we can keep running each new data point into the model for the entire length of the sequence. RNNs solved these and quickly became the de facto standard for most forms of NLP using Neural Networks [Li et al. (2018)].

However, what today may be considered "vanilla" RNNs have some problems. Training them through Backpropagation has a tendency to become hard, as their gradient either becomes vanishingly small or very large, making convergence unlikely. Vanishing or "exploding" gradients [Hochreiter (1998)] make RNNs difficult to train well. Additionally, these models have a tendency to become large, especially in NLP, and therefore take considerable computational resources to train. There have been some developments in order to improve convergence when training RNNs. Some of the same regularization algorithms used for general MLPs can be carried over, such as L1 [Tibshirani (1996)] or L2 [Tikhonov (1943)] regularization. Others, like Dropout [Srivastava (2014)] may even be considered very important for some models' convergence [Howard and Ruder (2018)].

In this study, we focus on NLP, but it is worth mentioning that RNNs are used in many kinds of sequential data such as video or general time series.

2.3.2 LSTMs

Hochreiter and Schmidhuber tried to solve many of the RNN's problems with the Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber (1997)]. They use gates in order to update the hidden states in a more efficient manner. LSTMs uses input, forget, cell and output gates, seen in Figure [3]. The purpose of the gates is to let the network learn when to update the hidden state, in effect adding a layer of de-noising in the hidden state, making it less likely to update the hidden state with non-important information. The model has been expanded by Gers et al. [Gers et al. (1999)] and Greff et al. [Gref (2017)] with additional gates from the original ones. While these gates improve the vanishing and exploding gradient problems associated with regular RNNs, they do not completely solve them. And while the model becomes easier to train as it is less sensitive to learning parameters, it is not trivial, especially for deep or stacked, LSTMs. This problem will be discussed further in the Transfer Learning section of the literature study, see chapter 2.4.

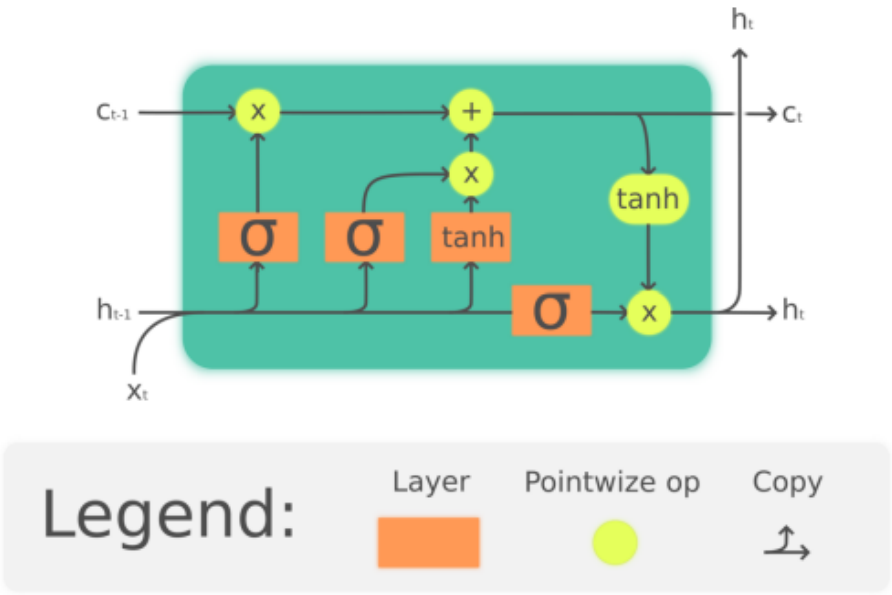

Figure 3: LSTM illustration [23]

One of the complexities of NLP is that text may have future dependencies as well as past ones. Thus takes the form of the meaning of words being determined by context not

yet expressed in a sentence. These problems are quite One of the complexities of NLP is that text may have future dependencies as well as past ones. Thus takes the form of the meaning of words being determined by context not yet expressed in a sentence. These problems are quite significant in NLP, as humans may read further into a sentence before comprehending the context of a word. As LSTMs cannot look ahead this problem is solved by bidirectionality [Schuster and Paliwal (1997)]. We create two independent LSTMs and let them look left to right and right to left respectively, then combine their output at inference, usually by simple concatenation. This is referred to as a Bidirectional LSTM or Bi-LSTM. This solves part of the problem, while it allows LSTMs to look ahead, they are still not conditioning the hidden state both forwards and backwards in time, but merely do both in parallel. And while LSTMs in theory can model dependencies at infinite distance, in practice the space in which it encodes useful information is significantly shorter than that. This limit in the range is a considerable problem in NLP, as a reference can be established several sentences back in time, or even paragraphs.

2.4 Transfer Learning

There are some special considerations that need to be taken into account with respect to feature engineering in NLP. Representing words as numerical vectors in a Neural Network have inherent problems. Commonly, a certain number of words from the training set are chosen called a dictionary that is represented in the model. The dictionary is built up of all, or a subset of, words in the training dataset. Any words not contained in the dictionary cannot be used for inference or prediction. The encoding of these words plays a crucial part in NLP. Previously, popular techniques for encoding the words included simple representations like Bag Of Words (BOW) [Michael McTear and Barresa (2016)] and Term Frequency–Inverse Document Frequency (Tf-idf) [Jones (1978)]. Where BOW only sums the number of occurrences of a word in the text, Tf-idf normalises words over the overall number of occurrences in the corpora, the idea being that less common words carry more relative importance. These techniques are computationally simple and work well for some problems, but have problems with nuances since all positional context is lost.

More recent developments in Neural Networks have created more complex techniques, such as Embeddings [Bengio (2006)]. Embeddings look at the words and try to decide from the context how to convert the word into numeric vector form. The Embedding layer takes the word and represents it as a vector in a space where it tries to place words that occur together closely and unrelated words far apart. This is usually done with euclidean or cosine distance between different pairs of words.

Embeddings are still fundamental to any NLP problem and still under active development. Older models include GLoVE [Jeffrey Pennington and Manning (2014)] and Word2Vec [Mikolov (2013)], both early instances of Transfer Learning. These were developed by Pennington et al. and Mikolov et al. respectively on a large amount of text and are supposed to work as general-purpose encoders of text. These Embeddings are part of what is generally referred to as feature engineering, and while we mostly discuss techniques used by neural methods here, it should be mentioned that features can be hand crafted as well. Hand-made features were common in Conditional Random Field classifiers and others, but recently Neural Networks have proven to be capable feature extractors and encoders, forgoing the need for hand-crafted features. While there are applications that benefit from handcrafting, it is much less common today.

2018 was a big year for NLP with significant improvement on several benchmarks throughout the year [Peters (2018)] [Howard and Ruder (2018)] [Devlin (2018)], partly enabled by Transfer Learning. Transfer Learning has had enormous success in Computer Vision with pre-trained models like ImageNet [Deng et al. (2009)]. These models try to create a general understanding of visual objects by looking at a very large number of images, sometimes with thousands of different classes. These models generalise well when fine-tuned to specific applications. With significant pre-training, the idea is that we could then fine-tune the model based on a much smaller dataset of domain-specific images and achieve good results there, even better than if we had trained a specific model from scratch. This pre-training can be seen as an extension of Dictionary Learning [Ron Rubinstein and Elad (2013)]. The idea is that we can decompose a signal, in this case an image, as a linear combination of other simpler functions. We can learn a general basis for a problem, and then modify it to suit our needs.

First to make significant waves in transfer learning for NLP was Peters et al. [Peters

(2018)] who introduced the ELMo model. The general idea is to pre-train using a large corpus and to create a Language Model from the text. This training makes clever use of semi-supervised data by not having to create annotated datasets [Dai and Le (2015)], but using unstructured text in a structured way. It is done by trying to capture the general structures in the language by predicting the next word given the observed sequence. ELMo works as a feature extractor in general language by passing the sequence through a series of bidirectional LSTMs. These features capture the general patterns in the language similarly to how lower-level convolutional layers capture lower-level image features. One of the breakthrough ideas in ELMo is that each embedding is conditioned on its context, meaning that the same token, or word, can have different embeddings. However, ELMo is not fine-tuned to a specific task, and is generally used as a supplement to other feature extraction techniques [Li et al. (2018)].

Closely following the results of ELMo, another development was made with ULMFiT by Howard et al. [Howard and Ruder (2018)]. ULMFiT is a deep LSTM based model that also does general Language Modelling to improve feature extraction. ULMFiT uses several techniques to ensure that models actually converge well during fine-tuning, such as using AWD-LSTMs proposed by Merity et al. [Stephen Merity and Socher], triangular learning rates and learning rate scheduler to avoid catastrophic forgetting. AWD-LSTMs do not just apply Dropout to the recurrent state h, but to each gate in the layer as well. These tactics, combined with the model's size again pushed state-of-the-art forward and enabled NLP Transfer Learning [Howard and Ruder (2018)] properly for the first time. Previous attempts at fine-tuning pre-trained models have proven tricky, and catastrophic forgetting or general poor convergence have long been a problem. These problems have usually become worse with deeper and larger models. While ULMFiT does solve many of these problems, it is hard to train and sensitive to hyperparameter settings for convergence.

2.5 Transformers

Vaswani et al. [Vaswani (2017)] proposed an alternative to LSTMs and other RNN methods with the introduction of Transformers. It does away with recurrence or convolutional methods for parsing sequences and instead uses only an attention mechanism. Transform-

ers have become important to several sequence tasks both in NLP and in other fields, such as time series prediction. The core technology in Transformers is the "Self-Attention" mechanism. Self-Attention is not recurrent like RNNs, and instead uses a token pair-wise scoring system. In the context of NLP it can be explained as following: Given a word in a sequence, the algorithm can learn which other words to prioritise in determining the meaning of a word. An example showing why Transformers are powerful is the sentence "The chicken did not cross the road because it was scared.", in this context, what does it refer to? For a human the relation to The chicken is obvious, but machine learning techniques have had problems decoding this. Partly because LSTMs parse the sequence left-to-right, and is in practice influenced the most by the closer words rather than the ones farther away. Self-Attention does away with the sequential updating of the hidden state at each time step and instead looks at each word in the entire sequence at once. In this case, the Self-Attention layer should learn to associate it with chicken rather than the road.

An illustration of the model is shown in Figure [4]. The architecture utilises a similar concept to that of Autoencoders, a form of unsupervised learning technique introduced by Ballard et al. [Ballard (1987)]. Autoencoder uses a encoder- decoder stack, where the encoder tries to create an improved representation of the input while the decoder reconstructs the original input. Transformers can use a similar technique but with additional information sharing between the encoder-decoder stacks (see Figure [5]). The Transformer uses several components for its network, with the calculations described below. In addition to the Value (V), Key (K) and Query (Q) matrices (see equation [2]) describing the Self-Attention mechanism the algorithm uses, residual connections proposed by [He (2016)] ensure that the network learns at each step. This in addition to the layer normalization proposed by [Jimmy Lei Ba and Hinton (2016)].

$$\begin{aligned} V &= W^V X \\ K &= W^K X \\ Q &= W^Q X \end{aligned} \tag{2}$$

At the core of the attention mechanism is the attention formula, described by equation [3]. Vaswani et al. [Vaswani (2017)] extended this model with Multi-Head Attention according to equation [4]. According to the authors: "Multi-Head Attention allows the

model to jointly attend to information from different representation subspaces at different positions." [Vaswani (2017)]. Transformer has an encoder-decoder structure, with each encoder layer consisting of a Self-Attention and Feed Forward layer. The decoder has a Self-Attention, encoder-decoder attention and Feed Forward layer. The Self-Attention looks at the input from the previous layer in the decoder stack, while the encoder-decoder attention looks at the combination of Self-Attention layer and input from the encoder stack. The encoder pushes the last encoded vector to all layers in the decoder stack (see Figure [4]).

