# Cross-sentence contexts in Named Entity Recognition with BERT

Master of Science Thesis
University of Turku
Department of Computing
TurkuNLP
2021
Jouni Luoma

Supervisors:
Sampo Pyysalo

UNIVERSITY OF TURKU
Department of Computing

Jouni Luoma: Cross-sentence contexts in Named Entity Recognition with BERT

Master of Science Thesis, 72 p.
TurkuNLP
June 2021

---

Named entity recognition (NER) is a task under the broader scope of Natural Language Processing (NLP). The computational task of NER is often cast as a sequence classification task where the goal is to label each word (or token) in the input sequence with a class from a predefined set of classes. The development of deep transfer learning methodologies in recent years has greatly influenced both NLP and NER. There have been improvements in the performance of NER models but at the same time the use of cross-sentence context, the sentences around the sentence of interest, has diminished in NER methods. Many of the current methods use inputs that consist of only one sentence of text at a time. It is nevertheless clear that useful information for NER is often found also elsewhere in text. Recent self-attention models like BERT can both capture long-distance relationships in input and represent inputs consisting of several sentences. This creates opportunities for making use of cross-sentence information in NLP tasks. This thesis presents a systematic study exploring the use of cross-sentence information for NER using BERT models in five languages. The study shows that adding context as additional sentences to BERT input systematically increases NER performance. Adding multiple sentences in input samples also allows the study of predictions for the sentences in different contexts. A straightforward method of Contextual Majority Voting (CMV) is proposed to combine these different predictions. The study demonstrates that using CMV increases NER performance even further. Evaluation of the proposed methods on established datasets, including the Conference on Computational Natural Language Learning CoNLL'02 and CoNLL'03 NER benchmarks, demonstrates that the proposed approach can improve on the state-of-the-art NER results for English, Dutch, and Finnish, achieves the best reported BERT-based results for German, and is on par with other BERT-based approaches for Spanish. The methods implemented for this work are published under open licenses.

Keywords: NLP, NER, BERT, CMV, Cross-sentence contexts, Deep learning

# Contents

# List of Figures

# List of Tables

# List of acronyms

| | |
|---|---|
| **BERT** | Bidirectional Encoder Representations from Transformers |
| **biLSTM** | Bidirectional LSTM |
| **CNN** | Convolutional Neural Network |
| **CoNLL** | Conference on Computational Natural Language Learning |
| **CRF** | Conditional Random Fields |
| **CSV** | Comma Separated Values |
| **LSTM** | Long short-term memory |
| **MLM** | Masked Language Model |
| **NER** | Named Entity Recognition |
| **NLP** | Natural Language Processing |
| **NSP** | Next Sentence Prediction |
| **POS** | Part of Speech |
| **RNN** | Recurrent Neural Network |
| **SGML** | Standard Generalized Markup Language |
| **WWM** | Whole Word Masking |

# 1 Introduction

Deep learning methods have been introduced to multiple domains of machine learning in recent years with great success. These methods have been able to remarkably improve results on various machine learning tasks [1–3]. One difficulty with deep learning methods has been that their efficient use often requires huge amounts of data for training the neural networks. This has been a major hindrance especially with supervised learning approaches as producing labeled data sets is a labour heavy and time consuming task. One of the keys to success in alleviating the situation has been the use of transfer learning. Transfer learning is based on the idea that a model is first generally trained with a large amount of readily available data and then fine-tuned for a purpose with task specific data. The data for initial training does not need to be specific to the task, but it is sufficient e.g. to use data from the same domain. For example, an image classifying Convolutional Neural Network (CNN) may be trained on readily available labeled image data sets to recognize completely different classes than what we are really interested in. The weights of this kind of pre-trained neural network can be then used as a starting point for training a task-specific neural network with task-specific data. The benefit of transfer learning comes from the fact that the amount of task-specific labeled data needed in this approach is only a fraction of what would otherwise be needed to train a corresponding model from scratch.

Named Entity Recognition (NER) is a task under the broader scope of Natural Language Processing (NLP). It is normally cast as a sequence classification task, assigning each item in a sequence (words or tokens in this case) to a class in a predefined set of classes. The goal of NER is to find mentions of named entities in a text and classify them into categories that are predefined for the task. The correctness of the found mentions is normally evaluated not only by the category that a named entity is classified to, but also by the span of text which constitutes the mention. The usual method for evaluation is to compare the output of a trained named entity recognition system against test data that is annotated with named entities of different types with the same annotation principles as the training data. A named entity mention is correctly recognised only if its predicted text span and category both exactly match to the gold standard labels in test data.

NER as computational task often operates on single sentences of input at a time. This thesis is a study of how to introduce cross-sentence information, meaning e.g. other sentences in a passage of text around the sentence of interest, to improve NER results. This is done by using BERT [1] as a basis for an NER system. BERT is a recently introduced language representation model based on the Transformer architecture [4] which is first pre-trained in an unsupervised/self-supervised fashion with unannotated data. This pre-trained model can then be fine-tuned for use in different NLP tasks using annotated task-specific data. This study is continued and expanded from a Master's project course work, which implemented an NER pipeline that was used for example in our study introducing BERT for Finnish [5]. This study introduces new methods to improve NER results for Finnish and other languages.

Chapter 2 takes a closer look on the task of NER. The task and the data used for NER are introduced. The chapter also briefly introduces the reader to some of the different methods used for NER and how the data is processed for performing the task. Chapter 3 describes the BERT model in more detail and explains how the model can be utilized for NER. Chapter 4 goes through the methods developed in this study to incorporate cross-sentence contexts in NER. Chapter 5 describes the experimental setup for evaluating the introduced methods. The implementation and the data are discussed. Chapter 6 takes a look at the results of the experiments and compares the results to those found in the scientific literature. The results and comparisons reflect the situation in May 2020, when the experiments were performed. The main results published in [6] are elaborated and some observations that were made throughout the study are presented in this chapter. Chapter 7 concludes this thesis with discussion on the relevance of the results, limits on the scope of this thesis, future ideas and possible topics for continuation of the research.

# 2 Named Entity Recognition

In this chapter the NER task is described with details on the data and methods used for performing the task. The chapter starts by introducing the task and data, followed by a short overview of the history and the methods used for NER as well as the current state of the art excepting for BERT-based approaches which are discussed in the next chapter.

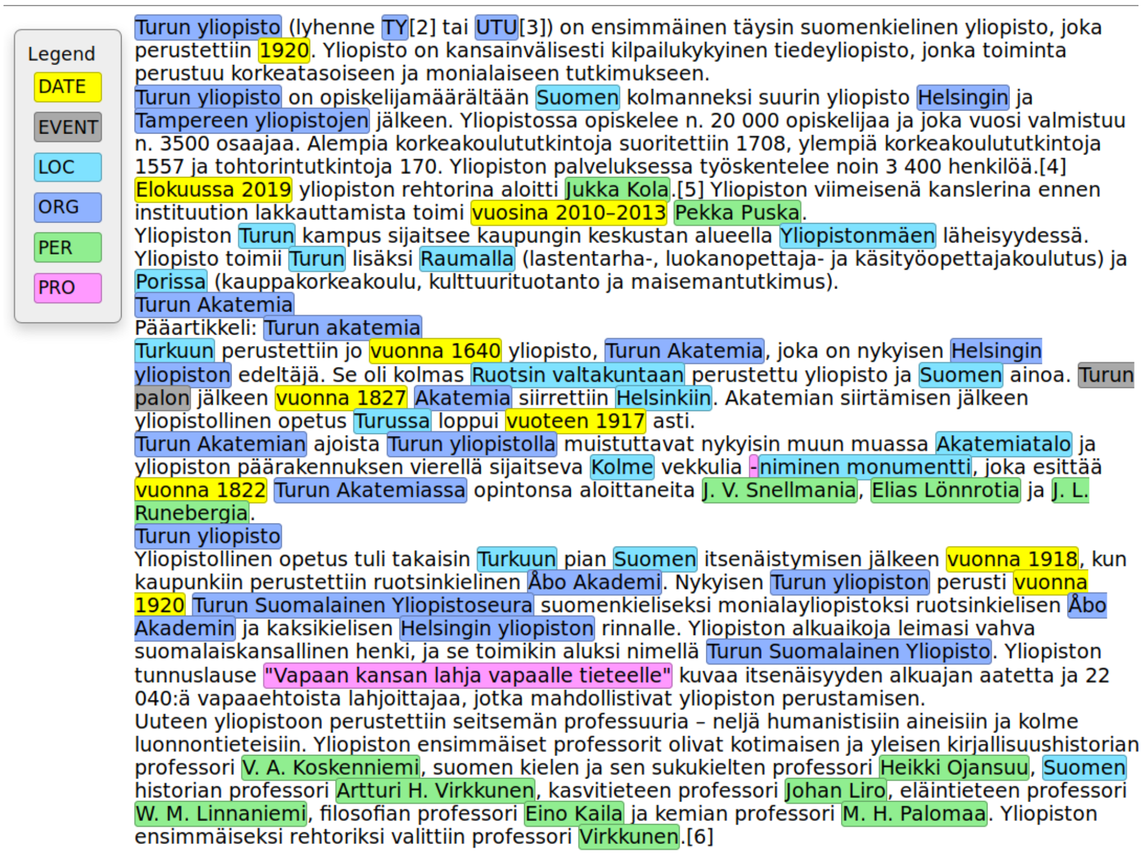
## 2.1 Named Entity Recognition task

NER is one of the key tasks in NLP and it is often used as a first step for downstream NLP tasks such as Information Extraction from documents, Information Retrieval, Question answering, automatic text summarization, and Co-reference resolution, among others [7, 8]. In historical perspective, the methods that now fall under the scope of NER were earlier considered to be a part of Information Extraction. One early mention of NER as a separate evaluation task was introduced in the sixth Message Understanding Conference (MUC-6) [9]. The reason for this separate evaluation task was a desire to develop technologies that are of practical use, domain independent and could be automated in a near term future. NER was seen as such a task [9].

The NER task consists of finding and classifying mentions of entities with proper names (proper noun or noun phrases that refer to individual person, organization, location etc.) or other identifying sequences such as abbreviations in passages of text. One named entity mention may consist of one or several consecutive words in text. The NER task is often expanded also to consider entities in categories such as dates and money, which in a strict sense are not named entities, but can be recognized with similar methods. The categories for classification are usually specific to the task and the data at hand.

NER is often cast as a sequence classification task. The input to a system is given as a sequence of words, and the desired output is a sequence of labels identifying the named entities found from the input text. In sequence classification approaches to NER, the goal is to output a sequence of labels, matching the length of the input, with an entity class assigned to each word (or token) found in input text. Some of the usual categories are persons, locations or organizations in news texts, or perhaps proteins, chemicals, species and diseases in medical texts, or stock market ticker codes in economic news.

A named entity mention may consist of different lengths of text and the target in NER is to simultaneously find the correct category of each named entity mention as well as to find the boundaries of each mention in text. Finding the boundaries of a named entity mention means the task of finding which words (or tokens) belong to that particular mention and which belong to other named entity mentions or are outside of any such mentions. Finding the correct category for an entity mention simply means that we assign the entity in question a class from the set of predefined categories according to the guidelines given for the task. If the system for performing the NER task is machine learning-based, the categories for classification are normally inferred from the annotated data that are used for training the machine learning model. An example of NER system predictions with entity types and spans on a Finnish language passage of text is shown in Figure 2.1.



Turun yliopisto (lyhenne TY[2] tai UTU[3]) on ensimmäinen täysin suomenkielinen yliopisto, joka perustettiin 1920. Yliopisto on kansainvälisesti kilpailukykyinen tiedeyliopisto, jonka toiminta perustuu korkeatasoiseen ja monialaiseen tutkimukseen.
Turun yliopisto on opiskelijamäärältään Suomen kolmanneksi suurin yliopisto Helsingin ja Tampereen yliopistojen jälkeen. Yliopistossa opiskelee n. 20 000 opiskelijaa ja joka vuosi valmistuu n. 3500 osaajaa. Alempia korkeakoulututkintoja suoritettiin 1708, ylempiä korkeakoulututkintoja 1557 ja tohtorintutkintoja 170. Yliopiston palveluksessa työskentelee noin 3 400 henkilöä.[4]
Elokuussa 2019 yliopiston rehtorina aloitti Jukka Kola.[5] Yliopiston viimeisenä kanslerina ennen instituution lakkauttamista toimi vuosina 2010–2013 Pekka Puska.
Yliopiston Turun kampus sijaitsee kaupungin keskustan alueella Yliopistonmäen läheisyydessä.
Yliopisto toimii Turun lisäksi Raumalla (lastentarha-, luokanopettaja- ja käsityöopettajakoulutus) ja Porissa (kauppakorkeakoulu, kulttuurituotanto ja maisemantutkimus).
Turun Akatemia
Pääartikkeli: Turun akatemia
Turkuun perustettiin jo vuonna 1640 yliopisto, Turun Akatemia, joka on nykyisen Helsingin yliopiston edeltäjä. Se oli kolmas Ruotsin valtakuntaan perustettu yliopisto ja Suomen ainoa. Turun palon jälkeen vuonna 1827 Akatemia siirrettiin Helsinkiin. Akatemian siirtämisen jälkeen yliopistollinen opetus Turussa loppui vuoteen 1917 asti.
Turun Akatemian ajoista Turun yliopistolla muistuttavat nykyisin muun muassa Akatemiatalo ja yliopiston päärakennuksen vierellä sijaitseva Kolme vekkulia -niminen monumentti, joka esittää vuonna 1822 Turun Akatemiassa opintonsa aloittaneita J. V. Snellmania, Elias Lönnrotia ja J. L. Runebergia.
Turun yliopisto
Yliopistollinen opetus tuli takaisin Turkuun pian Suomen itsenäistymisen jälkeen vuonna 1918, kun kaupunkiin perustettiin ruotsinkielinen Åbo Akademi. Nykyisen Turun yliopiston perusti vuonna 1920 Turun Suomalainen Yliopistoseura suomenkieliseksi monialayliopistoksi ruotsinkielisen Åbo Akademin ja kaksikielisen Helsingin yliopiston rinnalle. Yliopiston alkuaikoja leimasi vahva suomalaiskansallinen henki, ja se toimikin aluksi nimellä Turun Suomalainen Yliopisto. Yliopiston tunnuslause "Vapaan kansan lahja vapaalle tieteelle" kuvaa itsenäisyyden alkuajan aatetta ja 22 040:ä vapaaehtoista lahjoittajaa, jotka mahdollistivat yliopiston perustamisen.
Uuteen yliopistoon perustettiin seitsemän professuuria – neljä humanistisiin aineisiin ja kolme luonnontieteisiin. Yliopiston ensimmäiset professorit olivat kotimaisen ja yleisen kirjallisuushistorian professori V. A. Koskenniemi, suomen kielen ja sen sukukielten professori Heikki Ojansuu, Suomen historian professori Artturi H. Virkkunen, kasvitieteen professori Johan Liro, eläintieteen professori W. M. Linnaniemi, filosofian professori Eino Kaila ja kemian professori M. H. Palomaa. Yliopiston ensimmäiseksi rehtoriksi valittiin professori Virkkunen.[6]

Figure 2.1: Example of Named Entity Recognition system predictions

### 2.1.1 Named Entity Recognition Corpora

To be able to train and evaluate systems designed for NER, there is a need for annotated collections of text for NER. These annotated text collections are called NER corpora. In general, a corpus in NLP refers to a set of texts which optionally contain also corresponding annotations. Therefore collections of annotated texts for NER are called NER corpora.

A corpus for NER consists of body of text annotated in such a way that the spans and categories of named entity mentions can be distinguished from passages of text belonging to other named entity mentions or not belonging to any mentions. There are different ways to encode this information as annotations. For example, in the MUC-6 conference mentioned earlier, the named entity mentions were annotated with Standard Generalized Markup Language (SGML) tags inside running text [10]. Another way of annotating the data is to use IOB (Inside-Outside-Begin) encoding (or some variants of it) for labels [11] to mark spans and categories of entity mentions. More about the named entity annotation methods is explained in Section 2.1.2.

An NER corpus may also include some metadata or have some other granularity in addition to sentences and words. For example, sentences may be grouped under an original document from which they were extracted, or some metadata regarding the type or domain of the documents are preserved in the corpus. This kind of additional information can also be utilized by NER systems.

Some NER corpora are widely used in the scientific literature as a reference. An example of widely used corpora are CoNLL (Conference on Computational Natural Language Learning) shared task corpora from the years 2002 [12] and 2003 [13]. These data sets are often used as benchmarks for evaluating the performance of NER algorithms. The availability of these resources have also influenced the languages for which NER systems have been developed: for example, the research for English, German, Spanish and Dutch have been more active due in part to the resources published in the CoNLL tasks.

Today, there exists multiple resources for different languages and purposes available publicly (as well as proprietary resources) and more are published frequently for example in scientific conferences.

