Master's Thesis

Joint Entity Linking with BERT

Amund Faller Råheim

Supervisor: Natalie Prange

Examiners: Prof. Dr. Hannah Bast
Dr. Fang Wei-Kleiner

Albert-Ludwigs-University Freiburg
Faculty of Engineering
Department of Computer Science
Chair for Algorithms and Data Structures

May 10th, 2022

Writing period

13. 12. 2021 – 10. 5. 2022

Examiners

Prof. Dr. Hannah Bast

Dr. Fang Wei-Kleiner

Supervisor

Natalie Prange

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

Entity Linking aims to connect mentioned names in a text to unambiguous entities in a knowledge base. State-of-the-art approaches are deep neural network models based on one or more instances of BERT. With the use of a Candidate Generation step between Mention Detection and Entity Disambiguation, these approaches tend to break the end-to-end capabilities of training deep neural networks. This thesis aims to investigate an end-to-end approach that combines Mention Detection and Entity Disambiguation in a single BERT-based model. We find that we are able to reach near state-of-the-art performance, but that we are unable to reproduce previous results with a similar approach.

Contents

List of Figures

List of Tables

1. Introduction

Information extraction from text is an important component of many technologies in contemporary society, such as search engines, recommendation systems, question answering systems, and chatbots. Search engines are essential for navigating the World Wide Web, which shows an increase in global use every year (ITU, 2020), and for accessing the growing corpus of open-access scholarly literature (Ware and Mabe, 2015).

Search engines benefit from information extraction from both the search query and the target documents (e.g., web pages or scholarly articles). The aim is to extract features from the text to rank the search results according to relevance to the search query (Croft et al., 2010).

Entity Linking is a common method for information extraction from unstructured text. The task consists of two parts:

- first, **Mention Detection** discovers, in the target document, the text snippets that reference entities;
- and second, **Entity Disambiguation** identifies the correct entity for each discovered mention.

Figure 1 illustrates a simplified Entity Linking system.

An **entity** of interest can for example be a location, person or event. We define a **mention** as the text snippet that references an entity. A **Knowledge Base** stores entities of interest for disambiguation and may have additional information about the entities. Wikipedia and Wikidata are frequently used as Knowledge Bases.

Let the following two sentences be example documents:

1. The Sun King ruled for over 72 years.
2. Sun King appears on Abbey Road.

The first document has the mention “The Sun King”, and the second document has the two mentions “Sun King” and “Abbey Road”. We can infer that the documents

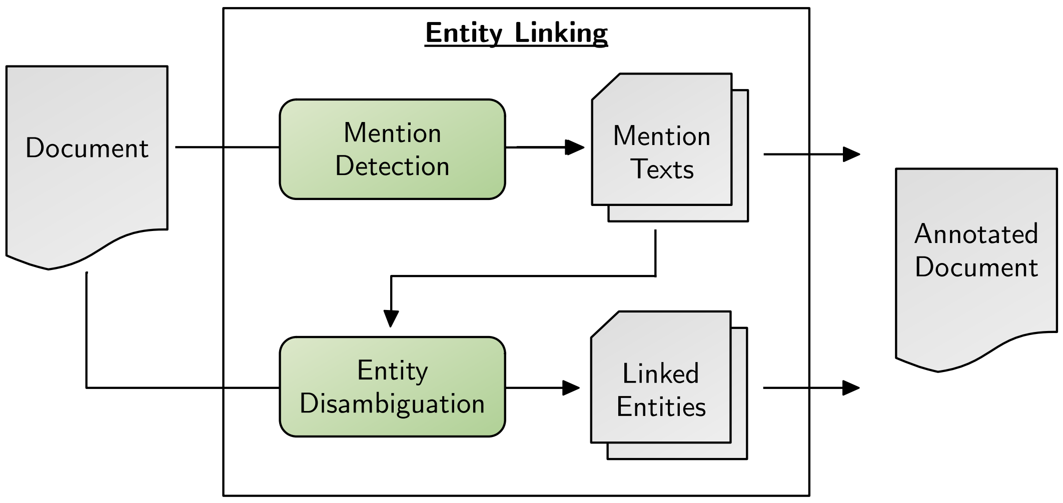


Figure 1.: A simple Entity Linking system.

refer to different entities with the mentions of "The Sun King" and "Sun King". Using the context of each document, we can **disambiguate** each mention to the entity to which they refer.

We assume that all the mentioned entities are in the Knowledge Base used by our Entity Linking method. In other words, the Entity Linking method should aim to disambiguate all these mentions.

The correctly annotated documents, with Wikipedia as a Knowledge Base, can be illustrated like this, where text in braces ({...}) represent detected mentions, and text in brackets ([...]) are the Wikipedia entities to which the mentions refer:

1. {The Sun King}[Louis_XIV] ruled for over 72 years.
2. {Sun King}[Sun_King_(song)] appears on {Abbey Road}[Abbey_Road].

A Knowledge Base can contain millions of entities and disambiguation can be a challenge. Most Entity Linking approaches use an additional step of **Candidate Generation** to narrow the search for potential entities to a subset of the Knowledge Base. This step traditionally comes between Mention Detection and Entity Disambiguation, using the mention text to search among aliases of entities in the Knowledge Base. The result is a list of **candidate entities** for each mention, which significantly simplifies the disambiguation task.

1.1. Task Definition

Given a Knowledge Base KB storing all entities of interest and an input document d, we define Entity Linking as finding the subset of entities $E \subset KB$ appearing in the document d and finding their respective mention spans M in the document. A mention span m has the start and end position of the mention in the document.

With an Entity Linking method EL:

$$EL(d) = \{M, E\}$$

Furthermore, we assume a labeled dataset of documents D. Each reference to an entity of interest in D is annotated with a position span m and an entity ID e from the Knowledge Base KB. Using a subset of documents $D_{\text{train}} \subset D$ (the "training set"), we fit a model M to reproduce the annotated mention and entity labels. This gives the followng approximation:

$$M(d) \approx EL(d) = \{M, E\}, \forall d \in D_{\text{train}}$$

We evaluate the model on a subset D_{test} (a discrete "test set", such that $D_{\text{test}} \cup D_{\text{train}} = \emptyset$) to evaluate the model performance on unseen documents.

1.2. Motivation

Deep neural networks (Section 2.1) built on BERT (Devlin et al. (2019), see Section 2.2 for details) are currently the state-of-the-art for Entity Linking and Entity Disambiguation. Using two independent BERT models for Mention Detection and Entity Disambiguation already shows good results on Entity Linking benchmarks (Ravi et al., 2021). However, recent research in Entity Linking suggests a co-dependence between Mention Detection and Entity Disambiguation, and modeling the two tasks with a joint training objective shows a better Entity Linking performance (Martins et al., 2019; Kolitsas et al., 2018).

In light of these findings, we hypothesize that models with joint Mention Detection and Entity Disambiguation have a better performance potential than disjoint models. In order to jointly model the tasks with a neural network, we need to design an end-to-end model with a joint training objective. In order to train end-to-end neural networks, the operations in the network need to be differentiable. Differentiable operations have a derivative, and thereby a gradient. This gradient is the basis for the training signal of the network. For more details on how we train neural networks, refer to Section 2.1.

Unfortunately, the classic Entity Linking pipeline (Mention Detection → Candidate Generation → Entity Disambiguation) does not easily translate to an end-to-end model. This is due to the Candidate Generation step where, to the best of our knowledge, there are no differentiable methods available. Consequently, Candidate Generation between Mention Detection and Entity Disambiguation creates a roadblock when training deep neural networks: the training signal (i.e., the gradient) cannot be evaluated for Candidate Generation and thus does not reach all the trainable weights of the model.

Innovations in Candidate Generation methods generally come from incorporating more prior knowledge about the Knowledge Base entities (Ganea and Hofmann, 2017; Ravi et al., 2021). Ideally, the evaluation of new Candidate Generation methods should be independent of Entity Disambiguation models. Similarly, Entity Linking methods that can employ any Candidate Generation method should be compared using the same Candidate Generation method. That way, Entity Linking methods can be compared on equal terms, and Candidate Generation methods can be evaluated across models.

There have been attempts to model Entity Linking without Candidate Generation (Broscheit, 2019), or where Candidate Generation is an optional step (Chen et al., 2019). When not using Candidate Generation, these methods do not perform on par with the state-of-the-art. However, they show a proof of concept for Entity Linking with a single model trained end-to-end. Both these approaches rely heavily on BERT as a basis for their models.

Of particular interest to us is the approach of Chen et al. (2019), who propose an end-to-end model entirely agnostic to Candidate Generation: the model can be evaluated with any Candidate Generation method, or indeed none at all.

In this thesis, we attempt to reproduce the results of Chen et al. (2019) and find that we are unable to do so. However, our model achieves a strong performance when using Candidate Generation. We also show that training on larger datasets can give a performance increase.

1.3. Contribution

In this thesis, we aim to answer the following questions:

- Can we reproduce the model and performance of Chen et al. (2019)?

- Can we improve performance by changing the model architecture and training procedure relative to Chen et al. (2019)?
- Does the model benefit from additional training data?
- How does Candidate Generation impact the performance of this model?
- How well does the model generalize to entities that it has not trained on?
- How well does the model perform on entities in the categories organizations, locations, persons and events?

1.4. Chapter Overview

In the next chapter, Chapter 2, we will provide the necessary background knowledge on deep learning — the methodology of training large neural network models, and introduce the neural network architecture "BERT", which we use in our approach.

Next, Chapter 3 gives an overview of the research on Entity Disambiguation and Entity Linking, focusing primarily on the recent neural network models that have improved the state-of-the-art. The chapter culminates in the method of Chen et al. (2019), which currently report state-of-the-art results.

In Chapter 4, we detail our model architecture, training procedure, Knowledge Base and Candidate Generation modules. We contrast this to Chen et al. (2019) and highlight the known differences to their method.

Chapter 5 introduces the evaluation criteria for Entity Linking and the datasets we use for training and evaluation. The chapter then details the experiments and results which motivate the modifications to the approach of Chen et al. (2019).

Next, Chapter 6 presents the final results of our best-performing models and compares them to the state-of-the-art for Entity Linking. We also give an in-depth analysis of some strengths and weaknesses of the model.

Finally, Chapter 7 summarizes the findings of this thesis and answers the research questions posed in Section 1.3.

2. Background

This chapter gives a theoretical introduction to methods we rely on in this thesis. Section 2.1 introduces artificial neural network models and the methods we use to train them. Section 2.2 details the neural network architecture known as BERT, which is the basis for our model.

2.1. Deep Learning

Large neural networks differ from other machine learning methods in that they largely remove the need to manually extract and select features of the data. In text documents, an extracted feature could for example be the frequency of words in the document. Instead, neural networks will often take raw, unprocessed input data and **learn to extract feature representations** that are specific to the task on which they train.

Deep learning is the methodology used to train deep neural networks. We focus on **supervised learning** in this thesis, where a model trains by predicting known ground-truth labels of a dataset. In contrast, unsupervised learning can be used, for example, to learn task-independent feature extraction.

2.1.1. Artificial Neural Network Models

Artificial neural networks are a collection of machine learning methods that are loosely inspired by biological neural networks in the animal brain. Networks may have different architectures and use different computations, and can train on a wide variety of tasks. Categories of artificial neural network architectures include feedforward neural networks, convolutional neural networks, recurrent neural networks, and transformers.

The feedforward neural network is the most basic neural network architecture. The input data is represented numerically, and transformed through the model by a sequence of linear transformations. Each linear transformation uses a weight matrix

$\mathbf{W}$ and a bias vector $\mathbf{b}$ to produce an "activation" $\mathbf{z}$. For an input vector $\mathbf{x}$, the linear transformation is:

$$\mathbf{z} = \mathbf{W}^T\mathbf{x} + \mathbf{b}$$

Each linear transformation is followed by a non-linear activation function $g(\mathbf{z})$. Without non-linear activation functions, a neural network can only model linear functions, and would be inadequate to model complex data such as images and text.

A linear transformation and an activation function make up a **layer** of the neural network. The layers between the first layer (the *input layer*) and final layer (the *output layer*) are called **hidden layers**. The output $\mathbf{h}$ of a layer with index n is computed as:

$$\mathbf{h}_n = g_n(\mathbf{h}_{n-1}\mathbf{W}_n^T + \mathbf{b}_n)$$

The input layer takes the data point vector $\mathbf{x}$ as input. The output layer produces a prediction $\hat{y}$. For a classification task, the prediction can, for example, be a vector of probabilities for different classes. We refer to the latent vectors $\mathbf{h}$ in the hidden layers as **hidden representations**.

Given a feedforward neural network with an input layer, a hidden layer, and an output layer (with indexes 1, 2, and 3, respectively), we can represent the computation of a **forward pass** of a data point $\mathbf{x}$ through the network as:

$$\hat{y} = g_3(g_2(g_1(\mathbf{x}\mathbf{W}_1^T + \mathbf{b}_1)\mathbf{W}_2^T + \mathbf{b}_2)\mathbf{W}_3^T + \mathbf{b}_3)$$

The weights in the weight matrix $\mathbf{W}$ and biases in the bias vector $\mathbf{b}$ are what we call **trainable parameters** of the model. When designing a neural network, we decide the number of hidden layers (the "depth" of the network) and the dimensionality of the hidden representations $\mathbf{h}$ (the "width" of the network). These architectural parameters decide the size of the network. The high representational power of **deep neural networks** comes from using dozens of hidden layers, each with thousands of trainable weights.

In addition to feedforward neural networks, multiple families of neural network architectures exist. For example, Convolutional Neural Networks are common in image processing; Recurrent Neural Networks allow for input and output sequences of arbitrary length; and Transformers apply the self-attention mechanism to associate data over longer distances, such as in text. Transformer networks are particularly relevant in this thesis, and is the basis of the BERT network discussed in Section 2.2.

2.1.2. Training Deep Neural Networks

In **supervised deep learning** the neural network learns to predict **ground-truth labels** from input data. Consequently, supervised learning requires a dataset where the true labels are known for all data points. In a **classification task**, the ground label of a data point is the correct class of that data point.

For example, in Mention Detection, each word is assigned one of three classes: the word is the first word of a mention (label “Beginning”); part of a mention, but not the first word (label “Inside”); or not in a mention (label “Outside”). For example, in the sentence “Abbey Road was released in 1969”, the word “Abbey” marks the beginning of a mention, “Road” is inside a mention, and the other the words are outside mentions.