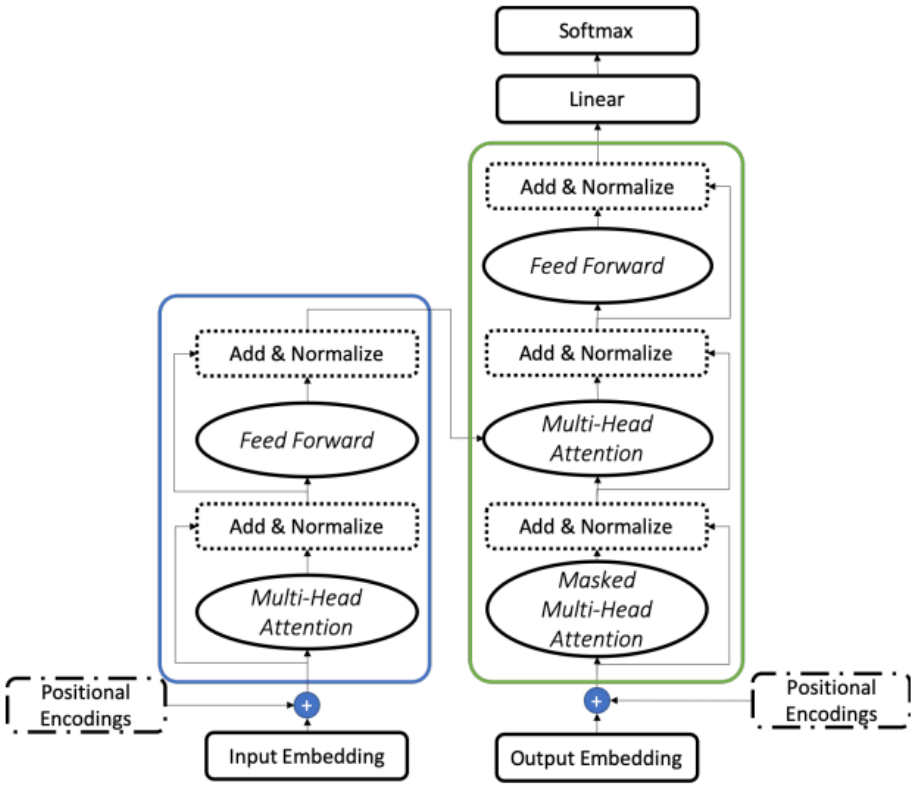

Figure 4: Transformer architecture illustration used by Vaswani et al. [Vaswani (2017)].Figure shows encoder to the left and decoder to the right. Many of these layers can be stacked to create a deeper model.

The Self-Attention layer is a key part of the success found by Transformers. For each

token in the sequence, the model will condition the observation on any other part of the sequence, allowing it to efficiently associate what vague tokens like it may refer to in each sequence. Calculation of Self-Attention is a six-step process outlined below:

$$Attention(Q, V, K) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3}$$

$$\begin{aligned} Z &= Multihead(Q, K, V) = concat(h1, ..., hh)WO \\ hi &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{4}$$

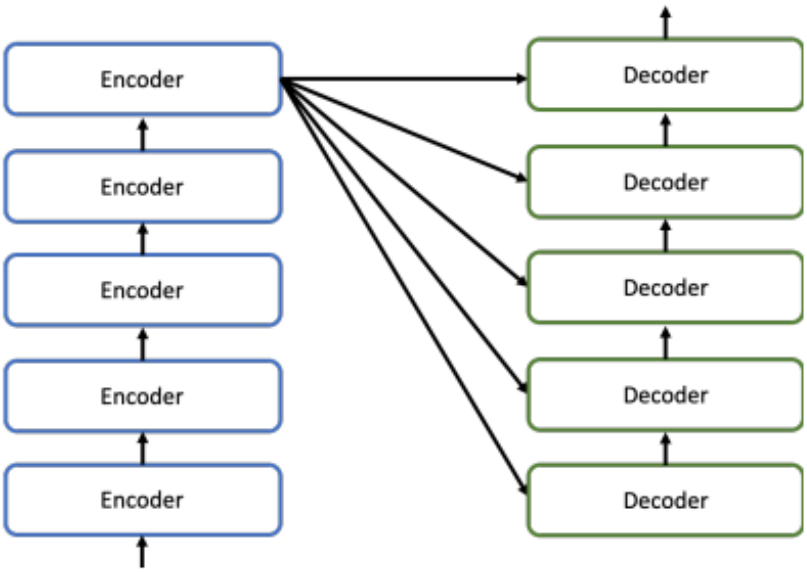

Figure 5: High level overview of how the encoder and decoder shares information in deep Transformer architectures.

Each output is feed through a layer normalization with a residual connection according to [Vaswani (2017)]. The activation function for F F N(x) is a ReLU and H = (h1, ..., hi) is the output from the encoder layer (see equation [5]). Equation [5] shows only a single

residual, but there will be an equal number of instances to the number of Attention Heads.

$$FFN(x) = max(0, xW1 + b1)W2 + b2 \tag{5}$$
$$H = LayerNorm(Z + FFN(x))$$

Stepping through the operations done at each step of a single Attention mechanism and how (K, Q, V) are used and interact.

- First, we create three vectors for each embedded token: Key, Query and Value vectors (K, Q, V). These vectors are learned during training.
- Secondly, we calculate the score for each token by multiplying the Query vector of each token with the key of itself and every other token. This corresponds to the QK^T operation in equation [5].
- This value is then scaled with the term $\sqrt{d_k}^{-1}.d_k$ is usually determined by the number of dimensions in the Key vector/matrix.
- The result of the scaling is pushed through the Softmax function. This functions as a scoring of each term in relation to all other terms.
- We then multiply the Softmax with V , which can be thought of as suppressing or putting the focus on the word we want and vice versa.
- The sixth, and final, step is to sum up the weighted value vectors producing the output of the layer. In the case of Multi-Head Attention, this corresponds to equation [4].

Taking a further look at the interpretations of the V, K and Q matrices, as they describe the different representational subspaces and hold the key to the Self-Attention mechanism. The three matrices will together help us understand how much focus to place on each neighbouring token in a sequence. This score is found by the Query and the Key vector multiplication. Since the Query for each token is multiplied with the Key vector for each other position we get a relative importance between each token pair combination in the

sequence. The Softmax operations ensure mathematical convenience by putting the relative score between 0 and 1. Lastly, the Value vector allows an additional step of ignoring words of low importance to a sequence. The above steps describe a single Self-Attention mechanism, but this can be extended to Multi-Headed attention, where multiple concurrent, independent, operations of the type described above are undertaken. The results are then concatenated and processed as normal. This allows multiple subspaces to form for each token, allowing greater representational power to be achieved.

For the full model, many of these heads are combined, allowing the model to find many different patterns in the sequence. Lastly, in order to encode position into the sequence, we add a positional encoding vector to the embedding.

One of the critical advantages of the Transformer architecture is that it lends itself to parallelisation, thus speeding up training time. Additional improvements on this front have been made with very recent developments such as Transformer-XL [Dai (2019)] that can speed up sequence parsing through connecting layers of past sequences with the current batch. Given the novelty of Transformers, and the speed at which it is developing we can expect to see further developments in the future.

2.6 BERT

The work of Devlin et al. [Devlin (2018)] extends previously discussed Transfer Learning and Transformer techniques to achieve state-of-the-art results within multiple areas of NLP. BERT stands for Bidirectional Encoder Representations from Transformers. BERT utilises only an encoder stack for classification, forgoing the entire decoder stack showen in Figure [5] , with suitable classifiers attached after the encoder. For NER this takes the form of a Feed-Forward layer [Devlin (2018)]. While BERT uses only an encoder stack, the most important concept in [Devlin (2018)] is the idea of Masked Language Modelling (MLM, also referred to as a Cloze task [Taylor (1953)]) which allows the model to look at the entire sentence or text when predicting a word. This is in contrast to RNNs or Transformers that are observing only a partial sentence when making predictions, otherwise they would be able to see the correct answer. BERT takes the normal semi-supervised Language Modelling task and masks 15% of all the tokens. To ensure that the model does

not get too simple, 10% of masked words are changed to a different random word, and an additional 10% is unchanged from their original word. The model is then asked to guess the word for these tokens, but the combination of random, masked and original words helps the model to generalise. Additionally, BERT extends the semi-supervised learning by asking if two sentences are related or not, that is, are they logical follow-ups from its predecessor. These two tasks in combination make sure that BERT generalises well and captures structures in language. Multi-task learning has some positive effects, as it can act as a form of regularizer on both word prediction and sentence classification. The fact that the weights need to balance the need for multiple tasks can prevent it from overfitting to a single task.

BERT has many uses within NLP, but we are going to apply it to NER in this thesis. In this context, BERT is used as a form of feature extractor that creates contextual embeddings, much like ELMo. These models then "attach" other layers after the feature extraction to make different forms of classification and prediction. Lastly, it should be said that even though some of the techniques used in BERT are innovative in and of themselves, part of BERT's success is the size. BERT has been published in two sizes, the smaller one with 110 million parameters, and the larger one with 340 million parameters. These models require specialized hardware like TPUs or GPUs to train because of their considerable memory requirements and size.

BERT was pre-trained in multiple different configurations depending on the dataset and need. The larger model, referred to as $BERT_{LARGE}$ with 340 million parameters, is built up with 16 attention heads and 24 encoder layers (see Figure [5]). The smaller model, referred to as $BERT_{BASE}$, has 12 encoder layers and 12 attention heads. It is designed to be closer in size to OpenAI's GPT [Radford (2018)] as mentioned by [Devlin (2018)]. The two models use different sizes of hidden space, where the larger models have 1024 dimensions and the base model 768. There is only one version of the large model, for English only. It was trained on English Wikipedia and BooksCorpus [Zhu et al. (2015)] with a total of 3.3 billion tokens. Additionally, there exists a small version of BERT that does not remove upper case in the text. It limits the vocabulary size, but is very useful in NER as many classes are names, where capitalization is useful for inference. There exists one model that has been trained on data other than English. This model is of the same

	#blocks	hidden_size	#heads	#parameters
$BERT_{BASE}$	12	768	12	110
$BERT_{LARGE}$	24	1024	16	340

Table 2: The size configurations of BERT.

size as $BERT_{BASE}$, keeps capitalization and is trained on Wikipedia for 104 languages. Since the largest language on Wikipedia is English, the languages are normalised over frequency, so that larger languages are sampled less than smaller ones.

2.6.1 Architecture and Interface

BERT is essentially the encoder side of the Transformer with some modifications. The first input token position to the model is a special [CLS] (classification) token. The succeeding token positions are reserved for the input text sequence. All input tokens are propagated in their own path through the encoder stack as usual. The last encoder block outputs a vector of size hidden_size for each token position, including the [CLS] position, whose purpose is to serve as input to a sentence classifier added on top. A common mistake is to use this [CLS] output as a sentence representation, which is not recommended by the authors. Creating good sentence representations [A. Conneau and Bordes (2017),Reimers and Gurevych (2019), F. Carlsson and Sahlgren (2021)] is a task of its own. However, the authors propose ways to extract contextualized word embeddings from BERT, by aggregating the encoder block outputs throughout the encoder stack. The model comes in two sizes, $BERT_{BASE}$ and $BERT_{LARGE}$. They vary in number of encoder blocks, hidden layer size in the feed-forward networks, number of attention heads, and thus in total number of trainable parameters. The configuration values of the two sizes can be seen in Table [2]. There is a clear pattern that indicates that larger models perform better, given sufficient data.