### 2.1.2 Named Entity encoding methods

NER corpora are available and distributed in different formats. The main categorization of text annotation methods may perhaps be made between inline annotation and stand-off annotation. Inline annotation contains the annotation within the text, and the data and the annotation reside in the same location (e.g. file). Examples of inline annotation are methods such as IOB/IOB2 [11, 14] and variants of them as well as the use of SGML in MUC-6 [10]. Stand-off markup or Stand-off annotations refer to ways where the text to be annotated and the annotation reside separately (e.g. separate files). One example method for stand-off annotation is to store annotations in a separate file with information on the location (e.g. character offset), span and category of each mention in the text file.

NER data used in this study are plain text files constructed in the following way (illustrated in Table 2.1). The input file contains tab-separated lines of text. Each line in the file contains information on one word in the text. A line consists of tab-separated values giving the string representation of the word and the class of the word encoded with an IOB2 tag, as explained below. Blank lines in the files separate sentences. This encoding is loosely similar to the later introduced CoNLL-U format[1] where all other fields beside the word form and label are removed. NER corpora may be distributed in different formats like the CoNLL-U but for this study the files are pre-processed to conform to the format described here.

IOB2 encoding [14] is a variant of IOB encoding that was first introduced in text chunking research. The text chunking task is shortly summarized by Tjong Kim Sang and Buchholz in CoNLL'00 shared task introduction [15]: *"Text chunking consists of dividing a text into phrases in such a way that syntactically related words become member of the same phrase. These phrases are non-overlapping which means that one word can only be a member of one chunk."* The aim of the original IOB encoding was to add an additional "chunk tag" to part of speech (POS) tags in such a way that chunk structure can be derived from the tags. A chunk tag set of `[I]`,`[O]`,`[B]` was used to mark noun phrases. The tag `[I]` was used for words inside a noun phrase, the tag `[O]` marked words outside of phrases, and the tag `[B]` was used to mark the leftmost word of a chunk immediately following another chunk to distinguish that a new chunk is starting. This encoding scheme was later adopted

[1]`https://universaldependencies.org/format.html`

in NER so that the category of a named entity was appended to a chunk tags. For example, a mention of category Person `[PER]` would be marked as `[I-PER]` and Organisation `[ORG]` as `[I-ORG]`. This encoding scheme would create 2N + 1 different tags for N categories of mentions, as there are `[B-Category]` and `[I-Category]` tags for each category and the `[O]` tag is used to mark words not belonging to any mention. This kind of encoding is sufficient to separate the different non-overlapping mentions in text from each other.

Variants of the IOB scheme have later been introduced to alleviate some of the problems encountered with the original IOB tagging. In the original tagging scheme the `[B-Category]` tags are quite rare, and encountering them requires multiple named entities of a same category to occur next to each other in the text. The imbalance in the number of different tags creates problems with statistical and machine learning approaches to the task. This study uses IOB2 encoding, which assigns `[B-Category]` tag for the beginning of every named entity mention. An example of a sentence in IOB2 encoding is shown in Table 2.1. Some variants of IOB tagging also employ separate tags for the last and the middle parts of multi-token named entity mentions and for mentions of single / unit length. These approaches have been referred in literature for example with names BILOU (Begin-In-Last-Out-Unit), IOBES (In-Out-Begin-End-Single), or BMEWO (Begin-Middle-End-Single-Out).

Some NER corpora also contain information on nested named entities where a part of a long multi-word named entity mention may contain also a mention of a shorter named entity. For example a mention of organisation like "University of Turku" may contain also a nested named entity mention of location "Turku". The methods introduced in this thesis are not using nested named entity mentions.

| | |
|---|---|
| Erik | B-PER |
| Justander | I-PER |
| oli | O |
| Turun | B-ORG |
| akatemian | I-ORG |
| professori | O |
| . | O |

Table 2.1: Example of IOB2 named entity annotation. Translation: Erik Justander was a professor for the Academy of Turku

### 2.1.3 Named Entity Recognition Evaluation

The performance of NER systems is often assessed in terms of mention-level exact match Precision, Recall and F-score. This type of evaluation has been applied in NER tasks for example in CoNLL'02, and CoNLL'03 conferences [12, 13].

Mention-level metrics are calculated separately for each of the named entity categories. The predicted outputs of an NER system for each category are compared to the annotated test data. A True Positive (TP) classification happens when both the category and the span of the predicted named entity mention are identical to an annotation in test data. A False Positive (FP) classification happens when the system predicts a named entity mention, which does not match any annotation in the test data (class, span or both do not align). A False negative (FN) happens when the system does not predict an entity mention of the correct type in a position where one exists in the annotated test data. The evaluation scores are then expressed as Precision (2.1), Recall (2.2) and $F_1$-score (2.3) on each type of the Named Entity categories in text. The total performance of the system is usually expressed as micro average calculated by totalling the TP, FP, and FN counts over all categories and taking overall Precision, Recall, and F$_1$-score.

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{2.3}$$

The performance measures applied in this study are calculated as defined above. The calculations are done with a version of the conlleval script, which has been used in CoNLL shared task evaluations. The connleval script was originally implemented in the Perl programming language but for this study a Python version[2] is used so that evaluation may be easily done inside a Python program.

## 2.2 Named Entity Recognition approaches

NER approaches have evolved through different methodological phases during the last decades. The approaches to NER can be roughly split to four different categories: rule- and knowledge-based approaches, unsupervised approaches, feature engineering and supervised learning approaches, and feature learning and deep learning approaches. The last one also includes the latest deep learning and transfer learning based approaches [7, 8].

### 2.2.1 Rule- and knowledge-based approaches

According to Nadeau and Sekine [16] one of the earliest research papers regarding named entities was by Rau [17] in 1991. NER, as defined in Section 2.1, was intro-

[2]`https://github.com/spyysalo/conlleval.py`

duced as a separate task in MUC-6 Conference in 1995 [9]. Before that, the methods for finding e.g. names in text were considered to be part of information extraction and NER was not defined as a standalone task.

Early NER methods applying rule- and knowledge-based approaches, often relied on gazetteers/lexicons (lists of names, locations, etc.) and on handcrafted approaches as Borthwick [18] describes the systems that are *"built by hand and rely heavily on the intuition of their human designers."* These systems for example matched words and phrases, or parts of them, found from text to lexicons and with different rules tried to determine if a sequence in text is a named entity mention and of which type. Some descriptions of these early information extraction and NER systems can be found e.g. in the Message Understanding Conference proceedings from the 1990s [19–23].

The methods used for finding entities often made several passes through the text while trying to increase the performance of the system on each pass. On the first passes, the methods presumed to have high precision, meaning that mentions that had a good probability to be correctly classified were tagged. This information was used in the next passes through the text, trying to find new mentions with higher confidence. One example of such a system is described by Mikheev *et al.* in [24] and the characteristics of this kind of systems were summed up by Borthwick [18]:

*"In sum, if one is smart enough and works hard enough, it is possible to build a strong named entity system using conventional handcoded techniques. However, these systems will still have a number of drawbacks."*

1. *They will be expensive, since they will rely on the expertise of trained computational linguists*
2. *They will have to be manually adapted to new domains*
3. *Their rules and lexicons must be completely rewritten when they are ported to new languages*
4. *Performance will be highly sensitive to the computational linguist's skill in writing the named entity patterns and to the amount of labor devoted to the task*

### 2.2.2 Unsupervised approaches

Unsupervised approaches to NER are not always easy to categorize. Some of the approaches presented here are classified under semi-supervised approaches in a survey by Nadeau and Sekine [16] and on the other hand are under unsupervised approaches in more recent surveys by Yadav and Bethard, and by Li *et al.* [7, 8]. The approaches falling under this scope vary, and some were used to automatically detect and disambiguate named entity mentions [25, 26] while some were aimed more at extracting information and building lists (gazetteers) of named entities e.g. from the web [27]. What is common to many of these approaches is that the they often use some "seed" information and try to create new rules by iterating through a large corpus. Therefore, these unsupervised approaches tend to rely on similar classification methods as the rule- and knowledge-based approaches. The difference is that here the rules for classification are automatically extracted from documents. One example of an unsupervised approach is presented by Collins and Singer [25]. Their system keeps track of `(Spelling, Context)` patterns, where `Spelling` is the written form of a named entity and `Context` refers to the surrounding text of that entity. A small set of seed rules are given to the system, and the system performs iterations through a large corpus. It first labels the corpus given the `Spelling` rules and induces `Context` rule candidates from the found matches, and keeps a predefined number of the most promising rules. Next the corpus is annotated again, now using the `Context` rules to find new candidates for `Spelling` rules. The most promising ones are again kept and added to the list of `Spelling` rules. The iteration continues again from `Spelling` rules, and this is continued until a predefined number of rule patterns are reached. Some approaches such as those by Nadeau *et al.* [26] and Etzioni *et al.* [27] also utilize web search engines in their approaches to validate candidates for new rules or entities to be inserted in knowledge base.

A lot of the research that could be categorized as unsupervised or semi-supervised learning has used these methodologies in connection with supervised learning approaches. The purpose of using unlabeled data has been to extract information from a large amount of unlabeled text to improve the results of supervised learning [28, 29]. Also, some of the more recent feature learning approaches such as BERT [1] use unsupervised or self-supervised approaches for pre-training of models. More about the current uses of unsupervised learning is covered in Section 2.2.4.

### 2.2.3 Feature engineering and supervised learning approaches

NER systems evolved from matching strings and patterns to calculating different features of words and surrounding text, and used machine learning approaches and supervised learning to find and classify entity mentions in text. The technologies applied in these supervised learning approaches were varied and they were sometimes used in combination with the earlier rule-based approaches. The developments within supervised learning continued with researchers continuously trying to find better classification methods and also better ways to extract features for classifiers. Some things that have greatly helped the development of the NER systems were the introduction of publicly available datasets for NER as well as the methods for utilizing the output of other NLP tasks and unlabeled data to prepare better features. According to Tjong Kim Sang and De Meulder [13] the competing systems in CoNLL'03 shared task were for example using the following kind of features for classification:

- Lexical features
- Part-of-speech (POS) tags
- Affix information (n-grams)
- Previously predicted Named entities
- Ortographic information and patterns
- Gazetteers, Lexicons (lists of entities)
- Chunk tags
- Case infomation of words

There were 16 competitors in the CoNLL'03 shared task, and according to Tjong Kim Sang and De Meulder [13] the learning methods used in the competition were the following:

- Maximum Entropy Modeling in [30–34]
- Hidden Markov Models in [33–36]
- Robust Risk Minimization in [33, 37]
- Voted Perceptrons in [38]
- Long Short-term Memory (LSTM) [39] used in [40]
- AdaBoost.MH in [41, 42]
- Memory based learning in [43, 44]
- Transformation based learning in [33]
- Support Vector Machines in [35]
- Conditional Random Fields (CRF) [45] in [46]

The list above shows that by the time of CoNLL'03 conference, many of the learning methods for the coming decade were already introduced, even though not all of them were in active use yet.

### 2.2.4 Feature learning and Deep Learning approaches

During the last decade, feature learning and deep learning-based approaches have gained more popularity as they usually require minimal feature engineering and are not as domain specific as earlier methods [7]. One of the largest changes behind the current developments has been on how the textual data is represented to the NLP systems. The earlier machine learning-based approaches often used sparse vector representations (thousands, even hundreds of thousands of dimensions). For example, each word in the text sequence was represented with features describing some properties of that word, its surroundings, statistical presence in some text corpus, etc. In feature learning approaches the main idea is, instead of having thousands of engineered features, to learn dense representation vectors (often under 1000 dimensions) of words or tokens automatically. This learning of representations is often achieved by using deep neural networks and large amounts of unannotated data that is readily available. The learned representations are then used as features with supervised learning algorithms as-is or by fine-tuning the representations simultaneously with learning NLP task-specific neural network weights. These learned representations are often called embeddings, which is a common name for different representations that map textual information to vectors of real numbers.

Recent survey studies have approached the classification of the modern neural architectures for NER in differing ways. Yadav and Bethard [7] base their classification of feature learning approaches broadly on how the text is represented to the classifier and include categories for Word level architectures, Character level architectures, Character + Word level architectures, and Character + Word + Affix architectures. In word level architectures the text is given to a model as sequence of individual words, represented by word embeddings. In character level architectures, the text is seen as sequence of characters. Character + Word level architectures combine the word embeddings with character level information. Character + Word + Affix architectures also include learned affix features as part of the combined embeddings. Li *et al.* [8], on the other hand, propose a new taxonomy for Deep Learning approaches to NER. Their approach divides methods based on three different aspects of a NER system: Distributed representations (embeddings) used as inputs, context encoder architectures, and tag decoder architectures. Distributed representations in this taxonomy are similar to the ones proposed by Yadav and Bethard: Word-level representations, Character-level representations, and Hybrid representations. Context encoder architecture refers to the ways to include context information of a word in text to the tagging process. These include among others CNN, Recurrent Neural Networks (RNN), and Transformer-based architectures. Tag decoder architectures refer to classifiers that produce a sequence of tags from the context-sensitive representations. The survey mentions for example Multi-layer perceptron + softmax, CRF and RNN as possible tag decoding architectures.

Word-level representations deal with whole words of input at a time. Some of the methods and tools for creating word embeddings include Word2Vec [47, 48], GloVe [49], fasttext [50], and SENNA [51]. One example of feature learning in NER scope (and other NLP tasks as well) was presented by Collobert *et al.* [51] where an embedding was learned for every word in a fixed size vocabulary. A special "RARE" token was used to represent every word in input data that was not present in the vocabulary. The embeddings were trained from unannotated English Wikipedia data as well as Reuters RCV1 corpus [52] for some models. Their approach did not outperform the state of the art in NER at that time, but was able to get close without using extensive feature engineering. Their experiments demonstrated that unsupervised training using a large amount of unannotated data was able to produce such word embeddings (vectors) that words with similar syntactic and semantic properties often reside near each other in the vector space. They were also able to separate the lengthy training process of language models with large unannotated data from faster supervised training with smaller data set for a task such as NER. This is the same idea of the transfer learning approaches that is used throughout the field nowadays. The embeddings calculated this way were also known as SENNA embeddings, named after the software the authors released in addition to their research. One shortcoming of word-level representations is related to the size of vocabulary, which easily grows due to word inflection as every inflected word is considered as its own token in the vocabulary. Other problems with word level representations include polysemy (the same word form may have different meanings) and their inability to handle out-of-vocabulary words well.

Character level representations work upwards from the level of single characters to get representations for text. The text is seen as a sequence of characters instead of predefined words, and word (or string) representations are constructed based on the character sequences. These representations are normally learned using end-to-end neural models, which are often CNN- or RNN-based. For RNN models, LSTM [39] and Gated Recurrent Unit (GRU) [53] are common choices for neural network basic units. Character-level embeddings are able to use sub-word-level information such as prefixes and suffixes as well as better handle out of vocabulary words [8]. Character-level representations are sometimes combined with word embeddings e.g. by concatenating to get better predictions. There also exist approaches where character representations are not mapped to word representations first: for example, character level representations try to directly tag characters and these predictions can be combined into word level predictions. Hybrid representations try to incorporate additional information besides the plain text into the final representations of words, for example by adding some engineered features as part of embeddings.

The current state-of-the-art results in NER and NLP in general have been achieved using contextual embeddings, which are categorized under Hybrid representations by Li *et al.* [8]. Contextual embeddings are a way to encode the context of a word into the embedding so that a word occurring in different contexts gets different embeddings. Contextual embeddings are a way to overcome the problem with same word having multiple meanings (polysemy). One example of this is the name Washington, which may be a name of Person, City, or State and only the context tells which of these is correct. Multiple contextual embedding approaches have been recently proposed [1, 54–56]. Some of the contextual embeddings approaches also use sub-word level representations that try to combine the benefits of both word- and character-based models. One of these sub-word methods is discussed in Section 3.1.

# 3 BERT

BERT: Bidirectional Encoder Representations from Transformers [1] was introduced in late 2018 with great impact, as it was able to improve the performance in multiple NLP benchmark tasks by a considerable amount.

BERT is a type of language representation model that benefits from training with vast amounts of unlabeled data and can later be adapted to specific tasks with a reasonable amount of labeled data in a transfer learning approach. BERT is based on the Transformer architecture [4], but it completely omits the decoder part of the original Transformer architecture and also introduces a different training objective function. These aspects of the design, combined with the advantages of the Transformer architecture such as the omission of recurrent connections, relying only on the attention mechanism, and better parallel computation capabilities made BERT-based models computationally efficient and highly accurate at various NLP tasks. The developers of BERT released the source code and pre-trained models for the English language as well as a multilingual model trained for 104 languages to public use.[1] However, the developers announced that they do not intend to release additional language-specific models, and this has led to development and public release of multiple language-specific models by different parties around the world, such as FinBERT, BERTje, and BETO [5, 57, 58].