Loss Function

In supervised deep learning, the loss function gives the model feedback on how close its predictions were to the ground-truth labels. The loss function J is a function of the model's prediction $\hat{y}$ on a given data point and the ground-truth label y of that data point, and returns a scalar number. This scalar number is the **loss** of the model on the data point.

The loss function is often a proxy for the actual performance of the model. For example, the loss function of a classification task may measure the distance between the prediction $\hat{y}$ and the label y (giving a floating point number in the range $[0, 1]$). For example, in binary classification, we may use the binary cross-entropy loss:

$$J(\hat{y}, y) = -y\log(\hat{y}) + (1-y)\log(1-\hat{y})$$

For performance, however, we are only interested in the model's accuracy in selecting the correct class (giving either 1 for correct or 0 for wrong). Many common performance measures do not qualify as loss functions because they are not differentiable. Consequently, they cannot be used for training a model.

Backpropagation and Optimization Strategies

We use data to train the model, and calculate the training signal as **the negative gradient of the loss function with regard to the model parameters** (weights and biases) with the given data. [1] Starting from the loss function, we calculate the

[1]The chain rule of calculus gives us the gradient as the partial derivative of the loss function with regards to the parameters.

gradient through the network in reverse order of the forward-pass. The network trains by changing the trainable weights in the direction of the negative gradient, causing the loss to decrease. When the loss decreases, the model improves at predicting the correct labels for the training data.

Given an example model with one hidden layer, we first train the weights and biases of the output layer, then the hidden layer, and finally the input layer. This method to find the gradient is called **backpropagation** (Rumelhart et al., 1986), and optimization following the negative gradient of a loss function is called **gradient descent**.

Calculating the gradient for **mini-batches** of data is faster than for the whole dataset, and allows the model to converge more quickly. Using **stochastic gradient descent**, we train on mini-batches with a fixed number of data points randomly sampled without replacement. A full iteration over the dataset is called an **epoch**.

We combine stochastic gradient descent with other optimization strategies to encourage faster convergence of the training procedure. Most importantly, we want to adapt the learning rate of the model for each trainable parameter. The **learning rate** decides the length of each training step, or how far we follow the gradient in the direction given by the current mini-batch of data.

Adam (Kingma and Ba, 2014) is a common optimization strategy to dynamically adapt the learning rate. Specifically, it accumulates an exponentially decreasing average of the squared gradient of each parameter. By multiplying the learning rate with the accumulated squared gradient, the model is able to learn faster for parameters that require large value changes, and slower for parameters that need small value changes.

Layer normalization (Ba et al., 2016) also aims to speed up the training. By normalizing the activations in a layer for each data sample, we achieve more stable gradients. Compared to other normalization strategies, layer normalization preserves the internal statistics of each sample. In turn, the input to the next layer in the network always follows a standard normal distribution.

Because we need to calculate the gradient through the network, the **operations in the network need to be differentiable**. The layers of the neural network consist of linear transformations and activation functions, and the training signal depends on the loss function. Consequently, the activation functions and the loss function are restricted to the class of differentiable functions. The linear transformations are always differentiable.

In this thesis, we use an end-to-end model for Entity Linking. An **end-to-end**

model trains with a joint loss function for all sub-tasks (for Entity Linking: Mention Detection and Entity Disambiguation). The training signal should not be interrupted by undifferentiable functions. As a result, the joint loss function is the source of the training signal for all parts of the model, and the model learns a shared representation for all tasks.

Regularization

In machine learning, we use disjoint datasets to train and evaluate the model. Firstly, we use a **training dataset** to calculate the training signal and train the model. However, we are mainly interested in how well the model generalizes to new data. Hence, we evaluate the model on a **test dataset** with data that does not appear in the training dataset. Additionally, we may use a third distinct **validation dataset** to evaluate the model during training, and to adapt the training procedure or the architecture of the model. These three distinct datasets are usually subsets of the same larger dataset. The data is often distributed as 80 % - 10 % - 10 % in, respectively, the training, validation and test sets.

During training, the model may improve its prediction of the training dataset while performance deteriorates on the validation and test datasets. This is what we call **overfitting**: by fitting too well to the training data, the model loses the ability to generalize to new data. Deep neural networks are particularly susceptible to this due to their high expressive power: they may be able to model the training data perfectly, and achieve a loss near zero.

Deep learning provides many **regularization strategies** to prevent overfitting. These strategies aim to restrict the expressive power of the neural network and thereby force the model to make useful generalizations during training.

Dropout is a common regularization strategy, where some of the weights in each layer are randomly deactivated in each training epoch. This prevents the network from becoming overly dependent on some weights in the network by forcing it to make predictions without the deactivated weights.

Hyperparameters

Hyperparameters are any parameters that influence the model or the outcome of the training procedure, but that are not trained by backpropagation. Both the model architecture and the training procedure have hyperparameters. The depth and width of the network are examples of **architectural hyperparameters**, while the learning rate is an example of a **training hyperparameter**. When evaluating and changing hyperparameters, we typically evaluate the model on the **validation dataset**.

2.2. BERT

The neural network architecture BERT (Devlin et al., 2019) is a large transformer-based model for natural language processing. BERT networks have been used to set new state-of-the-art results in multiple natural language processing tasks, including Entity Linking. We differentiate between *pretrained* BERT, which is trained on generic language understanding tasks, and *finetuned* BERT, which is further trained for specific tasks.

A BERT model creates a numerical representation of text. However, unlike standard language modeling methods, BERT produces **contextualized word embeddings**. In contextualized word embeddings, the vector used to represent a word depends on the other words in the text, thereby incorporating context into the word embedding.

The input data to BERT is close to raw text. BERT relies on a *WordPiece tokenizer* to parse words to tokens in a vocabulary of 30,000 word-piece tokens. A common word like "Sun" will be tokenized simply as "Sun". Less common words may be split in multiple tokens. For example, the word "tokenizer" becomes "token" and "##izer". We refer to the first token of a word as the "head token". The subsequent tokens are characterized as being part of the head token by the "##" prefix.

Figure 2 shows a schematic overview of the architecture of BERT, with three main modules: the input layer, a stack of encoders, and an output layer.

The BERT network takes sequences of tokenized text as input. Initially, BERT converts the input tokens to token embedding vectors. The vectors have the same dimensionality (length) throughout the network. These initial vectors are comparable to traditional word embedding methods, such as GloVe and Word2vec, which produce word embedding vectors without contextualization.

Next, the token embeddings go through a series of **transformer encoder** modules. The transformer encoders in BERT rely heavily on "self-attention" mechanisms, and follow the architecture proposed by Vaswani et al. (2017). The first encoder takes the initial vectors of the input layer as input, and the subsequent encoders take the output of the previous encoder as input.

Each encoder uses multiple parallel computations of self-attention (so-called self-attention "heads") to create a new representation of each token. For a given token vector, the **self-attention** is computed as a weighted dot product of the vector with

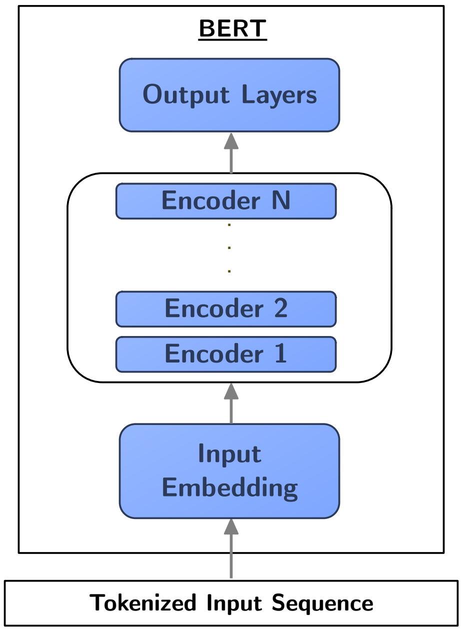


Figure 2.: Schematic view of the BERT architecture.

all vectors in the sequence. The weights are trained to extract relevant information from the context vectors. The attention of a token to its context tells us how that token is impacted by its context.

Figure 3 shows an example of how the token “Abbey” may attend to the context “Sun King appears on Abbey Road” in a given layer of the network. In this illustration, the strength of a connecting line shows how strongly the “Abbey” token attends to the context token. The subsequent representation of the “Abbey” token will depend more on the tokens with a higher attention-connection.

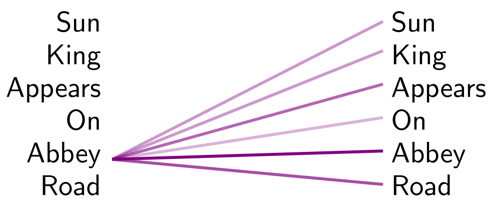


Figure 3.: An illustration of self-attention between a word and its context.

While passing through the sequence of transformer encoders, the token embeddings become increasingly contextualized (Ethayarajh, 2019). In fact, in the encoder transformers, the representation of a single token relies on all other tokens for context.

After the sequence of transformer encoders, the resulting token embeddings are passed through the remaining network to produce outputs that are specific to the task on which the network is trained. When finetuning a pretrained BERT network for a new task, we need to define new output layers. The output layers take the contextualized token vectors from BERT as input. While the output layers are trained from randomly initialized weights, the BERT module already has a general understanding of language. Consequently, a network based on a pretrained BERT model can quickly converge to good results.

During **pretraining**, BERT trains jointly on two tasks: it learns to fill in missing word-tokens in sentences through the Masked Language Model task, and it predicts whether one text sequence naturally follows another in a Next Sentence Prediction task. The former task requires predictions on individual tokens, while the latter task requires predictions on longer text sequences.

A BERT network is characterized by three architectural hyperparameters: the number of transformer encoders, the number of parallel self-attention computations in each encoder, and the length of the token embedding vectors passed between the encoders. The BERT instance we rely on in this work, dubbed *BERT base*, has twelve encoders, each with twelve attention computations, and a token embedding length of 768.

3. Related Work

This section gives an overview of existing research in Entity Disambiguation and Entity Linking methods, including the trends leading up to the current state-of-the-art in the field.

Most methods for Entity Linking treat the task of Mention Detection distinctly from Entity Disambiguation. However, more recent research has found a co-dependence between the two tasks and proposes models that perform both tasks end-to-end (Kolitsas et al., 2018; Martins et al., 2019). Frequently, these models aim to train a single model on a joint task of Mention Detection and Entity Disambiguation.

We will start this literature review with methods for (Named) Entity Disambiguation. In the Entity Disambiguation task, the disambiguation system is provided with ground-truth mention spans. Section 3.1 gives an overview of early works in the field, where graph-based methods dominated the state-of-the-art for a long time. Next, Section 3.2 presents the current state-of-the-art methods in Entity Disambiguation, using artificial neural networks.

Finally, in Section 3.3, we look at the more recent work in Entity Liking, where the model performs both Mention Detection and Entity Disambiguation. We have particular focus on models training on joint training objectives encompassing both Mention Detection and Entity Disambiguation.

The performance of a model on the Entity Disambiguation task is measured by the accuracy of the model's candidate assignments, and the results in Section 3.1 and Section 3.2 follow this format. On the other hand, Entity Linking has two sources of error: Mention Detection and Entity Disambiguation. Accordingly, we use the F1 score, defined as the harmonic mean of the precision and recall, for Entity Linking performance. In Section 3.3, we mainly look at the Micro F1 score, which is averaged over all mentions. Further details on the evaluation metrics we use in this chapter can be found in Section 5.1.

Most papers in the field use the AIDA-CoNLL dataset of Hoffart et al. (2011) as training data and testing benchmark. The results we list in this review are obtained from evaluating on the test set of this dataset. Section 5.2 gives more details on this dataset.

3.1. Entity Disambiguation with Coherence Graphs

One of the first attempts at Entity Disambiguation was on person names. Bunescu and Pasca (2006) rank a mention's candidates (given by entities in Wikipedia disambiguation pages) with a similarity score between features extracted from the context of the mention and the categories to which a candidate's Wikipedia article belongs. A Support-Vector Machine disambiguates each mention to a single candidate. This model considers each document mention independently of other entities in the document. Such a model is referred to as a **local model**.

Further improvements in the field came from Kulkarni et al. (2009) by modeling the coherence of all the entities linked in a document. Following the vocabulary of Ratinov et al. (2011), this is referred to as a **global method**. Like Bunescu and Pasca (2006), their Candidate Generation step uses a lookup table with aliases.

Introducing a landmark disambiguation model, Hoffart et al. (2011) combine multiple heuristics to improve on preceding methods. Namely, they use the prior probability of a mention referring to a candidate entity (extracted from Wikipedia hyperlink texts), the similarity between the context of the mention and a candidate entity (a local model), as well as the coherence of the selected candidate set for all mentions in a document (a global model). Before running the global coherence model, candidates that stand out with high prior probability and high similarity scores are assigned. The global assignment is cast as choosing a dense subgraph in a document graph of mentions and candidates.

The most widely used benchmark dataset to date, the *AIDA-CoNLL* dataset, was also introduced by Hoffart et al. (2011). They establish the first results for Entity Disambiguation on the dataset with an accuracy of 81.9 %. We will later rely on this dataset for training and evaluation and discuss it further in Section 5.2.

Improving the document coherence graph of previous work, Chisholm and Hachey (2015) complement the graph with a large corpus of web links. They follow the general approach of Hoffart et al. (2011) by first assigning entities with a high local score before using coherence to assign remaining mentions. Their new global model improves on the state-of-the-art with an accuracy of 88.7 %.

Further improvements from Pershina et al. (2015) also came from new global methods. They use a "Personal Page Rank" algorithm to improve the coherence of the global candidate graph. Their candidate graph contains candidate—candidate edges wherever one of the candidate entities' Wikipedia article links to the Wikipedia article of the other. Combined with heuristics on the local similarity score for each mention, they report an accuracy of 91.8 % on AIDA-CoNLL.

Inspired by new language models where words are embedded in a continuous vector space, Yamada et al. (2016) proposes embedding words and entities to the same vector space. The entity embeddings are combined with a graph model to perform global disambiguation of multiple document mentions jointly. They achieved a state-of-the-art at the time, with an accuracy on AIDA-CoNLL of 93.1 %. This model may be the last method to improve the state-of-the-art in Entity Disambiguation without using some artificial neural network component (Sevgili et al., 2020).

The embedding method of (Yamada et al., 2016) for words and entities has later become known as *Wikipedia2vec* and will be an essential module in the approach of this thesis.