2.6.2 Pre-training

The pre-training of BERT consists of two self-supervised tasks, one on word-level and one on sentence-level. The addition of a sentence-level task is motivated by the fact that there are various downstream tasks that require the model to handle the relationship between two text sequences.

The word-level pre-training objective is that of Masked Language Modelling (MLM). It consists of masking 15% of the input tokens, replacing 80% of these with the [MASK] token, 10% with a random token, and 10% with the original token. The model is then trained to predict the correct tokens for the masked positions, using a classification layer added on top of the output of each token position. Part of the intuition that led the authors to these rules is that the model must learn good representations for both masked and non-masked tokens. The bidirectional MLM task entails a slower model converges than unidirectional equivalents, due to only training on 15% of the input tokens. However, the authors showed that the bidirectional approach outperforms the unidirectional model after only a few pre-training iterations.

The sentence-level pre-training objective is then, given two input sentences, to predict the likelihood of the second one being the subsequent sentence in the original data. This way, the model learns to handle a high-level view of the input sequences. To distinguish between the sentences, since they are concatenated with only a [SEP] token between them, a positional segment encoding is added to the input along with the Transformer's positional token encoding and token word embedding. Whether the second sentence follows the first is a binary classification problem. To allow the model to output predictions, a simple classification layer is added on top of the [CLS] token output. The model is pre-trained with these two tasks together, minimizing their combined loss function.

2.6.3 Fine-tuning

When the model is pre-trained, it can be shared with anyone who desires to fine-tune it for their particular downstream task. Fine-tuning BERT models are in many cases simple and does not require much data as the model is already pre-trained language understanding.

This makes it possible to reuse the results of the immense pre-training process (16 Tensor Processing Unit (TPU)s running for 4 days for $BERT_{LARGE}$), saving time and resources, allowing models to be trained on commodity hardware in a reasonable time.

In many cases, fine-tuning is only a matter of adding a small layer to the pre-trained model. Figure [6] illustrates the composition of BERT and a custom model on top. A common example is that of sentence classification, e.g. spam detection or sentiment analysis, which only requires a classifier added on top of the [CLS] token, similar to the sentence prediction pretraining task. In the case of NER, each output token vector is simply passed through a shallow classification network to predict the token entity. In conclusion, BERT provides a way to reuse trained language models for a wide variety of downstream tasks with little effort.

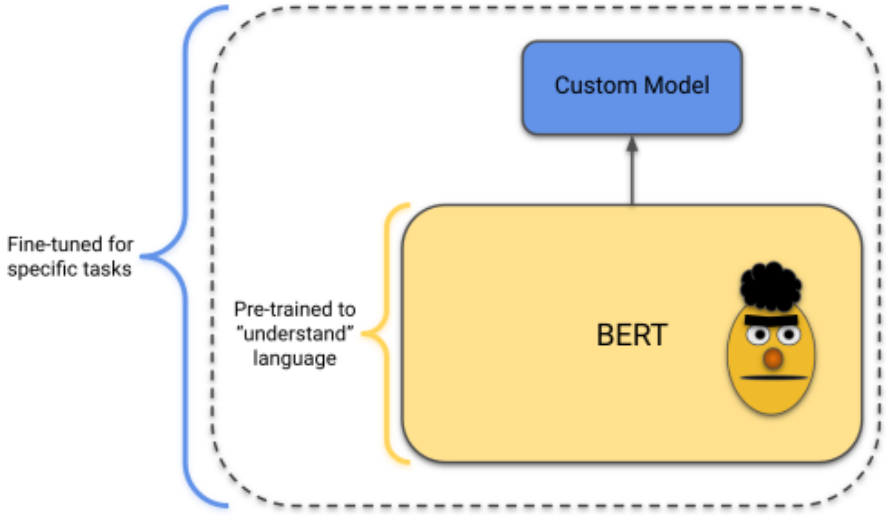

Figure 6: The composition of BERT with a task-specific neural network head on top.

2.7 Conditional Random Fields

One of the problems with both Feed Forward and RNNs is that their prediction is at token level. This means that their predictions ignore the neighbouring tokens when doing infer-

ence. However, many sequences cannot assume independence between the tokens. Some time-series data, like weather and stock prices have a noise component to them, but text is not nearly as random [Eduardo G Altmann and Esposti. (2012)]. Language's structure takes a hierarchical approach, where topics correlate with other topics, and lower-level features do the same on sentence-level. These hierarchical structures are complex and dependent on language, time and context. There has been work on this topic related to Conditional Random Fields (CRFs) before [Fei Liu and Cohn (2017)] [Greaves-Tunnell and Harchaoui (2019)]. CRFs, and other sequence classifier algorithms, observe the entire output in order to make predictions, making it possible to condition the prediction on neighbouring words.

2.7.1 General form

CRFs were proposed by Lafferty et al. [John D Lafferty and Pereira (2001)]. They are related to other sequential models like Hidden Markov Models (HMM), although CRFs can be considered a generalization of the transition and state probabilities from a graph. This is in contrast to how HMM works, where the model state transitions and emissions are constant. HMMs, in contrast to CRFs, are generative and model the joint probability $p(x, y)$ and therefore do not work in Neural Network applications as they are discriminative and model $p(y|x)$. As other sequential models, CRFs try to predict labels by considering the entire sequence together. This is very helpful in NLP applications since language has many dependencies, where certain word sequences are unlikely, or even grammatically forbidden. However, CRFs have found uses in other areas as well, such as Bioinformatics and Computer Vision. CRFs are able to relax the strong independence assumptions of state transitions required in HMMs (an assumption done so that the Markov Property holds). It can even be shown that logistic regression models are very simple CRFs. CRFs also solve part of HMM's bias towards states with fewer successor states.

The model is based on the fundamental random field theorem known as the Hammersley–Clifford theorem [John D Lafferty and Pereira (2001)] found in equation [6]. Random fields can be modelled as a graph with v representing vertices and e for edges in the graph, while f and g define features. For a classical CRF algorithm this feature is handcrafted

for the task and maybe things like “Is this word uppercase?”. f and g work as a collection of measurements on the edge and vertices of that graph which the model then combines to do inference upon. Given a sequence x with possible classes y, the core of the CRF problem formulation, is that we ignore the joint distribution of y and x, and only care about the conditional probability $p(y|x)$, therefore making the problem discriminative instead of generative in nature.

$$p\theta(y|x)\alpha exp(\Sigma_{(}eE,k)\lambda_k f_k(e,y|e,x) + \Sigma\mu_k g_k(v,y|v,x)) \quad (6)$$

The purpose of training is to find the parameters $\theta = \lambda_k; \mu_k$ such that the maximum-likelihood is found.

2.8 Named entity recognition

There are a wide variety of tasks within the field of NLP. One common task is that of NER, a sub-task of information extraction about entity identification and classification. For example, finding which tokens correspond to a person or location in the following sentence: ”Barack Obama lives in America”. This example also demonstrates that an entity can be composed of more than one word or token.

There are many use cases of NER, ranging from general applications of information extraction for analysis to specific tasks such as anonymizing documents. It can act as a pre-processing step for other downstream tasks such as question answering or search algorithms. Consequently, it could be seen as one of the most important NLP tasks.

The NER task is often formulated as a token-level multi-class classification problem. Meaning that a model processes a sequence of text and assigns a predefined entity class to each token. Previously, the standard entity-tag format has been to represent an entity simply as the entity class, with no consideration of whether it is a continuation of an entity or the start of a new entity. This can cause ambiguities, especially in processed data where stop words are omitted.

To address this problem, the Beginning-Inside-Outside (BIO) format [Ramshaw and Marcus (1995)] represents entity tags as B-entity or I-entity (or O if no entity is assigned to

the token). B-entity represents the beginning of an entity, the left-most token of the entity. I-entity represents an entity token inside the entity, a continuation of the current entity. So, for our previous example ”Barrack Obama lives in America” we would get different entity sequences for different tag representations. The plain representation would result in ”PER PER O O LOC” while BIO would yield ”B-PER I-PER O O B-LOC”.

If there are special cases of the entities of interest, a custom knowledge base could be more successful at correctly extracting the information needed. The next two sections explore the differences between standard and custom NER models and give examples of when each would be most relevant to use.

2.8.1 Standard Named Entity Recognition Model

Standard NER models seek to capture information that already has a known Wikipedia trained knowledge base. It is the problem of finding the members of various predetermined classes, such as person, organization, location, date/time, quantities, numbers, etc. (Goyal, Gupta, & Kumar, 2018). From the example in Figure 2 above, it is seen that the standard NER model can pick up that “WeWork” is an organization, “Adam Neumann” is a person, “Manhattan” is a location, and “$37.5 million” is a monetary value.

For many purposes, a standard model can accurately capture the entities as they are intended. It is still important to note that unique variations on these entities might not be picked up by a standard NER model. The unique supplier names in DoD are a prime example of when a standard model is not sufficient in picking up the intended entity with sufficient accuracy. When the standard model does not accurately pick up the entities of interest well, a custom NER model should be explored.

2.8.2 Custom Named Entity Recognition Model

One of the most limiting factors of NER is a fixed number of classes of entities that are currently available (Stepanyan, 2020). Custom NER models are used mainly when a standard NER model is not available or not sufficient at extracting the entities of interest accurately enough. This can be due to special cases of common entities or needing to

extract entities that are not present in a standard model. However, developing a corpus of custom-named entities (CNE) is a cumbersome task requiring an annotated dataset (Stepanyan, 2020). In the sections below, NER packages that can help with speeding up the annotation process will be explored. Once the annotations are complete, the NER model then gets trained on the annotated corpus of data.

For example, dates used in the DoD vary from how the rest of the world formats dates. This special case of a common entity can make it hard for the NER model to pick up that information. As well as dates, organizations are not standard in the DoD either. Some supplying organizations, such as Boeing, a standard NER model would be able to recognize but the unique suppliers to the DoD, such as 88th Air Base Wing, would not be recognizable as an organization to the model. In addition to special cases of common entities, there are some entities that standard models just do not have. Information such as NSN and Part Number, which are entities of interest in this analysis, do not exist in standard models as they are specific to the DoD. Cases such as these are where it is most beneficial to have a custom NER model help more accurately capture some entities that are not as mainstream as the ones included in a standard NER model.

2.8.3 NER Tools

There are many tools available to help analysts build named entity recognition models. Some of the more popular open-source, code-based packages to perform NER are StanfordNER, OpenNLP, GATE, Spacy, and NLTK. StanfordNER, OpenNLP, and GATE are available for use in Java while Spacy and NLTK are available in Python. For this analysis, packages available in Python are the focus because the contracts data needs to be kept on the HPC system for privacy reasons and Python is readily available for use on the HPC system.

In addition to these code-based approaches, there are some newer tools available to allow analysts to annotate data in a graphical user interface (GUI) format. A lot of these tools require you to download the software to your computer, thus giving up the privacy of your data. Two packages that were found that allow the analyst to use the GUI format of annotating data as well as keep the data secure are Prodigy (a subset of Spacy) and

PyLighter.

2.8.4 Evaluation

While NER models are usually trained as a token-level multi-class classification task with sequence to sequence data, the evaluation is often done on entity-level. This means that for the plain tag format, consecutive tokens with the same tag will be considered as one entity. In the case of the BIO format, entities are well-defined. Since a multi-token entity can be partly correct, there are multiple ways of evaluating a model. A common way to handle this is to treat any entity with any incorrect tokens as an incorrect entity prediction.