[1]https://github.com/google-research/bert

A detailed description of the Transformer architecture is out of the scope of this thesis, but for clarity it should be mentioned that it consists of stacked Transformer encoder blocks and stacked Transformer decoder blocks. The size and architecture of the BERT neural model can be characterized with the following parameters.

- Number of Layers $L$
- Hidden Size $H$
- Number of self-attention heads $A$

The number of layers refers to the number of stacked Transformer encoder blocks used in the network architecture. The hidden size refers to the dimension of the vectors that are input and output of each of the encoder blocks. This is also the size of the input embedding vectors, and the size of the vectors used in residual connections [59] and layer normalization [60] output inside each of the encoder blocks. The number of self-attention heads refer to the number of parallel self-attention operations done on the same input in each of the encoder blocks. The self-attention in each of the encoder blocks in BERT model is calculated on reduced size vectors with dimension of $H$ / $A$. The parameters used in linear transformations between the hidden size and the reduced size are learned on each of the encoder blocks during the model pre-training and the fine-tuning. Using multiple self-attention heads improves the performance of the model and enables more complex representations. The original pre-trained BERT models published were of two different sizes: BERT base (L=12, H=768, A=12, with 110M parameters in total) and BERT Large (L=24, H=1024, A=16, with 340M parameters in total).

In addition to the size of the BERT model, one parameter that is often mentioned with the BERT implementations is the maximum sequence length of the input. The maximum sequence length defines the longest possible sequence of text (measured in WordPieces, see Section 3.1) that will fit in one input sample. Attention mechanism calculations are quadratic in computational complexity in proportion to the sequence length, and the maximum sequence length provided by the pre-trained models is often capped to 512 WordPieces. One reason for this limitation is that BERT uses positional embeddings (see Section 3.2) which are trained during the pre-training, and an increase in maximum sequence length would also increase the time required for pre-training. Normal sentences of text are rarely longer than some tens of WordPieces, but for this research the maximum sequence length limits the number of sentences that fit into one input sample at a time.

It is mentioned by Devlin *et al.* [1] that there are two different strategies for utilizing the pre-trained contextual representations of BERT for language understanding tasks, namely fine-tuning and feature-based approaches. In feature-based approaches the pre-trained contextual word embeddings (i.e. the output of a BERT model that is not fine-tuned to a task) are used as features (or extra features) in a separate machine learning architecture, e.g. bidirectional LSTM (biLSTM) network, to benefit from the pre-training. In fine-tuning approaches, the whole network producing the contextual embeddings is fine-tuned to a task with extra layer(s) added on top of the pre-trained model. Also, the objective function of training is normally changed to a task-specific one.

## 3.1 WordPiece Tokenization

BERT uses WordPiece tokenization, categorized as hybrid representation by Li *et al.* [8], to represent text to the model. WordPiece tokenization was first introduced by Schuster and Nakajima [61] as WordPieceModel and later adopted by Wu *et al.* [62], to process text for neural machine translation. According to Wu *et al.* WordPiece tokenization aims to improve the handling of rare words by dividing words into a limited set of common sub-word units. WordPiece tokenization allows the neural model to “understand” also infrequent words like affixed, inflicted and compounded words, names, etc. There is much less need for special treatment for words not available in the vocabulary, as these words can in most cases be constructed from their constituent WordPieces. Naturally, there may still exist unknown WordPieces e.g. with foreign words and names. This is generally rare and an acceptable tradeoff, as restricting vocabulary to a manageable size helps the accuracy of the system in other ways. If a WordPiece is never seen in the pre-training data, its vector would be random as there is no signal to learn the embedding. Using WordPieces for encoding the data is a way to seek for a balance between the flexibility of processing input on the character level and efficiency of processing the input one word at a time. The idea behind building the WordPiece vocabulary is quite similar to Byte Pair Encoding (BPE) algorithm presented by Philip Gage in 1994 [63]. The vocabulary building is determined by an iterative process over the input data. The algorithm for building a WordPiece vocabulary is presented below and in more detail in [61].

1. Initialize the WordPiece vocabulary first with the characters needed for the data we want to present

2. Build a language model from training data using the current vocabulary

3. Add a new WordPiece to vocabulary by combining two existing WordPieces so that the selected new WordPiece increases the likelihood on training data the most

4. Repeat from 2. until a predefined number on WordPieces are created or likelihood increase drops below a selected threshold

In other words the goal of the algorithm is to find a set of WordPieces that produces the minimal number of WordPieces when segmenting the training data according to the model [62]. The size of the vocabulary therefore varies depending on the training corpus and chosen likelihood drop (or predefined number of WordPieces), but e.g. for the English language model released by the BERT team, the size of the vocabulary is around 30 000 tokens.

WordPiece tokens are used to represent text in a way that allows the reconstruction of the original words from the WordPiece representation. The exception to this with BERT are the whitespace characters, which are not tokenized. The WordPieces that are starting words (or representing whole words) are plain character sequences. If an encoded word is constructed from multiple WordPieces, every additional WordPiece after the first has a prefix of two hashmarks (##) to mark that the WordPiece continues an incomplete word in the sequence. This is illustrated with a WordPiece tokenized sentence in Table 3.1. The most frequent words in the data have just one WordPiece token representing them, while uncommon ones may in extreme cases be represented one character at time. This is however very rare in practice.

The code for generating the WordPiece vocabulary for the original BERT paper had dependencies to Google internal libraries and is not released as open source.[2] Therefore the different groups training their own BERT models have used other, similar approaches to create vocabularies for their BERT models. Sentencepiece [64] is one method providing essentially the same functionality as WordPiece and which also has code available.[3] In this work, all the sub-word vocabulary generating methods are referred to as WordPiece, regardless of which actual implementation was used to generate the vocabulary of the BERT model(s) in question.

```
Th, ##is, is, not, an, ex, ##am, ##ple, sen, ##ten, ##ce
```

Table 3.1: Example of WordPiece tokenized sentence

[2]Mentioned in `https://github.com/google-research/bert`

[3]`https://github.com/google/sentencepiece`

## 3.2 BERT input and output

The main input to the BERT model is written text, converted to WordPieces. The maximum sequence length that the model is capable of processing at a time is defined by the training configuration of the pre-trained BERT model. In addition to WordPiece encoded text, the input may contain some special tokens. Each sequence that is input to the model starts with a `[CLS]` token at the beginning of the input sample. The sequences of text (e.g. sentences) are usually separated from each other with a `[SEP]` token. If the input to the model is shorter than the maximum sequence length, the input sample is padded with `[PAD]` tokens to fill up the sample. There is also an `[UNK]` token available for encoding characters, symbols or sequences that are not available in the BERT vocabulary. The sequence of text is presented to the model as sequence of indices to the vocabulary file, which contain the WordPieces the model was pre-trained with. These indices are also used to fetch the corresponding vector representation of each WordPiece in the embedding layer of the model to use in the calculations. The vector representations of input tokens, the (*input embeddings*), are learned during pre-training and fine-tuning. In addition to the input embedding, the model is also given information on the position of each input token in the sequence (*position embeddings*) and information on what the different parts of the input represents (*segment embeddings*). The segment embeddings are used for example in pre-training (see Section 3.3) for the next sentence prediction task, but are not used in all applications. For example, the NER system developed in this study does not differentiate between segments of the input, and sets all the segment indices to zero.

The output of the BERT model from the last of the stacked Transformer blocks for one input sample is an array of size *Sequence length x Hidden size.* The output has a contextual embedding calculated for each of the input tokens. It is on top of this output that task-specific network structure is added.

## 3.3 Pre-training

The power of BERT lies in that it can be pre-trained on a vast amount of unannotated text and the pre-trained model then used as a basis for training task-specific NLP models. This transfer learning approach significantly reduces the amount of work needed for training a task-specific model either by fine-tuning (see Section 3.4) or a feature-based approach (see Section 3.5) by using a significantly smaller amount of task-specific data than would be needed for training a comparable model from scratch.

The pre-training of BERT is performed with unsupervised or self-supervised learning where no manual labeling is needed, but the targets for network predictions can be inferred from unannotated text. The study introducing BERT [1] proposes a masked language model (MLM) training objective, which has similar idea to Cloze task introduced by Taylor [65]. The MLM training objective is such that a certain percentage, e.g. 15% (the amount used by Devlin *et al.*) of input token positions in each sequence are masked at random, mostly by replacing input tokens with special `[MASK]` tokens, and the objective of training is to predict the tokens that are masked from the surrounding tokens that are visible to the model. This allows the model to take advantage of the bidirectional information available in context. One issue that Devlin *et al.* point out is that MLM training objective introduces `[MASK]` tokens in pre-training, and that token does not occur in the task-specific data or fine-tuning process. Instead of masking the tokens in all of the positions selected for masking, they decided to replace the token with `[MASK]` in 80% of the cases, replace the token with random token 10% of the time, and keep the token unchanged 10% of the time. The BERT team introduced Whole Word Masking (WWM) in pre-training approximately half a year after the introduction of BERT. In WWM, instead of masking individual WordPieces, the mask is applied to all of the WordPieces of a multi-token word. The team said that the original MLM task was too "easy" as a training task. Some WordPieces were often trivial to predict from the visible WordPieces of the same word in cases where a single word was expressed with multiple WordPieces. This improved the results from the original paper when the algorithm was otherwise kept the same.

The BERT team also introduced a second training objective that is optimized together with the MLM objective. The Next Sentence Prediction (NSP) objective aims to introduce some understanding of the relationships between sentences into the model, as this kind of information is largely not required to perform the masked language modeling task. The NSP objective is trained so that examples for pre-training contain two sentences or passages of text separated by the `[SEP]` token. These sentences, $A$ and $B$ for example, are generated from the original corpus used for training. The generation process is done in such way that 50% of the time sentence $B$ is the continuation to Sentence $A$ in the original data. The other 50% of the time the sentence $B$ is a random sentence from the corpus. In the next sentence prediction task the first output token of the BERT model (corresponding to the `[CLS]` token in the input) is used to predict if the sentence $B$ is the continuation of the sentence $A$ in the original corpus. This same first token output is also used in sentence-level classification tasks after fine-tuning a model for such a task.

## 3.4 Fine-tuning approach

The fine-tuning approach as presented by Devlin *et al.* works as follows. The network weights learned in pre-training are used to initialize the model to be trained for a certain NLP task. The output structure of the network corresponding to the MLM and NSP tasks that were used in pre-training are replaced by an output structure needed for the NLP task at hand. To get the state-of-the-art results, Devlin *et al.* used only one additional dense output layer on top of the BERT encoder layers as task-specific structure. The input samples are generally of the same size (with varying amount of padding) for implementation reasons, but some new implementations for using a trained model for predictions can also utilize the information of different lengths of the input for speeding up computations. For the NER task a time-distributed dense classification layer is added on top of the BERT encoder blocks so that the output of the NER model are softmax probabilities [66] of each input WordPiece to belong to one of the named entity tags defined in the training data. For the fine-tuning approach it is common that all of the trainable parameters in the network are fine-tuned according to the task-specific data. It is also possible to select which parameters or layers to train with task-specific data.

## 3.5 Feature based approach

In feature-based approaches the BERT model is used as a feature extractor. The input sequence is used to generate contextual embeddings for each of the input WordPieces and these embeddings are then used as an input to another model such as the biLSTM proposed by Huang *et al.* [67]. This approach is usually suitable in situations where a pipeline and architecture for an NER system already exists and one wants to use the context-sensitive embeddings as drop-in replacement to other embeddings.

# 4 Methods

The study of the adding context in NER with BERT was started when doing research for Finnish BERT [5]. In that study, cross-sentence context was added to the sentence of interest in BERT input samples. The sentence of interest is the only sentence in input sample that is used for prediction, and the other sentences in the sample serve as context. This was done by fitting as many as possible subsequent sentences from data to each input sample. The sentence of interest in this case was always the first sentence of the sample. In this approach the context was introduced to the right of the sentence of interest. However, BERT is bidirectional, and a decision was made to test how the predictions of a sentence of interest behave in bidirectional context and in different parts of the window of maximum sequence length.

## 4.1 Background

This section is based on material previously published by Luoma and Pyysalo [6].

As mentioned in Section 2.2, NER approaches have evolved through different methodological phases in recent decades. Two recent surveys classify NER methods roughly to four different categories: rule- and knowledge-based approaches, unsupervised approaches, feature engineering and supervised learning approaches, and feature learning approaches [7, 8]. The use of cross-sentence information in some form has been a normal part of many NER methods in the former categories, but its role has diminished with the current deep learning based approaches. Rule- and knowledge-based approaches such as that of Mikheev *et al.* [24] often matched strings to lexicons and similar domain knowledge sources for finding named entities. In Feature engineering and supervised approaches, manually engineered features were used to incorporate information from the surrounding text, whole documents, data sets, and also from external sources. As the number of different features and classifiers grew during the years, it was normal to include features that contained cross-sentence information in decision making [68, 69]. Dense vector representations of text such as word, character, string and sub-word embeddings first started to appear in NER methods as additional features to be used with classifiers [51]. Step by step, feature engineering has been demoted to a lesser role, as the most recent deep learning approaches learn to create context-sensitive representations of text by pre-training with vast amounts of unlabeled data. These contextual representations are often used directly as features for existing NER architectures or, in transfer learning, fine-tuned with labeled data to match a certain task.

In recent years, the development of NLP in general and NER in particular has been greatly influenced by deep transfer learning methods capable of creating contextual representations of words, to the extent that many of the state-of-the-art NER systems mainly differ from one another on the basis of how these contextual representations are created [1, 54–56]. Using such models, sequence tagging tasks are often approached one sentence at a time, essentially discarding any information available in the broader surrounding context, and there is only little recent study on the use of cross-sentence context – sentences around the sentence of interest – to improve sequence tagging performance. This study presents methods to introduce cross-sentence context to named entity recognition task with focus on the recent BERT deep transfer learning models.

One recent method taking sentence context into account is that of Akbik *et al.* [70], which addresses a weakness of an earlier contextual string embedding method also by Akbik *et al.* [54], specifically the issue of rare word representations occurring in underspecified contexts. In the continuation study [70] the authors make the intuitive assumption that such occurrences happen when a named entity is expected to be known to the reader, i.e. the name is either introduced earlier in text or is of general in-domain knowledge. Their approach is to maintain a memory of contextual representations of each unique string in text and pool together contextual embeddings of a string occurring in text with the contextual embeddings of the same string earlier in text. This pooled contextual embedding is then concatenated with the current contextual embedding to get the final embedding to use in classification.

Another recent approach taking broader context into account for NER was proposed by Luo *et al.* [71], where in addition to token representations, also sentence and document level representations are calculated and used for classification using a CRF model. A sliding window is used by Wu and Dredze [72] so that part of the input is preserved as context when the window is moved forward in text. Baevski *et al.* [56] states that they use longer paragraphs in pre-training their model, but it is not mentioned in the paper if such longer paragraphs are used also in fine-tuning the model or predicting tags for NER. Some other approaches such as that of Liu *et al.* [73] include explicit global information in the form of e.g. gazetteers. Also, some approaches formulate NER as a span finding task instead of sequence labeling [74, 75]. These approaches would likely allow the use of longer sequences, but the incorporation of cross-sentence information is not explicitly proposed by the authors. In the paper introducing BERT, the authors write in the description of their NER evaluation "we include the maximal document context provided by the data." However, no detailed description of how this inclusion was implemented is provided, and some NER implementations using BERT have struggled to reproduce the results of the paper.[1,2] The addition of document context to NER using BERT is discussed also by Virtanen *et al.* [5]. This approach is named as the method *First* in this study and discussed in Section 4.3

Of the related work discussed above, our approach most closely resembles that of Devlin *et al.* By contrast to other studies discussed above, the methods in this study do not introduce extra features or embeddings to represent cross-sentence information or incorporate extra information in addition to that captured by the BERT model. Instead, the BERT architecture is directly utilized and the methods rely only on self-attention and predictions for sentences in different contexts and additionally aggregations thereof.

## 4.2 Single

The method *Single* refers to the most simple and perhaps the most usual implementation of NER with BERT. The method simply uses one sentence of text per sample in training and prediction time. The sample is filled with padding tokens up to the maximum sequence length after the WordPieces corresponding to the actual sentence are placed in the beginning of the sample. This will be used as a baseline against which the other approaches are compared. This method is illustrated in Figure 4.1a.