3.2. Neural Entity Disambiguation

In recent years, deep neural network models have consistently been the state-of-the-art for Entity Disambiguation. Traditionally, networks for natural language processing have been recurrent models with Long Short-Term Memory (LSTM) units. However, more recent works in Entity Disambiguation have focused almost exclusively on large transformer networks such as BERT and have allowed further leaps on Entity Disambiguation benchmarks.

One of the first approaches using deep neural networks was by He et al. (2013). The network initially trains on an unsupervised denoising task using an auto-encoder architecture. The encoder thereby learns a general language model. Next, they finetune the encoder to find a maximum similarity between a document and entities that appear in the document. The entities are represented by their Wikipedia pages. They use a dataset of 40 million Wikipedia hyperlinks for the finetuning stage.

The results of He et al. (2013) suggest that local information is sufficient to reach a high performance on Entity Disambiguation without considering global coherence between entity sets. They point out that global information is used as a fall-back when the model is not sufficiently confident with local information. However, they found an increase in accuracy from 84.8 % to 85.6 % when combined with the best global method at the time.

Ganea and Hofmann (2017) proposes a new state-of-the-art system by successfully combining a neural network model for local information with a global disambiguation graph. Their neural network only receives the document as input and does not rely on prior information. In line with the advantages of deep learning, their model can implicitly learn to produce a document representation to solve the disambiguation task.

Their local model produces a contextualized embedding of the mention using an attention mechanism on the mention context. The model is one of the first examples of neural attention for Entity Disambiguation. Furthermore, using differentiable message-passing for the global model allows them to train the local and global models jointly and end-to-end. With an accuracy of 92.2 %, they improved on the previous best of Yamada et al. (2016).

They also introduce a more sophisticated Candidate Generation by combining prior probabilities from entity hyperlink statistics with alias searches from the YAGO Knowledge Base.

The current state-of-the-art in Entity Disambiguation is achieved by Yamada et al. (2019) using a model based on pretrained BERT. Inspired by the pretraining tasks of BERT (Devlin et al., 2019), they propose a denoising auto-encoder task on masked entities. Specifically, entities are masked at random, and the model needs to construct an embedding for the masked entity that allows it to predict the missing entity. They train their model on Wikipedia articles, using hyperlinks as labels.

Unlike previous works using global coherence graphs, Yamada et al. (2019) iteratively disambiguate entities in order of confidence while appending previously assigned entities to the input. After further fine-tuning the model on the AIDA-CoNLL training set, they report an accuracy of 95 %, establishing a new state-of-the-art performance.

3.3. Entity Linking

We define Entity Linking systems as any system that performs both Mention Detection and Entity Disambiguation. All the research articles in this section have reported results on Entity Linking.

Mention Detection and Entity Disambiguation have been treated mostly as separate problems even in the Entity Linking literature. Strong Entity Linking performance has come from using state-of-the-art Entity Recognition systems to detect mentions, followed by Candidate Generation, and finally novel Entity Disambiguation methods to pick candidates and filter over-sampled mentions. However, some recent works have shown that end-to-end deep learning can solve the two tasks jointly, and that it may be beneficial for both Mention Detection and Entity Disambiguation to do so.

The methods we describe here rely on deep neural networks to create contextual mention embeddings in a continuous vector space. Accordingly, Entity Linking methods align with the state-of-the-art Entity Disambiguation methods described in Section 3.2. However, with the added challenge of Mention Detection, we see a variation in the research in how the two tasks are combined. Approaches range from optimizing the traditional Entity Linking pipeline of *Mention Detection* $\rightarrow$ *Candidate Generation* $\rightarrow$ *Entity Disambiguation*, to circumventing the two former tasks by directly classifying all words to entities.

Performance in Entity Linking is not directly comparable to performance in Entity Disambiguation because the addition of Mention Detection gives a new source of errors. The combination of Mention Detection and Entity Disambiguation motivates using the F1 score rather than accuracy as a measurement of performance.

3.3.1. An End-to-End Model for Entity Linking

Arguably, Kolitsas et al. (2018) were the first to propose a joint model for Entity Linking. Given an input document, they over-generate potential mentions. Next, using the YAGO knowledge graph inherited from Hoffart et al. (2011) for Candidate Generation, they consider all spans that have at least one candidate. Finally, they use a bi-directional LSTM model to create vector embeddings for all the potential mention spans. The model is trained to maximize the vector-similarity between a mention embedding and its ground-truth candidate.

The similarity score between each mention embedding and their respective candidates is combined with other scores: firstly, the conditional probability of the candidate being referred to by the relevant mention text; and secondly, the similarity score from the attention model of Ganea and Hofmann (2017). This gives the local mention–candidate score. Finally, a global disambiguation network promotes stronger coherence by filtering the many over-sampled mentions and assigning a coherent entity set for the document.

After training and testing the model on the AIDA-CoNLL dataset, they achieved a Micro F1 score of 82.4, with 100 being the highest achievable score. When they train the model for only Entity Disambiguation and use a separate program for Mention Detection, they observe a drop in performance to 74.6 in Micro F1 score. This fall in performance indicates that the system indeed benefits from jointly modeling Mention Detection and Entity Disambiguation. Their Knowledge Base only contains around 500K entities, but is designed to contain virtually all entities that appear in the AIDA-CoNLL dataset. Notably, their model is only trained on the AIDA-CoNLL dataset.

3.3.2. The Potential of a Local Model for Entity Linking

Similarly, Martins et al. (2019) argue in favor of jointly training a model on Entity Recognition and Entity Disambiguation. Their model is trained on a multitask objective that covers both of these tasks. For comparison, they also train two specialist models on each respective task. When comparing the performance, they find that the joint model performs better on both tasks than the respective specialists.

Only local information is used in their bi-directional LSTMs model with attention. In other words, the model disambiguates each mention based on its context, and does not consider the other entities disambiguated in the document. Their results come close to that of Kolitsas et al. (2018), with a Micro F1 score of 81.9. It is also worth noting that the Entity Recognition task is more complex than Mention Detection, as the model also needs to classify the entity type of a mention.

3.3.3. A Simple Approach to End-to-End Entity Linking with BERT

As with Entity Disambiguation, the next wave of state-of-the-art Entity Linking models are based on BERT (Devlin et al., 2019). The first use of BERT for end-to-end Entity Linking in Broscheit (2019) uses a simplification of the Entity Linking task.

They cast Entity Linking as a simple classification task, where each word token is classified, and classes are the entities in the Knowledge Base. Additionally, there is the class of “no entity” for words that are not part of entities. They use a relatively small Knowledge Base of 500K entities. This method circumvents the need for Candidate Generation and unifies Mention Detection and Entity Disambiguation to a single task. The model classifies all document tokens at once, making this a local model.

Because of the large number of classes, the model needs to train on a large dataset. For pretraining, they train the model on Wikipedia hyperlinks in sentences from Wikipedia articles, before they finetune most of the network on the AIDA-CoNLL training set. Similar to Kolitsas et al. (2018), all entities that are in AIDA-CoNLL are added to the Knowledge Base. This addition results in 1000 extra entities that are outside of the top 500K Wikipedia entities. Training on one GPU, their model trains for 42 days.

By comparing three results from the ablation study of Broscheit (2019), we can learn more about how BERT works as a language model for Entity Linking. When training only on the AIDA-CoNLL training set (no pretraining on Wikipedia), the model achieves a Micro F1 score of 49.6. Despite the relatively low score, this shows that BERT has a potential to learn Entity Linking. However, this model cannot generalize to entities that it has not seen during training.

To familiarize the model with more entities, they pretrain with Wikipedia as a training set, before finetuning on AIDA-CoNLL. They compare two models: one where they only train the output classification layer, and one where they train the whole model, including BERT. The former achieves a Micro F1 score of 67.8, and the latter 79.3 on AIDA-CoNLL. The first model already shows a big increase from the model not trained on Wikipedia, which again suggests that BERT has a lot of prior knowledge useful for Entity Linking. The performance increase in the best model shows that BERT can learn additional entity knowledge with this pretraining task.

3.3.4. Entity Linking with Entity Embeddings for BERT

With a novel method to introduce entity knowledge to BERT, Poerner et al. (2020) takes a different approach to Entity Linking. To adapt entity vectors to BERT, Poerner et al. (2020) project the word and entity vectors of Wikipedia2vec (Yamada et al., 2016) to the domain of BERT input encodings using the word vectors that are in common for the two vector domains as guides. By introducing entity knowledge on BERT's premises, BERT requires no further training to learn about entities. They dub their entity-enhanced model *E-BERT*.

E-BERT is not trained end-to-end on Entity Linking because they rely on *KnowBert* (Peters et al., 2019) for Mention Detection and Candidate Generation. However, they report their findings on the Entity Linking rather than Entity Disambiguation.

To use E-BERT for Entity Disambiguation, they first use a local model to find a context embedding of each mention. Next, they replace the most probable mentions of the document with the predicted candidate, represented by its projected Wikipedia2vec vector. In the next iteration over the document, these entity vectors provide additional information to the context of the unassigned mentions for the next iteration. Reminiscent of Hoffart et al. (2011), this is the heuristic of first assigning the most obvious entities from the local context, followed by a better informed global assignment.

Raising the bar from Kolitsas et al. (2018), the result is a Micro F1 score of 85.0 on the AIDA-CoNLL test set.

3.3.5. End-to-End Entity Linking with a Joint Task

Broscheit (2019) is not alone in proposing a model without Candidate Generation and other heuristics. With a more flexible approach, Chen et al. (2019) propose an end-to-end Entity Linking model with BERT, where Candidate Generation is optional.

Unlike Broscheit (2019), they keep the distinction between Mention Detection and Entity Disambiguation. The model produces two outputs for each word token: one Mention Detection prediction and one entity embedding of each token. The training objective allows the network to learn a projection from the embeddings produced by BERT to entity Wikipedia2vec vectors from Yamada et al. (2016). A token belonging to an entity mention should be projected to the correct entity vector from Wikipedia2vec. Notably, this model does not pretrain on a large dataset, but rather trains directly on the target AIDA-CoNLL dataset. The model looks at a document only once to classify all its entities, making this another only-local Entity Linking approach.

This approach is discussed in detail in Section 4.1, as this is the approach we base ourselves on in this thesis.

When treating the entire Knowledge Base of 1M entities as potential candidates, Chen et al. (2019) report a Micro F1 score of 70.7. Using the YAGO knowledge graph of Hoffart et al. (2011) for Candidate Generation, the performance vastly improves to a state-of-the-art F1 score of 87.7. According to Sevgili et al. (2020), these is the best Entity Linking performance reported on the ADIA-CoNLL Test dataset. However, as pointed out by Poerner et al. (2020), these results may not be directly comparable to previous works due to a potential difference in Knowledge Bases. Specifically, Chen et al. (2019) do not explicitly include all entities that appear in the AIDA-CoNLL dataset in their Knowledge Base, nor report on the coverage of their Knowledge Base on the dataset.

4. Approach

In this thesis, we attempt to reproduce the results of Chen et al. (2019). This chapter outlines our general approach to Entity Linking. Section 4.1 describes our Entity Linking model for joint Mention Detection and Entity Disambiguation. Section 4.1 also features an analysis of the loss function and general aspects of our training procedure. Section 4.2 gives an overview of how we construct our Knowledge Base. Section 4.3 details our Candidate Generation method. Finally, we describe how to use our trained model for inference in Section 4.4. Some differences between our model and the model proposed by Chen et al. (2019) are further motivated by empirical results in Chapter 5.

Our Python source code is available at `https://github.com/amundfr/el-bert`.

4.1. Joint Mention Detection and Entity Disambiguation

Our model is based on a pretrained instance of BERT-base-uncased from Huggingface[1]. Additionally, we define two output heads: one for Mention Detection classification, and one to produce entity embeddings for Entity Disambiguation. Both heads take the final hidden state of the BERT model as input.

Figure 4 shows how data flows through the model. First, the tokenizer prepares an input document of raw text for the model. Next, the model produces a contextualized representation of each token, which finally passes to the output heads.

[1]`https://huggingface.co/bert-base-uncased`

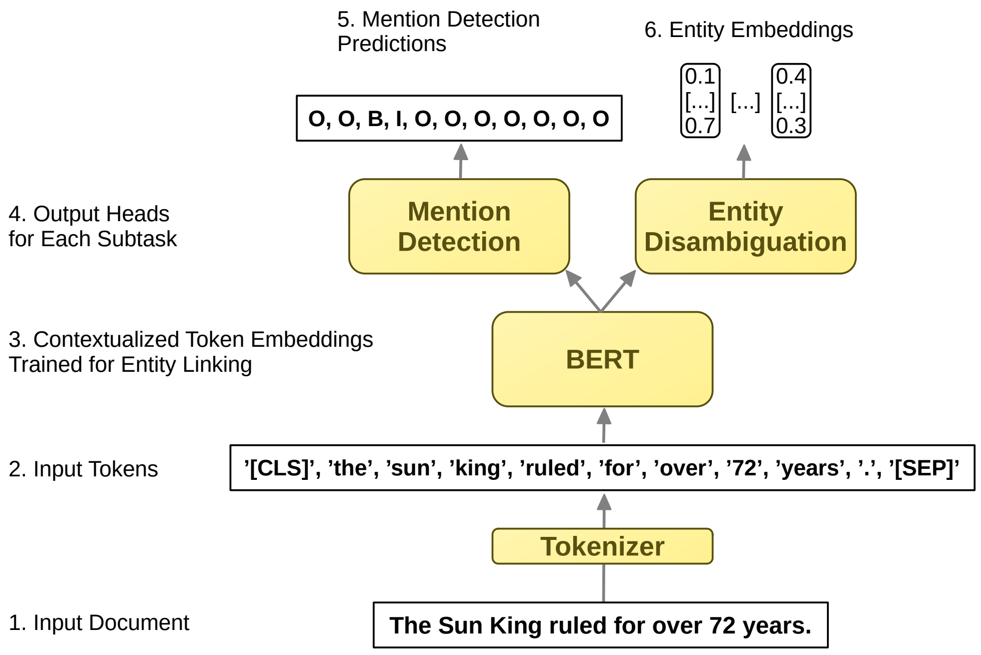

5. Mention Detection Predictions
6. Entity Embeddings
O, O, B, I, O, O, O, O, O, O, O
0.1 [...] 0.7
[...]
0.4 [...] 0.3
4. Output Heads for Each Subtask
Mention Detection
Entity Disambiguation
3. Contextualized Token Embeddings Trained for Entity Linking
BERT
2. Input Tokens
'[CLS]', 'the', 'sun', 'king', 'ruled', 'for', 'over', '72', 'years', '.', '[SEP]'
Tokenizer
1. Input Document
The Sun King ruled for over 72 years.