Furthermore, as with any machine learning problem, there are multiple metrics for model performance. In NER, F1 scores, harmonic means of precision and recall, are often used. However, there are multiple ways to calculate the F1 score, precision, and recall. The Language-Independent NER task [Sang and Meulder (2003)] introduced at CoNLL-2003, defines precision as the percentage of named entities correctly tagged, and recall as the percentage of named entities that are found. This definition also uses the strict definition of treating partly correct entities as errors. The F1 score is then defined as:

$$F_1 = \frac{2 \times precision \times recall}{precision + recall} \tag{7}$$

This results in an F1 score for each entity class. To aggregate these F1 scores into one final F1 score, a micro- or macro average can be used. The macro average will simply calculate the mean F1 score, considering the relevance of each class equal. Micro average is instead defined using a weighted average, where each class is weighted by the fraction of the total number of samples belonging to the class. So, macro average gives equal importance to each class, whereas micro average gives equal importance to each sample.

3 Related Work

Transformers have since then been used for numerous sequence-to-sequence tasks. These tasks include machine translation, time-series related tasks, text summarizing, name en-

tity recognition, sentiment analysis and even image generation [K. Ahmed and Socher (2017), C. Plizzari and Matteucci (2020), T. Cai and Dai (2019), H. Yan and Qiu (2019), U. Naseem and Imran (2020), N. Parmar and Tran (2018)]. These breakthroughs also allowed the development of models such as the Bidirectional Encoder Representations from Transformers (BERT) [J. Devlin and Toutanova (2019)] and the Generative Pre-trained Transformer 3 (GPT3) [T. B. Brown (2020)]. These model architectures are based on the Transformer architecture but they only rely on one of its components: encoders (BERT) or decoders (GPT3). This structural difference is also a main determinant of how differently the models operate, for example, while GPT3 is an auto-regressive language model, BERT is not. Nevertheless, these models are very powerful also because they're trained using large datasets and contain billions of parameters [T. B. Brown (2020)]. Transformers have also been used for Data-to-Text Generation (D2T) generation tasks. Some of them are similar to the task at hand which is generating text from tabular data. However, in most of these applications, the goal is to obtain a free text description related to a certain tabular data.

Brayan [Johannes Molza Maximilian Bryana (2020)] describes a two step processing pipeline for identifying text reuse of Shakespeare's Hamlet in a corpus of postmodern fiction by comparing n-grams from both sources. Our research did hardly find any previous work regarding neural networks explicitly for annotation of literary texts, whereas several approaches are aiming at similar problems while not being applied in the broader field of intertextuality in digital humanities. An approach that uses alignments is presented by [Burghardt and Liebl (2020)]. The authors use a pre-processed corpus and try to find text reuse for different input queries (=the actual quotes) using a modified Needleman-Wunsch operating on word embeddings. An approach that also uses word vectors to find similar sentences for a given query sentence is presented by [Kenter and de Rijke (2015)]. When comparing not just single words, but rather full sentences, it is necessary to respect the order of words in a sentence.

4 Proposed Method

There are different approaches to fine-tune BERT models for information extraction and labelling. In the scope of this thesis, we train models that are based on a BERT-CRF model architecture. The model architecture is explained in Section 4.1. Furthermore, Sections 4.2 and 4.3 discuss the choice of suitable BERT models and the choice of input formats for all models.

4.1 Model Architecture

The model architecture is based on a BERT-CRF architecture as can be seen in Figure [7]. It is composed of a BERT model with a linear classifier on top followed by a linear-chain CRF. Similar model architectures have also been used in other works ([W. Huang (2019); M. Liu (2020); J. Mao (2019);F. Souza (2019)]). For the following formal definition of our model architecture, we orientate ourselves at the explanations of Lample et al. [G. Lample (2016)] and Souza et al. [F. Souza (2019)].

Given an input sequence X = (x_1, x_2, ..., x_n) of n tokens, BERT generates an output (h_1, h_2, ...,h_n) with hidden dimension H for each output vector h_i . The classification layer maps each of these hidden vectors to the tag space, $R^H \rightarrow \mathrm{R}^T$, where T is the number of used tags (dimension of the tag space). Consequently, the produced output scores P by the linear layer have the dimension $R^{n \times T}$. They are passed to the CRF layer. The parameters of the CRF layer are represented by a matrix of tag transitions $A \in R^{(T+2)\times(T+2)}$. The dimension of the matrix is (T+ 2) × (T + 2) and not T x T because in CRFs two additional states are needed, namely the start and end of the sequence. Otherwise, y_0 and y_{n+1} would not be defined for the later score computations. In the matrix A,$A_{i,j}$ denotes the score of the transition from the i-th tag to the j-th tag. Furthermore, $P_{i,j}$ is the score of the j-th tag of the i-th word in a sequence. Then, the score for a sequence of predictions Y = (y_1, y_2, ..., y_n)is defined as:

$$score(X, Y) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} P_{i, y_i}$$

With the softmax function, one obtains the normalized probability of Y given X:

$$P(Y|X) = \frac{exp(score(X,Y))}{\sum_{Y' \in y_x} exp(score(X,Y'))}$$

, where y_x is the set of all possible tag sequences, given the input X. In the training phase of the model, the log-probability of the correct tag sequence is maximized:

$$log(P(Y|X)) = score(X,Y) - log(\sum_{Y' \in y_x} exp(score(X,Y')))$$

Finally, we can make predictions by choosing the output sequence Y^* that achieves the maximum score out of all other output sequences:

$$Y^* = argmax score(Y|X)$$

Losses and predictions are computed for all WordPiece tokens. However, for every token only the prediction of its first sub-token is taken into account and predictions of further sub-tokens are ignored

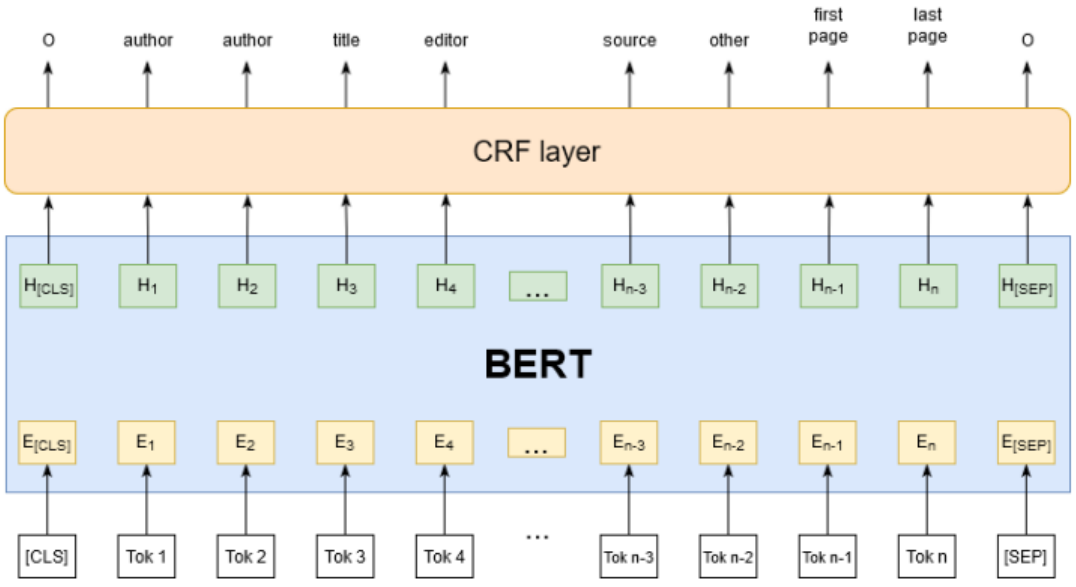

Figure 7: The model architecture of BERT-CRF

4.2 Choice of BERT Model

The choice of the BERT model is crucial. This is because the BERT model returns contextual word representations. The more efficient words of a text can be represented, the better predictions can be made on them. As different pre-trained BERT models can be used within the model architecture, we list our considered BERT models with their important characteristics:

- $BERT_{BASE}$ **uncased** : L=12, H=768, A=12, Total Parameters: 110M. This model was trained on cased English text from Wikipedia and BooksCorpus
- $BERT_{BASE}$ **multilingual uncased** : L=12, H=768, A=12, Total Parameters: 110M. This model was trained on cased text including the top 104 languages. For the text, the largest Wikipedia articles were considered.
- $BERT_{BASE}$ **german uncased** : L=12, H=768, A=12, Total Parameters: 110M. This model was trained on cased German text by Deepset.ai.

In the above provided list, L is the number of layers, H is the hidden size, and A is the number of attention heads.

Moreover, all considered pre-trained BERT models have the 'cased' property. Each of these models also has an analog 'uncased' version. The 'uncased' property of a BERT model indicates that during pre-training of the BERT model,

- cased letters were always lowercased and
- accent markers on words were always removed

before the WordPiece tokenization step. The upper two pre-processing steps were not performed for the training of cased BERT models. This means, choosing cased BERT models ensures that a word in the same context of a sentence is represented differently depending on whether it is in lower case or capitalized.

Devlin et al. [J. Devlin (2019)] used a cased WordPiece model for NER tasks. We state reasons for preferring cased BERT models over uncased BERT models for our task if the language is German:

- The German language uses accent markers which can appear in the German data that is used for the training of our models. Therefore, it is advantageous to use cased models. For instance, the German word “schön” means “beautiful”. When using an uncased BERT model, it will be converted to “schon” which means “already” in the English language. Thus, a cased BERT model is necessary to preserve the semantics of a German sentence.

To find out the most suitable BERT model for the training on our dataset, we will conduct an experiment including all the considered BERT models.

4.3 Input Format

Although the general model architecture is already defined and different pre-trained BERT models are considered, there is still another important decision to be made about the models. The efficient representation of tokens of input sequences depends on how input sequences that go into BERT are created. Therefore, the input format can have a decisive impact on the training and the resulting performance of the models. This section discusses three different input formats for our model. Therefore, it is important to explain how we decided to create input sequences and on what tokens of the input sequence classifications are made.

BERT takes a sequence of tokens as input. This token sequence is passed to a WordPiece model which may tokenize the tokens into further sub-tokens (WordPiece tokens or WordPieces). For example, the token “flightless” is broken down into further sub-tokens “flight” and “less”. Based on WordPiece tokens, the numerical input representation that goes into BERT is computed. Then, BERT outputs a hidden state for every WordPiece token. This means that for every WordPiece token a prediction can be made. In our model, the CRF layer makes predictions on every WordPiece token. However, some predictions are filtered out as we will explain later. In the further course of this chapter, the words “token” and “WordPiece token” are used interchangeably.

For the input format to BERT, there are different options. Before passing an input sequence to BERT, arbitrary pre-processing steps can be made. For example, available text

data can be passed line-wise, sentence-wise, as a randomly shuffled sequence of tokens, etc. Furthermore, the original text of the used data can be manipulated beforehand as this sometimes can come with advantages. Throughout this thesis, we use the terms "sentence" and "linguistic sentence" to refer to a sequence of tokens that form a normal sentence in a language. We chose to construct input sequences based on linguistic sentences due to the following reasons. In the pre-training of BERT [J. Devlin (2019)], long contiguous sequences are used for the input. Furthermore, BERT was pre-trained on next sentence prediction tasks where sequences are always pairs of sentences. As this implies that BERT was pre-trained with many linguistic sentences and learned the relationship between them, it is meaningful to use linguistic sentences as basic units in the input sequences.