## 4.3 Sentence in Context

The *Sentence in Context* method was the starting point for this research. The right side cross-sentence context was added to a sentence of interest by Virtanen *et al.* [5]. The input sample built this way also contained bidirectional context for the added

[1]`https://github.com/google-research/bert/issues/581`

[2]`https://github.com/google-research/bert/issues/569`

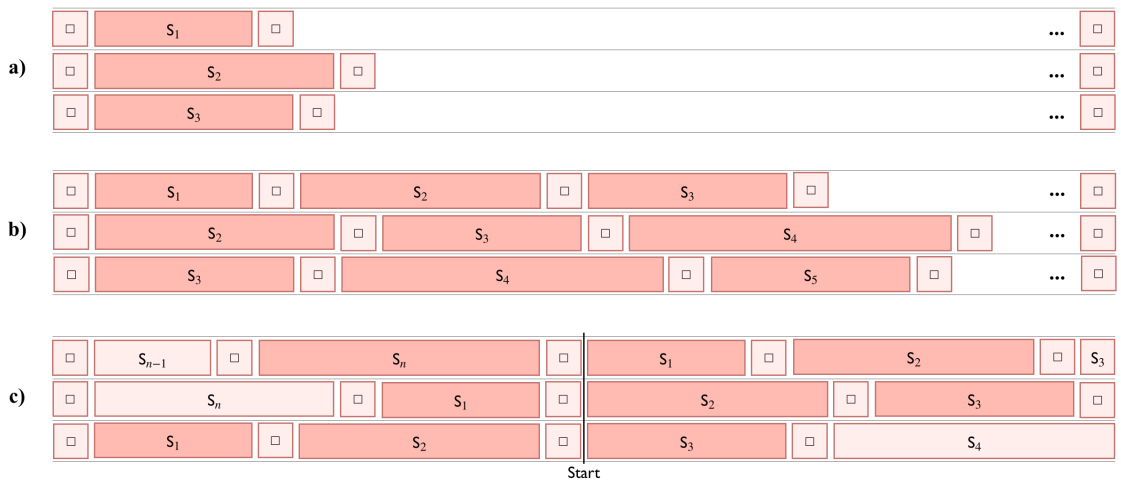


Figure 4.1: Illustration of various input representations for sequence labeling tasks. a) One sentence per example (*Single*), b) including following sentences (*First, CMV*), c) including preceding and following sentences (*Sentence in context*). CMV combines predictions for the same sentence (e.g. $S_3$ in b) in various positions and contexts. The empty square (□) stands for special separator symbols (e.g. `[CLS]`, `[SEP]` and `[PAD]` for BERT); a light background color is used to represent special symbols and incomplete sentences in c)

sentences, but that information was not used. Introducing the right side cross-sentence context to input also means that each sentence may be a part of multiple input samples, and therefore reside in different places inside the maximum sequence length window in different samples. The problem was that in contexts constructed in this way, we were not able to consistently measure the performance for a sentence of interest. The sentences were of different lengths and as more sentences were added as parts of input samples, the beginning of the sample was only place where the sentences would align. Also, the number of sentences fitting to a single sample vary sample by sample. We were not able to pick the N*th* sentence for study as there were no guarantees one will exist in every sample. Or, if it existed, we were not able to control where in the sample it actually resides.

For the reason stated above, a method was implemented to place the sentence of interest to start at a predefined location inside the window of maximum sequence length. The input sample was then filled by adding the previous and next sentences from input data around the sentence of interest. If the predefined starting location of a sentence of interest was such that the whole sentence would not have fit to the window, the sentence was moved backwards in the window so that whole sentence fits. This enabled the measurement of the performance with $F_1$-score in different starting locations inside the window of maximum sequence length. The sentence with a certain starting position in context was only used in prediction time. Fine-tuning of the BERT model was still done with the input samples containing consecutive sentences from data. This decision was done because taking also the starting position as one hyperparameter in training would have lead to an excessive computational cost. This method of arranging input samples is illustrated in Figure 4.1c.

The intuition behind the *Sentence in Context* approach was that samples in the middle of the maximum sequence length window would benefit from the context in both directions. This intuition was later empirically found not to hold in all cases (see Section 6.1)

For the rest of this thesis, adding only the right side context is referred to as the method *First*, which is a short form of *First sentence in context*, and otherwise the *Sentence in Context* is normally referred to with the starting place of the sentence inside the maximum sequence length window.

## 4.4 Contextual Majority Voting

When the cross-sentence context was added to BERT input samples (see Figure 4.1b), it was noted that the predictions for the same sentence in different contexts were not the same. The idea of having multiple classifications of the same input lead to thinking about ensemble methods. Knowing that ensembles of classifiers are commonly used to improve the performance of classification methods, the predictions of the same sentence in difference samples were combined to create an ensemble-like construct. This is not a real ensemble in the usual sense as we have only one model and the number of predictions for each sentence varies due to the length of sentence and the sentences around it. Additionally, some sentences did not fit into one input sample but had to be split into several samples to be able to process the complete input data, resulting in only a single prediction for some parts of sentences.

Two approaches to combining the results from predictions in different contexts were evaluated. The first approach was to assign labels to tokens according to the majority vote of the labels from different contexts. This method is called *Contextual Majority Voting (CMV)* throughout this study. The other approach was to sum the softmax probabilities of the predictions in different contexts and then take the maximum value of the combined probabilities as prediction. This method is called *Contextual Majority Voting on Probabilities (CMV-P)* in this study. Both of these methods are variations of the same underlying idea and the term *Contextual Majority Voting* is used to refer to both of them, unless there is a need to make a distinction between the two approaches.

Another feature of implementation is the construction of documentwise input samples. This was used in cases where the input data had boundaries between different documents in the original data marked e.g. with `-DOCSTART-` tokens. The subsequent sentences were filled into input samples only from the same original document. The sentences after the last sentence in the original document were taken from the beginning of the same original document and the sentences before a first sentence in document were taken from the end of the same original document instead of the previous document.

This research also studies the effect of moving the sentence of interest inside the window of maximum sequence length. The training data was constructed as in Virtanen *et al.* [5] but testing data was constructed so that the sentence of interest is started at given location inside the maximum sequence length and the input sample was filled to both directions with sentences before and after the sentence of interest in test data.

# 5 Experimental setup

This chapter provides details on the experimental setup of this study. The methods introduced in Chapter 4 were tested on five different languages to get a more general evaluation of the effect of adding cross-sentence information in BERT fine-tuning and prediction. The pre-trained BERT models and the annotated NER corpora used in this study are introduced and some details of the implementation and computational environment are presented in the following.

## 5.1 Data

The data used to run the experiments for this study consists of six pre-trained BERT models and six NER corpora in five different languages.

### 5.1.1 Data sets

For each of the five languages considered in this study, pre-trained BERT model(s) is fine-tuned for the NER task. Annotated data as defined in Section 2.1.1 is required for training and evaluating a fine-tuned model, and for that purpose we use an NER corpus for each tested language. For Finnish, the corpus is a combination of two corpora. The NER corpora used in this study are listed in Table 5.1.

The NER corpora for English, German, Dutch and Spanish were introduced as CoNLL shared task data sets in the years 2002 and 2003. These data sets have become established as benchmark standards for evaluating NER methods and that

| Language | Source |
|---|---|
| English | CoNLL'03 Shared task [13] |
| Dutch | CoNLL'02 Shared task [12] |
| German | CoNLL'03 Shared task [13] |
| Spanish | CoNLL'02 Shared task [12] |
| Finnish | FiNER news corpus[76] |
| Finnish | Turku NER corpus [77] |

Table 5.1: NER corpora used in this study

provides a good reason to select them for this study. Each of these four data sets consist of three different data files. Training data for training the algorithm, development data for hyperparameter optimization and tuning the algorithm, and test data for evaluating the final performance of the trained models. The Named Entity annotations in all of these four data sets have four different categories annotated. The categories in these datasets are presented in Table 5.2.

| Category | Entity type ID |
|---|---|
| Persons | [PER] |
| Organizations | [ORG] |
| Locations | [LOC] |
| Miscellaneous | [MISC] |

Table 5.2: Entity categories in CoNLL data sets

This study is a continuation from a work done for the Finnish BERT [5], and Finnish was naturally selected as one language for this study as well. For Finnish, a combination of two NER corpora is used: a Finnish news corpus for NER (FiNER news) [76][1], which was the first publicly available NER corpus for Finnish, and the

[1]`https://github.com/mpsilfve/finer-data`

| Category | Entity ID |
|---|---|
| Persons | [PER] |
| Organizations | [ORG] |
| Locations | [LOC] |
| Products | [PROD] |
| Events | [EVENT] |
| Date | [DATE] |

Table 5.3: Entity categories in Finnish data sets

Turku NER corpus [77][2], which was annotated to be compatible with the FiNER news corpus. In this study, the data of these two corpora are combined together. More specifically, the train, development and test data sets from both corpora are joined with simple concatenation to form combined train, combined development and combined test sets for Finnish. These Finnish NER corpora have six named entity categories annotated as listed in Table 5.3. The key statistics of the NER corpora used in this study are listed in Table 5.4. For Finnish language the numbers are for the combined corpus.

| **Tokens** | English | German | Spanish | Dutch | Finnish |
|---|---|---|---|---|---|
| Train | 203,621 | 206,931 | 264,715 | 202,644 | 342,924 |
| Development | 51,362 | 51,444 | 52,923 | 37,687 | 31,872 |
| Test | 46,435 | 51,943 | 51,533 | 68,875 | 67,425 |

| **Entities** | English | German | Spanish | Dutch | Finnish |
|---|---|---|---|---|---|
| Train | 23,499 | 11,851 | 18,798 | 13,344 | 27,026 |
| Development | 5,942 | 4,833 | 4,352 | 2,616 | 2,286 |
| Test | 5,648 | 3,673 | 3,559 | 3,941 | 5,129 |

Table 5.4: Key statistics of the NER data sets

[2]https://github.com/TurkuNLP/turku-ner-corpus

| Language | BERT model |
|---|---|
| English | BERT Large Whole Word Masking, Cased [1][3] |
| Dutch | BERTje base, Cased [57][4] |
| Finnish | FinBERT, Cased [5][5] |
| Spanish | BETO, Cased [58][6] |
| Spanish | BERT Multilingual, Cased[7] |
| German | German BERT, Cased[8] |

Table 5.5: Pre-trained BERT models used in this study

### 5.1.2 Pre-trained BERT models

Pre-trained BERT models are the basis for the fine-tuning approach to training an NER system. In this study, monolingual BERT models for each of the studied languages were preferred, as recent studies have suggested that well-constructed language-specific models outperform multilingual ones [5, 57, 78]. The pre-trained models chosen for this study are listed in Table 5.5. The models were selected for this study mainly because they were, besides Spanish, readily discoverable, and were freely available for the languages that have established benchmark data for NER, as described in Section 5.1.1.

The pre-trained BERT models are often released with different variants to be used with cased and uncased text. The cased variants accept as input both upper and lower case text and the vocabulary is constructed accordingly. The uncased variants accept only lower case text as input and the input text must be converted

[3]`https://github.com/google-research/bert`

[4]`https://github.com/wietsedv/bertje`

[5]`https://github.com/TurkuNLP/FinBERT`

[6]`https://github.com/dccuchile/beto`

[7]`https://github.com/google-research/bert`

[8]`https://deepset.ai/german-bert`

accordingly. All of the models used in this study were cased variants of the pre-trained models. All the models except English BERT are the size of the BERT base model. For English, the BERT Large model trained with whole word masking was chosen for the study to achieve the best possible results. Also, we noted that the BERT base model for English had some issues when using the maximum sequence length of 512 WordPiece tokens. This issue is discussed further in Chapter 6. The German BERT model was published by company deepset.

## 5.2 Implementation

This section describes in more detail the NER pipeline implemented for this study. The implementation is available under an open license.[9] The implementation is written in the Python programming language, and in addition to Python standard libraries it has dependencies on the following software and platforms:

- BERT, `https://github.com/google-research/bert`
- keras-bert, `https://pypi.org/project/keras-bert/`
- conlleval, `https://github.com/spyysalo/conlleval.py`
- Tensorflow, `https://www.tensorflow.org/`
- Keras, `https://keras.io/`
- numpy, `https://numpy.org/`

The pipeline itself is language independent and may be used with any BERT model and NER corpus, given that the requirements for the input file structure are fulfilled. The primary implementation for NER is found in the file `ner.py`, which contains the main program, and `common.py`, which contains the methods needed for data processing. The different processing steps of the pipeline are described in subsections below.

[9]`https://github.com/jouniluoma/bert-ner-cmv`

The computations for this study were made on the CSC Puhti supercomputing cluster and the source code also contains some scripts and and an argument parser implemented for the purpose of modifying the read and write locations as well as functionality of the program for different purposes.

### 5.2.1 Preprocessing data

Preprocessing the data consists of multiple steps starting from reading in the IOB2-encoded data from the input file(s) and ending in transforming the data into a format that is ready to be input to the model for training. First, the NER data is read with the `read_connl()` method, which returns the data as sentences (lists of words) and their corresponding tags (lists of labels). The format of the input data is described in the Section 2.1.2.

#### Tokenization

BERT uses WordPiece tokenization, as discussed in Chapter 3, in order to be able to represent complex words with a limited-size vocabulary. Every BERT model is distributed with its corresponding WordPiece vocabulary that was used when pre-training the model. The vocabulary gives unique ID:s to WordPieces, and also contains IDs for special tokens such as unknown `[UNK]`, separator `[SEP]`, and padding `[PAD]`. The sentences of data (lists of words) are next tokenized with WordPiece tokenizer and the corresponding vocabulary for the model in question. After tokenizing the sentences, it is checked that each tokenized sentence will fit into an input sample of maximum sequence length. If a sentence does not fit, it will be split to multiple "sentences" for further processing. As the result of tokenization, the lists of input sentences are represented as lists of WordPiece tokens, and these WordPiece token lists are each shorter than the maximum sequence length selected for the training.

### Process sentences

In this phase the cross-sentence context is added to sentences and BERT input samples are constructed from individual tokenized sentences. In this study, different methods to process the tokenized sentences are evaluated to study the effect of cross-sentence context on predictions. The methods are described in detail in Chapter 4.

The baseline processing, against which the other methods are compared, is as follows. Every input sentence forms its own input sample and no context is added. The input sample is simply filled with padding tokens after the input sentence. This is a common approach that different BERT implementations use for fine-tuning a BERT model. This baseline processing is enabled with `--nocontext` as a command line input argument when running `ner.py`. This is also called method *Single*.

For testing the methods *First* and *CMV*, the input samples are created so that subsequent sentences from the input data are added to the same input sample. The input data is looped through so that each sentence is starting a BERT sample exactly once, and the next sentences from the original data are added to the same sample as long as the next sentence in the input fits into the sample. At the end of the input data the last samples are constructed so that the input samples are filled with sentences from the beginning of the input data if no new sentences are available at the end of the data. The sentences in each sample are separated with a `[SEP]` token. This is the standard way of constructing input samples in this implementation.

If the input data contains `-DOCSTART-` tokens, also a documentwise construction is possible. In this case, the next sentences are added only from the same original document in input data, and the data is rotated at the end of that original document instead of the end of the whole file. This optional processing is enabled with the `--documentwise` command line input argument.

To evaluate the prediction performance of a model on sentences residing at a predefined location, the input samples are constructed so that a sentence of interest is placed to start at the given location inside the input sample as described in the Section 4.3. If needed, the sentence of interest is moved backwards in the window so that it will fit as the last sentence of the sample. For this reason the tested starting locations are not always exact but rather desired locations for starting each sentence.

The output from `read_conll()` is input to one of the processing functions, which tokenizes the input, combines the sentences according to the selected method, and optionally also splits sentences that are longer than maximum sequence length. The count of WordPieces of each original word are kept for further processing and for reconstructing the data after predictions. The processing function returns a named tuple `Sentences` which contains the original sentences, tokenized sentences with their tags and the lengths of each original word in WordPieces. Also, combined sentences with their tags are part of the `Sentences` tuple.

#### Encoding

Up to this point the data are kept in textual format in list structures. To be able to make calculations, the final step of pre-processing is to convert the textual data to input samples in numerical format. Every WordPiece in the vocabulary has a numerical ID corresponding to the index of the relevant embedding. The textual WordPieces are converted to numerical numpy arrays containing the IDs. Converting textual data to numerical format as well as padding and formatting the data for calculations is done by the `encode()` method. Also the labels for sequence classification are converted to numerical format with the `label_encode()` method.

### 5.2.2 Model construction and training

The BERT reference implementation and the keras-bert library do not directly have functionality for NER purposes. In this study, a new model for NER is constructed by attaching a time-distributed dense layer on top of the pre-trained BERT model. This model is then fine-tuned with data annotated for NER. The Adam optimizer [79] with warm-up, weight decay, and linear learning rate decay is used in training the model. The hyperparameters used for training are given as command line arguments to `ner.py` and the argument parser is used to relay the information to the training process. In this study, the Adam optimizer parameters were fixed to following:

- Linear warmup of 10% of the samples followed by
- Linear learning rate decay on the 90% of samples
- Weight decay with rate 0.01
- Adam optimizer parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e - 6$
- Norm clipping on 1.0

Training uses the Keras framework and the embeddings of the input WordPieces are propagated forward through the model to get softmax probabilities for each input WordPiece belonging to a certain class. Sparse categorical cross-entropy is used as a loss function in calculations, and the Adam optimizer with the parameters given above is used to update the neural network weights. The `[CLS]` token starting an input sample and the `[SEP]` tokens separating the sentences are excluded from the loss calculation.

Also adding a CRF [45] layer on top of the BERT model was briefly tested, but the fine-tuning method with just a dense layer provided similar or better results in preliminary experiments, and a CRF layer was not applied in the final experiments reported below. The experiments with a CRF layer were not exhaustive, and perhaps this option could be studied further later on.

### 5.2.3 Predictions

For evaluating the performance of a fine-tuned model, a separate set of annotated text is pre-processed and passed through the model to get predictions of classes for each WordPiece. The development data is used for hyperparameter search and the test data is used for evaluating the final results. The predictions are presented as softmax probabilities for an input token belonging to each of the classes. The maximum softmax probability is selected as the label for each WordPiece in the input sample (except for *CMV-P*). After this, there are different ways to construct the final predictions from the initial predictions. For the method *Single* (`--nocontext`), there is only one sentence per input sample and the predictions for that sentence are directly taken as the final result. If the starting place of a sentence inside a context is defined, the results are taken from the single sentence starting at that location in each input sample. If *Contextual Majority Voting* is activated, there are two different approaches to get the final predictions. Either the final prediction for a token is a majority vote for predictions of the same token in different samples (*CMV*), or the final prediction is the maximum of summed softmax probabilities of the same token in different samples (*CMV-P*).

The predictions are used as an input to the `write_sentences()` method, which combines the WordPiece representations of original tokens back to a format where each word has exactly one label attached to it. No extra heuristic is built into this stage e.g. to fix broken IOB2 sequences. If a word consists of multiple WordPieces, the label for the first WordPiece is used to represent the label of the whole word. The method writes out an evaluation file that can be later examined with an evaluation script. The evaluation file follows the format described in the Section 2.1.1, with the modification that in addition to a word and its correct label, also the predicted label is written as additional tab-separated field to this file. The method also returns the lines written so that the same data may be evaluated inside the program with methods provided in `conlleval.py`. There is also a “predict” mode for `write_sentences()` which is used when unlabeled data is passed through the system for labeling. This was added for building a demo application for NER. The demo application and related functionality (web service, web pages etc.) were added to the repository after building the main NER pipeline and are not contributions of the author of this study.

### 5.2.4 Evaluation

This study uses entity level Precision, Recall and $F_1$-score as metrics for measuring NER task performance. These are standard evaluation metrics for NER and were used for example in the CoNLL shared tasks that introduced the English, Dutch, German and Spanish data sets used in this study. This is also the evaluation metric that `conlleval.py` script implements. The total evaluation scores are expressed as micro average over the results of each named entity type. The result output of one evaluation done with `conlleval.py` script on CoNLL'03 English language development set is shown in Figure 5.1.

```
processed 51578 tokens with 5942 phrases; found: 6005 phrases; correct: 5767.
accuracy:  99.37%; precision:  96.04%; recall:  97.05%; FB1:  96.54
              LOC: precision:  97.72%; recall:  97.82%; FB1:  97.77  1839
             MISC: precision:  89.79%; recall:  92.52%; FB1:  91.13  950
              ORG: precision:  95.28%; recall:  96.27%; FB1:  95.77  1355
              PER: precision:  98.12%; recall:  99.13%; FB1:  98.62  1861
```

Figure 5.1: Example of Named Entity Recognition evaluation

The test implementation is built so that once a model is trained, multiple different tests can be run against the model. If the `--documentwise` command line option is used, the model is trained with training samples built documentwise and all the different tests are run against this model, including also the ones with input samples build without documentwise context. With the `--nocontext` command line option, the model is trained on single sentences and also tested on single sentences.

The default functionality of the `ner.py` program is to build BERT samples by adding subsequent sentences from input data. No documentwise wrapping of the samples are used in testing in this case.

Once the tests are run, the evaluation results are written into a Comma Separated Values (CSV) file for further analysis and visualization purposes. The file contains information on the pre-trained model, the training data set filename, the test data set file name, the training hyperparameters and the precision, recall and $F_1$-score for each test that was run and an identifier of the method used for prediction. As the tests were run in the Slurm environment of the CSC Puhti supercomputer, also the Slurm-job ID was saved. This same job ID was saved also as part of the log and output files from which the results were calculated.

## 5.3 Experiments

To evaluate the effect of introducing cross-sentence information to NER, the following scenarios were examined. The last scenario of single sentence only was selected as baseline against which the methods with cross-sentence information is compared.

- *CMV*, documentwise built samples
- *CMV*, subsequent sentence context (default)
- *CMV-P*, documentwise built samples
- *CMV-P*, subsequent sentence context
- *First*, documentwise built samples
- *First*, subsequent sentences context
- *Sentence in context*, starting positions 32 .. 512 with 32 WordPiece intervals
- *Single*, no context used

These methods were defined in the Chapter 4.

### 5.3.1 Hyperparameter search

A hyperparameter search was performed for all of the scenarios defined above. The creators of BERT suggested that an exhaustive grid search over a small selection of learning rates, batch sizes and epochs would be enough to reach state of the art results (at the time) using the fine-tuning approach. The hyperparameter range suggested by Devlin *et al.* is presented in Table 5.6:

| Hyperparameter | Suggested values |
|---|---|
| Epochs | 2, 3, 4 |
| Batch Size | 16, 32 |
| Learning rate | 2e-5, 3e-5, 5e-5 |

Table 5.6: Hyperparameter suggestions from study introducing BERT

The hyperparameter range for the grid search in this study was modified from the original, and the used hyperparameters are presented in Table 5.7. This was done mostly due to the fact that the implementation did not initially support multi-GPU processing. The fine-tuning was performed using a single NVIDIA Tesla V100 GPU with 32 GB of GPU memory in the CSC Puhti Supercomputer shared environment. The execution on a single V100 GPU using BERT base sized model failed with an Out-of-Memory error with batch size of 32 and the maximum sequence length set to 512. The implementation successfully ran on batch size of 16 or lower, and therefore the hyperparameter search was limited to those batch sizes.

Initial tests with the implementation were performed with batch sizes 4, 8, and 16. After some tests against the development set, also batch size 2 was added into the hyperparameter search range. This was due to the fact that the best hyperparameter combinations often included a batch size which was found on the lower end of the hyperparameter range. The same result was seen with the number of epochs as the system many times reached comparatively good results after training for a single epoch. An additional factor influencing the decision to take smaller batch sizes into use was that it enabled the experimentation on personal computer. Experiments with the personal computer ran the code on a single NVIDIA RTX2080Ti GPU with a maximum sequence length of 512. This was done only for development purposes, and all of the final results were calculated in the CSC Puhti supercomputer environment. For the reasons stated above, the following hyperparameter range was finally selected to be tested for all of the pre-trained models and languages.

| Hyperparameter | Values |
|---|---|
| Epochs | 1, 2, 3, 4 |
| Batch Size | 2, 4, 8, 16 |
| Learning rate | 2e-5, 3e-5, 5e-5 |

Table 5.7: Hyperparameters used in this study

The multi-GPU support was later implemented as the English language pre-trained model had to be changed from BERT base to BERT Large Whole Word Masking (WWM) model, and a single V100 GPU was only able to process the batch sizes 2 and 4 with a maximum sequence length of 512. Part of the motivation to change to BERT Large WWM was a discovery made during the analysis of the hyperparameter search results on the English data set. The English BERT base cased model appeared to lose prediction performance in certain locations inside the input sequence. More detail about this behaviour is provided in Section 6.1.1.

## 5.4 Result analysis implementation

The results from running the tests were gathered to CSV text files which were later analysed. The contents of these files are defined in Section 5.2.4. The results were analysed using the Jupyter notebook environment (Python) and the Pandas library for processing the data. Evaluation with each hyperparameter combination was repeated 5 times and the mean values are reported in Chapter 6. The result analysis was performed by combining the output result files to a Pandas DataFrame, grouping the results by different fields (e.g. hyperparameter, method, sentence starting location), and calculating measures based on these groupings (e.g. mean and standard deviation of precision, recall and $F_1$-score).

## 5.5 Software and Hardware

The hyperparameter search and the final evaluations were performed on the CSC Puhti supercomputer. More specifically, the `gpu` partition of Puhti was used and the calculations utilized one NVIDIA Tesla V100 GPU on all computations except for the BERT Large model with batch sizes 8 and 16. The BERT Large model with batch size 8 used 2 NVidia Tesla V100 GPUs and the computations with batch size 16 utilized 4 NVidia Tesla V100 GPUs in parallel.

| | |
|---|---|
| Tensorflow | 1.14 |
| Keras | 2.2.4 |
| keras-bert | 0.80.0 |
| python | 3.7.3 |

Table 5.8: Software versions used in results

The software versions used for calculating the results in this thesis are listed in Table 5.8. The versions of software were not fixed during development, as for example CRF layer on top of BERT model had different prerequisites for the used libraries. The source code has been updated after the evaluations and the released version now supports Tensorflow version 2.x and the `tensorflow.keras` version of the Keras library.[10]

[10]`https://github.com/jouniluoma/bert-ner-cmv`

# 6 Results

This chapter describes the results achieved by introducing cross-sentence contexts to NER. The main results have been published in [6], and the results are described here in more detail. First, a look is taken at the results on the development data sets obtained during hyperparameter search. These include testing the performance of the *Sentence in Context* method on different locations inside BERT samples. These results are also compared to the results of the method *CMV* on the development set. Next, the results on the test set are evaluated for the models trained with the hyperparameters that gave the highest development set performance in the search. The results on the test data reach new state of the art level for three languages when compared to NER benchmark results reported in domain literature at the time when the study was performed. Incidental findings that were observed during the study are also discussed. Not all of these findings were thoroughly tested, and they should therefore be considered more as preliminary indications than as verified results. The distinction between fully tested results and preliminary is clearly stated when presenting each.

## 6.1 Results on development sets

The results on development sets are next discussed one language at a time. Some general topics are additionally presented under the headings for relevant individual languages.

### 6.1.1 English

The initial tests for English were performed using the BERT base sized pre-trained model. The initial results for sentences starting in different locations are shown in Figure 6.1, which shows the *Sentence in context* (see Chapter 4) results, expressed as mean of all the results for each sentence starting position during hyperparameter search.

It appears that the NER classification performance with the English BERT base cased model drops noticeably if the sentence of interest resides in the latter part of the 512 WordPiece maximum sequence length window. The performance of the

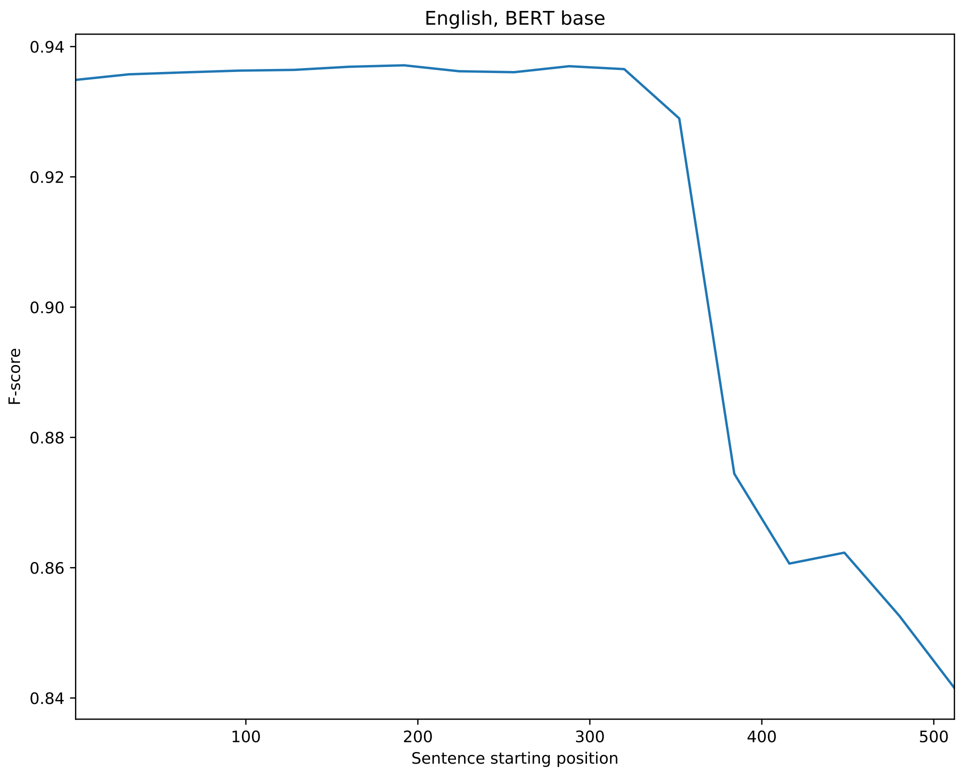


Figure 6.1: English, BERT base, performance on different starting positions. $F_1$-score presented as mean of the results over the whole hyperparameter range on the development data set.

model is quite stable for the sentence start locations up to approximately 300 WordPieces, having a development set $F_1$-score higher than 93% throughout this range. The performance drops after that from 93% to approximately 86%. This kind of drop was not observed for any other language. The first thoughts were that this issue must be related either to the pre-trained model or the training data. To rule out the possibility that the performance drop happens only due to taking the mean over all hyperparameter combinations, the best achieved $F_1$-score (as mean of 5 repetitions) for each sentence starting position was plotted in a similar way. This approach shows a similar drop in performance, although the scale is smaller in this case, as show in Figure 6.2.

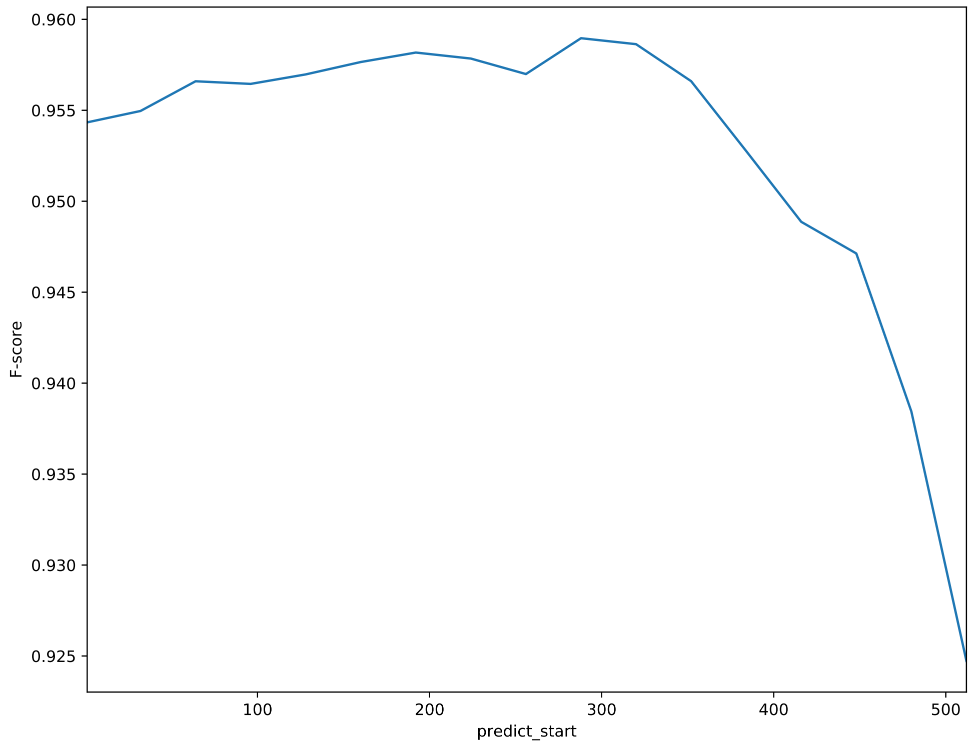


Figure 6.2: English, BERT base, performance on different starting positions, Best found performance on development data set

At the time of this evaluation no other data set for English with comparable size and entity types was available. To assess whether the model might be causing this effect, the experiment was repeated using BERT large model for English to see if the behaviour was similar. The results looked considerably more stable with respect to the starting position with BERT Large WWM, as seen in Figure 6.3, where the vertical axis is set roughly similarly as in Figure 6.1. It was decided to continue the study using that model instead of BERT base cased. Additionally, testing on BERT large was a good way to test the *Contextual Majority Voting* method against other state-of-the-art methods, which are commonly evaluated using BERT large.