Figure 4.: General model architecture and example document predictions.

4.1.1. Known Differences

We are aware of the following differences to Chen et al. (2019):

- using uncased instead of cased version of BERT;
- using hidden layers in the output heads;
- the Candidate Generation module;
- the Knowledge Base;
- the weight parameter of the loss function.

We will detail the Knowledge Base and the Candidate Generation in this chapter. The remaining differences are motivated by empirical results. These differences are described in Chapter 5.

4.1.2. Output Heads

The Mention Detection head is a feed-forward neural network, which applies the same transformation to each token embedding. For each token, it outputs a prediction vector of three scalar predictions $\mathbf{p}_{md} \in \mathbb{R}^3$; each scalar is a prediction for one of the three possible Mention Detection classes (the token being "Inside", "Outside" or "Beginning" of a mention, where "Beginning" is the first token of the mention). The class with the highest prediction is considered the predicted class for that token. We do not use an activation function for the Mention Detection predictions.

When using the WordPiece tokenizer, a word can be split in multiple tokens (e.g. the word "tokenizing" into tokens "tokeniz" and "##ing"). The first token for each word is called the *head token*. We only look at labels and predictions of head tokens, and not the following tokens.

The following example shows the correct Mention Detection labels for an example sentence (note that the the *non-head token* "##s" does not need a label):

Sun	King	appear	##s	on	Abbey	Road
Beginning	Inside	Outside	None	Outside	Beginning	Inside

The Mention Detection head takes a contextualized token embedding $\mathbf{h}_b \in \mathbb{R}^d_b$ from the BERT module, where d_b is the same hidden vector length used throughout BERT. The Mention Detection head first applies a linear transformation to get the hidden activation $\mathbf{z}_{md} \in \mathbb{R}^{d_b}$. For this computation, it uses the weight matrix $\mathbf{W}_h^{MD} \in \mathbb{R}^{d_b \times d_b}$ and the bias vector $\mathbf{b}_h^{MD} \in \mathbb{R}^{d_b}$:

$$\mathbf{z}_{md} = \mathbf{h}_b \mathbf{W}_h^{MD} + \mathbf{b}_h^{MD}$$

Next, it applies the GeLU activation function and a layer normalization (see Section 2.1.2) to give the hidden representation $\mathbf{h}_{md}$:

$$\mathbf{h}_{md} = \text{LayerNorm}(\text{GeLU}(\mathbf{z}_{md}))$$

Finally, the hidden token representation $\mathbf{h}_{md}$ passes through the output layer. The output layer is again a linear transformation with a weight matrix $\mathbf{W}_o^{MD} \in \mathbb{R}^{d_b \times 3}$ and a bias vector $\mathbf{b}_o^{MD} \in \mathbb{R}^3$. This produces a Mention Detection prediction $\mathbf{p}_{md} \in \mathbb{R}^3$:

$$\mathbf{p}_{md} = \mathbf{h}_{md} \mathbf{W}_o^{MD} + \mathbf{b}_o^{MD}$$

With hidden vector size $d_b = 768$, this output head has $768 \times 768 + 768 + 768 \times 3 + 3 = 592,899$ trainable weights.

The Entity Disambiguation head is also a feed-forward neural network working on each token embedding in turn. It transforms the entity embeddings $\mathbf{h}_b$ from BERT to the target domain of Wikipedia2vec entities with dimensionality d_w. The goal is to project a token embedding (specifically the token corresponding to the beginning of a mention) to the embedding of the corresponding Wikipedia2vec entity.

The output head first applies a linear transformation with weight matrix $\mathbf{W}_h^{ED} \in \mathbb{R}^{d_b \times d_b}$ and adds a bias vector $\mathbf{b}_h^{ED} \in \mathbb{R}^{d_b}$:

$$\mathbf{z}_{ed} = \mathbf{h}_b \mathbf{W}_h^{ED} + \mathbf{b}_h^{ED}$$

After activating with the GeLU activation function and normalization, we get the hidden representation $\mathbf{h}_{ed} \in \mathbb{R}^{d_w}$:

$$\mathbf{h}_{ed} = \text{LayerNorm}(\text{GeLU}(\mathbf{z}_{ed}))$$

Next, the output layer transforms the hidden representation $\mathbf{h}_{ed}$ to a predicted embedding vector with weight matrix $\mathbf{W}_o^{ED} \in \mathbb{R}^{d_b \times d_w}$ and bias vector $\mathbf{b}_o^{ED} \in \mathbb{R}^{d_w}$. Additionally, this head uses the hyperbolic tangent ("tanh") activation function to give the final entity embedding $\hat{\mathbf{e}}_{ed} \in \mathbb{R}^{d_w}$:

$$\hat{\mathbf{e}}_{ed} = \tanh(\mathbf{h}_{ed} \mathbf{W}_o^{ED} + \mathbf{b}_o^{ED})$$

The BERT embedding token $\mathbf{h}_b$ and the hidden representation $\mathbf{h}_{ed}$ have $d_b = 768$ dimensions, while the Wikipedia2vec embeddings in the target domain have $d_W = 100$ dimensions. The result is $768 \times 768 + 768 + 768 \times 100 + 100 = 667,492$ trainable weights for the Entity Disambiguation output head.

Our model contrasts with Chen et al. (2019) with hidden layer in the output heads. We motivate this with empirical results in Table 8.

In addition to the trainable weights in the output heads, the **BERT base model** has around 109 million trainable weights. In fact, 98.86 % of the trainable weights of our model are found in BERT. Table 1 shows a break-down of the number of trainable weights in each module of the network.

Network Module	# **Weights**
BERT	108,853,248
Mention Detection output head	592,899
Entity Embeddings output head	667,492
Total	110,113,639

Table 1.: Number of trainable weights in each module of the network, including bias weights.

4.1.3. Loss Function

The loss function needs to account for both the Mention Detection errors and the Entity Disambiguation embedding errors. Consequently, our joint Entity Linking loss function J_{el} is a weighted sum of two losses:

$$J_{el} = \lambda J_{md} + (1 - \lambda) J_{ed}$$

Figure 5 shows a graphical representation of the inputs to the loss function. For an analysis of how this loss function fulfills the criteria for end-to-end training, see Appendix A.

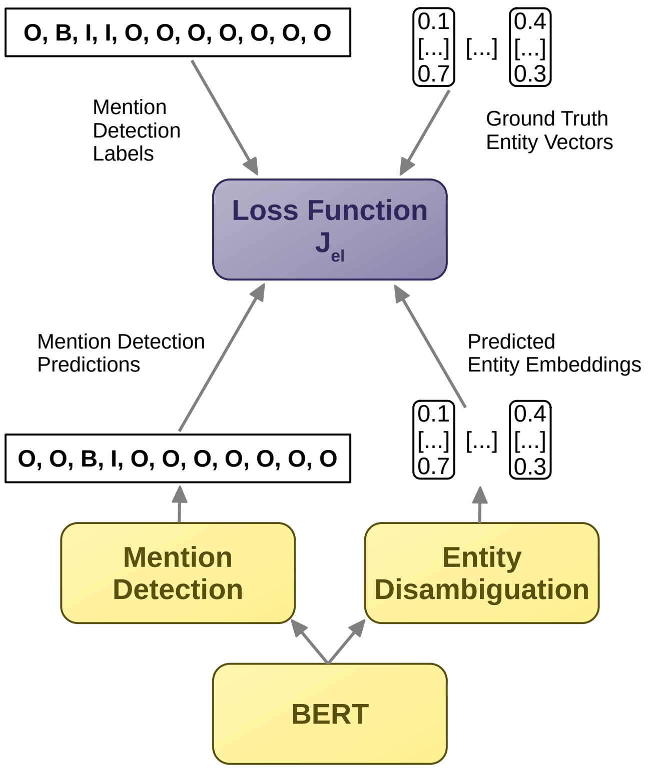


Figure 5.: The loss function in relation to the model outputs.

Mention Detection Loss

The Mention Detection loss J_{md} is a cross-entropy loss. It takes the Mention Detection predictions $\mathbf{p}_{md} \in \mathbb{R}^3$ (one scalar prediction for each class) and the ground-truth mention label $gt_{md} \in \{\text{Inside} = 1,\ \text{Outside} = 2,\ \text{Beginning} = 3\}$ as input:

$$\begin{aligned} J_{md}(\mathbf{p}_{md}, gt_{md}) &= -\ln\left(\text{softmax}\left(\mathbf{p}_{md}\right)_{gt_{md}}\right) \\ &= -\ln\left(\frac{\exp\left(\mathbf{p}_{md}\left[gt_{md}\right]\right)}{\sum_{i=1}^{3}\exp\left(\mathbf{p}_{md}\left[i\right]\right)}\right) \end{aligned}$$

Because there is only one correct class, the cross-entropy is simply the negative logarithm of the prediction of the ground-truth class. The cross-entropy takes a probability in the range of $[0, 1]$. The loss function incorporates the softmax function to transform the model output to a probability distribution.

Entity Disambiguation Loss

Next, the Entity Disambiguation loss J_{ed} is a cosine difference loss. It takes the entity embeddings $\hat{\mathbf{e}}_{ed}$ from the Entity Disambiguation head and the Wikipedia2vec vectors $\mathbf{e}_w$ of the ground-truth entities:

$$J_{ed}(\hat{\mathbf{e}}_{ed}, \mathbf{e}_w) = 1 - \cos(\theta) = 1 - \frac{\hat{\mathbf{e}}_{ed} \cdot \mathbf{e}_w}{\|\hat{\mathbf{e}}_{ed}\| \, \|\mathbf{e}_w\|}$$

The normalized dot product is the cosine of the angle θ between the two input vectors. This dot product represents the similarity between the two vectors. If the two vectors are the same (maximum similarity), the angle between the two vectors is 0, and the similarity score is $cos(0) = 1$. To turn the similarity into a loss function, we invert the similarity to a difference with the operation $1 -$ similarity. The final result is analogous to a sinusoidal difference.

Weighting the Loss

The Mention Detection loss is evaluated on all words in the document, whether or not they are part of a mention. On the other hand, the Entity Disambiguation loss is only evaluated once for each mention in a document. Because the Entity Disambiguation loss is sparser than the Mention Detection loss, we compensate by suppressing the Mention detection loss by a factor λ.

We compare the recommended setting of $\lambda = 0.1$ from Chen et al. (2019) with three alternatives in Table 10, and we find the best performance with $\lambda = 0.01$.

4.1.4. General Training Procedure

Our training procedure is similar to Chen et al. (2019). Like Chen et al. (2019), we use the Adam optimizer with the default parameters proposed by Kingma and Ba (2014): $\beta_1 = 0.9$, $\beta_2 = 0.999$.

For experiments with training hyperparameters such as mini-batch size, initial learning rate, and epochs, refer to Section 5.4.2.

Mentions Without Entity Annotation

For various reasons, a dataset may have mentions annotated with mention labels ("Inside", "Outside", "Beginning") but no corresponding entity label. In that case, we treat the mentions as labeled when calculating the MD loss but ignore it in the ED loss. In practice, that means the model learns to identify mentions regardless of whether they are in the Knowledge Base.

4.1.5. Document Pre-processing

We use the default WordPiece tokenizers described in Section 2.2 to prepare a document for the model. The BERT models we use as a basis have corresponding WordPiece tokenizers for cased and uncased text respectively. Our best-performing models use the *BERT-base-uncased* model and the corresponding uncased tokenizer with a vocabulary of around thirty-two thousand word pieces. When training a *BERT-base-cased* model, we used the case sensitive tokenizer.

The maximum sequence length of BERT is 512 tokens, including a [CLS] and a [SEP] token. That leaves space for 510 WordPiece tokens from the input document. Accordingly, we split documents with more than 510 tokens in multiple sequences.

Figure 6 illustrates three strategies to split documents. In Strategy A, there is no overlap between sequences from the same document. This strategy requires padding of the final sequence. Strategy B places an overlap between the last two sequences. Finally, Strategy C split the documents by distributing the overlap between all sequences. Strategy B and C do not require padding, assuring maximum context in each sequence.

We reason that Strategy C best distributes the context from the overlaps, and this is the strategy we use. In practice, we see negligible differences in performance between the strategies.

The tokens that are in the overlap between two sequences necessarily have predictions from both sequences. For tokens with two predictions, we choose the Mention

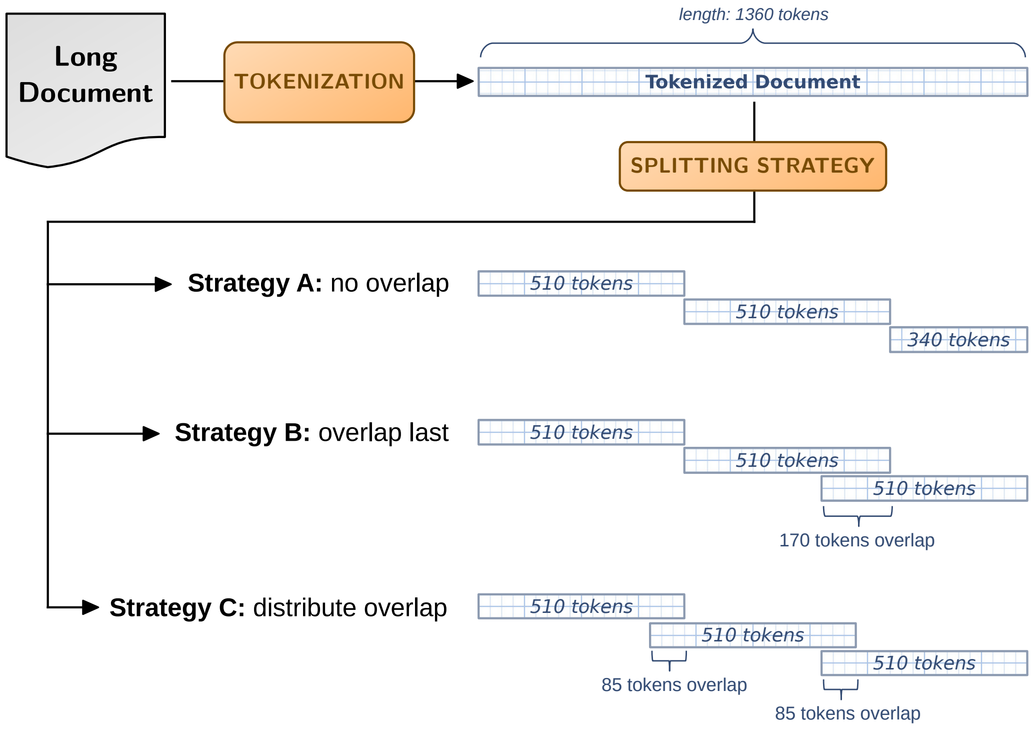


Figure 6.: Illustration of three strategies to split long documents into multiple sequences.