We construct input sequences using linguistic sentences. For the input sequence, linguistic sentences can be concatenated, overlapped, merged, etc. However, the input length to the model is limited by BERT. The sequence length of a BERT model allows at most 512 WordPiece tokens as input. This implies that when input sequences have more WordPiece tokens than the specified sequence length of the BERT model, the token sequence is truncated to the sequence length. If one linguistic sentence is tokenized by BERTs WordPiece model and is then already composed of more WordPiece tokens than the sequence length seq_len the tokens that come after the seq_len do not go into computation and are consequently also not predicted. However, this case only occurs very rarely as linguistic sentences are usually not that long. Before input sequences are passed to BERT, they are converted into a numerical representation. Therefore, they undergo necessary pre-processing steps.

4.4 Implementation

The implementation for this thesis is written in the Python programming language. The training of the models is implemented using the deep learning framework PyTorch as well as HuggingFace's Transformers library. For the sentence segmentation and tokenization of the data, the bert tokenizer was used. There are pre-trained models that provide both the functionality of sentence segmentation and tokenization, but often for only one language. As a consequence, we used different models, one for the documents in the German

language and one for the documents in the English language of our data used. To evaluate the performance of our models on different classification metrics (precision, recall, and F1-score), we use the Python module Scikit-learn as well as the Python seqeval package.

5 Data

In this chapter, we describe the data used for training and evaluation of the models. Therefore, the different annotations of the data are described in Section 5.1 and in Section 5.2 it is explained how the data are prepared to be used for our models.

To train and evaluate the different models that were presented in Chapter 4, text data is needed. For the training of our models, we use the two types of data. First one is the letters and second one is drama texts. So depending on the language of the documents inside the dataset, they are divided into two categories: documents in German language and documents in English language. In the following, we describe these two categories in more detail.

- **English articles:** This article contains letters annotated articles in the English language. In this we have letters that are encoded in TEI tags. Thus, in total there are 5060 annotated unique articles in the English language.
- **German articles:** German Drama Corpus (GerDraCor), a collection of TEI P5-encoded German-language plays from 1730 to the 1940s. The corpus is released under the Creative Commons Zero copyright waiver (CC0). These files are encoded in the tei tags. Thus, in total there are 501 annotated unique articles in the German language.

5.1 Data Annotations

Both the datasets are annotated in different TEI tags.

5.1.1 Annotation of Dramas

<**div**>**:** This tag indicates whether a new scene or act is starting and all other tags are stored in these div tags

eg:

<**head**>**:** This tag indicates the number of the act or scene like if its first scene or second scene.

eg: <head> I. Akt.</head>

<**stage**>**:** This tag contains all the stage directions like ”tall, strong man with a coal-black beard, throws the dice with a great noise”.

eg: <stage> großer starker Mann mit kohlschwarzem Bart, wirft die Würfel mit großem Geräusch auf.</stage>

<**speaker**>**:** This tag contains the name of the speaker which is currently speaking.

eg: <speaker>DIETRICH</speaker>

<**p or l**>**:** This tag contains the speech that is spoken by the speaker like ”I have to go home before sunset, noble woman ”

eg: Ich muß noch vor Sonnenuntergang heim, edle Frau

5.1.2 Annotation of Letters

<**TEI**>**:** This tag is the root tag and all other tags are included in this tag.
<**title**>**:** This tag contains the title of the letter.

eg: <title> William S. Ingram to Walt Whitman, 24 December 1890</title>

<**author**>**:** This tag indicates the name of the person who wrote the letter

eg: <author>William S. Ingram</author>

<**date**>**:** This tag contains the date on which the letter is written.

eg: <date> December 24, 1890</date>

<**address**>**:** This tag contains address of both sender and the receiver. It has few <**address**> to store the complete address

eg: <addrLine>University of Nebraska-Lincoln</addrLine>

<**p**>**:** This tag contains what sender wants to write to the receiver i.e. the actual content of the letter.

eg: Hoping you will have a pleasant one and many of them.

5.2 Data Preparation

Named entity recognition (NER) uses a specific annotation scheme, which is defined (at least for European languages) at the word level. An annotation scheme that is widely used is called IOB-tagging, which stands for Inside-Outside-Beginning. Each tag indicates whether the corresponding word is inside, outside or at the beginning of a specific named entity. The reason this is used is because named entities usually comprise more than 1 word.

Let's have a look at an example. If you have a sentence like "Robert Petison was born in Hawaï", then the corresponding tags would be [B-PERS, I-PERS, O, O, O, B-GEO]. B-PERS means that the word 'Robert' is the beginning of a person, I-PERS means that the word 'Petison' is inside a person, 'O' means that the word 'was' is outside a named entity, and so on. So one typically has as many tags as there are words in a sentence.

As we have files in deep hierarchical structures, so first we will remove the deep hierarchies and keep our data only in flat hierarchy. So after that, if we convert our data in IOB format as discussed above will look like something which is shown in Figure [8].

Then we will create 2 dictionaries: one that maps individual tags to indices, and one

that maps indices to their individual tags. This is necessary in order to create the labels (as computers work with numbers = indices, rather than words = tags).

Now, we have to ask ourselves the question: what is a training example in the case of NER, which is provided in a single forward pass? A training example is typically a sentence, with corresponding IOB tags. Let's group the words and corresponding tags by sentence as shown in Figure [9]

Let's only keep the $'sentence'$ and $'word\_labels'$ columns, and drop duplicates to train our model as shown in Figure[10]

	0	1	2
0	sentence:0	duk.00340.xml	<filename>
1	sentence:1	duk.00340.xml	<filename>
2	sentence:2		<?xml>
3	sentence:2		<?oxygen>
4	sentence:3	George	<title>
..	...	...	...
95	sentence:22	may	O
96	sentence:22	be	O
97	sentence:22	shared	O
98	sentence:22	in	O
99	sentence:22	accordance	O

Figure 8: Letters in IOB format

	Sentence	word	tag	sentence	word_labels
0	sentence:0	duk.00340.xml	<filename>	duk.00340.xml	<filename>
1	sentence:1	duk.00340.xml	<filename>	duk.00340.xml	<filename>
2	sentence:2		<?xml>		<?xml>,<?oxygen>
3	sentence:2		<?oxygen>		<?xml>,<?oxygen>
4	sentence:3	George	<title>	George Washington Whitman to Louisa Van Velsor...	<title>,O,O,O,O,O,O,O,O,O,</title>

Figure 9: Data after combining per sentence

.

	sentence	word_labels
0	duk.00340.xml	<filename>
1		<?xml>,<?oxygen>
2	George Washington Whitman to Louisa Van Velsor...	<title>,O,O,O,O,O,O,O,O,O,</title>
3	a machine readable transcription	<title>,O,O,</title>
4	George Washington Whitman	<author>,O,</author>

Figure 10: Actual training data

A tricky part of NER with BERT is that BERT relies on wordpiece tokenization, rather than word tokenization. This means that we should also define the labels at the wordpiece-level, rather than the word-level!

For example, if you have word like ”Washington” which is labeled as ”b-gpe”, but it gets tokenized to ”Wash”, ”##ing”, ”##ton”, then one approach could be to handle this by only train the model on the tag labels for the first word piece token of a word (i.e. only label ”Wash” with ”b-gpe”).

Note that this is a design decision. You could also decide to propagate the original label of the word to all of its word pieces and let the model train on this. In that case, the model should be able to produce the correct labels for each individual wordpiece. Another design decision could be to give the first wordpiece of each word the original word label, and then use the label “X” for all subsequent subwords of that word. All of them seem to lead to good performance.

Below, we define a regular PyTorch dataset class (which transforms examples of a dataframe to PyTorch tensors). Each sentence gets tokenized, the special tokens that BERT expects are added, the tokens are padded or truncated based on the max length of the model, the attention mask is created and the labels are created based on the dictionary which we defined above. Word pieces that should be ignored have a label of -100 (which is the default ignore_index of PyTorch’s CrossEntropyLoss) as shown in Figure [11].

```
{'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0]),
 'input_ids': tensor([  101, 18243,  1012,  2199,  2692,  2629,  1012, 20950,   102,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0]),
 'labels': tensor([-100,    0, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
        -100, -100, -100, -100, -100, -100, -100, -100]),
 'token_type_ids': tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0])}
```

Figure 11: Data after tokenization

6 Evaluation

6.1 Evaluation Procedure

The evaluation procedure comprises the used evaluation metrics, the calculation of the results and the model selection during training time.

6.1.1 Calculation of the Results

Given an input sequence, it is tokenized by the underlying WordPiece model of a selected BERT model. It is of high importance that each token prediction in the original input sequence is determined by taking into account only the prediction of its first sub-token. Thus, for the prediction of the word "flightless", only the prediction of its first sub-token "flight" is taken into account. The prediction of further sub-tokens of a word (here: "##less") is ignored. Depending on the model, the results are calculated differently. In the following explanation, the term "token" always refers to the first sub-token of a word (WordPiece token) after the word has been tokenized by a WordPiece model:

- In the data, we used the BIO tagging format which is a common tagging scheme for the task of NER. For the evaluation of the models, we treat reference strings as named entities. We evaluate on the entity-level which means that a reference string is only considered to be classified correctly by the model if and only if all its tokens are classified as reference. Furthermore, the tag O is not considered in the calculation of the results since it is not a meaningful tag. This is because (a) it annotates plain text and (b) if the model was to predict every token with the tag O, still high performance would be achieved for the tag O (because it is by far the most dominant tag in the annotated data).

6.1.2 Experimental Setup

We split the data ch that 80% are used for training and the remaining 10% for testing. Model selection describes the process of choosing the model's "best state" during the training time based on its performance. We use AdamW algorithm which is an improved version of Adam for optimizing our networks, with batches of size 256.For developing our models we used PyTorch and Scikit-learn. To calculate the loss we are using CrossEntropyLoss (which is used by BertForTokenClassification) uses mean reduction by default, it will compute the mean loss for each of the tokens in the sequence for which a label is provided. Here we define the model, BertForTokenClassification, and load it with the pre-trained weights of specific bert which is chosen as per experiment. The only thing we

need to additionally specify is the number of labels (as this will determine the architecture of the classification head).

Note that only the base layers are initialized with the pre-trained weights. The token classification head of the top has just randomly initialized weights, which we will train, together with the pre-trained weights, using our labelled dataset.

6.2 Experiments

In this section, we investigate the empirical performance of our proposed model on various tasks and compare the results. We will split this dataset into two sets : train and test. For every task, we will split the dataset randomly. We will fine-tune the model using the train set and make predictions for the test set.

6.2.1 Experiments on letters

In this we will extract the data from the letters which are encoded in tei tags. In total, we have 5060 files. So we will randomly split these files into train and test data. We have 4048 files for training and 1012 for testing. After applying sentence tokenization it results into 86677 sentences which is the size of our training dataset. For testing, we have 17402 sentences. For this task, we have used the 'bert-base-uncased' as a baseline model. So we will perform experiments using the different number of epochs and learning rate.

Experiment 1 In this we will train the model on *10 epochs* with the *learning rate of 1e-05*. For training data we achieved the accuracy of 0.97 and loss is 0.12 and for the testing data f1-score is 0.9045, accuracy is 0.94 and loss is 0.25 as shown in Table [3].

	loss	accuracy
Training dataset	0.12729	0.97403
Testing dataset	0.25185	0.94566

Table 3: Results of Experiment 1 on letters.

Experiment 2 In this we will train the model on *15 epochs* with the *learning rate of 1e-05*. For training data we achieved the accuracy of 0.98 and loss is 0.06 and for the testing data f1-score is 0.9283, accuracy is 0.96 and loss is 0.19 as shown in Table [4].

	loss	accuracy
Training dataset	0.06424	0.98799
Testing dataset	0.19193	0.96147

Table 4: Results of Experiment 2 on letters.