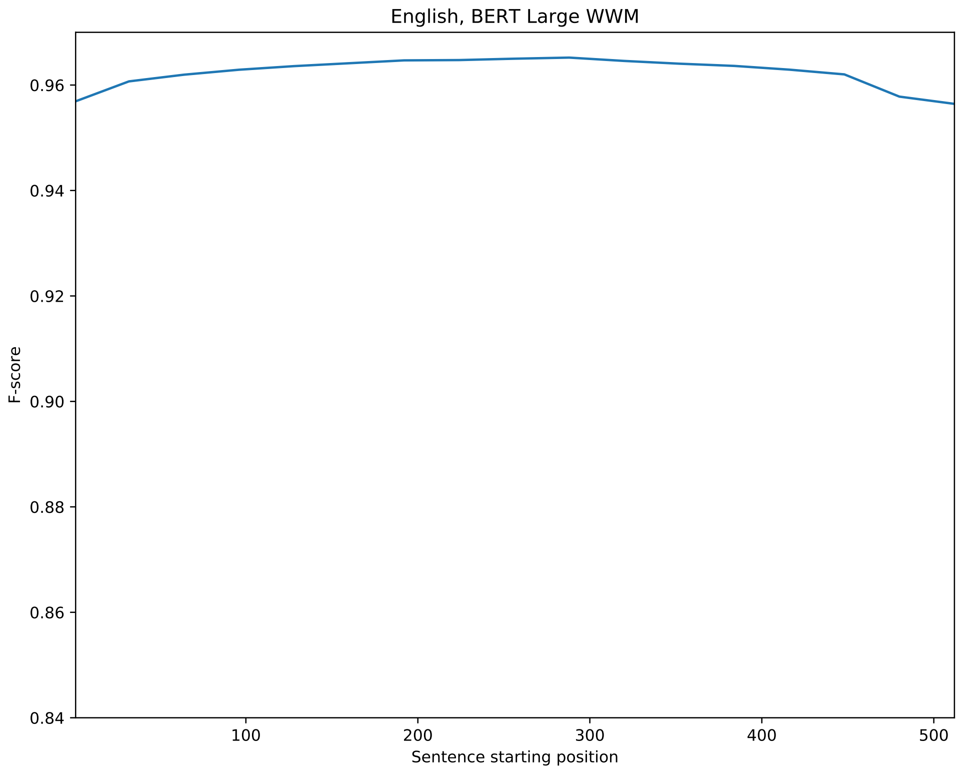


Figure 6.3: English, BERT Large WWM, performance on different starting positions, development data set

The BERT Large WWM model has stable performance throughout the range of starting locations when using the same data as with the BERT base pre-trained model. This appears to confirm that the BERT base cased pre-trained model has some issues learning to perform the NER task when using WordPieces at offsets greater than approximately 350. While this is a preliminary finding only, this suggests that careful evaluation should be performed if one is to use the BERT base model in real-world applications.

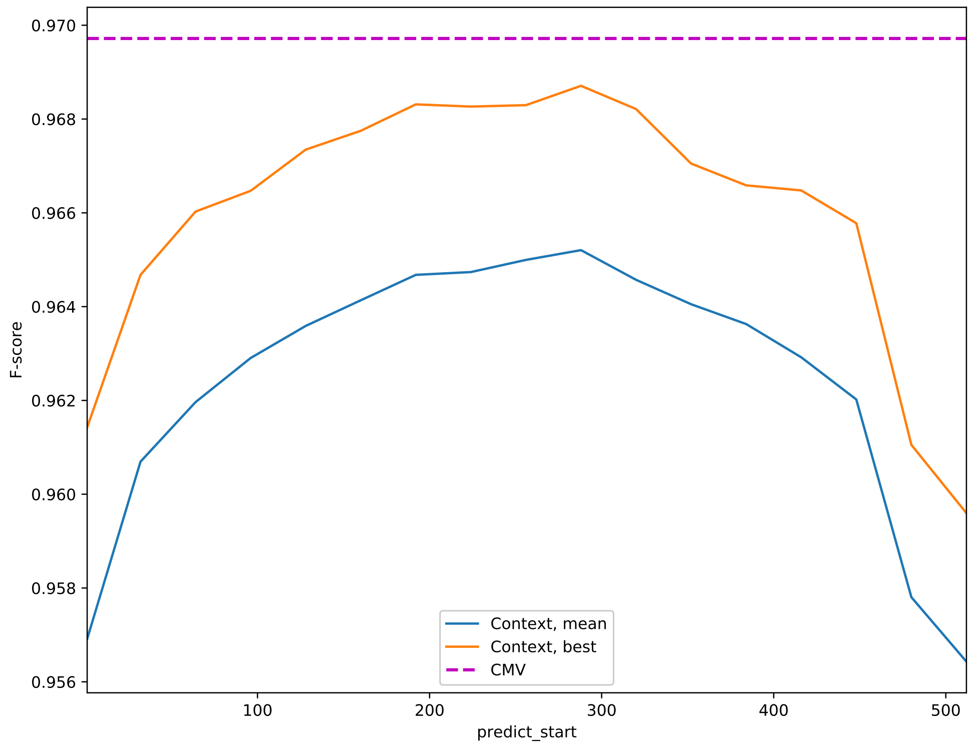


Figure 6.4: English, BERT Large WWM, Comparison of methods, development data set

Figure 6.4 shows the performance of the BERT Large WWM model on the English development set with different starting positions. The curve "Context, mean" (blue colour) is the average performance at a specified starting position (*Sentence* *in context*), averaged across all the results for that position and all of the hyperparameter combinations tested. The curve "Context, best" (orange colour) is a curve which shows the best result found in hyperparameter search for each position. The best result for each position is expressed the best mean $F_1$-score of 5 repeated experiments with same hyperparameter combination. The result for *Contextual Majority Voting (CMV)* is the best mean of 5 repetitions with same hyperparameters. From Figure 6.4 it is apparent that the *Contextual Majority Voting* gives a benefit over any specific starting position when comparing the results on the development set.

### 6.1.2 Dutch

The hyperparameter search for Dutch was performed in a similar manner as for English. Figure 6.5 shows that the *Sentence in Context* results vary according to the starting position of the sentence, but for Dutch model the differences are not very notable. An observation was made that both the *Sentence in context* and *Contextual Majority Voting* methods provided considerably better results than the ones presented by de Vries *et al.* [57] where the Dutch BERTje model was introduced and achieved a development set $F_1$-score of 87.8% for the NER task with the same data. This is remarkably promising result, as we observe an almost 5 percentage point improvement in the results.

Before starting the experiments evaluating the effect of adding cross-sentence context to BERT training samples, the initial expectation was that placing the sentence of interest near the middle of the sequence would generally yield the best performance. However, while this effect could be clearly observed for English (Figure 6.4), the pattern does not hold in all cases. While performance does appear to improve in most cases when moving the starting position away from either end of the context window, in some cases the problem was that the performance in the middle of the context did not appear to be stable enough to reliably pick a starting

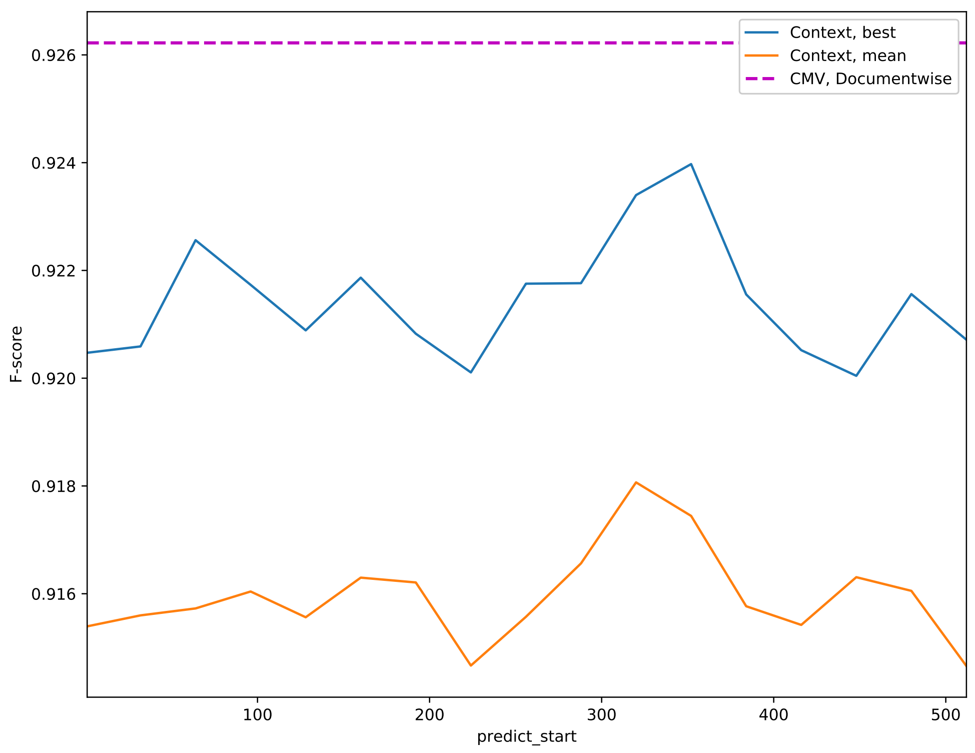


Figure 6.5: Dutch, performance on different starting positions, development data set

position. The results for Dutch deviated the most from the initial expectation, and a possible reason for this was later found from the source data: the sentence order of the documents inside the original Dutch language data set has been randomized for copyright reasons. To see if randomizing the sentence order of documents has an effect on results, we tested adding such shuffling to data in other languages. However, in our initial experiments randomizing sentences inside each document did not result in notable performance drop on any of the tested languages. [6]

### 6.1.3 Finnish

Finnish results on the development set appear to be in line with results for the other languages: prediction performance for a starting location appears to increase when moving away from either end of the maximum sequence length window (Figure 6.6). Also in this case the *Contextual Majority Voting* method seems to provide an increase in performance compared to any single starting location inside the window.

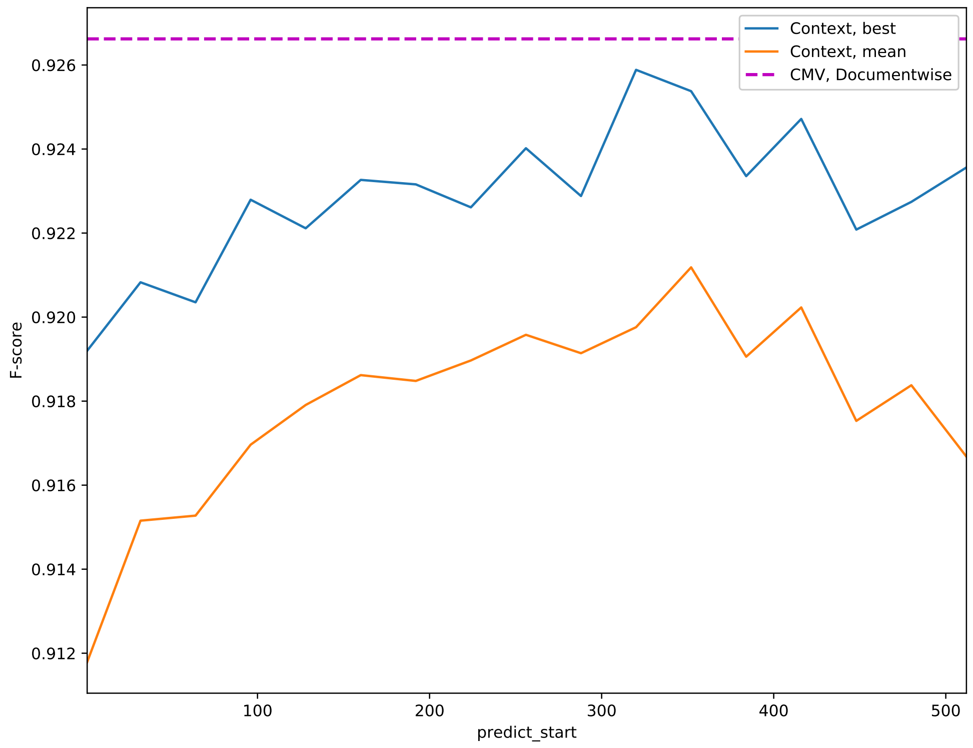


Figure 6.6: Finnish, performance on different starting positions, development data set

### 6.1.4 Spanish

Spanish was added to the tested languages in comparatively late in the study as no monolingual BERT model was available for Spanish when this work was first started. During the study a Spanish BERT model, named BETO, was introduced to public [58][1]. In experiments using *Contextual Majority Voting*, the results on the development set appeared to be quite similar to those achieved using multilingual BERT by Wu and Dredze [72] and for comparison purpose the NER evaluation was extended to also include multilingual BERT for the Spanish data. In our experiments on the development data, the Multilingual BERT model was found to produce slightly better results than the monolingual pre-trained model (Figure 6.7).

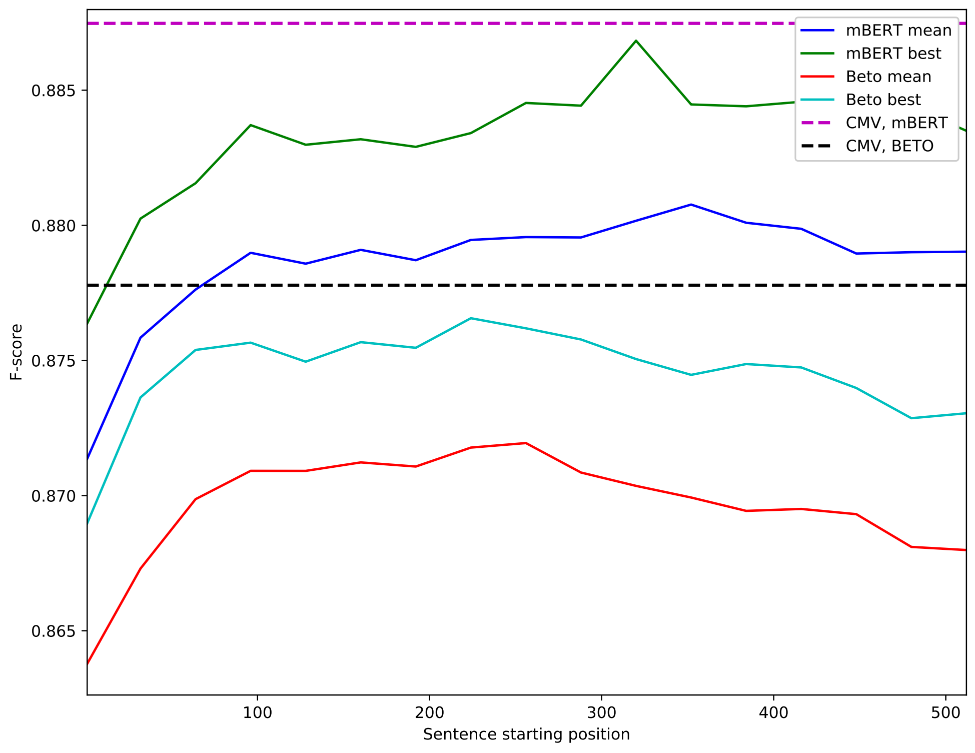


Figure 6.7: Spanish, performance on different starting positions, Comparison of mBERT and BETO, development data set

[1]`https://github.com/dccuchile/beto`

### 6.1.5 German

The hyperparameter search results on German were, in absolute terms, somewhat disappointing. The results seemed underwhelming after observing some of the high levels of absolute performance ($F_1$-score over 90%) achieved with BERT for other languages, and performance of some other methods on German [54, 70]. On the other hand, comparing the results e.g. to those achieved with multilingual BERT [72], things did not look that bad. Perhaps German is a challenging language for BERT models, or perhaps there are some issues with the CoNLL'03 German corpus. The results on development set for German are shown in Figure 6.8

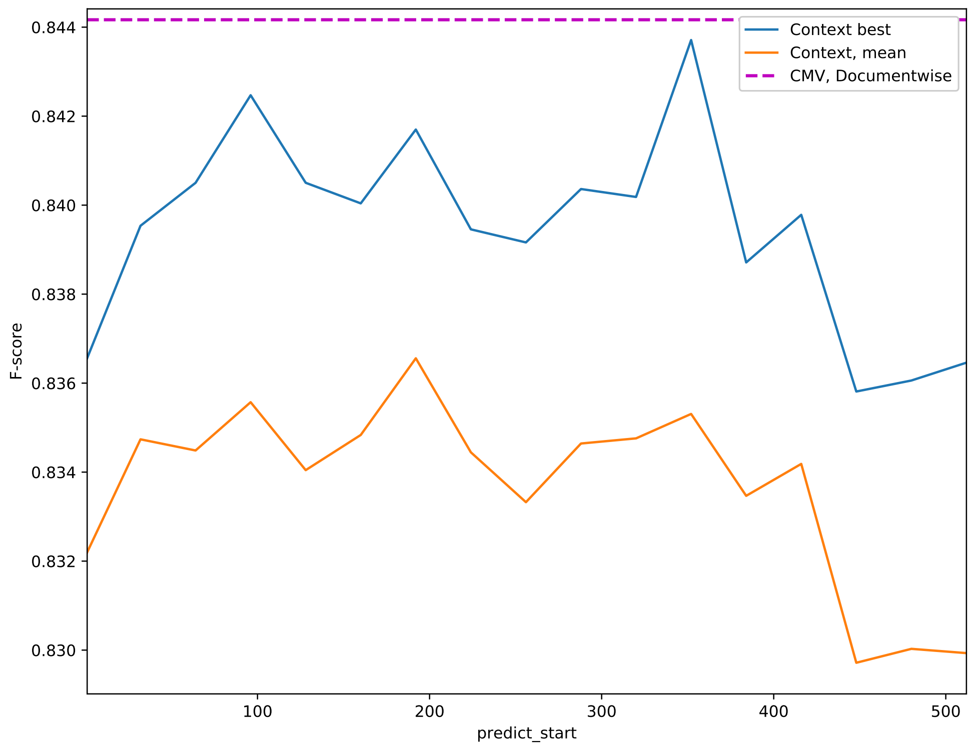


Figure 6.8: German, performance on different starting positions, development data set

It was also later learned that there exists a “revised” version of the CoNLL corpus for German and at least some other published NER results have used the revised corpus in their experiments. The problem was that it was not always clearly stated which studies have used the revised version. This study is made on the original CoNLL’03 version of the corpus. Despite the lower than expected $F_1$-scores, the results show a similar pattern of benefit of the *CMV* model seen with the other languages.