Detection classification with the highest predicted probability. We also select the Entity Disambiguation embedding from the same sequence. We reason that the sequence which produces the highest magnitude Mention Detection prediction has the best context and, consequently, will have produced the best Entity Disambiguation embedding.

4.2. Knowledge Base

A Knowledge Base is a collection of unique entities with unique identifiers. We define our Knowledge Base around Wikipedia2vec: entities that have a vector in our instance of pretrained Wikipedia2vec[2] are considered entities of interest. This may filter out some entities from the Knowledge Base that show up in the dataset. We use the pretrained Wikipedia2vec trained on English Wikipedia from 2018 with vector dimensionality 100.

[2]Provided here: `https://wikipedia2vec.github.io/wikipedia2vec/pretrained/`

The Wikipedia2vec vectors are the targets of the Entity Disambiguation head. These vectors are our search domain and the Entity Disambiguation embeddings are the queries.

For details on how we use the Knowledge Base for search during inference, refer to Section 4.4. We evaluate the Knowledge Base in Section 5.2.

4.3. Candidate Generation

The Entity Linking system permits the use of Candidate Generation during evaluation. It is important to note that the model does not rely on Candidate Generation in any way during training and may also be evaluated without Candidate Generation.

We use a mapping from entity aliases to entities in the Knowledge Base to generate our candidate sets. Given a mention text m, we get the set of candidate entities E_c for which m is an alias. The resulting candidate set does not depend on the context of the mention.

We use the candidate sets of Ganea and Hofmann (2017), who have collected candidates from multiple sources. They collect aliases and candidates from hyperlinks in Wikipedia articles and a web corpus inherited from Spitkovsky and Chang (2012) and supplement with candidates from the YAGO knowledge graph.

We process the aliases of Ganea and Hofmann (2017) by changing them to lower case and removing accents. This may merge some candidate sets, as compared to using the aliases with accents and casing. Furthermore, we do not restrict the number of candidates for a mention or use the prior probability of a mention referring to a given candidate as provided by Ganea and Hofmann (2017). This is in line with Chen et al. (2019). For an evaluation of the Candidate Generation module, see Section 5.3.2.

When Candidate Generation fails and there are no candidates for a given mention text, we fall back to default behavior without Candidate Generation. That is, we treat all Knowledge Base entities as potential candidates.

Chen et al. (2019) uses the YAGO knowledge graph introduced by Hoffart et al. (2011) when evaluating with Candidate Generation. Our Candidate Generation module inherited from Ganea and Hofmann (2017) includes YAGO candidates. This module has also been used for Entity Linking by Kolitsas et al. (2018).

4.4. Model Inference

We can use the trained models for inference. In particular, we are interested in evaluating the model's performance on documents it has not seen during training.

First, we resolve overlaps between sequences from the same document as described in Section 4.1.5. Next, the predicted Mention Detection classes decide the mention spans in the document. Tokens with predicted class "Beginning" designate the start of mentions. All tokens with predicted class "Inside" that directly follow are considered part of the same mention. In the rare case that an "Inside" prediction does not follow a "Beginning" prediction, we consider the token as the start of a mention. Any WordPiece tokens that are not head tokens (because a word is tokenized to multiple tokens) are assigned the prediction of their head token.

In the Entity Disambiguation stage, we treat the mentions independently.[3] When using Candidate Generation, we generate a set of candidates for each mention. Without Candidate Generation, we consider all Knowledge Base entities as candidates for each mention.

For each mention, we rank the set of candidates E_c by a similarity score. Unless we use Candidate Generation, the candidate set contains all entities in the Knowledge Base. In the similarity computation, the mention is represented by the model's Entity Disambiguation embedding on the first token in the mention (the "Beginning" token). Each candidate is represented by their respective Wikipedia2vec vectors. The similarity score between a candidate vector c and a token embedding e is a normalized dot product:

$$sim(c, e) = \frac{e \cdot c}{\|e\|_2 \, \|c\|_2}$$

where $\|v\|_2$ is the Euclidean norm of a vector v.

Finally, we assign the candidate with the highest similarity score to the mention in question.

[3]This makes our model a local-only model, as it does not consider the global coherence of the assigned candidates in a document.

5. Experiments

We train and evaluate our models on the AIDA-CoNLL benchmark dataset to compare Entity Linking methods on equal ground to previous models. In Section 5.1, we will describe the evaluation criteria we use for Entity Linking. In Section 5.2, we present the two datasets we use in our experiments. Next, in Section 5.3, we evaluate the Knowledge Base and Candidate Generation system through these datasets. Finally, in Section 5.4, we derive some hyperparameter settings for the model architecture and the training procedure.

5.1. Evaluation Criterion

The evaluation criteria for Entity Linking need to account for all sources of error; that is, Mention Detection and Entity Disambiguation. For Mention Detection, a predicted mention span is a **True Positive** prediction if there is a corresponding ground-truth mention and a **False Positive** prediction otherwise. In addition, we may have **False Negative** ground-truth mentions in the document that the model does not detect.[1]

For Entity Disambiguation, we require a correctly predicted mention. For Entity Disambiguation alone, we can measure performance with the accuracy of correct predictions. We combine this with the Mention Detection errors by defining correct entity assignments on True Positive mentions as True Positive predictions. Next, incorrect entity assignments on True Positive mentions are both False Negative and False Positive predictions.

Using these categories of correct and incorrect predictions, we derive the following performance metrics for Entity Linking:

- **Precision**: the ratio of True Positive predictions to number of predictions:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

[1]True Negative predictions are not relevant for model performance.

- **Recall**: the ratio of True Positive predictions to number of ground-truth mentions:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

- **F1 score**: the harmonic mean of the precision and recall:

$$\text{F1 score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The **precision** shows the likelihood that a model predicts correctly when it makes a prediction. The **recall** tells us the likelihood of a ground-truth mentions being correctly predicted by the model. Finally, the **F1 score** is of particular interest, as it scores performance using both precision and recall. To achieve a high F1 score, the Entity Linking system must strike a balance between predicting correctly (precision) and finding all entities in a document (recall).

We use the F1 score as the primary performance measure of our models. In particular, we look at the *Micro F1 score*, which denotes the average F1 score over all mentions. [2]

We only report performance on mentions of entities that are in our Knowledge Base. Consequently, we ignore mentions of ground truth labels that are not in the Knowledge Base, and we ignore predictions on these mentions. We treat mentions without ground truth labels in the same way.

5.2. Datasets

We present two datasets that we use for training the model. The AIDA-CoNLL dataset described in Section 5.2.1 is the primary dataset on which we train and evaluate models. The Wikipedia Articles dataset described in Section 5.2.2 serves as complimentary training data.

5.2.1. AIDA-CoNLL

The AIDA-CoNLL dataset from Hoffart et al. (2011) serves as the standard benchmark dataset for Entity Disambiguation and Entity Linking. We rely on this dataset for training and evaluation. The dataset has 1393 documents, distributed to a training dataset, a validation dataset, and a test dataset. Table 2 shows the number of

[2] The Micro F1 score treats the dataset as if it was one document. In contrast, the *Macro F1 score* calculates the average F1 score over documents.

documents, unique Wikipedia entities and mentions with entity annotations in the respective subsets of the dataset.

Dataset	**Documents**	**Mentions**	**Annotated Mentions**	**Unique Entities**
Train	946	23,396	18,541	4,084
Validation	216	5,917	4,791	1,644
Test	231	5,616	4,485	1,536
Total	1,393	34,929	27,817	5,593

Table 2.: Number of documents, mentions, mentions annotated with Wikipedia entities and unique mentioned entities in the AIDA-CoNLL datasets.

As a result of our method of splitting documents described in Section 4.1.5, we get a total of 1562 sequences from the 1393 AIDA-CoNLL documents. Table 3 shows the distribution of document splits per subset of the dataset.

		Sequences				
Dataset	**Documents**	**5**	**4**	**3**	**2**	**1**
Train	946	1	1	7	88	849
Validation	216	1	0	3	30	182
Test	231	0	0	0	20	211
Total	1393	2	1	10	138	1242

Table 3.: Count of how many AIDA-CoNLL documents are split in the respective number of sequences.

5.2.2. Wikipedia Articles Dataset

We experimented with pretraining on a dataset of 50.000 annotated Wikipedia articles before finetuning on the AIDA-CoNLL dataset. We split the dataset in a training dataset, a validation dataset, and a test dataset. Table 4 shows the breakdown of documents and mentions in each of the subsets.

Of the 187,826 unique entities in the Wikipedia Articles Train dataset, 2150 entities also appear in the AIDA-CoNLL training set. Consequently, 1934 entities in AIDA-CoNLL Train do not appear in the Wikipedia Articles training dataset.

Dataset	Documents	Mentions	Annotated Mentions	Unique Entities
Train	48,500	595,458	474,008	187,826
Validation	500	5,599	4,432	3,538
Test	1,000	12,562	10,086	7,543
Total	50,000	613,619	488,526	191,838

Table 4.: Number of documents, mentions, mentions annotated with Wikipedia entities and unique mentioned entities in the Wikipedia Articles dataset.

5.3. Module Evaluation

In this section, we evaluate key modules of our system on the AIDA-CoNLL dataset. First, we assess the Knowledge Base in Section 5.3.1. Next, we review the Candidate Generation system in Section 5.3.2.

5.3.1. Knowledge Bases

As described in Section 4.2, each Knowledge Base entity requires a Wikipedia2vec vector. The pretrained version of Wikipedia2vec has 2,592,608 Wikipedia entity vectors. When evaluating **with Candidate Generation**, we include **all Wikipedia2vec entities** in our Knowledge Base.

However, when evaluating the model **without Candidate Generation**, we use a **smaller Knowledge Base**. Because each predicted entity embedding is compared to all entity vectors in the Knowledge Base, using a smaller Knowledge Base significantly reduces the evaluation time.

To construct the smaller Knowledge Base, we follow Chen et al. (2019) and make a selection of the most popular entities. To decide the popularity of entities, we count the number of times other Wikipedia articles link to a given entity. By setting the minimum number of incoming links to 15, we get 1,081,226 entities. Similarly, Chen et al. (2019) report using a knowledge base with one million entities based on the same criterion of popularity.

Including AIDA-CoNLL Entities

Kolitsas et al. (2018); Broscheit (2019); Poerner et al. (2020) explicitly include all entities that appear in the AIDA-CoNLL dataset in their Knowledge Bases. As Poerner et al. (2020) point out, this assumes that we know which entities are in the test and validation datasets and may give an unfair advantage.

Regardless, we follow the same approach for better comparability with the mentioned methods. As a result, we include an additional 558 entities in the smaller Knowledge Base that appear in the AIDA-CoNLL dataset but whose popularity scores were too low. The resulting Knowledge Base covers 95.40 % of the unique entities in the AIDA-CoNLL dataset and 98.33 % of the mentions. The entities missing from the Knowledge Base have no Wikipedia2vec vectors.

Evaluating a model on AIDA-CoNLL Validation takes 60 minutes when using all 1,081,784 Knowledge Base entities as candidates. This corresponds to 0.75 seconds per mention. When using Candidate Generation and 2,592,608 entities, evaluation takes around 30 seconds. Some characteristics of the Knowledge Base are summarized in Table 5.

	No Candidate Generation	**Candidate Generation**
Entities	1,081,784	2,592,608
Gold Recall on AIDA-CoNLL entities	95.40 %	95.40 %
Gold Recall on AIDA-CoNLL mentions	98.33 %	98.33 %
Evaluation time* (minutes)	60	0.5
Evaluation time per mention* (seconds)	0.75	0.007

Table 5.: Key characteristics of the Knowledge Base with and without using Candidate Generation. *Evaluation time measured on AIDA-CoNLL Validation.

We define entities that appear in a Knowledge Base as "entities of interest" and ignore entities that are not in the Knowledge Base during evaluation. This type of evaluation is referred to as "In-Knowledge-Base evaluation" and is common in the Entity Linking literature.[3] This excludes 254 unique AIDA-CoNLL dataset entities with a total of 461 mentions (80 mentions in Validation, 106 in Test, and 275 in Train).

[3]The alternative is to treat entities that are not in the Knowledge Base as Entity Disambiguation errors.

5.3.2. Candidate Generation

Evaluation with Candidate Generation is significantly faster than searching the entire Knowledge Base because the generated candidate sets never have more than a few hundred candidates. Consequently, we do not need to limit the size of the Knowledge Base to speed up evaluation, and we evaluate the Candidate Generation approach described in Section 4.3 with all 2,592,608 Wikipedia2vec entities in the Knowledge Base. Regardless, the Knowledge Base has the same coverage of AIDA-CoNLL mentions and entities. Table 6 shows some statistics of the Candidate Generation module when using this Knowledge Base.

Total Aliases	11,395,243
Knowledge Base entities with at least one alias	2,105,095
Knowledge Base entities without an alias	487,513

Table 6.: Candidate Generation statistics for entities in the Knowledge Base.

Importantly, we evaluate how the Candidate Generation system performs on the target AIDA-CoNLL dataset. For this evaluation, we only consider mentions of ground-truth entities that **are in the Knowledge Base** (see Table 5).

The *gold recall* is a measure of the performance of a Candidate Generation system. We measure the gold recall as the proportion of mentions in a dataset for which the ground-truth entity is one of the candidates. Assuming the model finds all the correct mentions, the gold recall is the ceiling for recall when using Candidate Generation. Table 7 lists the gold recall for the different AIDA-CoNLL datasets.

	Gold Recall	
Dataset	**Unique Entities**	**Mentions**
AIDA-CoNLL Train	98.75 %	98.92 %
AIDA-CoNLL Validation	99.24 %	99.05 %
AIDA-CoNLL Test	98.71 %	99.11 %
AIDA-CoNLL	98.84 %	98.97 %

Table 7.: Candidate Generation gold recall on the AIDA-CoNLL dataset for entities in the Knowledge Base.