Experiment 3 In this we will train the model on *20 epochs* with the *learning rate of 1e-05*. For training data we achieved the accuracy of 0.99 and loss is 0.04 and for the testing data f1-score is 0.9295, accuracy is 0.96 and loss is 0.17 as shown in Table [5].

	loss	accuracy
Training dataset	0.04045	0.99053
Testing dataset	0.17504	0.96566

Table 5: Results of Experiment 3 on letters.

Experiment 4 In this we will train the model on *10 epochs* with the *learning rate of 2e-05*. For training data we achieved the accuracy of 0.98 and loss is 0.05 and for the testing data f1-score is 0.9321, accuracy is 0.96 and loss is 0.18 as shown in Table [6].

	loss	accuracy
Training dataset	0.05755	0.98797
Testing dataset	0.18846	0.96148

Table 6: Results of Experiment 4 on letters.

Experiment 5 In this we will train the model on *15 epochs* with the *learning rate of 2e-05*. For training data we achieved the accuracy of 0.99 and loss is 0.03 and for the testing data f1-score is 0.9264, accuracy is 0.96 and loss is 0.19 as shown in Table [7].

	loss	accuracy
Training dataset	0.03847	0.99037
Testing dataset	0.19676	0.96075

Table 7: Results of Experiment 5 on letters.

Experiment 6 In this we will train the model on *20 epochs* with the *learning rate of 2e-05*. For training data we achieved the accuracy of 0.99 and loss is 0.02 and for the testing data f1-score is 0.9394, accuracy is 0.96 and loss is 0.16 as shown in Table [8].

	loss	accuracy
Training dataset	0.02861	0.99174
Testing dataset	0.16970	0.96746

Table 8: Results of Experiment 6 on letters.

So from above experiments, we observed that the best combination of hyperparameters : **epochs = 20 and learning rate = 2e-05**. Now we will use these for further experiments on drama to save our time.

Experiment 7 In this we will train the model on *20 epochs* with the *learning rate of 2e-05* and *'bert-base-multilingual-uncased'* as baseline model. For training data we achieved the accuracy of 0.97 and loss is 0.12 and for the testing data f1-score is 0.9472, accuracy is 0.96 and loss is 0.19 as shown in Table [9].

	loss	accuracy
Training dataset	0.12029	0.97500
Testing dataset	0.19215	0.96353

Table 9: Results of Experiment 7 on letters.

6.2.2 Experiments on German Drama

In this we will extract the data from the german drama texts which are encoded in tei tags. In total, we have 501 files. So we will randomly split these files into train and test data. We have 401 files for training and 100 for testing. After applying sentence tokenization it results into 828718 sentences which is the size of our training dataset. For testing, we have 201552 sentences. For this task, we have used the 'bert-base-uncased' and 'bert-base-german-uncased' as a baseline model. So we will perform experiments using different bert as baseline model.

Experiment 1 In this we will train the model on *20 epochs* with the *learning rate of 2e-05* and *'bert-base-uncased'* as baseline model. For training data we achieved the accuracy of 0.98 and loss is 0.04 and for the testing data f1-score is 0.9466, accuracy is 0.94 and loss is 0.22 as shown in Table [10].

	loss	accuracy
Training dataset	0.04215	0.98596
Testing dataset	0.22311	0.94632

Table 10: Results of Experiment 1 on German Drama.

Experiment 2 In this we will train the model on *20 epochs* with the *learning rate of 2e-05* but *'bert-base-german-uncased'* as baseline model. For training data we achieved the accuracy of 0.99 and loss is 0.02 and for the testing data f1-score is 0.9578, accuracy is 0.959 and loss is 0.17 as shown in Table [11].

	loss	accuracy
Training dataset	0.02596	0.99159
Testing dataset	0.17208	0.959766

Table 11: Results of Experiment 2 on German Drama.

Experiment 3 In this we will train the model on *20 epochs* with the *learning rate of 2e-05* but *'bert-base-multilingual-uncased'* as baseline model. For training data we

achieved the accuracy of 0.99 and loss is 0.02 and for the testing data f1-score is 0.9528, accuracy is 0.958 and loss is 0.18 as shown in Table [12].

	loss	accuracy
Training dataset	0.02581	0.99157
Testing dataset	0.183433	0.95860

Table 12: Results of Experiment 3 on German Drama.

6.2.3 Experiments on French Drama

In this we will extract the data from the french drama texts which are encoded in tei tags. In total, we have 1560 files. So we will randomly split these files into train and test data. We have 1248 files for training and 312 for testing. After applying sentence tokenization it results into 1183912 sentences which is the size of our training dataset. For testing, we have 350992 sentences. For this task, we have used the 'bert-base-uncased' and 'bert-base-french-uncased' as a baseline model. So we will perform experiments using different bert as baseline model.

Experiment 1 In this we will train the model on *20 epochs* with the *learning rate of 2e-05* and *'bert-base-uncased'* as baseline model. For training data we achieved the accuracy of 0.98 and loss is 0.03 and for the testing data f1-score is 0.9259, accuracy is 0.93 and loss is 0.25 as shown in Table [13].

	loss	accuracy
Training dataset	0.03530	0.98858
Testing dataset	0.25389	0.93249

Table 13: Results of Experiment 1 on French Drama.

Experiment 2 In this we will train the model on *20 epochs* with the *learning rate of 2e-05* but *'bert-base-french-cased'* as baseline model. For training data we achieved the accuracy of 0.98 and loss is 0.05 and for the testing data f1-score is 0.9195, accuracy is 0.93 and loss is 0.31 as shown in Table [14].

	loss	accuracy
Training dataset	0.05014	0.98238
Testing dataset	0.310868	0.93249

Table 14: Results of Experiment 2 on French Drama.

Experiment 3 In this we will train the model on *20 epochs* with the *learning rate of 2e-05* but *'bert-base-multilingual-uncased'* as baseline model. For training data we achieved the accuracy of 0.97 and loss is 0.07 and for the testing data f1-score is 0.9271, accuracy is 0.93 and loss is 0.20 as shown in Table [15].

	loss	accuracy
Training dataset	0.07151	0.97189
Testing dataset	0.20074	0.93684

Table 15: Results of Experiment 3 on French Drama.

6.2.4 Experiments on Russian Drama

In this we will extract the data from the Russian drama texts which are encoded in tei tags. In total, we have 212 files. So we will randomly split these files into train and test data. We have 168 files for training and 44 for testing. After applying sentence tokenization it results into 222783 sentences which is the size of our training dataset. For testing, we have 66224 sentences. For this task, we have used the 'bert-base-uncased' and 'bert-base-multilingual-uncased' as a baseline model because we do not have uncased russian bert baseline model. So we will perform experiments using different bert as baseline model.

Experiment 1 In this we will train the model on *20 epochs* with the *learning rate of 2e-05* and *'bert-base-uncased'* as baseline model. For training data we achieved the accuracy of 0.97 and loss is 0.06 and for the testing data f1-score is 0.9397, accuracy is 0.94 and loss is 0.18 as shown in Table [16].

	loss	accuracy
Training dataset	0.06589	0.97466
Testing dataset	0.180588	0.94144

Table 16: Results of Experiment 1 on Russian Drama.

Experiment 2 In this we will train the model on *20 epochs* with the *learning rate of 2e-05* but *'bert-base-multilingual-uncased'* as baseline model because there is not any russian base bert which we can use as baseline model. For training data we achieved the accuracy of 0.99 and loss is 0.01 and for the testing data f1-score is 0.9493, accuracy is 0.95 and loss is 0.22 as shown in Table [17].

	loss	accuracy
Training dataset	0.01731	0.99409
Testing dataset	0.22157	0.95165

Table 17: Results of Experiment 2 on Russian Drama.

7 Discussion

In this chapter, the results are discussed in more detail. First, the obtained results in the experiments performed above are interpreted by relating them to the sub-research questions. After that, the main research question is answered. Finally, this section is concluded by describing the limitations of this thesis and by pointing out potential future work.

7.1 Interpretation of the Results

We begin the interpretation of the results by focusing on the first sub-research question of this thesis:

Is BERT a suitable choice for the task of labelling of TEI tags?

Yes, BERT is a suitable choice for the task of labelling of TEI tags. We justify this answer with the following arguments: We have fine-tuned BERT-based models with little data like Russian drama and still achieved high performance. This indicates that little data is sufficient such that BERT models can learn word representations efficiently on the downstream task of reference extraction and segmentation by fine-tuning. Furthermore, this partly shows the great capability of neural-based embeddings for reference extraction and segmentation. With a suitable pre-trained BERT model and meaningful input format, we were able to label data with high quality. Although our dataset contains English, German, French and Russian language articles with references that strongly vary in their content, length, and tags, our model labelled tags with an average accuracy of 96%. This shows that BERT is capable of learning the differences and similarities of high-variance tags and is therefore suitable for the task of labelling tei tags.

In the following, the second sub-research question is discussed:
Which BERT model and input format achieve the best performance on the task?

We saw that the performance of the models varies on the same dataset, depending on what BERT model is used. So if we have to choose one then 'bert-base-multilingual-uncased' worked very well on every dataset. So 'bert-base-multilingual-uncased' is the best as baseline model. The reason for this is that it learned word embeddings for German, English, French and Russian texts in its pre-training procedure. Consequently, it achieved better results than the BERT models 'bert-base-german-uncased', 'bert-base-french-cased' or 'bert-base-uncased' that only learned word embeddings for one of the four languages. From this follows that it was important to always choose 'bert-base-multilingual-uncased' in experiments.

Our observations and interpretations show that BERT models have a noticeable impact on the labelling of tags. We argue that the better a BERT model represents words of a reference in their context, the more accurate predictions can be made. Furthermore, our findings lead to the statement that the choice of the BERT model is highly dependent on what data is used for training and testing. Depending on the data, a BERT model should be selected based on

- the language(s) it was trained on (e.g. English, German, etc.)

- the domain-specific text corpus it was trained on (e.g. Wikipedia articles, scientific articles, etc.),
- whether a case-preserving WordPiece Model is used or not.

The more criteria match between the BERT model's pre-training data and the BERT model's fine-tuning data, the better the BERT model captures both syntactic and semantic meanings of tags.

Next, we discuss what input format achieves the best performance. The obtained results indicate that the input format to a BERT model has a significant impact on the model's overall classification performance. Models with the single input format had relatively poor performance compared to the two other input formats. The reason behind this is that the maximum context and CMV-based input formats both provide significantly more context around single sentences, allowing the model to utilize more information and make predictions with higher certainty. Extracting a reference solely using the information of its inner structure and content is harder than using further contextual information. Thus, tags can be identified better when the information around the reference is used. The information around tags can provide many clues to the model. Based on this observation, for high classification performance, rather long input sequences should be preferred such that there is sufficient context around each token. We conclude that input formats that generally define long, context-based input sequences significantly improve the performance of models. We argue that such input formats lead to better contextual word representations.

In what follows, the last sub-research question is discussed:

What is the best approach for training a given BERT-based model on the task?

From the above experiments, we conclude that if we train the bert on epochs = 20 and use Adam optimizer with learning rate = 2e-05, it gives the best results. We define the model, BertForTokenClassification, and load it with the pretrained weights of "bert-base-uncased". The only thing we need to additionally specify is the number of labels (as this will determine the architecture of the classification head).

Note that only the base layers are initialized with the pretrained weights. The token

classification head of the top has just randomly initialized weights, which we will train, together with the pretrained weights, using our labelled dataset.

Finally, the main research question is discussed and answered in the following:

To what extent can active learning techniques accelerate the model training of state-of-the-art language models and hence minimize the manual annotation effort of XML files in the context of Named Entity Recognition?