### 6.1.6 On reporting different methods

As seen in the preceding language-specific subsections, the results are shown only for *Contextual Majority Voting (CMV)* and the results when sentences are starting in a designated positions inside the input sample (*Sentence in Context*). Selecting the *Contextual Majority Voting (CMV)* as the method to report (over *CMV-P*) was due to the fact that in hyperparameter searches both the *CMV* and *CMV-P* consistently provided nearly identical numerical results, and both methods usually had the best performance with the same hyperparameters. The differences were too small to draw meaningful conclusions on whether one of these methods performed better than the other.

For cross-sentence context, the input samples built documentwise were preferred as the documentwise grouping appeared to work marginally better in comparison to the setting that does not use document information from input data. The exception to this was Spanish, which did not contain document information in the input data.

One challenge in assessing the impact of starting positions on NER performance is that often the results are quite similar regardless of the starting position. Averaging results over different hyperparameter combinations or over the whole hyperparameter range indicates some peaks and valleys in the results, but it is not guaranteed that a certain starting location would robustly improve results and generalize to good results on test set. The results between different repetitions of the same experiment with different random initializations of the NER layer weights varied in some cases more than the difference between different starting positions. This is one of the reasons that the *Sentence in Context* results on test data will be published only for the method *First*, meaning the sentence starting immediately after the `[CLS]` token in input sample. It would have been very challenging to make a reliable choice on any single location to use in final experiments on the test data. Another reason for selecting only the first position is to enable more direct comparison of the results to the selected Baseline, *Single* sentence, where the model is fine-tuned only by using single sentences in input samples without the context on the right side as in method *First*.

For the above-mentioned reasons the results provided in the tables and figures in the rest of this thesis will be for *Contextual Majority Voting (CMV)* on documentwise (except Spanish) built input samples, First sentence in Context (*First*) and Single sentence baseline (*Single*).

### 6.1.7 Best hyperparameter combinations

Table 6.1 presents the best $F_1$-scores and the corresponding hyperparameters found when evaluating the different methods against the development set during hyperparameter search. These hyperparameter combinations were used when fine-tuning the pre-trained models for final performance evaluation on test data sets.

| Model | Epochs | Batch size | Learning rate | $F_1$-score (dev) |
|---|---|---|---|---|
| Finnish, CMV | 2 | 2 | 2e-5 | 92.66 |
| Finnish, First | 2 | 2 | 2e-5 | 92.54 |
| Finnish, Single | 4 | 4 | 2e-5 | 91.48 |
| English, CMV | 3 | 2 | 3e-5 | 96.99 |
| English, First | 3 | 2 | 3e-5 | 96.86 |
| English, Single | 4 | 4 | 2e-5 | 96.17 |
| Dutch, CMV | 2 | 2 | 3e-5 | 92.62 |
| Dutch, First | 2 | 2 | 2e-5 | 92.37 |
| Dutch, Single | 4 | 2 | 2e-5 | 90.87 |
| German, CMV | 4 | 2 | 3e-5 | 84.43 |
| German, First | 2 | 2 | 3e-5 | 83.96 |
| German, Single | 3 | 4 | 3e-5 | 83.89 |
| Spanish, CMV | 3 | 8 | 2e-5 | 87.78 |
| Spanish, First | 3 | 8 | 2e-5 | 86.90 |
| Spanish, Single | 3 | 2 | 3e-5 | 86.56 |
| S-mBERT, CMV | 1 | 4 | 3e-5 | 88.75 |
| S-mBERT, First | 3 | 4 | 3e-5 | 87.64 |
| S-mBERT, Single | 4 | 4 | 3e-5 | 86.91 |

Table 6.1: Hyperparameter combinations resulting in the best performance on the development sets for different languages and methods

From the selected hyperparameters we can clearly observe that training BERT for NER with these data sets tends to favor smaller batch sizes as well as smaller learning rates. Almost all of the best hyperparameter combinations have batch size of 2 or 4. The highest batch size of 16 or the highest learning rate of 5e-5 is not selected for any of the many combinations of languages and methods.

### 6.1.8 CMV vs. First

In the preceding development set experiments, it was not only the best hyperparameter combinations selected for each method where the *Contextual Majority Voting* appeared to perform the better than method *First*: for most of the individual hyperparameter combinations, the best results were achieved when using *CMV* (or *CMV-P*) rather than any of the starting locations separately. The performance of *CMV* with different hyperparameters was also analyzed from the full results of the hyperparameter search and compared to the other methods. An analysis for all the languages and all of the hyperparameter combinations was performed to check if the method *First* provides better results than *CMV* for some combinations. It was found that for all of the tested languages and hyperparameter combinations besides Dutch, the method *CMV* provided better results on development data than method *First* in all cases. For Dutch there were 7 hyperparameter combinations out of the total 48 where *CMV* performed worse than *First*. The results for these 7 cases are shown in Table 6.2. These results show that the margin between the two methods with all of these parameters is quite small. Based on this information, one can quite safely recommend the usage of *CMV* method to improve results in almost all situations.

| Epochs | Learning Rate | Batch Size | CMV | First | CMV better |
|---|---|---|---|---|---|
| 2 | 0.00002 | 4 | 92.03% | 92.05% | False |
| | 0.00003 | 8 | 91.84% | 92.02% | False |
| 3 | 0.00002 | 4 | 92.12% | 92.24% | False |
| | | 8 | 91.87% | 91.89% | False |
| 4 | 0.00002 | 4 | 92.02% | 92.20% | False |
| | | 16 | 91.92% | 92.00% | False |
| | 0.00005 | 16 | 91.93% | 92.17% | False |

Table 6.2: Dutch, Hyperparameters for which method *First* outperforms *CMV*

## 6.2 Results on test sets

The results of experiments with the best found hyperparameter combinations are summarized in Table 6.3. From these results it is obvious that BERT predictions benefit from taking into account cross-sentence context. For all of the languages tested on the models which are fine-tuned and tested with samples containing context information outperform the models which do not use any context but rely only on single sentences. This difference is clearly significant for languages other than Spanish, with several percentage points separating the results for *CMV* from the results for *Single* with standard deviations generally well below half a percentage point. The ordering of the methods for Spanish is the same, but the results fall within one standard deviation of each other. It can also be observed from the results that using contextual information in both directions and using *CMV* aggregation tends to produce better results than the models which only use the right side context information. Exceptions to this rule seem to be English and Dutch languages.

One thing to note here is that also with English and Dutch the *CMV* outperforms

| | Precision | | Recall | | F1 | | F1 train+dev | |
|---|---|---|---|---|---|---|---|---|
| English, CMV | 93.06 | (0.25) | **93.78** | (0.08) | 93.42 | (0.12) | 93.57 | (0.33) |
| English, First | **93.15** | (0.15) | 93.73 | (0.04) | **93.44** | (0.06) | **93.74** | (0.25) |
| English, Single | 91.12 | (0.25) | 92.28 | (0.23) | 91.70 | (0.24) | 91.94 | (0.15) |
| Dutch, CMV | **93.12** | (0.26) | 93.26 | (0.18) | 93.19 | (0.21) | **93.49** | (0.23) |
| Dutch, First | 93.03 | (0.65) | **93.38** | (0.38) | **93.21** | (0.51) | 93.39 | (0.26) |
| Dutch, Single | 91.57 | (0.35) | 91.49 | (0.41) | 91.53 | (0.37) | 91.92 | (0.30) |
| Finnish, CMV | **92.91** | (0.18) | **94.42** | (0.13) | **93.66** | (0.13) | **93.78** | (0.26) |
| Finnish, First | 92.56 | (0.14) | 94.24 | (0.08) | 93.39 | (0.10) | 93.65 | (0.26) |
| Finnish, Single | 90.74 | (0.10) | 92.11 | (0.24) | 91.42 | (0.16) | 91.97 | (0.21) |
| German, CMV | **86.91** | (0.31) | **84.38** | (0.32) | **85.63** | (0.30) | **87.31** | (0.27) |
| German, First | 86.37 | (0.39) | 84.07 | (0.10) | 85.21 | (0.22) | 86.91 | (0.11) |
| German, Single | 85.55 | (0.20) | 81.81 | (0.31) | 83.64 | (0.21) | 85.67 | (0.25) |
| Spanish, CMV | **87.80** | (0.25) | **87.98** | (0.18) | **87.89** | (0.21) | **87.97** | (0.21) |
| Spanish, First | 86.71 | (0.31) | 87.41 | (0.28) | 87.06 | (0.28) | 87.27 | (0.25) |
| Spanish, Single | 87.43 | (0.53) | 87.90 | (0.34) | 87.66 | (0.43) | 87.52 | (0.41) |
| S-mBERT, CMV | **87.25** | (0.50) | **88.67** | (0.46) | **87.95** | (0.47) | **88.32** | (0.26) |
| S-mBERT, First | 86.92 | (0.40) | 87.88 | (0.44) | 87.40 | (0.42) | 87.54 | (0.25) |
| S-mBERT, Single | 87.19 | (0.28) | 87.81 | (0.26) | 87.50 | (0.26) | 87.57 | (0.29) |

Table 6.3: NER results for different methods and languages (standard deviation in parentheses)

the method *First* if we compare the *CMV* result with the same hyperparameter combination that is used for *First*. It seems that with *CMV* the differences between the best hyperparameter combinations are so small that the parameter combination chosen as best, even with five repetitions, may not represent the best choice of hyperparameters for the final model. As noted in Section 6.1.6, to get a better understanding on the effect of *CMV*, a comparison between the method *First* and *CMV* on different hyperparameters was performed. From those results it was seen that *CMV* improves the results at almost all of the points throughout the evaluated hyperparameter range.

From Table 6.3 it can also be seen that adding more training data seems to improve results on almost all of the languages and the methods. For testing this, the versions of final models were trained alternatively with the combined data of the training and development data sets. The combined data was used to fine-tune the pre-trained models with the best hyperparameter combinations selected for each method and language, and the resulting model was evaluated against the test set of the corresponding language. When using the extra data in training, the *CMV* model outperformed the method *First* also for Dutch. All of the results in Table 6.3 are expressed as the mean of five repetitions of the experiment, and their standard deviation written in parentheses. The best results for each language is shown with bold font.

### 6.2.1 Comparison with the state of the art

In this section the results of the implemented NER system are compared to the results of methods published in the literature and that represented the state of the art at the time of performing this study (May 2020). Table 6.4 presents the results with added cross-sentence context, state of the art results as well as the best results achieved with BERT-based systems for all of the languages considered in this study.

### 6.2.2 English

English and perhaps Chinese are the most studied languages in NER at the moment and therefore the results achieved in English language carry particular weight. The implementation adding cross-sentence context to input sample, evaluated on the

| Model | Our F1 | Our F1 (t+d) | BERT best | SOTA |
|---|---|---|---|---|
| English | 93.44 | **93.74** | 93.47 [80] | 93.5 [56] |
| Dutch | 93.21 | **93.49** | 90.94 [72] | 92.69 [81] |
| Finnish | 93.66 | **93.78** | 93.11 [77] | 93.11 [77] |
| German | 84.89 | 86.97 | 82.82 [72] | **88.32** [54] |
| Spanish | 87.89 | 87.97 | 88.43 [58] | **88.81** [81] |
| Spanish, mBERT | 87.95 | 88.32 | 88.43 [58] | **88.81** [81] |

Table 6.4: NER result comparison

CoNLL'03 English test data set (testb) achieved a mention level $F_1$-score of 93.44% as a mean of five repetitions. This result was improved to an $F_1$-score of 93.74% when additional data (CoNLL'03 English development set) was used in fine-tuning the Bert Large WWM pre-trained model. The $F_1$-score of 93.74%, which set the new state of the art for English CoNLL'03 NER benchmark, was reached using the method *First*. As the method *CMV* outperformed the method *First* with every hyperparameter combination on the development set, it was a surprise to see the method *First* performing better. In fact, the results of method *CMV* with the same hyperparameters as the method *First* resulted in an even better $F_1$-score of 93.88%, this simply was not achieved with the hyperparameters that happened to be selected for the method *CMV* in hyperparameter selection and therefore were not in our published study [6].

### 6.2.3 Dutch

For Dutch, the *CMV* method achieves the state-of-the-art results with $F_1$-score of 93.21% using only training set to fine-tune the models and $F_1$-score of 93.49% when also development set data was added in training the final models. Both of these results surpass the previous state of the art of 92.69% presented by Straková *et al.* [81]. One thing of interest to note that all the methods in Dutch (*CMV*, *First*, *Single*) outperform the NER results presented by de Vries *et al.* [57] clearly, and this implementation uses the pre-trained model that was introduced in their study.

### 6.2.4 Finnish

The *CMV* approach manages to improve on the previous state-of-the-art NER results in Finnish achieved when introducing the Turku NER corpus [77]. The performance on the combined Finnish corpus was increased from $F_1$-score of 93.11% to $F_1$-score of 93.66% without extra training data and to 93.78% using the development set in training in addition to the training set. The previous state of the art was achieved with essentially the same setup as with the method *First*. The difference is that in this study the results for method *First* were obtained using input samples with documentwise built context. The method *First* with $F_1$-score of 93.39% without extra training data and 93.65% with adding the development set to training also outperforms the previous state of the art.

### 6.2.5 Spanish

The results achieved with multilingual BERT outperform the results with the language-specific BERT model for Spanish. The result using multilingual BERT and the *CMV* method was an $F_1$-score of 87.95% without additional training data and 88.32% when using the development set for training in addition to the training set. Both of these results outperform the previous multilingual BERT-based results presented by Wu and Dredze [72] and fall somewhat short from the state of the art for Spanish claimed by Straková *et al.* [81]. The performance of the multilingual model over a language specific model was unexpected as monolingual BERT models have in many case reported to improve on the performance of multilingual BERT [5, 57, 78].

### 6.2.6 German

The results achieved by the creators of German BERT were published on the same web page with the model weights[2] without providing detailed information on the test setup used to achieve those results. The *CMV* method, with $F_1$-scores of 84.89% for a model trained without extra data and $F_1$-score of 86.97% for a model with extra data, outperformed the NER results published (80.4% for CoNLL'03 German) on that web page quite substantially. This implementation also outperformed the best BERT-based NER result of 82.82 (multilingual) published by Wu and Dredze [72] but falls behind from the state of the art on German NER of 88.81% published by Straková *et al.* [81]. The German model in particular appeared to benefit from additional training data as the performance increased by over 2 percentage points when introducing the additional data from the development set for training the final models.

[2]`https://deepset.ai/german-bert`

# 7 Conclusion

This study has introduced methods for including cross-sentence contexts to the Named Entity Recognition task. The proposed methods are using BERT language representation models, which are based on the Transformer deep neural network architecture and the transfer learning approach. The introduced methods provide a simple and easy-to-implement approach for the NER task using only pre- and post-processing of the inputs and outputs of the BERT model, keeping the model architecture intact. This way the approaches introduced here can be easily utilised with other BERT-based implementations. The proposed methods established new state-of-the-art results in NER for three languages and were near the state of the art for two other languages, demonstrating how simple ideas can boost the performance of even very strong models.

Naturally, there exist many possible areas for further study of NER that were left out of the scope of this work. The implemented methods and the NER pipeline as whole provide a good starting point for further studies with BERT and other Transformer-based methods. Some further developments could be to analyse the errors in predictions on a detailed level and study if there is some information there that could be utilized in developing better methods. Also, an observation was made that results seemed better when more training data was used for the final models. This raises the question if there are methods to easily generate more training data or perhaps augment the current data in some way as is commonly done e.g. with image data. Another possible direction for further study could be to study how to utilize other available NER resources better. There is already a long history of study in Named Entity Recognition, different training data and also potent models for the task released before and after BERT. It is perhaps worthwhile sometimes to take a look at what we already have and not always rush to the newest neural network architecture. There is so much to learn, just go for it!

# References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding”, in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. DOI: `10.18653/v1/N19-1423`. [Online]. Available: `https://www.aclweb.org/anthology/N19-1423`.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks”, in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: `https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of Go with deep neural networks and tree search”, *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin, “Attention is All You Need”, in *Proceedings of the 31st International Conference on Neural Information Processing Systems*,

ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010, ISBN: 9781510860964.

[5] A. Virtanen, J. Kanerva, R. Ilo, J. Luoma, J. Luotolahti, T. Salakoski, F. Ginter, and S. Pyysalo, "Multilingual is not enough: BERT for Finnish", *ArXiv*, vol. abs/1912.07076, 2019.

[6] J. Luoma and S. Pyysalo, "Exploring cross-sentence contexts for named entity recognition with BERT", in *Proceedings of the 28th International Conference on Computational Linguistics*, Barcelona, Spain (Online): International Committee on Computational Linguistics, Dec. 2020, pp. 904–914. DOI: `10.18653/v1/2020.coling-main.78`. [Online]. Available: `https://www.aclweb.org/anthology/2020.coling-main.78`.

[7] V. Yadav and S. Bethard, "A Survey on Recent Advances in Named Entity Recognition from Deep Learning models", in *Proceedings of the 27th International Conference on Computational Linguistics*, Santa Fe, New Mexico, USA: Association for Computational Linguistics, Aug. 2018, pp. 2145–2158. [Online]. Available: `https://www.aclweb.org/anthology/C18-1182`.

[8] J. Li, A. Sun, J. Han, and C. Li, "A Survey on Deep Learning for Named Entity Recognition", *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2020, ISSN: 2326-3865. DOI: `10.1109/tkde.2020.2981314`. [Online]. Available: `http://dx.doi.org/10.1109/tkde.2020.2981314`.

[9] R. Grishman and B. Sundheim, "Message understanding conference- 6: A brief history", in *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, 1996. [Online]. Available: `https://www.aclweb.org/anthology/C96-1079`.

[10] "Appendix C: Named entity task definition (v2.1)", in *Sixth Message Understanding Conference (MUC-6): Proceedings of a Conference Held in Columbia,*

*Maryland, November 6-8, 1995*, 1995. [Online]. Available: https://www.aclweb.org/anthology/M95-1024.

[11] L. Ramshaw and M. Marcus, "Text Chunking using Transformation-Based Learning", in *Third Workshop on Very Large Corpora*, 1995. [Online]. Available: https://www.aclweb.org/anthology/W95-0107.

[12] E. F. Tjong Kim Sang, "Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition", in *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*, 2002. [Online]. Available: https://www.aclweb.org/anthology/W02-2024.

[13] E. F. Tjong Kim Sang and F. De Meulder, "Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition", in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 142–147. [Online]. Available: https://www.aclweb.org/anthology/W03-0419.

[14] A. Ratnaparkhi, "Maximum entropy models for natural language ambiguity resolution", Ph.D. dissertation, University of Pennsylvania, 1998.

[15] E. F. Tjong Kim Sang and S. Buchholz, "Introduction to the CoNLL-2000 shared task chunking", in *Fourth Conference on Computational Natural Language Learning and the Second Learning Language in Logic Workshop*, 2000. [Online]. Available: https://www.aclweb.org/anthology/W00-0726.

[16] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification", *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[17] L. Rau, "Extracting company names from text", in *[1991] Proceedings. The Seventh IEEE Conference on Artificial Intelligence Application*, vol. i, 1991, pp. 29–32. DOI: 10.1109/CAIA.1991.120841.

[18] A. E. Borthwick, "A Maximum Entropy Approach to Named Entity Recognition", AAI9945252, Ph.D. dissertation, USA, 1999, ISBN: 0599472324.

[19] *Third Message Uunderstanding Conference (MUC-3): Proceedings of a Conference Held in San Diego, California, May 21-23, 1991*, 1991. [Online]. Available: `https://www.aclweb.org/anthology/M91-1000`.

[20] *Fourth Message Uunderstanding Conference (MUC-4): Proceedings of a Conference Held in McLean, Virginia, June 16-18, 1992*, 1992. [Online]. Available: `https://www.aclweb.org/anthology/M92-1000`.

[21] *Fifth Message Understanding Conference (MUC-5): Proceedings of a Conference Held in Baltimore, Maryland, August 25-27, 1993*, 1993. [Online]. Available: `https://www.aclweb.org/anthology/M93-1000`.

[22] *Sixth Message Understanding Conference (MUC-6): Proceedings of a Conference Held in Columbia, Maryland, November 6-8, 1995*, 1995. [Online]. Available: `https://www.aclweb.org/anthology/M95-1000`.

[23] *Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29 - May 1, 1998*, 1998. [Online]. Available: `https://www.aclweb.org/anthology/M98-1000`.

[24] A. Mikheev, C. Grover, and M. Moens, "Description of the LTG system used for MUC-7", in *Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29 - May 1, 1998*, 1998. [Online]. Available: `https://www.aclweb.org/anthology/M98-1021`.

[25] M. Collins and Y. Singer, "Unsupervised Models for Named Entity Classification", in *1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999. [Online]. Available: `https://www.aclweb.org/anthology/W99-0613`.

[26] D. Nadeau, P. D. Turney, and S. Matwin, “Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity”, in *Conference of the Canadian society for computational studies of intelligence*, Springer, 2006, pp. 266–277.

[27] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, “Unsupervised Named-Entity Extraction from the Web: An Experimental Study”, *Artif. Intell.*, vol. 165, no. 1, pp. 91–134, Jun. 2005, ISSN: 0004-3702.

[28] L. Ratinov and D. Roth, “Design Challenges and Misconceptions in Named Entity Recognition”, in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, Boulder, Colorado: Association for Computational Linguistics, Jun. 2009, pp. 147–155. [Online]. Available: `https://www.aclweb.org/anthology/W09-1119`.

[29] R. Ando and T. Zhang, “A High-Performance Semi-Supervised Learning Method for Text Chunking”, in *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL’05)*, Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2005, pp. 1–9. DOI: `10.3115/1219840.1219841`. [Online]. Available: `https://www.aclweb.org/anthology/P05-1001`.

[30] O. Bender, F. J. Och, and H. Ney, “Maximum Entropy Models for Named Entity Recognition”, in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 148–151. [Online]. Available: `https://www.aclweb.org/anthology/W03-0420`.

[31] H. L. Chieu and H. T. Ng, “Named Entity Recognition with a Maximum Entropy Approach”, in *Proceedings of the Seventh Conference on Natural Lan-*

*guage Learning at HLT-NAACL 2003*, 2003, pp. 160–163. [Online]. Available: https://www.aclweb.org/anthology/W03-0423.

[32] J. Curran and S. Clark, "Language independent NER using a maximum entropy tagger", in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 164–167. [Online]. Available: https://www.aclweb.org/anthology/W03-0424.

[33] R. Florian, A. Ittycheriah, H. Jing, and T. Zhang, "Named Entity Recognition through Classifier Combination", in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 168–171. [Online]. Available: https://www.aclweb.org/anthology/W03-0425.

[34] D. Klein, J. Smarr, H. Nguyen, and C. D. Manning, "Named Entity Recognition with Character-Level Models", in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 180–183. [Online]. Available: https://www.aclweb.org/anthology/W03-0428.

[35] J. Mayfield, P. McNamee, and C. Piatko, "Named Entity Recognition using Hundreds of Thousands of Features", in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 184–187. [Online]. Available: https://www.aclweb.org/anthology/W03-0429.

[36] C. Whitelaw and J. Patrick, "Named Entity Recognition Using a Character-based Probabilistic Approach", in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 196–199. [Online]. Available: https://www.aclweb.org/anthology/W03-0432.

[37] T. Zhang and D. Johnson, "A Robust Risk Minimization based Named Entity Recognition System", in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 204–207. [Online]. Available: https://www.aclweb.org/anthology/W03-0434.

[38] X. Carreras, L. Màrquez, and L. Padró, "Learning a Perceptron-Based Named Entity Chunker via Online Recognition Feedback", in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 156–159. [Online]. Available: `https://www.aclweb.org/anthology/W03-0422`.

[39] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory", *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: `10.1162/neco.1997.9.8.1735`.

[40] J. Hammerton, "Named Entity Recognition with Long Short-Term Memory", in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 172–175. [Online]. Available: `https://www.aclweb.org/anthology/W03-0426`.

[41] X. Carreras, L. Màrquez, and L. Padró, "A simple named entity extractor using AdaBoost", in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 152–155. [Online]. Available: `https://www.aclweb.org/anthology/W03-0421`.

[42] D. Wu, G. Ngai, and M. Carpuat, "A Stacked, Voted, Stacked Model for Named Entity Recognition", in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 200–203. [Online]. Available: `https://www.aclweb.org/anthology/W03-0433`.

[43] F. De Meulder and W. Daelemans, "Memory-Based Named Entity Recognition using Unannotated Data", in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 208–211. [Online]. Available: `https://www.aclweb.org/anthology/W03-0435`.

[44] I. Hendrickx and A. van den Bosch, "Memory-based one-step named-entity recognition: Effects of seed list features, classifier stacking, and unannotated data", in *Proceedings of the Seventh Conference on Natural Language Learning*

*at HLT-NAACL 2003*, 2003, pp. 176–179. [Online]. Available: https://www.aclweb.org/anthology/W03-0427.

[45] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data”, in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML ’01, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282–289, ISBN: 1558607781.

[46] A. McCallum and W. Li, “Early results for Named Entity Recognition with Conditional Random Fields, Feature Induction and Web-Enhanced Lexicons”, in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 188–191. [Online]. Available: https://www.aclweb.org/anthology/W03-0430.

[47] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient Estimation of Word Representations in Vector Space*, 2013. arXiv: 1301.3781 [cs.CL].

[48] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality”, in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2013, pp. 3111–3119. [Online]. Available: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[49] Jeffrey Pennington and Richard Socher and Christopher D. Manning, “Glove: Global vectors for word representation”, in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162.

[50] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information", *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, Dec. 2017, ISSN: 2307-387X. DOI: 10.1162/tacl_a_00051. [Online]. Available: http://dx.doi.org/10.1162/tacl_a_00051.

[51] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural Language Processing (Almost) from Scratch", *J. Mach. Learn. Res.*, vol. 12, no. null, pp. 2493–2537, Nov. 2011, ISSN: 1532-4435.

[52] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "RCV1: A New Benchmark Collection for Text Categorization Research", *J. Mach. Learn. Res.*, vol. 5, pp. 361–397, Dec. 2004, ISSN: 1532-4435.

[53] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation", in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. [Online]. Available: https://www.aclweb.org/anthology/D14-1179.

[54] A. Akbik, D. Blythe, and R. Vollgraf, "Contextual String Embeddings for Sequence Labeling", in *Proceedings of the 27th International Conference on Computational Linguistics*, Santa Fe, New Mexico, USA: Association for Computational Linguistics, Aug. 2018, pp. 1638–1649. [Online]. Available: https://www.aclweb.org/anthology/C18-1139.

[55] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep Contextualized Word Representations", in *Proceedings of the 2018 Conference of the North American Chapter of the Association for*

*Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. [Online]. Available: https://www.aclweb.org/anthology/N18-1202.

[56] A. Baevski, S. Edunov, Y. Liu, L. Zettlemoyer, and M. Auli, "Cloze-driven Pretraining of Self-attention Networks", in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 5360–5369. DOI: 10.18653/v1/D19-1539. [Online]. Available: https://www.aclweb.org/anthology/D19-1539.

[57] W. de Vries, A. van Cranenburgh, A. Bisazza, T. Caselli, G. v. Noord, and M. Nissim, *BERTje: A Dutch BERT Model*, arXiv:1912.09582, Dec. 2019. [Online]. Available: http://arxiv.org/abs/1912.09582.

[58] J. Cañete, G. Chaperon, R. Fuentes, and J. Pérez, "Spanish Pre-Trained BERT Model and Evaluation Data", in *PML4DC at ICLR 2020*, 2020.

[59] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[60] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization", *arXiv preprint arXiv:1607.06450*, 2016.

[61] M. Schuster and K. Nakajima, "Japanese and korean voice search", in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2012, pp. 5149–5152.

[62] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation

system: Bridging the gap between human and machine translation", *arXiv preprint arXiv:1609.08144*, 2016.

[63] P. Gage, "A new algorithm for data compression", *C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.

[64] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing", in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 66–71. DOI: `10.18653/v1/D18-2012`. [Online]. Available: `https://www.aclweb.org/anthology/D18-2012`.

[65] W. L. Taylor, "Cloze procedure: A new tool for measuring readability", *Journalism quarterly*, vol. 30, no. 4, pp. 415–433, 1953.

[66] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition", in *Neurocomputing*, Springer, 1990, pp. 227–236.

[67] Z. Huang, W. Xu, and K. Yu, "Bidirectional LSTM-CRF models for sequence tagging", *arXiv preprint arXiv:1508.01991*, 2015.

[68] A. Passos, V. Kumar, and A. McCallum, "Lexicon infused phrase embeddings for named entity resolution", in *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2014, pp. 78–86. DOI: `10.3115/v1/W14-1609`. [Online]. Available: `https://www.aclweb.org/anthology/W14-1609`.

[69] V. Krishnan and C. D. Manning, "An effective two-stage model for exploiting non-local dependencies in named entity recognition", in *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, Sydney, Australia:

Association for Computational Linguistics, Jul. 2006, pp. 1121–1128. DOI: 10.3115/1220175.1220316. [Online]. Available: https://www.aclweb.org/anthology/P06-1141.

[70] A. Akbik, T. Bergmann, and R. Vollgraf, "Pooled Contextualized Embeddings for Named Entity Recognition", in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 724–728. DOI: 10.18653/v1/N19-1078. [Online]. Available: https://www.aclweb.org/anthology/N19-1078.

[71] Y. Luo, F. Xiao, and H. Zhao, "Hierarchical Contextualized Representation for Named Entity Recognition", in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, AAAI Press, 2020, pp. 8441–8448. [Online]. Available: https://aaai.org/ojs/index.php/AAAI/article/view/6363.

[72] S. Wu and M. Dredze, "Beto, Bentz, Becas: The Surprising Cross-Lingual Effectiveness of BERT", in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 833–844. DOI: 10.18653/v1/D19-1077. [Online]. Available: https://www.aclweb.org/anthology/D19-1077.

[73] T. Liu, J.-G. Yao, and C.-Y. Lin, "Towards improving neural named entity recognition with gazetteers", in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy: Association for

Computational Linguistics, Jul. 2019, pp. 5301–5307. DOI: 10.18653/v1/P19-1524. [Online]. Available: https://www.aclweb.org/anthology/P19-1524.

[74] P. Banerjee, K. K. Pal, M. Devarakonda, and C. Baral, *Knowledge Guided Named Entity Recognition for BioMedical Text*, 2019. arXiv: 1911.03869 [cs.CL].

[75] X. Li, J. Feng, Y. Meng, Q. Han, F. Wu, and J. Li, "A unified MRC framework for named entity recognition", in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online: Association for Computational Linguistics, Jul. 2020, pp. 5849–5859. DOI: 10.18653/v1/2020.acl-main.519. [Online]. Available: https://www.aclweb.org/anthology/2020.acl-main.519.

[76] T. Ruokolainen, P. Kauppinen, M. Silfverberg, and K. Lindén, "A Finnish news corpus for named entity recognition", *Language Resources and Evaluation*, pp. 1–26, 2019.

[77] J. Luoma, M. Oinonen, M. Pyykönen, V. Laippala, and S. Pyysalo, "A Broad-coverage Corpus for Finnish Named Entity Recognition", in *Proceedings of The 12th Language Resources and Evaluation Conference*, Marseille, France: European Language Resources Association, May 2020, pp. 4615–4624. [Online]. Available: https://www.aclweb.org/anthology/2020.lrec-1.567.

[78] H. Le, L. Vial, J. Frej, V. Segonne, M. Coavoux, B. Lecouteux, A. Allauzen, B. Crabbé, L. Besacier, and D. Schwab, "FlauBERT: Unsupervised Language Model Pre-training for French", in *Proceedings of The 12th Language Resources and Evaluation Conference*, Marseille, France: European Language Resources Association, May 2020, pp. 2479–2490. [Online]. Available: https://www.aclweb.org/anthology/2020.lrec-1.302.

[79] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[80] Y. Liu, F. Meng, J. Zhang, J. Xu, Y. Chen, and J. Zhou, "GCDT: A Global Context Enhanced Deep Transition Architecture for Sequence Labeling", in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 2431–2441. DOI: 10.18653/v1/P19-1233. [Online]. Available: https://www.aclweb.org/anthology/P19-1233.

[81] J. Straková, M. Straka, and J. Hajic, "Neural architectures for nested NER through linearization", in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 5326–5331. DOI: 10.18653/v1/P19-1527. [Online]. Available: https://www.aclweb.org/anthology/P19-1527.