The AIDA-CoNLL entities have an average of 144 aliases. Each alias is a potential mention text to refer to an entity. The ground-truth entity is the only candidate for 18.4 % and 17.0 % of mentions in the AIDA-CoNLL Validation and AIDA-CoNLL Test sets. If the model detects these mentions correctly, it will predict the correct entity regardless of the predicted Entity Detection embedding. In AIDA-CoNLL Validation, there are on average 53 candidates per mention, and in AIDA-CoNLL Test, there are on average 56 candidates per mention.

When the Candidate Generation finds no candidates for a given mention text, we fall back to using all Knowledge Base entities as candidates. There are only four mentions with no candidates in AIDA-CoNLL Validation and seven such mentions in AIDA-CoNLL Test. In theory, this raises the gold recall on mentions to 99.13 % on AIDA-CoNLL Validation and 99.27 % on AIDA-CoNLL Test. In practice, however, the ground-truth entity only appears as a candidate if the model has predicted a valid Entity Disambiguation vector for that entity.

5.4. Experiments

This section presents the empirical results motivating our hyperparameter settings. The experiments in Section 5.4.1 motivate modifications to the model architecture, and, in Section 5.4.2, we arrive at parameters for the training procedure. Finally, we look at the impact of pretraining on the Wikipedia Articles dataset in Section 5.4.3.

We leave the AIDA-CoNLL Test dataset for final evaluation in Section 6.1 and evaluate on the AIDA-CoNLL Validation dataset. The Entity Linking performance results without Candidate Generation reveal the contrast between models best. However, we report Entity Linking results both with and without Candidate Generation and on Mention Detection.

5.4.1. Model Architecture

We propose some modifications to the model architecture of Chen et al. (2019) motivated by the empirical results presented in this section. We trained models for 30 epochs to evaluate these architectures, compared to 180 for the final model.

Extra Hidden Output Layer

The word tokens have a shared representation for Mention Detection and Entity Disambiguation through the BERT module of the network. At the output heads, these representations diverge and the output heads need to transform the BERT embeddings for the respective tasks. We aim to evaluate whether the model proposed by Chen et al. (2019) with only one layer in the output heads has sufficient expressive power to make this transformation. To that end, we compare their model to a model with additional hidden layers in the output heads (the model described in Section 4.1).

With only one layer in each output head, the Mention Detection head has $768 \cdot 3 + 3 = 2,307$ trainable weights, while the Entity Disambiguation head has $768 \cdot 100 + 100 = 76,900$ trainable weights. The hidden layers in our model give an additional $768 \times 768 + 768 = 590,592$ trainable weights in each output head.

Table 8 summarizes the results of the two models. Evaluating without Candidate Generation, the model with extra hidden layers achieves an F1 score 3.14 points higher than the model with single-layer output heads. When evaluating with Candidate Generation, the difference is only 0.85 points. Interestingly, the difference in Mention Detection performance is only 0.02 points, suggesting that the extra output layer is redundant for Mention Detection. In light of these results, we propose using extra hidden layers on the output heads and use this in all following experiments.

		Entity Linking (Micro F1)	
Model	**Mention Detection (Micro F1)**	**No Candidate Generation**	**Candidate Generation**
Hidden layers in output heads	96.91	55.87	89.01
Single layer in output heads	96.93	52.73	88.16

Table 8.: Micro F1 scores on the AIDA-CoNLL Validation dataset of a model with hidden layers in the output heads and a model with only a single layer in each output head.

Cased or Uncased Model

In contrast to Chen et al. (2019), we use the case insensitive (*uncased*) version of BERT.

Table 9 shows results of two models trained with the same parameters with *BERT-base-cased* and *BERT-base-uncased* as base models. The cased model shows a weaker performance in Mention Detection and Entity Linking.

After comparing the models' predictions, we find that the cased model has a lower precision on Mention Detection. We observe that it depends strongly on upper-cased

Model	Mention Detection (Micro F1)	Entity Linking (Micro F1): No Candidate Generation	Entity Linking (Micro F1): Candidate Generation
BERT-base-uncased	96.91	55.87	89.01
BERT-base-cased	96.23	55.52	86.96

Table 9.: Results on the AIDA-CoNLL Validation dataset of two models trained with respectively case sensitive and case insensitive BERT models.

first letters to find mentions and produces more False Positive predictions due to the inconsistent casing in the AIDA-CoNLL dataset. From these results, we conclude that we do not need information about casing to achieve a strong Mention Detection performance, and we will henceforth use the uncased version of BERT as a basis.

5.4.2. Training

In this section, we investigate some training hyperparameters that are significant for the performance of the resulting model. Following Chen et al. (2019), we train all models using a linear learning rate scheduler with an initial learning rate of $2 \cdot 10^{-5}$ and four input sequences in each mini-batch.

The Loss Function Hyperparameter

The loss function has two components: a Mention Detection loss and an Entity Disambiguation loss. These are balanced by a parameter λ, as described in Section 4.1.3:

$$J_{el} = \lambda J_{md} + (1 - \lambda) J_{ed}$$

Following Chen et al. (2019), we train a model with $\lambda = 0.1$. We compare this model to three other models trained with $\lambda = 0.03$, $\lambda = 0.01$ and $\lambda = 0.005$.

Table 10 shows the evaluation results of the various models trained with different values for λ. We find that $\lambda = 0.005$ gives a marginally better Entity Disambiguation result than $\lambda = 0.01$ when evaluating without Candidate Generation.

As expected, the Mention Detection performance decreases with lower λ values. Interestingly, when using Candidate Generation, the Entity Linking performance also decreases at $\lambda = 0.01$ and $\lambda = 0.05$. This performance decrease may be explained by a stronger correlation between Entity Linking performance and Mention Detection performance because Candidate Generation significantly simplifies the disambiguation task. To balance performance with and without Candidate Generation, we use $\lambda = 0.01$ in future experiments.

		Entity Linking (Micro F1)	
λ value	**Mention Detection (Micro F1)**	**No Candidate Generation**	**Candidate Generation**
0.1	96.91	55.87	89.01
0.03	96.66	59.64	89.09
0.01	96.53	61.10	88.88
0.005	96.20	61.16	88.57

Table 10.: Evaluation results on AIDA-CoNLL Validation for four models trained with different λ values in the loss function.

Dropout

We use the default dropout behavior when training BERT, with a dropout probability of 0.1. Unlike Chen et al. (2019), we do not use dropout on the final BERT embeddings before the output heads. When trained with the dropout behavior of Chen et al. (2019), we observe a performance penalty of 1.45 points in the F1 score on the AIDA-CoNLL Validation dataset.

Epochs

The models discussed so far in this chapter train for only 30 epochs. Chen et al. (2019) use 50,000 training steps and four sequences per mini-batch. After splitting long documents, the AIDA-CoNLL Train dataset has 1055 sequences. Accordingly, 50,000 training steps are equivalent to $\frac{50000}{1055/4} \approx 190$ training epochs.

To investigate the impact of the number of training epochs on model performance, we trained models for respectively 30, 60, 90, 120, 150, 180, 210, and 240 epochs. Figure 7 shows the performance results of evaluating on AIDA-CoNLL Validation.

We observe diminishing returns in performance when models train for more than 120 epochs. Between 120 and 180 epochs, the increase is 0.47 points in F1 score, and, from 180 to 240, the increase is 0.23 points. To stay closer to the training budget of Chen et al. (2019), we train our final models for 180 epochs.

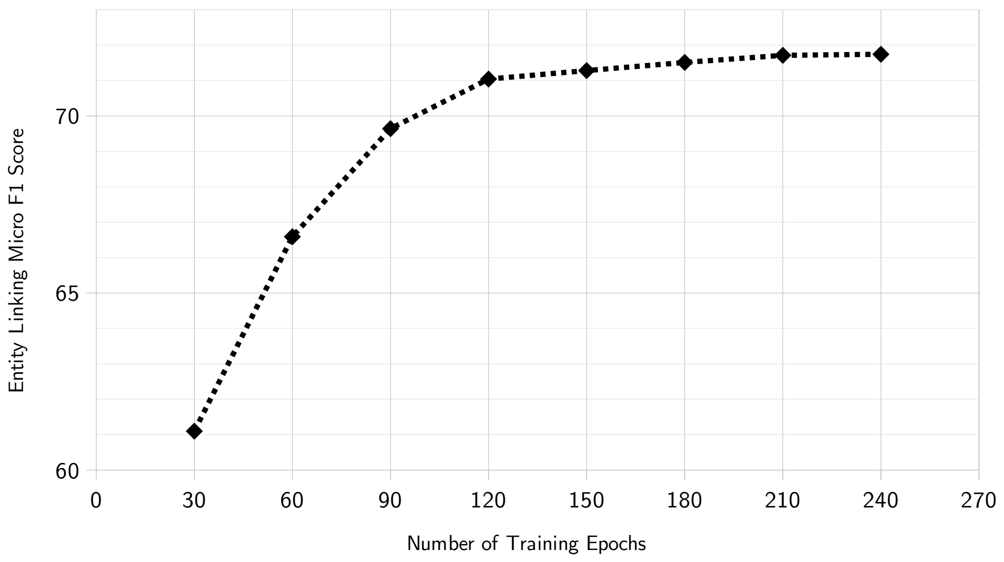


Figure 7.: Evaluation results on AIDA-CoNLL Validation for eight models trained with different number of epochs.

5.4.3. Pretraining on Wikipedia Articles

We hypothesize that the model can benefit from training on more data than the AIDA-CoNLL Train dataset provides. We pretrain a model on the Wikipedia Articles dataset described in Section 5.2.2 and finetune on the AIDA-CoNLL Train dataset to explore this hypothesis.

Because we are using two datasets, we double the training budget and allocate 50,000 training steps to pretraining and finetuning, respectively. The result is four training epochs of pretraining on Wikipedia Articles Train and 180 epochs of finetuning on AIDA-CoNLL Train. That is a total of 99,807 training steps.

Table 11 shows the evaluation results of the pretrained and finetuned model on the AIDA-CoNLL Validation dataset. We compare the performance to the base model trained for 180 epochs, with the hyperparameter settings derived in this chapter. Compared to the base model, the pretrained model shows a performance increase of 1.55 points on AIDA-CoNLL Validation when evaluating without Candidate Generation.

By training on the Wikipedia Articles dataset, the pretrained model trains on entities that appear in the AIDA-CoNLL Test and Validation sets which it would not see from training only on the AIDA-CoNLL training set. We investigate the impact

		Entity Linking (Micro F1)	
	Mention Detection (Micro F1)	**No Candidate Generation**	**Candidate Generation**
Pretrained Model	96.52	73.66	89.93
Base Model	96.47	72.11	89.16

Table 11.: Evaluation results on AIDA-CoNLL Validation for two models: a model pretrained for 4 epochs on Wikipedia Articles and finetuned for 180 epochs on AIDA-CoNLL Train; and a model trained for 180 epochs on AIDA-CoNLL Train.

of pretraining on entities that do not appear in the training set in Section 6.2.1.

6. Results

In this chapter, we evaluate and analyze the results of the models derived in Chapter 5. In Section 6.1, we present the results of our model on the AIDA-CoNLL Test dataset and compare them to other state-of-the-art systems. In Section 6.2, we analyze the results in more depth to expose some strengths and weaknesses.

6.1. Final Model Results

In this section, we evaluate the models from Section 5.4 on the AIDA-CoNLL Test and Validation datasets. Section 6.1.1 reports the results without Candidate Generation, and, conversely, Section 6.1.2 reports the results with Candidate Generation.

Table 12 lists the architecture and training parameters used for the three models that we evaluate: a model which is pretrained on the Wikipedia articles dataset and finetuned on AIDA-CoNLL Train; a model which is only trained on AIDA-CoNLL Train; and a model trained with the settings inherited from Chen et al. (2019).

6.1.1. Without Candidate Generation

Table 13 shows the performance of our best-performing model on the AIDA-CoNLL Validation and Test sets when evaluated without Candidate Generation. We compare with the results reported by Chen et al. (2019) and Broscheit (2019), who also report Entity Linking results without Candidate Generation.

We find that our results are far from the performance reported by Chen et al. (2019), both with and without pretraining. Without pretraining, we see a difference to Chen et al. (2019) in F1 score of 11.4 points on AIDA-CoNLL Validation and 13.0 points on AIDA-CoNLL Test. On AIDA-CoNLL Test, our implementation trained with the settings of Chen et al. (2019) achieves a score 2.7 points lower than our own model without pretraining and 15.7 points below Chen et al. (2019).

From these results, we conclude that we are not able to reproduce the results of Chen et al. (2019). However, we are able to improve the performance of our implementation

	Our Model	Our Model Pretrained	Our Implementation of Chen et al. (2019)
BERT-base Model	Uncased	Uncased	Cased
Hidden Layers in Output Heads	Yes	Yes	No
Training Epochs	180	4 + 180	190
Training Time	4 hrs 50 mins	8 hrs 37 mins	5 hrs 6 mins
Loss Function λ	0.01	0.01	0.1
Dropout Before Output Heads	No	No	Yes

Table 12.: Characteristics of our best-performing models with and without pretraining and our implementation with the settings of Chen et al. (2019).

	Validation		**Test**	
	Mention Detection (Micro F1)	**Entity Linking (Micro F1)**	**Mention Detection (Micro F1)**	**Entity Linking (Micro F1)**
Our Model	96.47	72.11	95.10	56.41
Our Model Pretrained	96.52	73.66	95.15	59.88
Our version of Chen et al. (2019)	95.84	69.26	94.28	53.74
Broscheit (2019)		86.0		79.3
Chen et al. (2019)		83.5		69.4

Table 13.: Evaluation results on AIDA-CoNLL Validation and Test for our best-performing model without Candidate Generation compared with other Entity Linking models that do not rely on Candidate Generation.

of their model using the modifications presented in Section 5.4. Pretraining on a larger dataset also gives an increase in performance.

6.1.2. With Candidate Generation

We evaluate the same three model with Candidate Generation. Table 14 shows the performance of the three models on the AIDA-CoNLL Validation and Test sets. Here, we compare with the results of all papers mentioned in Section 3.3. For Chen et al. (2019) the results are obtained with Candidate Generation. The results of Broscheit (2019) are the same as above, as their model cannot use Candidate Generation.

We see a large performance increase when evaluating our models with Candidate Generation. This is not surprising, given the advantage that Candidate Generation gives in restricting the search space. The absolute difference from our base model to the results reported by Chen et al. (2019) is significantly lower, with a difference of 4.4 points in F1 score on AIDA-CoNLL Validation and 4.7 on AIDA-CoNLL Test.

6.2. Error Analysis

To better understand the strengths and weaknesses of the proposed model, we conduct a comprehensive analysis of its performance. Firstly, in Section 6.2.1, we will look at how well the model generalizes to entities it has not seen during training and how pretraining affects this. Next, in Section 6.2.2, we will look at characteristics of entities where the model is likely to make errors.

6.2.1. Evaluation by Seen and Unseen Entities

By evaluating with data that is disjoint from the training dataset, we can assess how well the model generalizes to unseen data. Although no documents appear twice in the AIDA-CoNLL dataset, many entities appear in multiple contexts across the AIDA-CoNLL Train, Validation, and Test datasets.

We evaluate a model's capacity to **generalize to new entities** by splitting the mentions in the AIDA-CoNLL Test dataset in two categories:

- mentions of *"seen"* entities which appear as labels in the AIDA-CoNLL Train dataset and which the model has seen during training; and
- *"unseen"* entities, which do not appear in AIDA-CoNLL Train.

	Validation		**Test**	
	Mention Detection (Micro F1)	**Entity Linking (Micro F1)**	**Mention Detection (Micro F1)**	**Entity Linking (Micro F1)**
Our Model	96.47	89.16	95.10	83.03
Our Model Pretrained	96.52	89.93	95.15	85.08
Our version of Chen et al. (2019)	95.84	87.95	94.28	80.71
Broscheit (2019)		86.0		79.3
Martins et al. (2019)	95.72	85.2	92.52	81.9
Kolitsas et al. (2018)		89.4		82.4
Poerner et al. (2020)*		90.8		85.0
Chen et al. (2019)		93.6		87.7

Table 14.: Evaluation results on AIDA-CoNLL Validation and Test for our best-performing model with Candidate Generation compared with other Entity Linking models. *Poerner et al. (2020) does not perform Mention Detection, but reports Entity Linking results with third-party Mention Detection.

The AIDA-CoNLL Test dataset has 4485 mentions of 1536 unique entities. There are 1853 mentions of 883 unique *unseen* entities and 2632 mentions of 653 unique *seen* entities. This yields 41.3 % *unseen* mentions and 57.5 % *unseen* entities.

We calculate a model's **Entity Disambiguation accuracy** as the In-Knowledge-Base recall excluding Mention Detection errors. As defined in Section 5.1, recall is the ratio of correct predictions to mentions.

Without Candidate Generation

Table 15 shows the results of our base model and the pretrained model on *seen* and *unseen* entities, evaluated without Candidate Generation on the AIDA-CoNLL Test dataset. The models disambiguate the *seen* entities with Entity Disambiguation accuracies above 93 %. We conclude from this that the models learn to recognize the entities that appear in the training data and are able to predict them in new contexts with high accuracy without the aid of Candidate Generation.

	Entity Disambiguation Accuracy Without Candidate Generation		
	All Mentions (%)	***Seen* Entities (%)**	***Unseen* Entities (%)**
Our Model	59.36	93.05	7.58
Our Model Pretrained	62.98	93.79	15.86

Table 15.: Entity Disambiguation accuracy on In-Knowledge-Base entities in AIDA-CoNLL Test categorized by whether or not they appear in AIDA-CoNLL Train. Evaluation is without Candidate Generation and ignoring Mention Detection errors.

In comparison, the models perform poorly on the *unseen* entities. The weak performance suggests that the models do not learn to generalize to new entities well enough for entity disambiguation without Candidate Generation. A remedy for this problem is to train on more data, as exemplified by the pretrained model. The Entity Disambiguation accuracy of the pretrained model is twice that of the base model on unseen entities.

However, the pretrained model has encountered some *unseen* AIDA-CoNLL Test entities in the Wikipedia Articles dataset. Accordingly, we evaluate the Entity Disambiguation accuracy on entities that appear in neither training set. Around 22.5

% of the mentions and 26.0 % of the entities in AIDA-CoNLL Test are unseen by both models. On these mentions, the pretrained model achieves an Entity Disambiguation accuracy of 3.5 %. The base model only achieves an accuracy of 1.4 %. We conclude that the pretrained model generalizes better than the base model, but that neither model is able to generalize reliably to unseen entities without Candidate Generation.

With Candidate Generation

When using Candidate Generation for evaluation, the models perform significantly better. Table 16 shows the performance of the two models on AIDA-CoNLL Test on *seen* and *unseen* AIDA-CoNLL Test mentions. The performance on *unseen* entities is much higher than without Candidate Generation, and the difference in performance between the two models is reduced. These results show that the models have learned sufficiently to achieve high Entity Disambiguation accuracies with Candidate Generation.

	Entity Disambiguation Accuracy With Candidate Generation		
	All Mentions (%)	***Seen* Entities (%)**	***Unseen* Entities (%)**
Our Model	88.07	97.15	74.60
Our Model Pretrained	90.18	97.74	79.02
Baseline Model	63.90	70.55	54.04

Table 16.: Entity Disambiguation accuracy on In-Knowledge-Base entities in AIDA-CoNLL Test categorized by whether or not they appear in AIDA-CoNLL Train. Evaluation is with Candidate Generation and ignoring Mention Detection errors.

We evaluate a **baseline model**, which **predicts the candidate with the highest popularity** (defined in Section 5.3.1 as the number of incoming links to an entity's Wikipedia article). The results of the baseline model are listed in Table 16, and give us an indication of Candidate Generation on *seen* and *unseen* entities. Interestingly, the baseline model also shows a significant performance gap between *seen* and *unseen* entities.

The candidate sets for the *unseen* entities have, on average, half as many candidates as the *seen* entities. Consequently, the probability of finding the correct candidate should be higher for *unseen* entities. Furthermore, a third of the candidate sets for *unseen* mentions have the ground truth entity as the only candidate. In these cases, the models default to the correct prediction. Accordingly, the *unseen* entities should benefit more from the candidate sets than *seen* entities.

Contradictory to this, the performance gap of the baseline model between *seen* and *unseen* entities tells us that the *unseen* entities are 16.5 % less likely to be the most popular candidates. Moreover, the ground truth entity for *seen* entities are on average the 1.8 most popular candidate, while for *unseen* entities, they are only the 3.7 most popular candidate.

To investigate the importance of entity popularity for our models, we look at the correlation of entity popularity and entity frequency in the AIDA-CoNLL datasets. We find a strong correlation between the frequency of *seen* entities in AIDA-CoNLL Train and their popularity (Pearson's r = 0.7992)[1]. On the other hand, the *unseen* entities are poorly correlated with popularity, as confirmed above by their rank in their respective candidate sets.

In conclusion, we hypothesize that the performance discrepancy between *seen* and *unseen* entities could have a two-fold explanation: trivially, the models have not trained on *unseen* entities and thus are unfamiliar with the target entities; additionally, the *unseen* entities are inherently more difficult to disambiguate. Further research is needed to investigate whether the models learn a prior probability distribution from the training set.

Regardless of the models' disadvantage on *unseen* entities, we see a large performance increase on these entities when using Candidate Generation. The performance increase is also much higher for *unseen* entities than for *seen* entities, where the potential for improvement is lower.

The models show a similar performance difference between *seen* and *unseen* entities in the AIDA-CoNLL Validation dataset. Moreover, 23.7 % of the mentions in the Validation dataset are of *unseen* AIDA-CoNLL entities, while the same is true for 41.3 % of mentions in the Test dataset. This explains why the models perform better on AIDA-CoNLL Validation than on AIDA-CoNLL Test.

6.2.2. Performance by Entity Types

We combine In-Knowledge-Base entities in AIDA-CoNLL Test with information from Wikidata. Using the attributes of the corresponding Wikidata entities, we categorize

[1]The Pearson's correlation coefficient is highly significant with p-value $<< .001$.

the entities as *organizations* (2985 mentions), *locations* (1704 mentions), *persons* (890 mentions), and *events* (116 mentions). **Entities may fall into multiple categories.** These categories cover 68.5 % of the mentions in AIDA-CoNLL Test. Following the approach in Section 6.2.1, we further split each category into entities that the models have *seen* in AIDA-CoNLL Train, or that otherwise are *unseen*.[2]

For example, the Wikipedia Entity *El Salvador* corresponds to the Wikidata entity *Q792*. Using the *instance-of* and *subclass-of* attributes of the Wikidata entity, we categorize it as an *organization* and a *location*.

We evaluate the Entity Disambiguation accuracy for entity mentions in each category. The accuracy excludes Mention Detection errors and entities that are not in the Knowledge Base. Table 17 shows the evaluation results of the base model and the pretrained model with and without Candidate Generation on categories of mentions in AIDA-CoNLL Test. The table header also lists the number of mentions in each category.

[2]We do not consider whether or not entities appear in the Wikipedia Articles dataset.

	Entity Disambiguation Accuracy (%)									
	All Mentions		**Organizations**		**Locations**		**Persons**		**Events**	
	Seen 2627 mentions	**Unseen** 1753 mentions	**Seen** 2127 mentions	**Unseen** 858 mentions	**Seen** 1513 mentions	**Unseen** 191 mentions	**Seen** 229 mentions	**Unseen** 661 mentions	**Seen** 25 mentions	**Unseen** 91 mentions
Our Model - Without CG	85.37	6.49	87.98	11.52	87.09	11.05	72.93	2.44	59.09	17.39
Our Model Pretrained - Without CG	86.08	14.68	89.04	25.46	87.38	25.00	71.05	3.52	55.00	41.67
Our Model - With CG	97.00	73.64	97.22	62.69	97.55	76.44	96.71	94.78	71.43	60.00
Our Model Pretrained - With CG	97.52	78.14	97.80	70.54	97.89	80.90	96.71	93.32	68.42	61.76

Table 17.: Entity Disambiguation accuracy on entities in AIDA-CoNLL Test by categories. "All Mentions" includes all AIDA-CoNLL Test entities with Wikidata labels, regardless of category. The accuracy measure excludes Mention Detection errors and out-of-knowledge-base entities. An entity may fall in multiple categories. The models are evaluated with and without Candidate Generation, denoted with *CG*.

Before looking at the performance, we note some characteristics of the categories. Firstly, we observe from the ratio of *seen* and *unseen* mentions in each category that *person* and *event* entities in AIDA-CoNLL Test are unlikely to have appeared in AIDA-CoNLL Train. Conversely, *organizations* and *locations* are likely to have appeared in AIDA-CoNLL Train.

Moreover, we find that the average popularity score[3] for *organizations* and *locations* is over six times higher than the popularity of *persons* and *events*. Additionally, *organization* and *location* entities have on average twice as many mentions per unique entity as compared to *persons* and *events*.

Reflecting the results in Section 6.2.1, the models consistently perform better on *seen* entities than on *unseen* entities across all categories. Furthermore, Candidate Generation increases performance on both *seen* and *unseen* entities for all categories.

Without Candidate Generation

Without Candidate Generation, the models perform below the average for *seen persons* and *events*. Interestingly, *seen events* see the poorest performance of *seen* entities, while *unseen events* have the best performance.

To exemplify the base model's predictions on the category of *events*, we look at the two *event* entities most frequently mentioned in AIDA-CoNLL Test:

- The *unseen event* entity "National Hockey League" appears twelve times AIDA-CoNLL Test. The model predicts it correctly in seven occurrences, all of which with the mention text "NHL". The remaining five occurrences have mention text "National Hockey League", and the model disambiguates incorrectly to the entity "National League" (a North American baseball league) four times. This entity is a *seen* entity appearing nine times in AIDA-CoNLL Train. For the fifth incorrect prediction, the model predicts the related entity "Season structure of the NHL".

- The *unseen event* entity "FIS Alpine Ski World Cup" appears nine times in AIDA-CoNLL Test, always with the mention text "World Cup". However, the model consistently predicts the *seen* entity "UCI Road World Cup" (a cycling championship held in Europe) which appears five times in AIDA-CoNLL Train with "World Cup" as the mention text. Interestingly, six entities appear in AIDA-CoNLL Train with "World Cup" as mention text, and two of these appear

[3]The popularity of an entity is defined in Section 5.3.1 as the number of incoming links to its Wikipedia article.

at least as frequently as "UCI Road World Cup". It is not clear from the context why the model predicts this entity instead of other *seen* entities.

In both of these examples, the model predicts *seen* entities which appear with a similar mention text when the ground truth label is an *unseen* entity. In these cases, the model has overfit to the training data and recalls a *seen* entity without properly considering the context.

The annotations for *event* entities are often ambiguous in the AIDA-CoNLL dataset. Consider the following excerpt from a document with ground truth entity annotations from AIDA-CoNLL Test:

> SOCCER - {SPAIN} [Spain] PICK UNCAPPED ARMANDO FOR **{WORLD CUP} [1998_FIFA_World_Cup]** CLASH. {MADRID} [Madrid] 1996-12-06 {Spain} [Spain] coach {Javier Clemente} [Javier_Clemente] has added uncapped Deportivo Coruna midfielder Armando Alvarez to his squad for the **{World Cup} [FIFA_World_Cup]** qualifier against Yugoslavia on December 14.

The document has two mentions of "World Cup" annotated with "1998 FIFA World Cup" and "FIFA World Cup" (both are *seen* entities). In both instances, the model predicts "1998 FIFA World Cup", which is arguably correct for both mentions.

We observe the poorest performance in the category of ***unseen persons***. Only a third of the *person* entities in AIDA-CoNLL Test are *seen* entities. Furthermore, *seen person* entities have a lower average frequency in AIDA-CoNLL Train than other categories. We hypothesize that the models have little training experience with disambiguating *person* entities in multiple contexts.

With Candidate Generation

When evaluating with Candidate Generation, *seen events* see the lowest performance among *seen* entities. For *unseen* entities, the performance on *persons* is well above average, and *events* again see the poorest performance.

On *unseen persons*, the performance increases considerably when using Candidate Generation. In fact, the models perform almost as well on *seen* and *unseen persons.* This is less surprising when we consider that there are only ten candidates on average for *persons*, compared to over 50 candidates on average for other categories.

We hypothesize that the poor performance on both *seen* and *unseen events* is related to the issues of annotation in AIDA-CoNLL as argued above. However, we also note that the model relies heavily on small candidate sets for *event* entities: incorrectly predicted *events* have six times more candidates than correctly predicted *events*. In contrast, *organizations* and *locations* have virtually the same number of candidates for correct and incorrect predictions. These three categories have comparable number of candidates, with an average of 56 candidates.

The difficulty of *event* entities may be related to the distribution of these entities in the Wikipedia2vec vector space. For example, the entity "FIFA World Cup" is only the 16th most similar entity to "1998 FIFA World Cup", measured by cosine similarity in Wikipedia2vec. This highlights a challenge of using the Wikipedia2vec vector space for entity disambiguation: entities that appear in similar contexts need to be disambiguated using features extracted from the context. The **projection of token vectors in the Entity Disambiguation head** from 768 dimensions to 100 dimensions leads to a loss of information. Using a higher dimensional label space may lead to better results, at the cost of more training data.

Similarity of Candidates by Entity Category

The **similarity of candidates in each category** gives us a comparison of disambiguation difficulty. If candidates are more similar to the ground truth entity, the model must project the Entity Disambiguation vector more precisely to disambiguate correctly. We define similarity as the cosine-similarity of two entity vectors in Wikipedia2vec. To compare each category of entities, we average the similarity of the ten most similar candidates to the ground truth entity. In each category, we consider all mentions with at least ten candidates:

- *Organizations*: For 2204 mentions, the ten most similar candidates are 64.6 % similar.
- *Locations*: For 1474 mentions, the ten most similar candidates are 64.1 % similar.
- *Persons*: For 147 mentions, the ten most similar candidates are 55.3 % similar.
- *Events*: For 57 mentions, the ten most similar candidates are 76.3 % similar.

From this, we can see that *events* have candidates that are relatively similar to the target entity compared to *organizations*, *locations* and *persons*. This may explain why this category still shows poor performance with Candidate Generation. We also see that *persons*, in addition to having considerably fewer candidates on average, have less similar candidates.

In conclusion, the models benefit more from candidate sets with maximally different candidates. This requires less precise embedding predictions from the model. As illustrated by the category of *person* entities, this leads to a high performance even on *unseen* entities.

In contrast, the relatively small differences between *event* candidates requires proportionately more precise embedding predictions from the models. We have observed that the models perform above average on *unseen events* without Candidate Generation. However, with Candidate Generation the performance is below average for both *seen* and *unseen event* entities. We believe this behavior is due to the high similarity of candidates for *event* entities.

7. Conclusion

Answering the research goals stated in Section 1.3, we conclude that

- **We are not able to reproduce the results of Chen et al. (2019)**, neither with the same architectural and training hyperparameters nor with custom hyperparameters. Our attempt at reproducing their model performs 15.7 points in F1 score below the performance reported by Chen et al. (2019) when evaluating without Candidate Generation on the AIDA-CoNLL Test dataset. In the absence of any clear differences to their method, we propose no hypotheses to explain this discrepancy.

- **Relative to the results we obtain** with the proposed settings from Chen et al. (2019), we are able to **improve the performance** of the model. Using an additional fully connected layer in the output heads, a case-insensitive BERT model, and improved training hyperparameters, we improve the performance by 2.7 points in F1 score on the AIDA-CoNLL Test dataset when evaluating without Candidate Generation.

- **The results of our best performing model improve** by an additional 3.5 points in F1 score **when pretraining the model** on a dataset of 50,000 Wikipedia articles annotated with anchor text links. We found that the main reason for this performance improvement is that the pretrained model has been exposed to more of the entities that appear in the ADIA-CoNLL Test set. However, the pretrained model generalizes slightly better than the base model to entities that it has not seen during training on either dataset.

- **When evaluating with Candidate Generation** using the candidate sets from Ganea and Hofmann (2017), **the performance improves** by 26.6 points in F1 score on AIDA-CoNLL Test. **This result is near the state-of-the-art**, but is still 4.7 points lower than the F1 score reported by Chen et al. (2019) with Candidate Generation.

- Comparing the model's performance on **entities that it has *seen* during**

training to entities that it has not seen (*unseen*), we find that **the model generalizes relatively poorly to the *unseen* entities**. Without the aid of Candidate Generation, the model correctly disambiguates 93 % of *seen* entities, and 8 % of *unseen* entities. When evaluating with Candidate Generation, we observe an accuracy on *seen* entities of 97 %, and, conversely, 75 % for *unseen* entities.

- By **categorizing entities** as organizations, locations, persons and events, we revealed **large differences in performance between categories**. We found that the performance increased the most with Candidate Generation when the generated candidate set contained relatively dissimilar candidates and when the candidate sets were small.

8. Acknowledgments

First and foremost, I would like to thank Natalie Prange for advice and discussions during the work on this thesis. Under your supervision, I regularly had to summarize my progress and defend my decisions.

Furthermore, I sincerely thank Haotian Chen at Blackrock for answering in detail all my questions about his research, which helped me out of a ditch at important moments during this project.

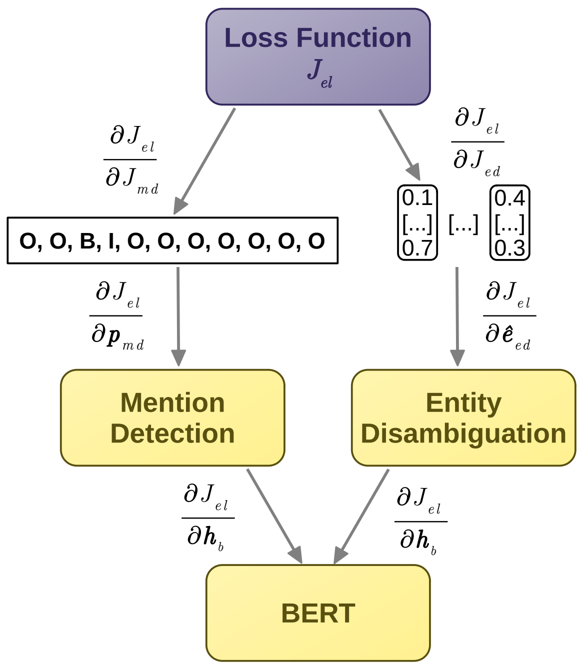
Next, I humbly thank Prof. Dr. Hannah Bast and Dr. Fang Wei-Kleiner for agreeing to supervise my thesis.

Finally, I would like to thank my willing and patient proofreaders: my good friend Jakob Voigt, for his keen eye for the pedagogic and his mind for complexity; my father Erlend Råheim for his grammatical scrutiny and patience in facing the unfamiliar; and finally to my partner Clémence Koren for her support and patience, and for keeping me focused on the bigger picture.

Appendices

A. The Gradient of the Loss Function

Recall that, when training neural networks with backward propagation, the training signal propagates backward from the loss function, through the output heads and the network layers, towards the input layer. The **training signal is the gradient of the loss function** with regard to the trainable weights at each layer of the network. Figure 8 shows how the gradient moves from the loss function through the main modules of the model.



Loss Function
J_{el}
$\frac{\partial J_{el}}{\partial J_{md}}$
$\frac{\partial J_{el}}{\partial J_{ed}}$
O, O, B, I, O, O, O, O, O, O, O
0.1 [...] 0.7
[...]
0.4 [...] 0.3
$\frac{\partial J_{el}}{\partial \boldsymbol{p}_{md}}$
$\frac{\partial J_{el}}{\partial \hat{\boldsymbol{e}}_{ed}}$
Mention Detection
Entity Disambiguation
$\frac{\partial J_{el}}{\partial \boldsymbol{h}_{b}}$
$\frac{\partial J_{el}}{\partial \boldsymbol{h}_{b}}$
BERT

Figure 8.: The propagation of the training signal from the loss function through the modules of the network.

Hence, to start the evaluation of the training signal, we calculate the gradient of the loss function J_{el} with regard to its inputs.[1] The inputs to the loss function are

[1] We do not calculate the gradient with regard to the labels, even though they are also inputs to the loss function. The labels are not produced by the network and do not create a training signal for model weights.

the outputs of the network $\mathbf{p}_{md}$ and $\hat{\mathbf{e}}_{ed}$.

First, we calculate the gradient with regard to the Mention Detection predictions $\mathbf{p}_{md}$:

$$\begin{aligned}\frac{\partial J_{el}}{\partial \mathbf{p}_{md}} &= \frac{\partial \left(J_{ed}(\hat{\mathbf{e}}_{ed}, \mathbf{e}_w) + J_{md}(\mathbf{p}_{md}, gt_{md})\right)}{\partial \mathbf{p}_{md}} \\ &= \frac{\partial J_{ed}}{\partial \mathbf{p}_{md}} + \frac{\partial J_{md}}{\partial \mathbf{p}_{md}} = \frac{\partial J_{md}}{\partial \mathbf{p}_{md}}\end{aligned}$$

We can see that the gradient with regard to the Mention Detection error does not depend on Entity Disambiguation errors. Next, the gradient with regard to the Entity Detection embeddings $\hat{\mathbf{e}}_{ed}$ does not depend on the Mention Detection predictions:

$$\begin{aligned}\frac{\partial J_{el}}{\partial \hat{\mathbf{e}}_{ed}} &= \frac{\partial \left(J_{ed}(\hat{\mathbf{e}}_{ed}, \mathbf{e}_w) + J_{md}(\mathbf{p}_{md}, gt_{md})\right)}{\partial \hat{\mathbf{e}}_{ed}} \\ &= \frac{\partial J_{ed}}{\partial \hat{\mathbf{e}}_{ed}} + \frac{\partial J_{md}}{\partial \hat{\mathbf{e}}_{ed}} = \frac{\partial J_{ed}}{\partial \hat{\mathbf{e}}_{ed}}\end{aligned}$$

After training the network weights in the output layers, we look at the input to the output layers. Both the Mention Detection head and the Entity Disambiguation head take BERT embeddings $\mathbf{h}_b$ as input. Hence, the training signal again depends on both the individual loss functions. The gradient of the loss with regard to $\mathbf{h}_b$ is:

$$\frac{\partial J_{el}}{\partial \mathbf{h}_b} = \frac{\partial J_{el}}{\partial \hat{\mathbf{e}}_{ed}} \frac{\partial \hat{\mathbf{e}}_{ed}}{\partial \mathbf{h}_b} + \frac{\partial J_{el}}{\partial \mathbf{p}_{md}} \frac{\partial \mathbf{p}_{md}}{\partial \mathbf{h}_b}$$

We see that the two gradients from the output heads merge to a sum that depends on both output heads. Consequently, the BERT network producing the embeddings $\mathbf{h}_b$ for Mention Detection and Entity Disambiguation trains to tackle both tasks simultaneously. This is the result and the aim of using a joint learning objective for both Mention Detection and Entity Disambiguation.

Bibliography

Haotian Chen, Andrej Zukov-Gregoric, Xi David Li, and Sahil Wadhwa. Contextualized end-to-end neural entity linking. *arXiv preprint arXiv:1911.03834*, 2019.

Nina Poerner, Ulli Waltinger, and Hinrich Schütze. E-bert: Efficient-yet-effective entity embeddings for bert. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 803–818, 2020.

ITU. Measuring digital development: Facts and figures 2020. `https://www.itu.int/en/ITU-D/Statistics/Documents/facts/FactsFigures2020.pdf`, 2020.

Mark Ware and Michael Mabe. The stm report: An overview of scientific and scholarly journal publishing. `https://www.stm-assoc.org/2012_12_11_STM_Report_2012.pdf`, 2015.

W Bruce Croft, Donald Metzler, and Trevor Strohman. *Search engines: Information retrieval in practice*, volume 520, pages 113–118. Addison-Wesley Reading, 2010.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423`.

Manoj Prabhakar Kannan Ravi, Kuldeep Singh, Isaiah Onando Mulang, Saedeh Shekarpour, Johannes Hoffart, and Jens Lehmann. CHOLAN: A Modular Approach for Neural Entity Linking on Wikipedia and Wikidata. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2021*, pages 504–514, 2021.

Pedro Henrique Martins, Zita Marinho, and André FT Martins. Joint learning of named entity recognition and entity linking. In *Proceedings of the 57th Annual Meet-*

ing of the Association for Computational Linguistics: Student Research Workshop, pages 190–196, 2019.

Nikolaos Kolitsas, Octavian-Eugen Ganea, and Thomas Hofmann. End-to-end neural entity linking. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 519–529, 2018.

Octavian-Eugen Ganea and Thomas Hofmann. Deep joint entity disambiguation with local neural attention. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2619–2629, 2017.

Samuel Broscheit. Investigating entity knowledge in BERT with simple neural end-to-end entity linking. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 677–685, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/K19-1063. URL `https://www.aclweb.org/anthology/K19-1063`.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

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

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Kawin Ethayarajh. How contextual are contextualized word representations? comparing the geometry of bert, elmo, and gpt-2 embeddings. 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)*, pages 55–65, 2019.

Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. Robust disambiguation of named entities in text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 782–792, 2011.

Razvan Bunescu and Marius Pasca. Using encyclopedic knowledge for named entity disambiguation. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006.

Sayali Kulkarni, Amit Singh, Ganesh Ramakrishnan, and Soumen Chakrabarti. Collective annotation of wikipedia entities in web text. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 457–466, 2009.

Lev Ratinov, Dan Roth, Doug Downey, and Mike Anderson. Local and global algorithms for disambiguation to wikipedia. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 1375–1384, 2011.

Andrew Chisholm and Ben Hachey. Entity disambiguation with web links. *Transactions of the Association for Computational Linguistics*, 3:145–156, 2015.

Maria Pershina, Yifan He, and Ralph Grishman. Personalized page rank for named entity disambiguation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 238–243, 2015.

Ikuya Yamada, Hiroyuki Shindo, Hideaki Takeda, and Yoshiyasu Takefuji. Joint learning of the embedding of words and entities for named entity disambiguation. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 250–259, 2016.

Özge Sevgili, Artem Shelmanov, Mikhail Arkhipov, Alexander Panchenko, and Chris Biemann. Neural entity linking: A survey of models based on deep learning. *Semantic Web*, 2020. URL `http://www.semantic-web-journal.net/system/files/swj2986.pdf`.

Zhengyan He, Shujie Liu, Mu Li, Ming Zhou, Longkai Zhang, and Houfeng Wang. Learning entity representation for entity disambiguation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 30–34, 2013.

Ikuya Yamada, Koki Washio, Hiroyuki Shindo, and Yuji Matsumoto. Global entity disambiguation with pretrained contextualized embeddings of words and entities. *arXiv preprint arXiv:1909.00426*, 2019.

Matthew E Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A Smith. Knowledge enhanced contextual word representations. 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)*, pages 43–54, 2019.

Valentin I Spitkovsky and Angel Chang. A cross-lingual dictionary for english wikipedia concepts. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 3168–3175, 2012.