By answering the sub-research questions we come to this conclusion that active learning techniques accelerate the model training of state- of-the-art language models and hence minimize the manual annotation effort of XML files in the context of Named Entity Recognition.

7.2 Error Analysis

To do the error analysis we will generate the XML file from the predicted data and then we will compare the differences between the original file and the newly generated file.

For letters In letters, we achieved an accuracy of 0.97. The generated XML file is shown in Figure [12].

```
<TEI>
 <title>
  Walt Whitman to Unidentified Correspondent, 8 April1887
 </title>
 <title>
  a machine readable transcription
 </title>
 <author>
  Walt Whitman
 </author>
 <editor>
  Kenneth M. Price
 </editor>
 <editor>
  Ed Folsom
 </editor>
 <resp>
  Transcription and encoding
 </resp>
 <persName>
  Ryan Furlong
 </persName>
```

Figure 12: XML file for letters

Now if the XML file is generated, we will compare the similarity between the original file and the new file generated with predictions by using **xmldiff**. In figure [13] the differences are shown between the two files and the no of diff nodes, total nodes and percentage of similarity is shown in figure [14].

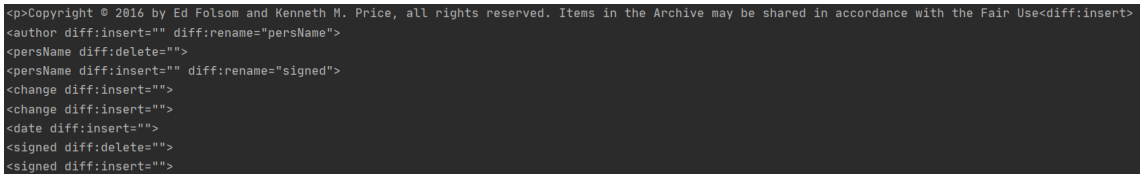

Figure 13: Differences between two XML files

Figure 14: Similarity Percentage

By using an online XML Comparison tool we can see the differences also as shown in figure[15]

Figure 15: XML Comparison using online tool

Let's take a deep look into the sentences which are not labelled correctly:

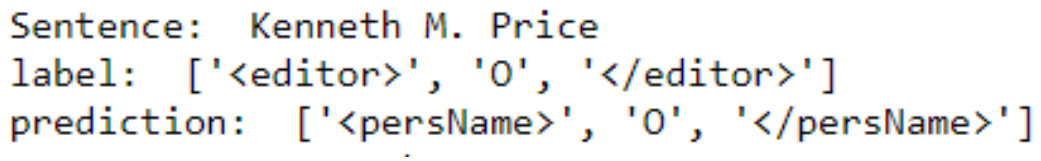

Figure 16: In this, <editor> is labelled as <persName> which we can understand because it is a name of the person.

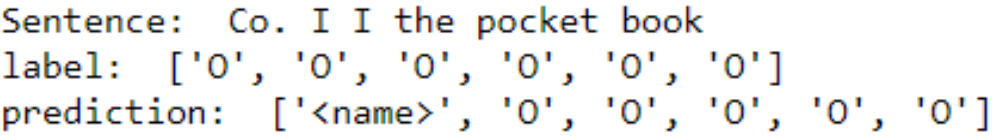

Figure 17: In this, 'Co.' is labelled as <name> because usually Co. stands for Company , so it can be name of Company.

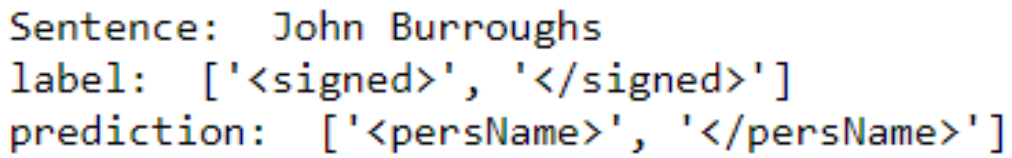

Figure 18: In this, <editor> is labelled as <persName> which we can understand because it is again name of the person.

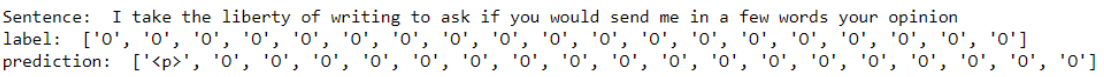

Figure 19: In letters, the text is contained in <p> tags and it is a text which sender wants to write to receiver.

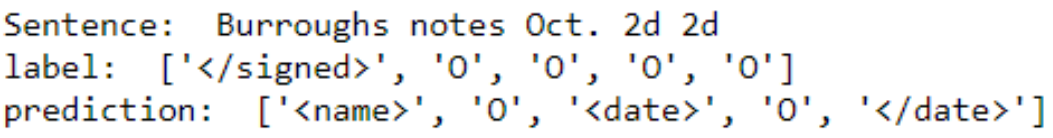

Figure 20: In this, again name is a problem and Oct. 2d stands for date so it is correct.

So from above examples, we conclude that model is having a problem to label names because the <signed> , <name> and <persName> all are almost similar.

For Drama Below we are attaching some examples which are not labelled correctly.

```
Sentence:  Reihe. Weißt du, nach wem ich gesprochen habe?
label:  ['O', 'O', 'O', 'O', 'O', 'O', 'O', '</p>']
prediction:  ['</stage>', 'O', 'O', 'O', 'O', 'O', 'O', '</p>']
```

Figure 21: Example 1

```
Sentence:  Ein Jahr lang hab' ich dich nicht gesehen. Laß dich begrüßen. Er will sie
label:  ['<p>', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '<stage>', 'O', 'O']
prediction:  ['<p>', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '<stage>', 'O', 'O', 'O', 'O']
```

Figure 22: Example 2

```
Sentence:  Nie hat eine Frau ein prächtigeres Hochzeitsgeschenk bekommen. Komm, Léocadie; auf
label:  ['<p>', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
prediction:  ['<p>', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '<stage>', '<stage>', '<stage>']
```

Figure 23: Example 3

```
Sentence:  Mach' dich nicht lächerlich! Zu den Dreien. Ihr braucht noch nicht
label:  ['<p>', 'O', 'O', 'O', '<stage>', 'O', '</stage>', 'O', 'O', 'O', 'O']
prediction:  ['<p>', 'O', 'O', 'O', '<stage>', '<stage>', '</stage>', '</stage>', 'O', 'O', 'O']
```

Figure 24: Example 4

In drama we have conflicts between normal sentences and stage directions.

7.3 Limitations And Future Work

Although satisfactory results were achieved, the trained labelling models are far from being perfect. There are some important limitations to consider when interpreting the results. First is that for now we are only able to label the flat hierarchies and we know all these dramas and letters have text encoded in deep hierarchies and we also removed the attributes of the tags but in the dataset we have tags with attributes.

Another limitation to be considered when interpreting the results is the amount of data that the models were trained with. All models in this thesis were fine-tuned with data, consisting of roughly 3,478,000 sentences or less. Thus, all models are trained with little data and are evaluated on little data. To set it in contrast, the published BERT models were trained with about 3,300 million words. Future research in this field could fine-tune BERT models with more data (with annotated tags) and consequently evaluate on more data, finding out how well BERT generalizes on generating TEI-based XML for literary texts.

For future work, we discuss potential improvements to the constructed models in this thesis. As mentioned in limitations that we are currently labelling only flat hierarchies so in the future we can try to label deep hierarchies. And we can also try to add attributes also to the tags. Or maybe we can use Masked Classification where we will mask the tags and let the bert predict the tags. But it can be challenging because bert has not seen these tags during pre-training. Furthermore, the performance of BERT ensembles should be investigated. In our implementation, we always used the last checkpoint of pre-trained BERT models. However, the performance of BERT models can be examined using different checkpoints that were made earlier than the last checkpoint during the pre-training phase.

8 Conclusion

This thesis aimed to annotate literary text with their corresponding tei tags with the use of BERT. To train the most principled classification models, different approaches, BERT models, and input formats were examined. Our findings showed that the 'bert-base-multilingual-uncased' is more suitable. Furthermore, we found out that the more context is provided around the words of an input sequence, the more precisely do the models extract and segment references. Also, the choice of the BERT model plays an important role. Choosing a BERT model whose pre-training data fits the fine-tuning data leads to better results since better word embeddings are used. Overall, our results confirm that BERT is a suitable choice for TEI tagging. This is because sentences extracted from

text data are tagged with a high classification performance and correctly with high accuracy. We achieved results, partly illustrating that neural-based embeddings are a strong alternative to the manual annotation of literary texts. As we pointed out earlier, there is still much potential to improve the obtained results in the research field of XML file generation. We believe that BERT combined with a feature engineering approach would lead to a promising model, as additional valuable information would be preserved and could be utilized by the model. Ultimately, automatic XML file generation is the main factor to build and maintain large corpus databases. It would be very interesting and practical to develop a BERT-based service that goes from the tagging of documents with tei tags to the automatic XML file generation, finding its use in production. The insights gained from this Master thesis could help build models with good performance. For example, such an application could find its usage in the online server where documents are published daily and XML file generation is an essential part of their databases.

List of Figures

1 A problem solved easily by Perceptron, with a decision boundary (dotted line) drawn between the two classes. 13

2 The XOR problem, that cannot be solved by a Perceptron. No decision boundary can be drawn for a single layered Perceptron, as no single straight line can separate the two different classes shown. The XOR problem can be solved with MLPs. 14

3 LSTM illustration [23] 16

4 Transformer architecture illustration used by Vaswani et al. [Vaswani (2017)].Figure shows encoder to the left and decoder to the right. Many of these layers can be stacked to create a deeper model. 21

5 High level overview of how the encoder and decoder shares information in deep Transformer architectures. 22

6 The composition of BERT with a task-specific neural network head on top. 28

7 The model architecture of BERT-CRF 36

8 Letters in IOB format 43

9 Data after combining per sentence 43

10 Actual training data 44

11 Data after tokenization 45

12 XML file for letters 57

13 Differences between two XML files 58

14 Similarity Percentage 58

15 XML Comparison using online tool 58

16 In this, <editor> is labelled as <persName> which we can understand because it is a name of the person. 59

17 In this, 'Co.' is labelled as <name> because usually Co. stands for Company , so it can be name of Company. 59

18 In this, <editor> is labelled as <persName> which we can understand because it is again name of the person. 59

19 In letters, the text is contained in <p> tags and it is a text which sender wants to write to receiver. 59

20 In this, again name is a problem and Oct. 2d stands for date so it is correct. 59

21 Example 1 60

22 Example 2 60

23 Example 3 60

24 Example 4 60

List of Tables

1 Three common activation functions. 14

2 The size configurations of BERT. 26

3 Results of Experiment 1 on letters. 47

4 Results of Experiment 2 on letters. 48

5 Results of Experiment 3 on letters. 48

6 Results of Experiment 4 on letters. 48

7 Results of Experiment 5 on letters. 49

8 Results of Experiment 6 on letters. 49

9 Results of Experiment 7 on letters. 49

10 Results of Experiment 1 on German Drama. 50

11 Results of Experiment 2 on German Drama. 50

12 Results of Experiment 3 on German Drama. 51

13 Results of Experiment 1 on French Drama. 51

14 Results of Experiment 2 on French Drama. 52

15 Results of Experiment 3 on French Drama. 52

16 Results of Experiment 1 on Russian Drama. 53

17 Results of Experiment 2 on Russian Drama. 53

References

H. Schwenk L. Barrault A. Conneau, D. Kiela and A. Bordes. "supervised learning of universal sentence representations from natural language inference data". page 670–680, 2017. URL https://aclanthology.org/D17-1070.

Dana H. Ballard. "modular learning in neural networks". *Proceedings of the Sixth National Conference on Artificial Intelligence*, 1:279–284, 1987. URL http://dl.acm.org/citation.cfm?id=1863696.1863746.

Yoshua Bengio. "neural probabilistic language models". *Innovations in Machine Learning: Theory and Applications*, page 137–186, 2006. URL https://doi.org/10.1007/3-540-33486-6_6.

M. Burghardt and B. Liebl. "the vectorian - eine parametrisierbare suchmaschine für intertextuelle referenzen". *Book of Abstracts, DHd*, 2020.

M. Cannici C. Plizzari and M. Matteucci. "skeleton-based action recognition via spatial and temporal transformer networks". *arXiv:2008.07404*, 2020. URL http://arxiv.org/abs/2008.07404.

Andrew M. Dai and Quoc V. Le. "semi-supervised sequence learning". *Advances in neural information processing systems*, page 3079–3087, 2015. URL https://arxiv.org/abs/1511.01432.

Zihang Dai. "transformer-xl: Attentive language models beyond a fixed-length context". 2019. URL http://arxiv.org/abs/1901.02860.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "imagenet: A large-scale hierarchical image database". *2009 IEEE Conference on Computer Vision and Pattern Recognition*, page 248–255, 2009. URL https://ieeexplore.ieee.org/document/5206848.

Jacob Devlin. "bert: Pre-training of deep bidirectional transformers for language understanding". *(2018)*, 2018. URL https://doi.org/10.48550/arXiv.1810.04805.

Stuart Dreyfus. “the evolution of sentiment analysis—a review of research topics, venues, and top cited papers”. *Computer Science Review*, 5.1:16–32, 2018. URL https://doi.org/10.1016/0022-247X(62)90004-5.

Giampaolo Cristadoro Eduardo G Altmann and Mirko Degli Esposti. “on the origin of long-range correlations in texts”. *Proceedings of the National Academy of Sciences*, 109.29:11582–11587, 2012.

E. Gogoulou E. Y. Hellqvist F. Carlsson, A. C. Gyllensten and M. Sahlgren. “semantic re-tuning with contrastive tension”. *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Ov_sMNau-PF.

R. de Alencar Lotufo F. Souza, R. F. Nogueira. “portuguese named entity recognition using bert-crf”. *CoRR*, 2019. URL http://arxiv.org/abs/1909.10649.

Timothy Baldwin Fei Liu and Trevor Cohn. “capturing long-range contextual dependencies with memory-enhanced conditional random fields”. *Proceedings of the Eighth International Joint Conference on Natural Language Processing*, 1:555–565, 2017.

S. Subramanian K. Kawakami C. Dyer G. Lample, M. Ballesteros. “neural architectures for named entity recognition”. *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 260–270, 2016. URL https://doi.org/10.18653/v1/n16-1030.

Deepali K. Gaikwad and C. Namrata Mahender. “a review paper on text summarization”. *International Journal of Advanced Research in Computer and Communication Engineering*, 5(3):154–160, 2016.

Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. “learning to forget: Continual prediction with lstm”. *Proc ICANN’99 Int. Conf. on Artifical Neural Networks*, 2: 850–855, 1999. URL https://doi.org/10.48550/arXiv.1011.1669.

Alexander Greaves-Tunnell and Zaid Harchaoui. “a statistical investigation of long memory in language and music”. *arXiv preprint*, 2019.

Klaus Gref. “lstm: A search space odyssey”. *IEEE Transactions on Neural Networks and Learning Systems*, 2017. URL https://doi.org/10.48550/arXiv.1503.04069.

X. Li H. Yan, B. Deng and X. Qiu. “tener: Adapting transformer encoder for named entity recognition”. *arXiv:1911.04474*, 2019. URL http://arxiv.org/abs/1911.04474.

Kaiming He. “deep residual learning for image recognition”. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1:770–778, 2016. URL https://arxiv.org/abs/1512.03385.

Sepp Hochreiter. “the vanishing gradient problem during learning recurrent neural nets and problem solutions”. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6.02:107–116, 1998.

Sepp Hochreiter and Jürgen Schmidhuber. “long short-term memory”. *Neural Computation*, 9.8:1735–1780, 1997. URL https://doi.org/10.48550/arXiv.1206.2944.

John J. Hopfield. “neural networks and physical systems with emergent collective computational abilities”. *Proceedings of the national academy of sciences*, page 2554–2558, 1982.

Jeremy Howard and Sebastian Ruder. “universal language model finetuning for text classification”. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 1:328–339, 2018. URL https://arxiv.org/abs/1801.06146.

K. Lee J. Devlin, M.-W. Chang and K. Toutanova. “bert: Pre-training of deep bidirectional transformers for language understanding”. *arXiv:1810.04805*, 2019. URL http://arxiv.org/abs/1810.04805.

K. Lee K. Toutanova J. Devlin, M. Chang. “bert: Pre-training of deep bidirectional transformers for language understanding”. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human*

Language Technologies, 1:4171–4186, 2019. URL https://doi.org/10.18653/v1/n19-1423.

W. Liu. J. Mao. "hadoken: a bert-crf model for medical document anonymization". *Proceedings of the Iberian Languages Evaluation Forum co-located with 35th Conference of the Spanish Society for Natural Language Processing*, 2421:720–726, 2019. URL http://ceur-ws.org/Vol2421/MEDDOCAN_paper_11.pdf.

Richard Socher Jeffrey Pennington and Christopher Manning. "glove:global vectors for word representation". *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, page 1532–1543, 2014.

Jamie Ryan Kiros Jimmy Lei Ba and Geoffrey E. Hinton. "layer normalization". 2016. URL https://arxiv.org/abs/1607.06450.

Manuel Burghardt Johannes Molza Maximilian Bryana. "a computational expedition into the undiscovered country - evaluating neural networks for the identification of hamlet text reuse". 2020. URL http://ceur-ws.org/Vol-2723/long22.pdf.

Andrew McCallum John D Lafferty and Fernando C N Pereira. "conditional random fields: Probabilistic models for segmenting and labeling sequence data". *Proceedings of the Eighteenth International Conference on Machine Learning.*, 2001. URL https://doi.org/10.48550/arXiv.1011.4088.

Karen Spärck Jones. "a statistical interpretation of term specificity and its application in retrieval". *Journal of Documentation*, 28:11–21, 1978.

N. S. Keskar K. Ahmed and R. Socher. "weighted transformer network for machine translation". *arXiv:1711.02132*, 2017. URL http://arxiv.org/abs/1711.02132.

T. Kenter and M. de Rijke. "short text similarity with word embeddings". *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2015. URL http://doi.acm.org/10.1145/2806416.2806475.

Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. “a survey on deep learning for named entity recognition”. *CoRR XX*, (1), 2018. URL https://arxiv.org/abs/1509.00685.

Z. Wang X. Xu. M. Liu, Z. Tu. “ltp: A new active learning strategy for bertcrf based named entity recognition”. *CoRR*, 2020. URL http://arxiv.org/abs/2001.02524.

Zoraida Callejas Michael McTear and David Griol Barresa. volume 1. Springer International Publishing, 2016.

Tomas Mikolov. “efficient estimation of word representations in vector space”. *Proceedings of Workshop at ICLR*, 2013.

Marvin Minsky and Seymour A. Papert. *Perceptrons, An Introduction to Computational Geometry*. MIT Press, 1969.

Mika V. Mäntylä, Daniel Graziotin, and Miikka Kuutila. “the evolution of sentiment analysis—a review of research topics, venues, and top cited papers”. *Computer Science Review*, page 16–32, 2018.

J. Uszkoreit Kaiser N. Shazeer A. Ku N. Parmar, A. Vaswani and D. Tran. “image transformer”. *arXiv:1802.05751*, 2018. URL http://arxiv.org/abs/1802.05751.

Vinod Nair and Geoffrey E Hinton. “rectified linear units improve restricted boltzmann machines”. *Proceedings of the 27th international conference on machine learning (ICML-10)*, page 807–814, 2010.

Matthew Peters. “deep contextualized word representations”. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1:2227–2237, 2018. URL http://allennlp.org/elmo.

Alec Radford. Improving language understanding with unsupervised learning. 2018. URL https://openai.com/blog/languageunsupervised/.

L. Ramshaw and M. Marcus. "text chunking using transformation-based learning". *Third Workshop on Very Large Corpora*, 1995. URL https://www.aclweb.org/anthology/W95-0107.

N. Reimers and I. Gurevych. "sentence-bert: Sentence embeddings using siamese bert-networks". page 3973–3983, 2019. URL https://aclanthology.org/D19-1410.

Tomer Peleg Ron Rubinstein and Michael Elad. "analysis k-svd: A dictionary-learning algorithm for the analysis sparse model". *IEEE Transactions on Signal Processing*, 61.3:661–677, 2013.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "learning representations by back-propagating errors". *Cognitive modeling*, 5(3):1, 1988. URL https://doi.org/10.1088/1751-8113/44/8/085201.

E. F. Tjong Kim Sang and F. De Meulder. "introduction to the conll2003 shared task: Language-independent named entity recognition". *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL*, 2003. URL https://www.aclweb.org/anthology/W03-0419.

Mike Schuster and Kuldip K Paliwal. "bidirectional recurrent neural networks". *IEEE Transactions on Signal Processing*, 45.11:2673–2681, 1997.

Nitish Srivastava. "dropout: A simple way to prevent neural networks from overfitting". *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Nitish Shirish Keskar Stephen Merity and Richard Socher. "regularizing and optimizing lstm language models". *(Aug. 2017)*. URL http://arxiv.org/abs/1708.02182.

N. Ryder M. Subbiah T. B. Brown, B. Mann. "language models are few-shot learners". *arXiv:2005.14165*, 2020. URL http://arxiv.org/abs/2005.14165.

H. Peng L. Jiang T. Cai, M. Shen and Q. Dai. “improving transformer with sequential context representations for abstractive text summarization”. *Natural Language Processing and Chinese Computing*, 2019.

Wilson L. Taylor. “”cloze procedure”: A new tool for measuring readability”. *Journalism Bulletin*, 1953. URL https://doi.org/10.1177/107769905303000401.

Robert Tibshirani. “regression shrinkage and selection via the lasso”. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58.1:267–288, 1996. URL http://www.jstor.org/stable/2346178.

Andrey Nikolayevich Tikhonov. “on the stability of inverse problems”. *Doklady Akademii Nauk SSSR*, 39.5:195–198, 1943.

K. Musial U. Naseem, I. Razzak and M. Imran. “transformer based deep intelligent contextual embedding for twitter sentiment analysis”. *Future Generation Computer Systems*, 2020. URL https://www.sciencedirect.com/science/article/pii/S0167739X2030306X.

Ashish Vaswani. “attention is all you need”. *Advances in neural information processing systems*, page 5998–6008, 2017. URL https://doi.org/10.48550/arXiv.1706.03762.

T. Wang W. Chu. W. Huang, X. Cheng. “bert-based multi-head selection for joint entity-relation extraction”. *Natural Language Processing and Chinese Computing - 8th CCF International Conference, NLPCC 2019*, 11839:713–723, 2019. URL https://doi.org/10.1007/978-3-030-32236-6_65.

B. W. White and Frank Rosenblatt. “principles of neurodynamics: Perceptrons and the theory of brain mechanisms”. *The American Journal of Psychology*, 1963. URL https://doi.org/10.48550/arXiv.1706.02515.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *2015 IEEE Interna-*

tional Conference on Computer Vision (ICCV), pages 19–27, 2015. URL https://ieeexplore.ieee.org/document/7410368.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Druck-Exemplaren überein.

01.07.2022 ,
Datum und Unterschrift:

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

01.07.2022 ,
Date and Signature: