Natural Language Set Expansion

Project Report

submitted to the

Marshallplan-Jubiläumsstiftung
Austrian Marshall Plan Foundation
Fostering Transatlantic Excellence

by

Florian Gwechenberger, BSc

FH Salzburg

Supervisor SUAS: DI Cornelia Ferner, BSc
Supervisor UMass: Prof. James Allan

Salzburg, September 2020

Details

Author	Florian Gwechenberger, BSc
University	Salzburg University of Applied Sciences
Degree Program	Information Technology and Systems Management

Keywords	Natural Language Processing, BERT, Entity Set Expansion

Supervisor	DI Cornelia Ferner, BSc

Abstract

This work elaborates on how contextualized word embeddings can be used to expand a set of seeds given a predefined text corpus. Here, it is investigated how the so-called BERT model can be used in a feature-based and a fine-tuned way for this particular problem. The gained insights are subsequently used to design a complete pipeline that is able to solve this problem even on a raw text corpus. The final experiments show that BERT embeddings can successfully be used for such a task .They exceed all previous approaches regarding precision and MAP score.

Contents

1 Introduction

In the last years, the technologies behind Machine Learning (ML) algorithms, especially in the field of Natural Language Processing (NLP), have improved beyond many expectations. Because of that, a lot of research is currently trying to continually improve state-of-the-art technologies, which can substitute past implementations. While in the early work NLP was done by only using high-level features of a text, it further evolved to neural methods, which are now further substituted by attention architectures. This work describes how concepts like contextualized embeddings and attention mechanisms can be applied to an unsolved problem, the so-called Entity Set Expansion (ESE) task and outperform recent implementations.

ESE describes the task of expanding a predefined set of seeds with related entities to expand the set. A simple example of such a system is illustrated in Figure 1.1. Here, the input set contains the seeds ”Massachusetts”, ”Florida” and ”Texas”. This query set seems to consist entirely of entities from the semantic group ”States in the USA”, thus this set could be completed with the entities ”New York” or ”Ohio”. While this issue in particular seems very trivial to solve, many sets can hardly be expanded without any broader understanding of the context of the word. For instance, if the given set consists of the countries ”France”, ”Great Britain” and ”Russia”, it can either be concluded that the correct solution would be countries in Europe or allies in the second world war. In previous work, there have been essentially two categories of approaches to this problem. The first are web search-based applications like SEAL [1] or Lyretail [2], which rely on an external search engine to parse documents for entities that match the same pattern as the query seeds. The other approaches consider only a predefined text corpus of one or multiple documents to expand the set. Thus the system needs to be more sophisticated to recognize semantic similarities between the seeds. But on the contrary, the system can solve a set in particular for this corpus and can also expand queries with unique seeds that only appear in these documents.

Corpus-based set expansion applications are applied in various fields. For instance, they can be a powerful tool to reduce the workload for other research projects in the area of NLP. At this moment, creating a proper dataset is often done by humans, which is a time-consuming, expensive as well as a cumbersome task. Furthermore, corpus-based ESE systems are also used as a stand-alone application. For instance, as described in [3], such a system is integrated into a recruiting tool to find the most suited employees or in software development to find duplicates with a different description in error reports.

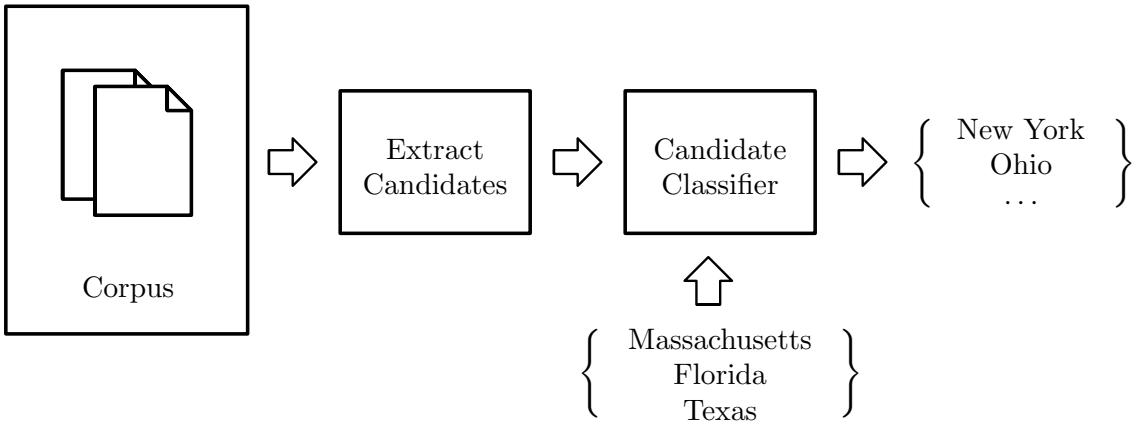


Figure 1.1: Entity Set Expansion pipeline visualized.

In the previous work, a corpus-based ESE basically consists of two components, as illustrated in Figure 1.1. At first, candidates are extracted from a given corpus. These candidates are then classified and all valid solutions are added to the set. In some of the latter mentioned work, the extraction step is skipped and a set of proper candidates is preselected. Therefore, these systems do not offer a concept for a complete pipeline, which can be implemented for an actual system. These implementations solely focus on creating a classifier optimized for these sets. Since the quality of the extracted candidates subsequently affects the results of the classifier, this work considers the candidate extractions step as well. The remaining restriction is that the pipeline only expands sets containing named entities. A named entity is a specific instance of a given group. For example, "Canada" would be a named entity for the group countries, while "David Bowie" would be a named entity for the group musicians. Because the dataset provided to evaluate the system consists merely of such named entities, the extraction step only parses named entities. Still, the pipeline could also be used for different inputs.

Also, instead of using traditional technologies, this work employs state-of-the-art ML technology called contextualized embeddings, already in the pre-processing as well as for the final classifier, which leads to remarkable improvements of the results.

To establish a decent knowledge of this contemporary technology, the work elaborates in Chapter 2 on the developments in the last years, which lead to the current technology applied in this work. As the described work covers a variety of contemporary topics in this field, the following work assumes basic knowledge of neural ML concepts from the reader to keep this work as cohesive as possible. Here, the reader is referred to the additional literature mentioned in the work. Based on these information, the subsequent Chapter 3 describes the architecture of the actual ESE system, while Chapter 4 shows the results

of the experiments of the system and investigates the impact of certain elements of the pipeline.

2 Literature Review

Before any of the methods for the ESE task or the proposed findings can be further described, it is important to introduce the related topics first. In Section 2.1, it is elaborated on how the meaning words can be represented as embeddings and how these have been utilized for the ESE task. Furthermore, it is demonstrated in Section 2.2 how the context of words can be represented by a sequential Language Model (LM). In Section 2.3, a novel attention network without any sequential elements called Transformer is presented, which concepts are further developed in Bidirectional Encoder Representations from Transformers (BERT) described in Section 2.4 to create strong contextualized embeddings.

2.1 Word Embeddings

Before text can be utilized in any NLP task, it needs to be processed into a suitable form for applications. To create a more meaningful representation of the words than just simply reading in each character after another, text is often converted into a distributed vector representation, also referred to as embeddings. A common example of this is the bag of word representation. A bag of words is defined over a predefined vocabulary and represents the occurrences of each word in a sentence. Here, the binary representation of a word is usually referred to as one-hot encoding. Although this representation seems intuitive, it does not consider any relationship between the words themselves or the context they appear in. In [4], it is elaborated how the textual context can be used to preserve the similarities between words to some extend. If a word which is not common like "tezgüino" appears in the contexts "A bottle of ___ is on the table", "Don't have ___ before you drive" and "___ is made out of corn", it can be inferred that this word might be some kind of beverage. This assumption is especially useful for words that do not appear during the training of a system.

The exact context matching of words is used in [5] for the ESE task. The authors propose an unsupervised system called SetExpan, which is divided into two major steps. The first extracts the exact context information from the seed entities out of the raw text. Entities which share the exact same context are considered as candidates. The second step is ranking the candidates based on how often the entities occur in these contexts. SetExpan adds a solution after each iteration instead of ranking all the candidates at once. Therefore, the seed set is growing and the performance of the system increases if the solution is correct. But this does also allow the system to make incorrect assessments

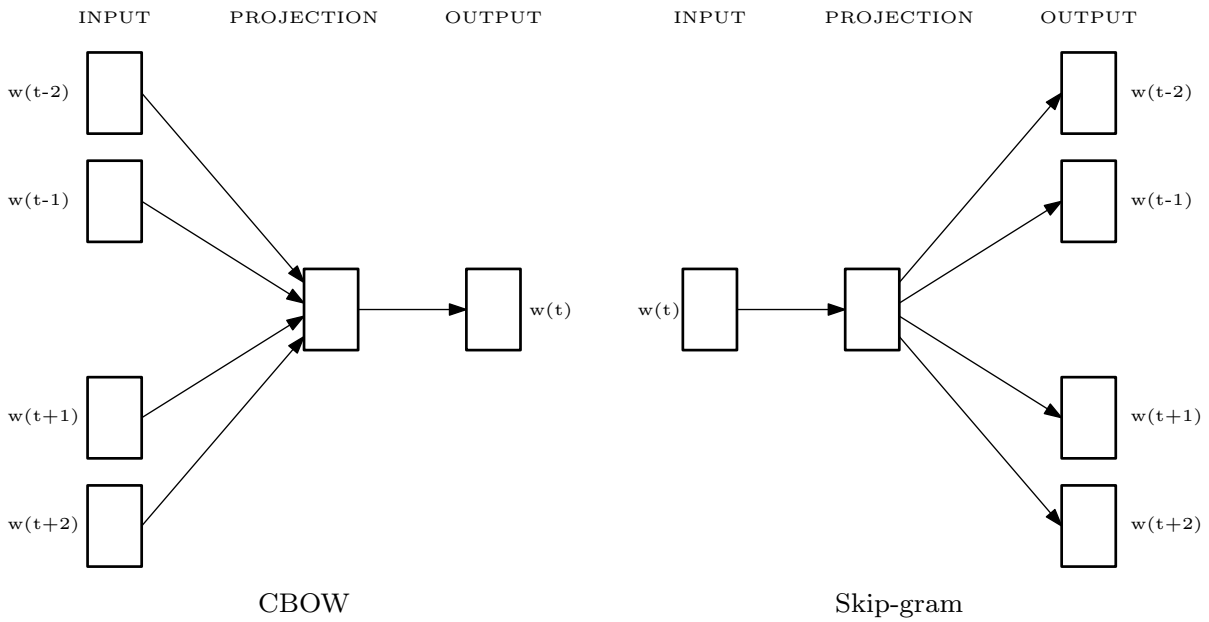


Figure 2.1: Word2Vec models illustrated as described in [6].

and thus, it might impair the precision of the system. This issue is named the semantic thrift problem.

The SetExpan approach does only consider the lexical context information and ignores any semantic information of the seed entities. Despite that this circumvents the problem of polysemy words, there is no guarantee that each solution is represented in the exact semantic matches. Especially in smaller corpora where candidates only appear very sparsely in the whole corpus, it seems impracticable that there would be enough common contexts. Therefore, the semantic information of the seeds itself should also be considered for the ESE task.

2.1.1 Word2Vec

Mikolov et al. [6] proposes a neural word embedding representation, often referred to as Word2Vec, which creates a single distributional representation for each word in the corpus. It processes a vast amount of text into a low dimensional word embedding in a reasonable amount of time while preserving the semantic similarities between words. Word2Vec considers only a small context window for each word and can be trained on two different tasks. As illustrated in Figure 2.1, one approach is skip-gram, which predicts the words in the context with only the pivot word as input. The other is called the Continuous Bag of Words (CBOW) approach and here, the model is trained on the context to predict the pivot word. Here, the input and output of the model are represented as one-hot encoding

and the model for both objectives consists of a projection layer and a hidden layer. The hidden layer consists of a single vector for each word in the vocabulary. In this embedding space, words are closer to each other if they share similar semantic properties than others. The vectors of the hidden layers are later used as word embeddings for other downstream applications instead of the one-hot encoding. This procedure creates a valid representation of words that only have a single distinctive meaning. It also makes it possible to pre-train a model on a different and bigger text corpus to get a more generalized representation of each word. Because of that, the use of Word2Vec or advanced models like Global Vectors for Word Representation (GloVe) [7] or fastText [8] improved many NLP downstream task, like word analogy tasks or Named Entity Recognition (NER).

However, the single context-independent representation is usually not desired for many NLP applications and also not for the ESE task. There are a lot of scenarios where a context-dependent representation is beneficial. A common issue while working with single word representations are processing words that have multiple semantic meanings. A trivial example of this is the word ”mouse”, which can be an animal or electric device.

In [3], SetExpander is proposed, which utilities the Word2Vec method for the set expansion task. SetExpander tries to circumvent the issue of a single representation per word by training multiple Word2Vec embeddings with different window size, to extract a representation for each context window. Then, for every representation, the system calculates the cosine distance to the centroid of all the seed embeddings. These metrics are then forwarded into a Multilayer Perceptron (MLP) and ranked by the highest probability to get the most likely candidates.

Based on the work in [5] and [3], CaSE is introduced in [9], which combines both the context information as well as the semantics of the seeds. The candidates are extracted by matching the context like in [5], though it uses a more sophisticated approach to select more meaningful features of the context. Moreover, it also uses a one time ranking of the words by comparing Word2Vec embeddings instead of iteratively increasing the solution.

As mentioned before, for many downstream tasks, it is crucial to have a distinctive semantic representation for every context. Since Word2Vec, there have been multiple approaches to create representation from Word2Vec for each context. In [10], Word2Vec embeddings are used to create a different static word representation for each word sense. But ambiguous word meanings are not the only issues that come with the use of static word embeddings. In [11], it is stated that a unique definition of similarity can not be explained by even monoseme words. For instance, the word ”cat” is closely related to the

word ”dog” but also to the word ”tiger”. Therefore, it depends on the context of the word, which of these proximities should be neglected.

2.2 Contextualized Representations

More recent work on embeddings in NLP focuses on contextualized representations, which represent solely the semantic meaning of the context of a word and not the word itself. Therefore, contextualized representations make it possible to classify a word even if it has never been seen before during training. Also, polyseme words can be distinguished between their multiple meanings for each different context the word appears in. Figure 2.2 shows how the same word can have various contexts and therefore have different proximities to other words.

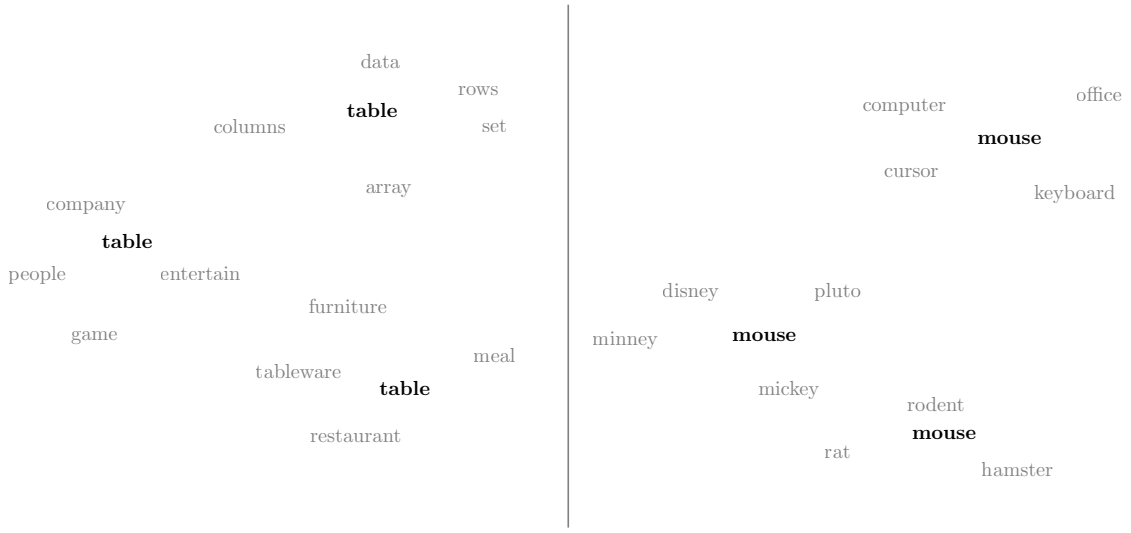


Figure 2.2: Ambigous meaning of words

2.2.1 ELMo

The mentioned approaches in Section 2.1 are either capable to represent the literal context of a word or the syntax and semantic context like Word2Vec. Combining these two elements already showed an increase of performance in [9].

Embeddings from Language Models (ELMo) introduced in [12] is a novel system that creates contextualized representations by employing multiple Long Short-Term Memory

(LSTM) networks [13]. Here, a bidirectional LSTM (biLSTM) is used to create an LM from both sides of the context. This leads to the following probability distributions

$$p(t_1, t_2, t_3, \ldots, t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2, \ldots, t_{k-1}) \tag{2.1}$$

$$p(t_1, t_2, t_3, \ldots, t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, t_{k+2}, \ldots, t_N) \tag{2.2}$$

where the probability for a sequence is predicted based on the previous context tokens. The second LSTM has the same objective but in the reversed order.

ELMo is employing a deep biLSTM architecture, which means that there are several LSTM layers stacked on each other. The output of the highest layer is then used for the training of the LM task. This makes it possible to pre-train the model on a big training dataset in an unsupervised way, which can be reused for many downstream applications. Here, a context-independent representation can be used as input for the ELMo model. Once the model is trained, it can be used to create a representation, as illustrated in Figure 2.3. Here, the output of different layers, called the intermediate states of the model, can be utilized to create a single representation that can then be used as the input for the following application. The embedding is a combination of each of the intermediate states. Therefore, the resulting representation is not representing a single word, it is representing the complete context. This means that if the word itself gets replaced by a word with a similar semantic meaning, the context should barely change. Therefore, this representation is supposed to be reliable even for ambiguous words.

For a downstream application, the ELMo model is applied to create a contextualized representation of each word in the corpus. As mentioned before, this makes it possible to have multiple embeddings for the same word as input. This increases the number of training data significantly compared to static word embeddings. The downstream application is using this representation for supervised objectives, whereby the weights of the ELMo model can either be frozen or trained together with the downstream task. The second method is known as fine-tuning of a model. Because the ELMo weights are pre-trained in an unsupervised manner, but the downstream task is then trained supervised, it is stated in [12] that ELMo is a semi-supervised model.

There have been several approaches that are using biLSTM similar to context2vec described in [14] or Contextualized Word Vectors (CoVe) in [15]. Despite that, ELMo received more acknowledgment. It is one of the few implementations that increased the

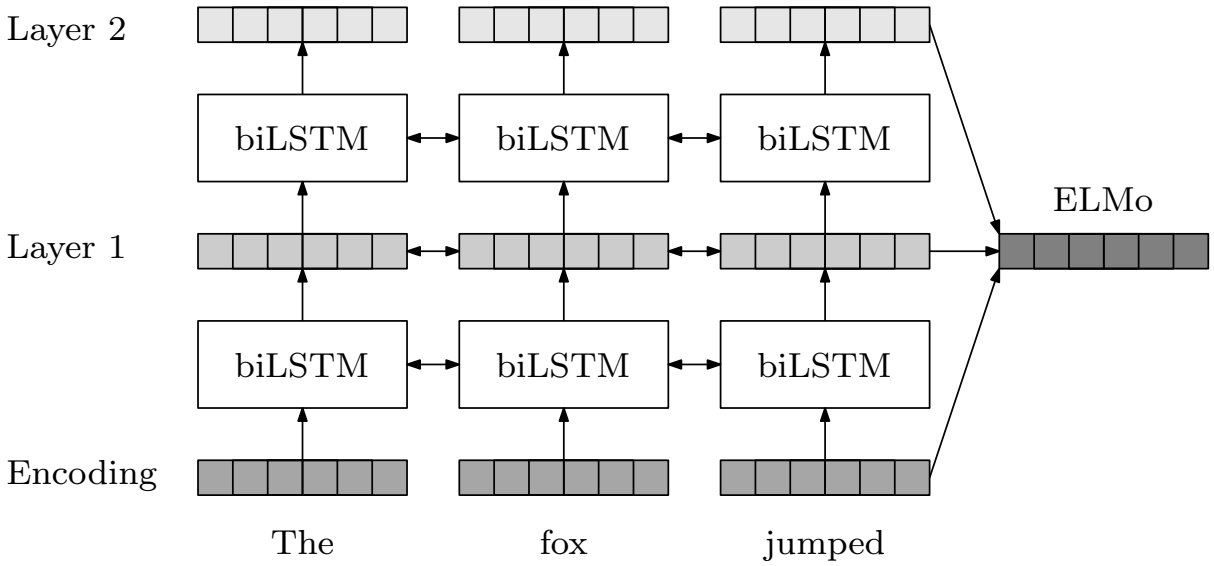


Figure 2.3: ELMo illustration as described in [12].

performance for several different NLP tasks, including question answering, textual entailment, semantic role labeling, NER and sentiment analysis.

Despite that ELMo has led to strong improvements, it still suffers from some major weaknesses. Although LSTMs have been a significant improvement to traditional Recurrent Neural Network (RNN), they are still prone to the vanishing gradient problem described in [16] and it is stated in [11] that the single representation for a context representation causes a bottleneck problem. Also, even with a biLSTM, it is only possible to train an LM in one direction. A recent development named self-attention promises to counteract these problems.

2.3 Attention

Basically, an attention mechanism focuses stronger on particular elements of a given input. For instance, in computer vision, attention might be drawn to a specific area of the image, which is more relevant for a task than the rest. In NLP, attention architectures have been trained on Neural Machine Translation (NMT) objectives. In language translation, one of the key problems is that languages usually not only differ from their vocabulary but also their grammar rules. Therefore, a simple mapping of words in the language is not reasonable.

In [17] and [18], an encoder-decoder architecture is used in which a biLSTM creates a single representation of one word in a sentence and an LSTM creates the translated word. Therefore, all of the information needs to be stored in this single representation, which neglects the information of the contexts. In [19], an encoder-decoder architecture with an additional attention component in-between is proposed. Instead of only processing the representation output of the word, it is taking the weighted sum of all representations in the context. In other words, each weight draws different attention to each word in the context. The trained model should be able to focus on weights that are related to the word and ignore irrelevant ones. For instance, in the sentence "The animal didn't cross the street because it was too tired." for a model, it might not be clear if the pronoun "it" describes the noun "animal" or "street". Especially if a recurrent model is used, it is difficult for a model to infer a connection to the word at the beginning of the text. As illustrated in Figure 2.4, an attention mechanism is able to take words into account, which are more distant. This clarifies the actual meaning of a word, which can further be used in the other language as well.

In [21], this attention is referred to as soft-attention, whereas hard-attention weights

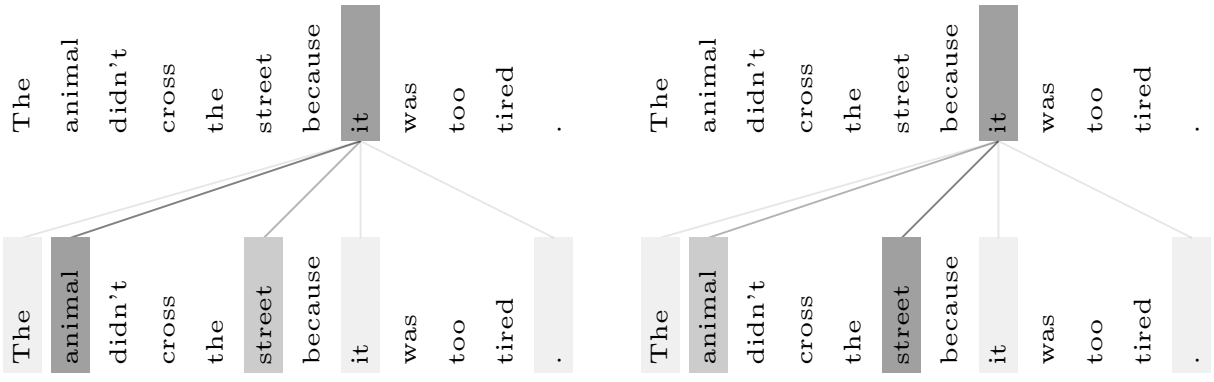


Figure 2.4: An illustration of the attention mechanism as described in [20].

each value as either true or false similar to a one-hot encoding. Here, attention is also divided into global and local attention. While global attention takes the whole document into account, local attention only processes a small context window. In [22], a model is proposed that applies attention weights to the intra-relations of the current sequence. This attention network is described as intra-attention, or nowadays usually referred to as self-attention, is able to infer based on words that are distant in the previous sequence. Although adding an attention mechanism to a recurrent network reduces the mentioned bottleneck problem as well as the vanishing gradient problem, the architecture still suffers from these problems caused by the LSTM.

2.3.1 Transformers

In 2017, Vaswani et al. [23] introduced the Transformer, which is an NMT encoder-decoder architecture, without any recurrent or convolution layers. Instead, it employs so-called self-attention layers, which do not process the context sequentially but utilize the whole context unidirectional at once. For that, it uses the encoded input and applies the dot product to three weighted matrices, subsequently generating three different matrices called query, key and value.

In many systems, for instance, a database or a search engine, a certain query request is used to retrieve a value. Here, the query is matched with the most likely key, which then returns the corresponding value. In self-attention, the key and query should amplify the corresponding value. Because the relationship between query, key and value is barely elaborated in the paper and also due to that the Transformer is a relatively new technology, there is currently no contemporary literature that covers the Transformer architecture in detail. Therefore, the work also refers to the content of the blog article in [24], which deconstructs the elements of the Transformer.

The weight matrices for query, key and value have the same shape, which is composed of the sequence length times a predefined size for the embeddings. Each of these matrices is multiplied with the input embedding, which creates the corresponding matrices Q, K, and V. These matrices are combined as follows:

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.3}$$

where K and Q are multiplied together to get the magnitude of the attention for every word to each other. Taking the dot product of similar vectors lead to the highest value,

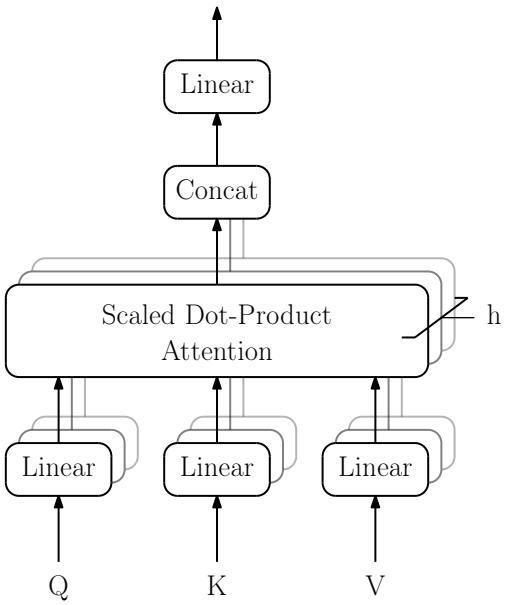


Figure 2.5: A multi-head attention layer as described in [23].

while vectors pointing in the opposite direction to the lowest negative score. To reduce the magnitude of strong attention, the score is normalized by the dimensionality of the embeddings. This score is put into a softmax function to get a probability distribution of the words, and, therefore, the weights for each of the words. These weights are then multiplied to the matrix V to get the attention score for each of the self-attention layers. As a result of this architecture, the three weight matrices are the only trainable parameter of the self-attention layer.

Vaswani [23] states that a single self-attention layer can not consider all facets of the attention. A single layer would need to take all characteristics of a sentence into account and thus, the resulting attention would be an averaged attention lacking significant information. To counteract this, the Transformer is employing, as illustrated in Figure 2.5, multiple self-attention layers, also referred to as attention heads. Here, each attention head creates a smaller embedding, which is then concatenated into a single representation. The model proposed in the Transformer paper consists of 8 attention heads. Therefore, each head is creating an embedding with 64 dimensions so that the resulting embedding has a dimensionality of 512 again.

This multi-head attention component itself consists only of matrices calculations and linear functions. Therefore, it is not possible to train the parameter of the multi-head attention layer without adding any additional non-linear component. Because of that, a feedforward network is added to each multi-head attention layer in the architecture.

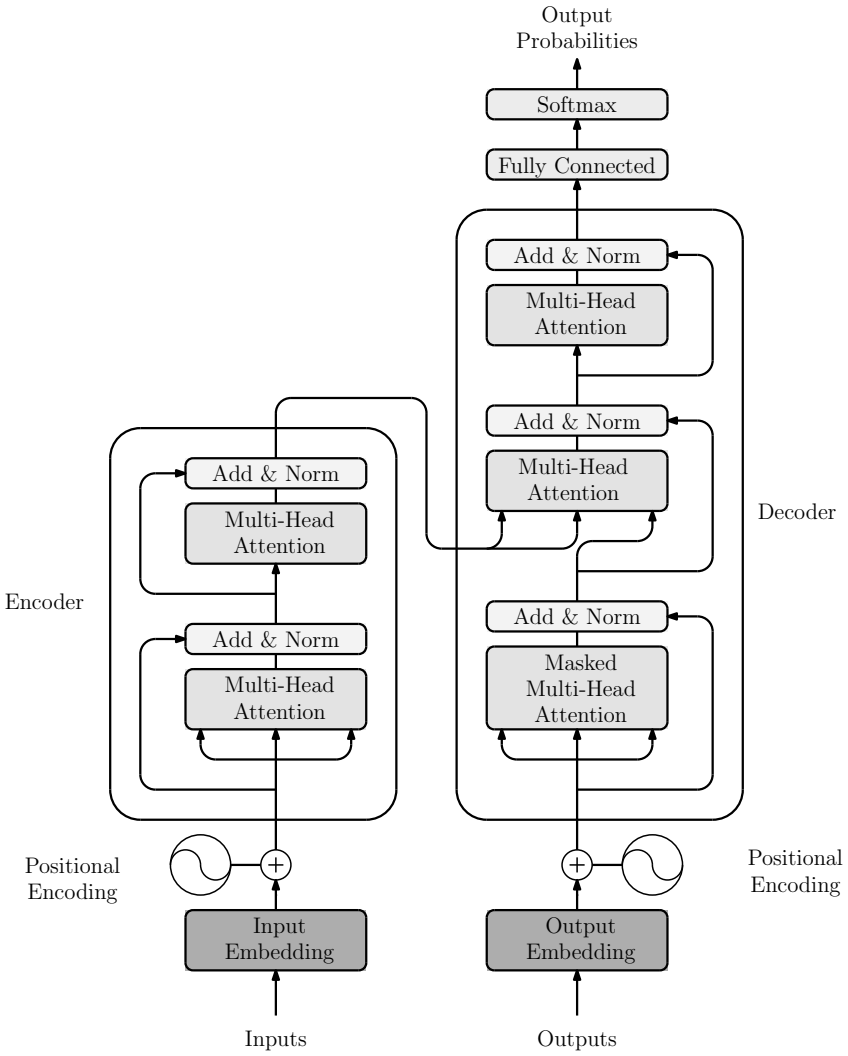


Figure 2.6: The Transformer architecture as illustrated in [23].

As illustrated in Figure 2.6, the final architecture employs these multi-head-attention layers in N encoder and decoder blocks. The encoder utilizes the input embedding and the decoder the output embedding. Depending on the task, the output embedding can be equal to the input or, for instance, in language translation, it can be the same text but in a different language. The overall architecture of the transformer encoder consists of one multi-head attention, a Fully Connected (FC) layer and for each of them a normalization layer. The normalization layer reduces the co-variate shift by scaling the output of the components.

The decoder of the Transformer, illustrated in Figure 2.6, is predicting the next token based on the output of the last encoder layer, also referred to as the intermediate state, and the output sequence before the current element. Therefore, the decoder adds an additional multi-head attention layer to the input, which masks all future words. Also,

the normalized output of this masked multi-head attention layer is the input embedding for the value and the intermediate state is the input for query and key. The output of the last decoder block is then forwarded into a single dense layer to calculate the probabilities for the next word in the sequence. To be able to efficiently train the model to predict a sequence, a method described by Goodfellow et al. [25] as teacher-forcing is applied. Here, instead of adding the predicted word to the output embedding, the correct solution is added. Therefore, the model is able to learn to make correct assumptions. At test time, with no reference output available, the model prediction is concatenated to the sequence.

In a model that merely consists of non-recurrent elements, positional information about the input sentence is not taken into account at all. Thus, the model is not able to keep track of the relative position of the embeddings in the sentence, which might be useful information for inference. Therefore, the authors add an additional positional embedding to the input representation. Instead of using simple positional information like incrementally increasing a counter, the Transformer employs the two oscillating functions

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{2.4}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{2.5}$$

where PE is not a scalar value, but an embedding of the size of the model. Also, the first Equation 2.4 is concatenated to all even and Equation 2.5 to all odd embeddings. The authors choose these functions since they assume that the Transformer is able to process longer sequences in testing than in training with such a positional signal. It is worth considering if adding the positional embedding to the input representation would distort the information of the input. Usually, concatenating the embedding avoids this issue. However, it seems that adding the positional embeddings has no negative effect on the final results.

Also, even after adding this encoding, the positional information are distorted after each layer. Therefore, to keep this positional encoding information also in the deeper layers, each normalization layer is connected with a residual connection to its input.

Vaswani et al. [23] states that the Transformer does not only exceeds the performance of many state-of-the-art systems. Its architecture can be better trained in parallel than recurrent approaches because LSTMs are processing the inputs sequentially, making it

difficult to run them efficiently on a GPU. The Transformer circumvents this problem by reading in the complete input at once. Also, using a Transformer architecture has direct effects on the run-time of the model. While each layer of an LSTM has a computational complexity of $\mathcal{O}(n \cdot d^2)$, where n is the sequence length and d the dimensions of the embedding, the complexity of a self-attention layer is $\mathcal{O}(n^2 \cdot d)$. Although longer sequences have a stronger negative impact on the run-time of the Transformer architecture, it can be used for much deeper architectures.

2.3.2 Implementations of the Transformer

Because the Transformer architecture introduced many innovative concepts, it has inspired a new area of research. Recent work is experimenting with the components of the Transformer in a variety of ways to improve the performance for specific downstream applications. For instance, a Transformer architecture called Transformer-XL is proposed in [26], which can process much longer context windows by adding recurrence to the Transformer and therefore mitigating the run-time problem of long input sequences aforementioned.

In 2018, researchers of the institution OpenAI proposed in [27] a Transformer network, which is nowadays known as Generative Pretrained Transformer (GPT). The authors propose a transformer network, which is merely using the decoder blocks of the original paper. Furthermore, the model is composed of only the masked multi-head attention layer, due to that it receives no intermediate state from an encoder block, which makes the second multi-head attention head obsolete. Here, similar to the ELMo model described in Section 2.2.1, the model is initially pre-trained on the LM task to increase the performance of the system. Later, the pre-trained system is fine-tuned for each downstream task.

However, all of these models are processing the input in an undirectional LM, where the task remains directional. To counteract this issue, BERT offers a different approach, which leads to significant improvements.

2.4 Bidirectional Encoder Representations from Transformers

In 2018, a language representation architecture called BERT had drawn major attention because it added only some simple conceptional modifications to the Transformer model, which lead to significantly better results for wide fields of tasks like question answering, sentence-pair completion or the General Language Understanding Evaluation (GLUE) score described in [28]. The BERT model is introduced in [29] and in contrast to the vanilla implementation of the Transformer, the model solely consists of encoder blocks to create contextualized embeddings. Since the findings on BERT are relatively recent, the following elaboration relies strongly on the original publication of BERT described in [29].

As mentioned before, contextualized embeddings of a pre-trained model have successfully been used in a feature-based approach, like in ELMo described in Section 2.2.1 or a fine-tuned matter like in GPT described in Section 2.3.2. Although BERT can be applied for both of these approaches, the model is often fine-tuned on a specific task. As mentioned in the sections before, the predecessors of BERT are only trained on a directional LM task, which is contrary to the actual concept of context in text. Finding a suited nondirectional task is more difficult than it seems in the first moment because observing a task like next word prediction from both directions at the same moment would already give away the solution.

The BERT paper proposes a training method that combines two different tasks. The first task is inspired by the so-called cloze task, which describes the challenge of predicting masked out words in a text. Therefore, the model is referred to in the paper as Masked Language Model (MLM). This task makes the training completely undirectional and thus, the model learns both sides of the context. The second task is the so-called Next Sentence Prediction (NSP) task. Here, two sentences are given to the model, and the classifier decides whether the second sentence is subsequent to the previous one. The original paper states that LM lacks the ability to capture relationships between sentences. Adding the NSP task should counteract this issue. This statement is further discussed in Chapter 5. Both tasks make it possible to pre-train BERT in a completely unsupervised manner on an unlabeled text corpus. The paper describes that although MLM have a marginally longer convergence time compared to classical left to right models, they increase the performance significantly.

Despite its slightly misleading name, a sentence in BERT does not necessarily represent a valid sentence but rather a longer text sequence. To keep the sentence consistent with the paper, the phrasing sentence is also used in the work.

2.4.1 BERT Input

As depicted in Figure 2.7, BERT requires a modified input to train on the two described tasks. The authors convert the input to a specific sequence, referred to as the token embedding. Every sequence starts with a classifier token, the two sentences are separated by a separation token, as well as the second sentence ends with another separation token. As an addition to the MLM task, 15% of the tokens are predicted during training simultaneously. From these training tokens, 80% are replaced by a mask token, 10% by an arbitrary token and for the remaining 10% the tokens are not replaced at all. This permutation increases the bias towards the actual word, while other terms still cannot be neglected completely.

In contrast to the previously described models, BERT does not use the representation of the whole words as input. As it is inefficient to train a model on a vocabulary that contains any possible combination of characters, BERT utilizes subword-tokens instead. Therefore, words with similar meaning like ”reading”, ”reads”, ”reader” can be split up into ”read” and the suffix of the word and subsequently trained together. To tokenize words into subwords, BERT employs the pre-trained WordPiece model. As described in [30], the WordPiece model has a fixed vocabulary that is created once. At first, the model is initialized with all possible characters to create the complete vocabulary. Therefore each word is tokenized into single characters. Next, an LM is trained on these tokens and all possible combinations of the vocabulary are combined together to predict the most likely solution for the LM. This token-pair is then added to vocabulary and this process is repeated until the vocabulary reaches a predefined size. This model can then be used to tokenize words. If a word is still not present in the vocabulary, it gets split up into smaller subwords. Therefore the WordPiece model can process any arbitrary word.

Because the WordPiece model is pre-trained, the segmentation of the words is deterministic, which means that a word, no matter in what context it appears in, will always be tokenized the same. Alternatively, in [31], the SentencePiece model is described, which tokenizes words based on the sentence they appear in. Therefore, SentencePiece is only deterministic if words are used in the same context. However, SentencePiece requires only a fraction of the vocabulary size to create similar results.

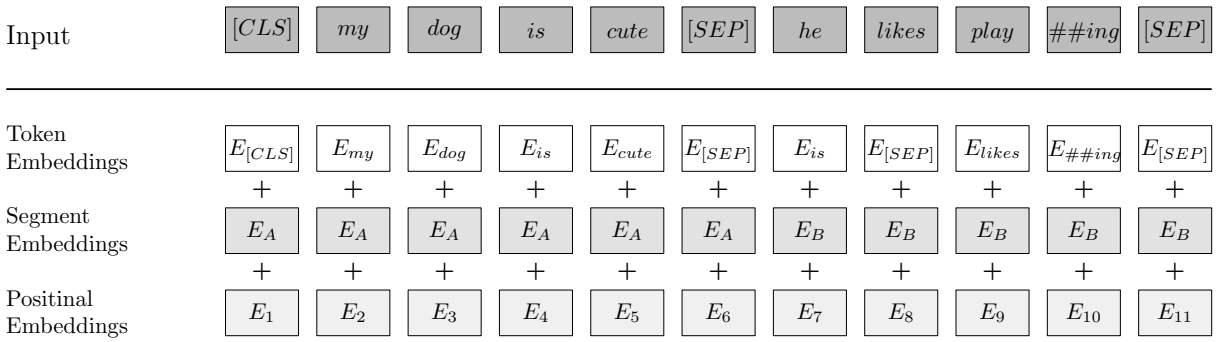


Figure 2.7: BERT token input illustration as described in [29].

Similar to the Transformer architecture, a positional embedding is added to the tokenized embedding. While the positional embedding serves the same purpose for BERT as in the Transformer described in 2.3.1, it is not using a predefined signal. Instead, the positional embedding of BERT is randomly initialized and subsequently trained to create compatible positional embeddings for the model.

The authors of the paper also add the so-called segment embedding to increase the ability to distinguish the input between the two sentences based on the separation token. There are only two different types of segment embeddings. The first is added to each token before the first separation token and the second for the rest of the tokens. Both are randomly initialized embeddings with the shape of maximum sequence length times the size of the embedding.

This complete input is forwarded through the BERT model to create a contextualized embedding for each input token, including the special tokens.

2.4.2 Fine-Tuning BERT

After the complete BERT model is trained on the NSP and the MLM task, the model can be applied to different downstream tasks. For that, the pre-trained model is extended by other components to be able to fulfill the task. For instance, as illustrated in Figure 2.8, in NSP classification, an additional FC Layer is added on top of the output of the last layer of the model. Since the classifier token represents a combination of all the token embeddings in the previous layer, it is possible to neglect all other output embeddings of the sequence.

It is currently unknown how the classifier token can represent the complete sequence. One

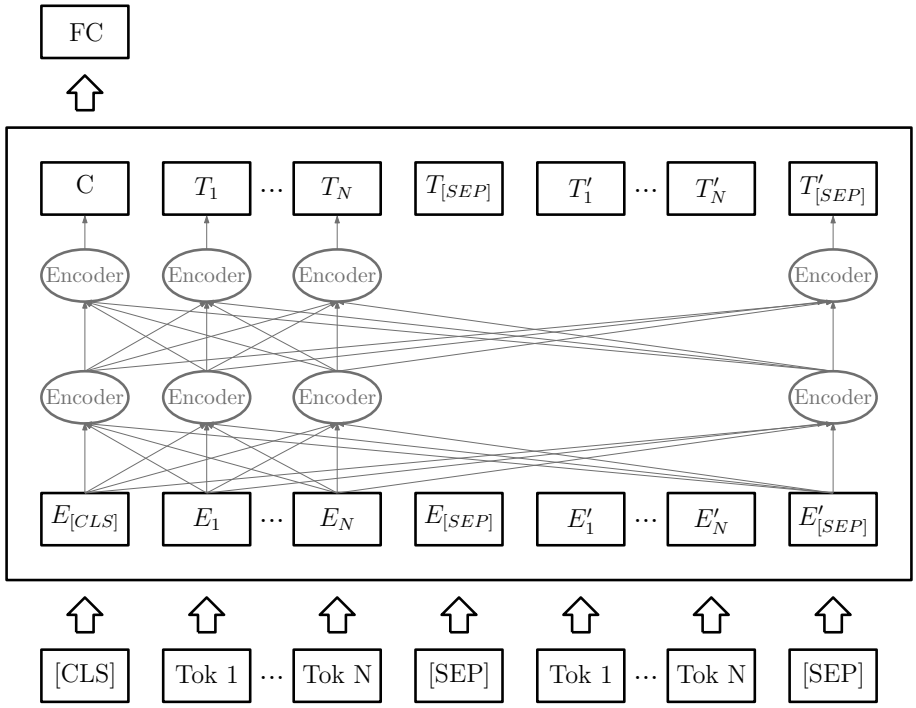


Figure 2.8: BERT fine-tuned for sentence classification described in [29].

of the authors of the paper Jacob Devlin pointed out[1] that it is unclear how the classifier token is representing the sentence. He suspects that the classifier token is some kind of average pooling over the embeddings of the sentence. Therefore, it seems not reasonable to compare the representations of solely the classifier token of a different sentence like it is done with words in Word2Vec described in 2.1.

For other tasks like NER, the pre-trained BERT model is fine-tuned differently. Here, the output embedding of each token is used for the classification because the system is not classifying the complete sequence rather than certain subword sequences of the sentence. In contrast to the classifier token, single subword tokens can be directly compared to other subword tokens.

During training, the weights of the BERT model can either be frozen and only the weights of the FC layer can be trained or the complete model can be trained for the downstream task. Because BERT consists of many trainable parameters, the first approach does require fewer resources, while training the complete model does not necessarily lead to better results.

[1] https://github.com/google-research/bert/issues/164#issuecomment-441324222

2.4.3 Feature-Based BERT

Besides fine-tuning the complete BERT model for a specific task, it is also possible to extract token representations from different layers. The original paper states that BERT can also be used as a feature-based model for downstream tasks similar to ELMo described in Section 2.2.1. For this approach, the weights of the pre-trained model are frozen and after the input is forwarded through the model, the embeddings can be extracted from each layer. These embeddings are latter used for the downstream task. It is also possible to combine multiple of these intermediate embeddings, which can achieve even better results than with embeddings from the top layer.

In order to provide a better understanding of why some layers might be more suitable for certain downstream than others, the attention weights for the layers are further elaborated. In [32], the behavior of the underlying attention mechanism for different layers of BERT is investigated. The authors state that attention heads can capture syntactic and co-reference attributes without any label data during training. This might be an indicator, why the performance of BERT exceeds other models. They also describe that attention heads in the same layer often focus on these attributes in a similar way. While attention heads in the early layers focus more generally on all tokens, the later layers focus stronger on specific tokens or patterns. For instance, attention heads in the last layer focus strongly on the punctuation tokens of a sentence. Despite that this behavior might be important for the NSP task, it could also explain why other downstream tasks perform better from features from previous layers.

2.4.4 BERT Architecture

As mentioned before, the BERT architecture is closely related to the original Transformer architecture. However, because BERT is only creating the contextualized embedding of the input sequence, it is only using the encoder blocks. Also, BERT increases the number of parameters to be able to create more extensive representations. The original BERT paper proposes two models, the BERT base model and the BERT large model. As shown in Table 2.1, the base model has, to make it more comparable, the same parameter as the GPT model described in Section 2.3.2. The BERT large model does require much more parameter and thus more resources to train. Still, this leads to a remarkable increase in the performance of the system and therefore, it might be reasonable to choose the more complex architecture after-all.

Parameter	GPT	BERT_Base	BERT_Large
Encoder Blocks	12	12	24
Embedding Dim	768	768	1024
Attention Heads	12	12	16
FC Neurons	3072 & 768	3072 & 768	4096 & 1024
$\sum$ Parameters	117 million	110 million	340 million

Table 2.1: Parameters of Transformer based models in comparison as described in [29].

The BERT paper also states that a higher number of parameters consequently increases the performance of the model. Recent work investigating the complexity of the model showed that a model with much less parameter could lead to similar results. Also, a model with the same number of parameters can be utilized more efficiently to exceed results. In [33], a BERT architecture called distilBERT is proposed. This model requires only 60% of the parameters, making it 60% faster, while 97% of the original performance is retained. Here, the original BERT model is distilled, which means that the model with less parameter is trained based on the results of the original model and, therefore, only recreates the model's behavior. While the final model is much smaller, it still requires a pre-trained BERT model, as training the distilBERT architecture from scratch does not lead to satisfying results.

The rapid improvements of the BERT model have also been applied to previous architectures to fortify the approaches. The aforementioned GPT model is further optimized in [34], now known as GPT-2. As a comparison, the GPT-2 architecture adopts the idea of a tokenized input, but instead of using the WordPiece tokenization, it applies the Byte Pair Encoding (BPE) algorithm described in [35]. The algorithm is in essence identical to the to WordPiece, with the exception that it adds the most frequent subword pairs instead of the most likely one.

3 Methodology

After elaborating over the recent developments in the field of NLP, from static word embedding to contextualized BERT models, this chapter proposes a novel ESE pipeline. This system combines the previously mentioned technologies which exceed the performances of all other compared implementations. Further, the pipeline is able to run as a real-time application on any text corpus, after the corpus is pre-processed. This chapter describes the inner-working of each component of the pipeline and how they are finally used in an application.

As mentioned in the previous chapters, an ESE system is composed of two major components, a candidate extraction tool as well as a candidate ranker or classifier. Despite that this structure leads to descent results, the described methods suffer from extracting entities only based on lexical patterns in the corpus, especially in small datasets. This work proposes a system that extracts all possible named entities in the corpus. Therefore, the recall in the extracting step of entities should be maximized. Unfortunately, the system extracts a humongous amount of candidates, which can not be classified by any state-of-the-art classifier. Thus, a new component, an entity reducer, is added to the system depicted in Figure 3.1 to remove unlikely candidates and therefore be able to employ any classifier. For this purpose, this work processes contextualized embeddings to reduce the number of candidates as far as necessary and employs a more sophisticated classifier, a fine-tuned BERT architecture, to rank the remaining candidates. The first tests described in Section 4 on two datasets have already shown the potential of such a setup.

The following sections describe each of these three components in more detail and elaborate on how the state-of-the-art technology explained in the previous chapter has been applied to solve this task. It is also briefly discussed why certain technologies have been chosen and also the reasons why certain setups lead to better results.

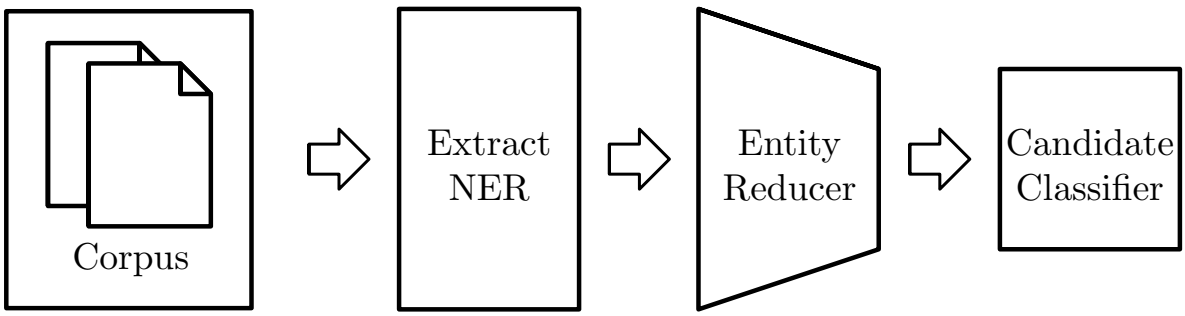


Figure 3.1: Entity Set Expansion pipeline

3.1 Problem Definition

In this work, the ESE problem is also defined as extracting a list of named entities $\hat{E}$ from a predefined text corpus T. The pre-processing of the list $\hat{E}$ resolves into a set of entities E. Further, the system is reducing the vast number of entities E to a candidate set C_S based on a query seed set S, while $C_S \subset E$ and $C_S \nsubseteq S$. After that, the candidates C_S are ranked and the top n_c candidates are selected as set C_E. Finally, the seed set is expanded as $S_{expanded} = S \cup C_E$.

3.2 Extraction of Named Entities

As described before, the first step of the system is to retrieve all possible named entities $\hat{E}$ from the text corpus. Because this can be a time-consuming task, it is important to use a lightweight module, which is still extracting enough relevant entities. Due to that, it seems beyond the purpose of this work to implement a complete NER tool. Thus, a third-party module is chosen to extract the entities from the text corpus. Here, the module spaCy[2] is included in the system, which applies a Convolutional Neural Network (CNN) architecture with an attention mechanism. Since CNN implementations are highly scalable, spaCy has a fast run-time and in addition to that, it offers an intuitive Application Programming Interface (API) to integrate the module into other systems. It also achieves similar recall values as, for instance, the well-known Stanford Core NLP Toolkit[3], which employs a Conditional Random Fields (CRF) architecture described in [36]. For further details on using CNN in NLP, the reader is referred to [37].

The extracted seeds $\hat{E}$ are then further pre-processed to filter out obvious misclassifications and also duplicate entities are merged together, resulting into a smaller set E. Though the reducer element filters out implausible candidates, it still important to scale the number of candidates as good as possible, considering each of these candidates has a direct effect of $\mathcal{O}(\#E)$ on the run-time of the subsequent system.

[2]https://spacy.io/
[3]https://stanfordnlp.github.io/CoreNLP/ner.html

3.3 Feature-Based BERT Candidate Reducer

As elaborated earlier, the entity extraction component retrieves a vast amount of candidates. Many of these candidates E are valid named entities. However, they are not related to most of the seeds in the set S. Therefore, the vast number of candidates severely slow down the system and can also impair its precision. Because of that, an ESE system benefits if a scalable component is added, selecting only candidates related to the given seed sets.

A common way to solve such a problem is to group data with a fast clustering algorithm. To be able to cluster the candidates, it is first necessary to create some kind of representation for the candidates. Here, many implementations use Word2Vec embeddings described in Section 2.1 to create a static embedding for each candidate. However, first experiments with these representations do not lead to satisfying results.

In the following section, the more novel BERT embeddings are further investigated to approach this problem. However, it is not possible to directly use BERT as representations for specific words. As described in Section 2.2, contextualized embeddings from models like BERT are representing the examined context, rather than single words. A straightforward approach to create a representation for the words with a contextualized model would be to merely use the tokenized candidate without any context as input and create a joined representation out of the output tokens. Nevertheless, this approach completely contradicts the concept of contextualized models and would not lead to satisfying results. Therefore, it is essential to process the full context of a candidate, even if only a few tokens are relevant.

In [38], a baseline is implemented for the ESE task, which classifies contextualized BERT embeddings with a K-Nearest Neighbors (KNN) classifier to select the best candidate. Here, the context of every appearance of an entity is joined to a single embedding. Intuitively, creating a single embedding out of many contextualized embeddings seems to stand in contrary to the basic concept and the benefits of contextualized embeddings elaborated in Section 2.2. However, it is described in [39] that although contextualized representation can scatter over the complete 768-dimensional embedding space, words from a similar context still tend to cluster in a certain area of the space. Based on this assumption, words can be grouped together depending on their position in the embedding space, but especially for polyseme words, losing information might be unavoidable.

Since candidates usually appear several times in a corpus and the WordPiece model splits up the candidates into several tokens, it is necessary to join these words somehow to-

gether. In the proposed pipeline, these embeddings are united by simply averaging all of these embeddings to keep the classifier as fast as possible, instead of using a more complex pooling method. At first, averaging high dimensional embeddings appear to be too trivial. Nonetheless, as elaborated in [40], averaging WordPiece token leads to the best results, compared to other approaches like max-pooling or just selecting the first token. Furthermore, as mentioned in Section 2.4.2, one of the authors suspects that the classifier token in BERT is merely applying something similar as an average pooling over the complete sentence. There has also been work which inspected the effect of averaging a different kind of embeddings. Here, as described in [39], it can be sufficient to just average unrelated embeddings.

Based on these findings, in the reducer, each appearance of a candidate is forwarded through the BERT model and the corresponding embeddings of the candidates are then averaged to a single embedding, which are then ranked by a KNN classifier. First implementations of this concept already showed some adequate results, although the system misclassifies entities that are quite similar to each other. However, this setup barely loses any recall by reducing a lot of unrelated entities to only a few candidates. These findings indicate that this feature-based BERT approach can also be used as a very fast entity reducer.

As introduced earlier, it is also possible to select the intermediate embeddings for the BERT model for downstream tasks. To investigate this idea further, this work analyses the performance of each layer of a pre-trained distilBERT model in Section 4.3.2. The gathered results show that processing the embeddings from an intermediate layer of the model, instead of the last layer, increases precision significantly. Therefore, the final model uses the embedding of the fourth layer from the distilBERT model.

3.3.1 Implementation of the Reducer

The following section describes the implementation of the reducer in more detail. For that, Algorithm 1 outlines the procedure in a simplified form in pseudo-code.

Since the BERT model can only process a fixed sequence length, the input is utilized in sentences, as shown in line 2. This makes it also possible to run the following steps of the reducer in parallel. As described in line 3, the whole sentence is tokenized into a list by the WordPiece model, described in Section 2.4.1. If the length of this list exceeds the BERT model's predefined input length, this list is split into smaller parts. The tokens are then forwarded through the model, which results in a list of embeddings.

Algorithm 1 BERT KNN Reducer

Input:
E ... Extracted Entities
S ... Query Seeds
T ... Text Corpus
n_c ... Number of Candidates
Output:
C_S ... Set of Candidates

```
1:  function CREATE CANDIDATE SET
2:      for sentence ∈ T do
3:          tokens ← [CLS] + WordPiece(sentence) + [SEP]
4:          embeddings ← BERT(tokens)
5:          for e ∈ E do
6:              if sentence contains e then
7:                  posistions ← find WordPiece(e) in tokens
8:                  sumemb_e ← sumemb_e + sum(embeddings[positions])
9:                  counter_e ← counter_e + length(positions)
10:     for e ∈ E do
11:         avgemb_e ← sumemb_e / counter_e
12:     for s ∈ S do
13:         C_S ← C_S add nearest_neighbors(s, e_i, k = n_c) : e_i in E ∧ E ⊈ S
```

Because named entities in the input are usually split into several tokens, it is necessary to assign the entities to the tokens they are composed of. Thus, it is possible to trace the embeddings to the entities later. As shown from line 5 to line 7, the token positions are obtained for every entity which appears in the sentence. In line 8, the embedding of each position of the entity is summed and subsequently added to another embedding. After the complete text corpus is processed, in line 11, the summed embedding for a specific embedding is divided through the number of appearances of the entities in the overall text corpus to get a single averaged embedding for the entity finally.

Subsequently, these averaged embeddings can be utilized to find the n_c nearest neighbors in a similar way as for word embeddings explained in Section 2.1.1. Here, the euclidean distance is calculated instead of other similarity measures like the cosine distance. Evaluating the cosine distance can not be optimized and thus requires a brute-force approach with a run-time of $\mathcal{O}(N^2)$. In contrast, the run-time of the euclidean distance can be reduced by applying an optimization algorithm like the ball algorithm described in [41], which leads to a run-time of $\mathcal{O}(N \log N)$. The run-time of the reducer makes a significant difference in the run-time of the whole pipeline. The number of entities might increase significantly, severely impacting the usability of the entire system. Also, the reducer al-

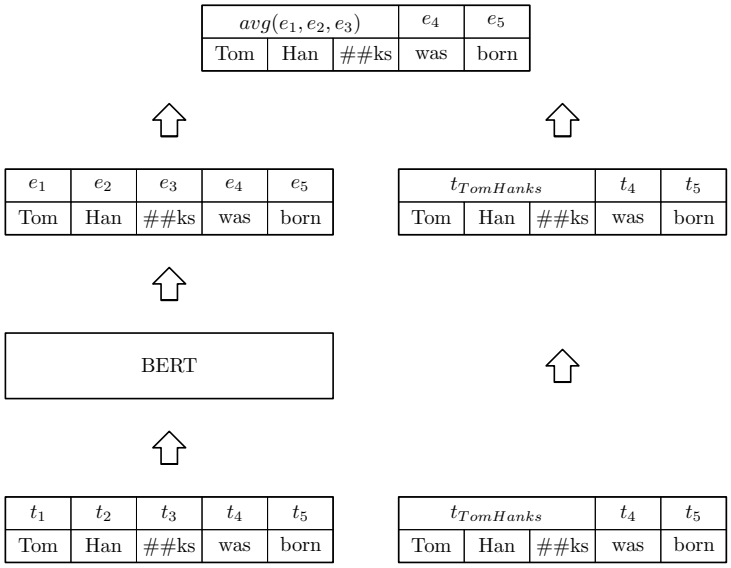


Figure 3.2: Lookup of entities split into several WordPiece tokens.

ways mitigates the set of candidates to almost the same number of entities, no matter how many entities are given as input. Therefore, the run-time of the classifier is not affected by the actual number of inputs. After checking for redundancies in the set, the candidates are used for the classification task.

The previously described algorithm is merely a simplification of the actual implementations and therefore, it does not consider many details. For instance, in the implementation, a lookup list is created, as illustrated in Figure 3.2, to make it more efficient to find the embedding of the entities in the tokenized list. Here, the tokenized list *tokens* are duplicated. For each position of an entity, the WordPiece token is replaced by a unique token id for this entity, which is higher than the highest valid WordPiece token id. After the original tokens are forwarded through the BERT model, it is only necessary to consider tokens that exceed the WordPiece model's vocabulary and average them.

Further, the reducer is divided into several steps to make the system more efficient. At first, the complete text corpus is tokenized and all of the entities are matched within the tokenized list. The following step is to calculate the average embeddings and subsequently, the last step is to find the nearest neighbors.

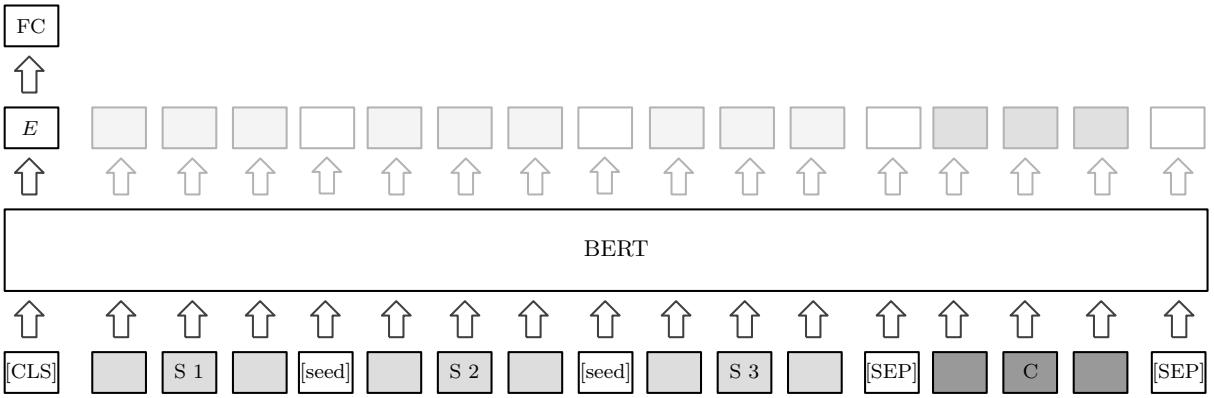


Figure 3.3: Input for fine-tuned BERT classifier.

3.4 Fine-Tuned BERT as a Classifier

As established in Section 2.4, recent work showed promising results in employing a fine-tuned BERT model as a classifier. This work, therefore, explores the potentials of using such a model as a final classifier for the ESE task.

As explained previously, the performance of Transformer networks like BERT benefit if the information from the context is utilized. Thus it is important to process candidates with contexts they appear in instead of simply using only the entities. Therefore, each entity is extracted inside a particular context window to capture a specific context. Here, the size of the window is an important parameter. While too small windows do not capture enough information about the context, bigger context windows might capture too much unrelated context information and thus lead to misleading inferences. It is also important to capture the context information of all the query seeds, due to that a single seed can usually not be addressed to only one semantic group. Another requirement is that a classifier should also be able to process multiple inputs without changing the architecture or even retraining the system to a specific number of input seeds.

To fulfill all of these criteria, this work proposes a fine-tuned BERT model, illustrated in Figure 3.3. Here, the input for a BERT classifier consists of several seeds in a textual context as the first sentence and one candidate in a context as the second sentence. The context is extracted in the same step as the reducer extracts the average embedding for the candidates. Additionally, a new special token, a seed token, is placed between each of the seed contexts. This token supports the model to recognize different contexts and make it more convenient to vary in the number of input seeds without retraining the complete model. Therefore, the candidates are classified based on their context, while the number of input seeds remains flexible.

It is quite common that entities appear multiple times inside a corpus. An intuitive approach would be to predict the probability of a candidate by combining each combination of the input context and then, for instance, averaging the results or choosing the highest value. For example, if a query consists of 3 seeds and each entity appears 100 times in the corpus, the system needs to predict the value 1.000.000 times before making a final classification. Hence, only a few contexts should be selected from all possible contexts. Experimenting with an additional classifier to predict the best context has not improved the results. Therefore, the textual contexts of the values are chosen randomly.

To be able to fine-tune the described model, it is necessary to add a classifier to the model. For that, a simple FC layer is used to process the embedding of the classifier token. The training of this architecture is based on the Cross-Entropy (CE) described in [42], which is in essence defined as

$$CE(x) = -\sum_{k} p_k(x) \cdot log(q_k(x)) \tag{3.1}$$

where p describes the actual distribution and q the prediction distribution for each class k. The model itself is trained on the so-called Binary Cross-Entropy (BCE), which is merely considering two possible classes, 0 and 1. For the optimizer, Adam, with an initial learning rate of 0.00002, is applied.

Since each test set only consists of a few entities, it is necessary to sample the dataset. Otherwise, the model learns to classify all entities as 0. Because of that, in each epoch, only twice as many incorrect entities are selected then the correct ones. It is also possible to weight the given contexts to circumvent the class imbalance problem. The BCE loss in the implementation uses the loss function from the PyTorch library[4]. This offers a parameter to weight the impact for specific input. This would make it possible to train one context of every entity in each epoch.

[4]https://pytorch.org/docs/master/generated/torch.nn.BCEWithLogitsLoss.html

3.5 Running the Pipeline in Production

The proposed system can also be implemented as an actual application. Here, as illustrated in Figure 3.4, the candidate extraction step, as well as the embedding and context extraction part from the reducer, can be considered as pre-processing. It is only necessary to run this pre-processing step initially a single time. The subsequent steps, the KNN reduction of the candidates and the actual BERT classifier, are considered the application. As mentioned before, these are the only necessary elements to train the classifier and latter to expand the seed set. Since the pre-processing consists of the by far most time-consuming steps, the actual application is quite fast.

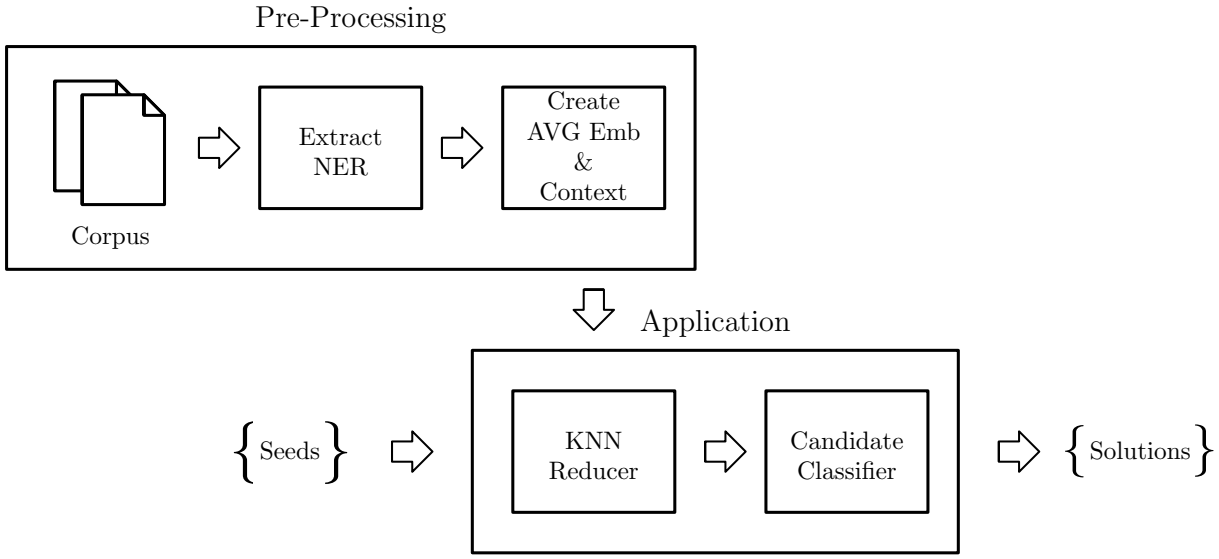


Figure 3.4: Proposed Entity Set Expansion pipeline in two steps.

4 Experiments and Results

This chapter gives an overview of the conducted experiments and the corresponding results. First, the hardware and the dataset used for the tests are explained in more detail. Furthermore, this chapter analyses the candidate extraction component in Section 4.2, the subsequent candidate reducer component in Section 4.3 and the fine-tuned classifier in Section 4.4. The results of this pipeline are compared to a baseline introduced in Section 4.5 to show the final classifier's impact. The actual results are then shown and analyzed in Section 4.6.

The following results are implemented and executed on two different clusters provided by the Center for Intelligent Information Retrieval, which is a research facility of the University of Massachusetts Amherst.

Especially for a bigger dataset with many named entities, applying the whole pipeline can require a lot of resources. Therefore, the pipeline is split up into smaller steps and all elements of the pipeline except for the classifier can be run in parallel on different clusters.

	Gypsum	Swarm2
number nodes	25	100
GPU	Tesla M40	-
CPU	Xeon E5-2620 v3	Xeon E5-2680 v4 Xeon Gold 6240
number CPUs	5	10
GB/CPU	10	15

Table 4.1: Clusters used for the experiments.

The two clusters described in Table 4.1 are used for the experiments. The Gypsum cluster can run jobs on a GPU, while the Swarm2 cluster is designed solely to run CPU intensive jobs. Therefore, to limit the CPU load on the Gypsum cluster, the spacy NER with the subsequent pre-processing, the tokenization of the sentence, the NER lookup in the sentence and the context extraction required for the classifier are executed on the Swarm2 cluster. The remaining steps, the forwarding of the tokenized sentence to create the averaged embedding and the final training and testing of the model, are run on the Gypsum cluster.

The system itself is implemented in Python with the following third-party software tools:

Software	Vesion
Python	3.7.4
CUDA	9.2.88
cuDNN	5.0
spaCy	2.2.4
scikit-learn	0.22
Hugging Face Transformers	2.5.1
PyTorch	1.4.0

Table 4.2: Information about software used in the experiments.

As already mentioned before, the pipeline integrates the spaCy library for the NER task. Further, it applies the scikit-learn library[5] for the KNN classification of the reducer. Here, the algorithm option is set to "auto", which automatically chooses the fastest KNN algorithm for the given data. Further, the pre-trained un-cased distilBERT model is integrated from the Hugging Face library[6], which is used for the reducer element as well as for the fine-tuned classifier. The classifier itself is fine-tuned with the PyTorch library[7].

4.1 Dataset for Experiments

The following sections elaborate on the text corpus and the corresponding labeled sets on which the systems are trained and tested on.

For the final evaluation, two datasets are processed through the complete pipeline described in Chapter 3. Both datasets are provided by the Text REtrieval Conference (TREC)[8]. The first corpus is referred to as AP89. It is composed of a collection of news articles of Associated Press from the year 1989. The second dataset, called WaPo contains news and blog articles from the years 2012 until 2017 from The Washington Post. These datasets have already been used in previous publications, making the retrieved results better comparable to other systems.

Generating a fair test dataset for a specific corpus to evaluate an ESE application can be an extensive task. The test data should contain as many sets as possible. In contrast,

[5]https://scikit-learn.org/stable/modules/neighbors.html
[6]https://huggingface.co/transformers/model_doc/distilbert.html
[7]https://pytorch.org/
[8]https://trec.nist.gov/

the sets should not tend to a specific semantic topic and the solutions should remain reasonable considering the given query seeds. Also, because text documents are usually quite massive, creating a test dataset manually can not be considered as a valid option. For instance, previous work, as in [38], is evaluating the whole system on 40 sets for the wiki corpus and 15 for the AP89 corpus. The author did not release the used dataset, which makes it hard to assess if the sets are valid.

This work employs a toolkit called DBpedia[9], which is further fortified by a research group at the CIIR for a similar project in this field. Here, DBpedia is creating, based on the current information on Wikipedia, a graph-based ontology that links entities to their semantic group. DBpedia can also be used to annotate entities from any text corpus, which is further utilized to create the entity sets. However, this entity extraction toolkit is considered as web-based set and not as a corpus-based set expansion because it extracts information from an external source, in this case, Wikipedia.

	AP89	WaPo
size	147MB	1.5GB
sets	120	528
sets > 8	60	242
sets > 10	48	197

Table 4.3: Information about the entity sets used for the final evaluations

As illustrated in Table 4.3, applying the toolkit to the two datasets extracts a descent amount of sets.

Still, extracting labeled sets from the documents lead to several issues. One is the so-called co-reference problem, which essentially describes the issue that different words can have the same meaning. For instance, ”John F. Kennedy” is also often referred to as ”JFK”. Therefore, even if an ESE system predicts the correct named entity, there may be a different name for the entity used in the test dataset. Also, because the DBPedia parses the topic name of a Wikipedia page and not the actual extracted entity from the text corpus, the used name of some solutions do not even appear in the text corpus.

Another issue by evaluating the ESE system with these sets is that entities are grouped based on the current information from DPedia. For instance, the AP89 dataset contains the entity ”Donald Trump” who was already a known entity back in 1989 but was not a politician. Still, in the test set, he is a correct solution for the queries ”Republican Party

[9]https://wiki.dbpedia.org/

Presidents of the United States". Here, a perfect corpus-based ESE system would not expand this set correctly.

However, every system tested on these documents is also affected by this issue. Thus the co-reference problem is neglected.

4.1.1 Experiments on the Perfect Candidates

Specific components of the system are first applied on a small selection of candidates to create a better understanding of the effect on each component of the pipeline. This makes it easier to evaluate the performance of one specific component. For instance, employing the classifier to all candidates would not only impair the performance of the classifier, but it would also take an enormous amount of time to finally get results.

For that reason, the set of candidates for some experiments consist solely of entities, which are the correct solution in at least one set. Therefore, this list is referred to in the following sections as "perfect candidates". Although applying these sets leads to good results, the results are not representative for the final application. To get similar results for the actual system, both the entity extractor and the reducer would need a recall of 100%.

4.2 Extracting Candidates and Pre-Processing

Before named entities can be extracted from the provided text corpus, the original file must be transformed into a generic text file format. In the case of the AP89 dataset, the provided file was in an XML format, while the WaPo was in a JSON format. Here, to get merely the relevant text, any tags or meta-information are discarded. While the resulting text is processed in the following NER step, both the reducing as well as the classification step are processing a lowercased version of the text.

As mentioned before, the test set of correct solutions consists of names for entities that do not appear in the original text corpus. Many of the given entity names do not appear in the corpus at all, while the NER tool also misses some entities. Due to that, the recall is even before the reducer is applied already below 0.931 for the AP89 and 0.806 for the WaPo dataset.

Furthermore, after the reducer is applied, a simple pre-processing is done to remove obvious incorrect or redundant candidates. For instance, if one of the query seeds is "New York" an allegedly obvious correct solution for the pipeline is the candidate "the New

York". The NER tool is not supposed to have any meta-information of any named entities. Therefore, it is not aware that "New York" is the actual named entity and makes based on a misleading context a wrong assumption. Since both entities occur in very similar contexts, the reducer considers it as a valid candidate and thus, the final ranker classifies it as a correct solution. This particular problem can easily be circumvented by simply removing every candidate, which contains a seed entity.

This entity extraction step is applied to both AP89 and the WaPo dataset. This subsequently leads to 335.009 entities for AP89 and 2.575.017 entities for the WaPo dataset. Despite that the content of the two datasets is completely different, it almost seems that there is a linear correlation between the size of the dataset shown in Table 4.3 and the number of extracted entities.

4.3 KNN Candidate Reducer

The section analysis the performance of the entity reducer, which is described in more detail in Section 3.3. In essence, the reducer's main purpose is to mitigate the number of candidates that would otherwise impair the classifier's performance while preserving the correct solutions. This is done by averaging and subsequently clustering the contextualized embeddings of each word.

Based on the actual test data, this section shows how the candidates are reduced to only a few candidates. This process is recreated visually with a t-Distributed Stochastic Neighbor Embedding (tSNE) to create a more general understanding of this process. After that, the actual performance of the reducer is analyzed. For the reducer, the more crucial information is the number of remaining candidates, rather than the ratio between correct and wrong candidates. Therefore, the reducer is evaluated on the recall.

4.3.1 Analyzing Contextualized Embddings with tSNE

Section 3.3 already discussed the question, how the concept of averaging embeddings introduced is sufficient. To further demonstrate that these assumptions apply for this particular task, the contextualized embeddings are further investigated. For that, the 768 dimensional embeddings are illustrated with a tSNE introduced in [43]. The following plots only consist of the 648 perfect candidates of the smaller AP89 dataset, to keep the plot as clear as possible.

At first, the contextualized embedding of each appearance of a word is illustrated in Figure 4.1. In the scatter plot, each color represents a different perfect candidate.

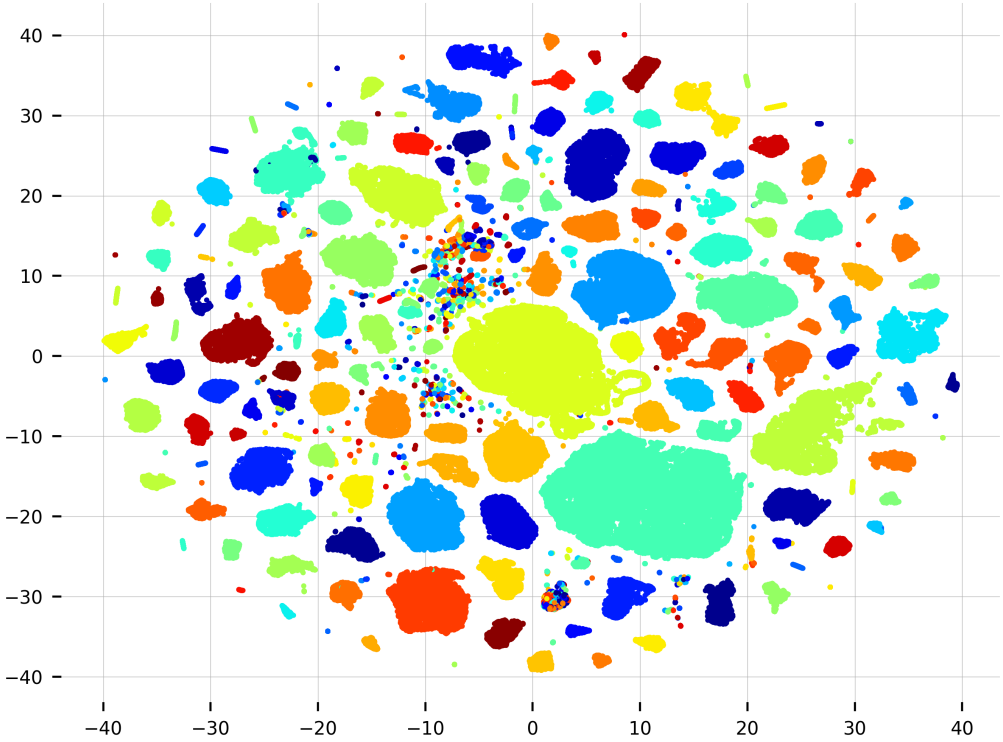


Figure 4.1: A tSNE visualization of all contextualized embeddings of the entities from the 648 perfect candidates of the AP89 dataset.

It is visible that especially entities that frequently appear in the corpus create a big, but still consistent cluster. This supports the assumption that the average embedding is, for most clusters, a valid representation of all the contextual embeddings.

Even for the perfect candidates, there are clusters that consist of embeddings from many different entities. Also, as described in Section 4.1, in the complete pipeline, many wrong assumptions are candidates that are often closely related to the correct solution or even a different name for the same entity. These incorrect solutions are often part of the entity clusters.

After the embeddings are combined to a single representation, the word representations are clustered. The illustration Figure 4.2 shows a tSNE for word representations with the corresponding semantic group from the test dataset as the color for this specific word.

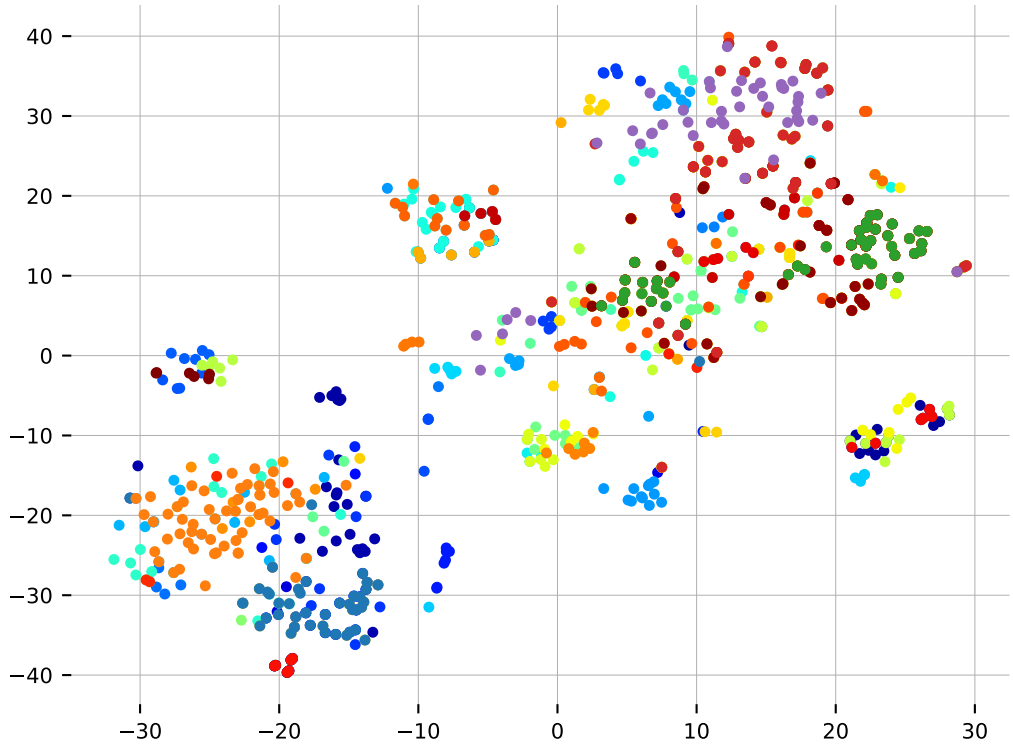


Figure 4.2: A tSNE visualization of all averaged contextualized embeddings of the entities. Each color represents a semantic group, to which the entity belongs to.

Despite that many words of the same semantic group are in the same cluster, many cluster consist of multiple semantic groups. For many smaller clusters, this might be an accurate visualization, since many entities might be valid solutions for several different seed sets. Also, there are several bigger clusters that contain several smaller sub-clusters. Here, a big cluster represents geographic entities and a smaller cluster represents geographic entities in Europe.

In order to better illustrate these big clusters, Figure 4.3 shows the same scatter plot with only a few selected semantic groups. The plot shows that the bigger cluster on the bottom left consists of people entities, while the cluster on the top-right consists of geographical entities.

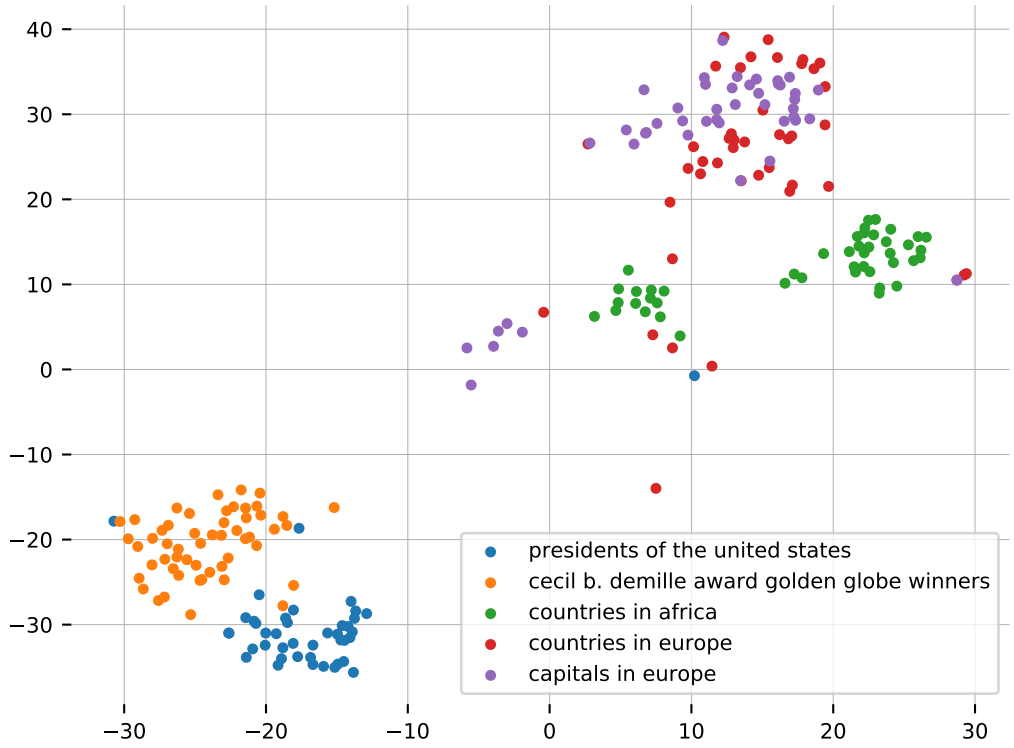
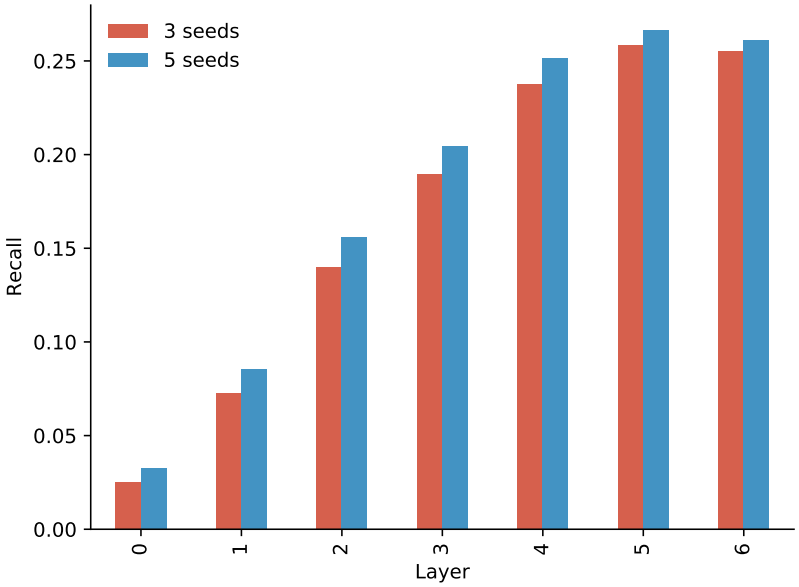


Figure 4.3: A tSNE visualization of the averaged embeddings from the five biggest semantic groups.

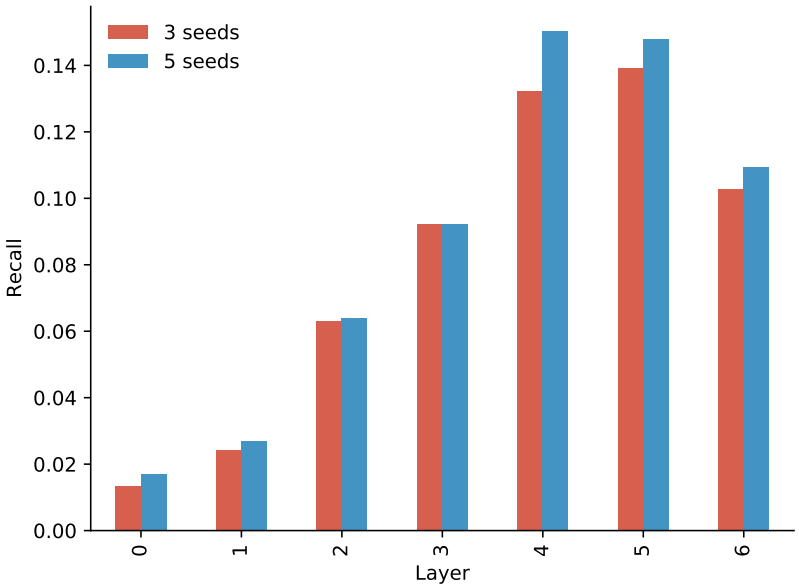
This illustration also reveals that entities closely related but do not belong to the semantic group, like countries to their capitals, still tend to be near each other. This behavior leads to many misclassifications, even if only the perfect candidates are considered.

4.3.2 Recall of the Different Layers for the Reducer

Since the intermediate layer's features have increased the performance of other applications in NLP, the recall values are also investigated for the reducer in more detail. For that, the recall values of every layer are compared in Figure 4.4. The plots reveal that the use of the intermediate layer's embeddings can increase performance. If only the reducer is considered, the layers four and five lead to the best recall. However, in the final pipeline, in which all components are put together, the fourth layer leads to the best results. It is also visible that the recall values for the AP89 dataset are up to about 10% higher than for the WaPo dataset.



(a) AP89



(b) WaPo

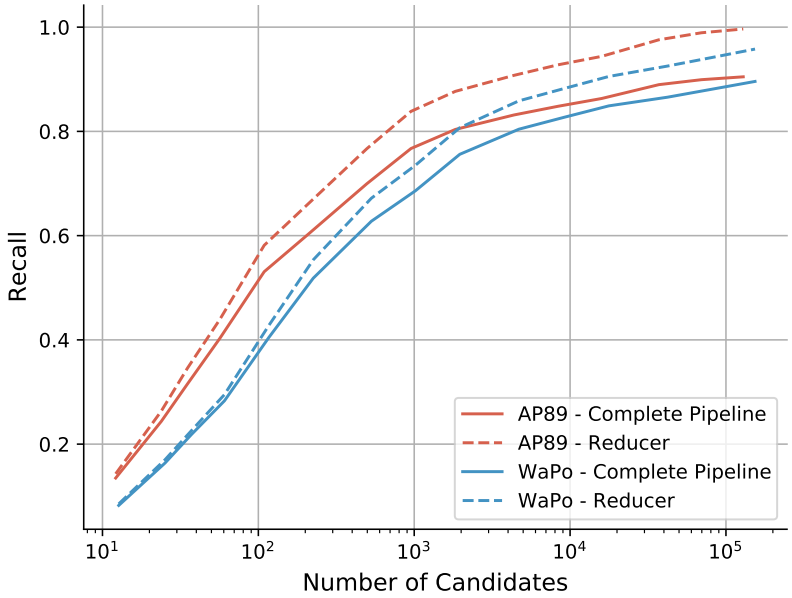
Figure 4.4: Recall of the reducer for around 100 candidates applied to all candidates for both datasets. The bar plots show the results for two different query sizes three and five seed.

4.3.3 Recall Curve of the Reducer

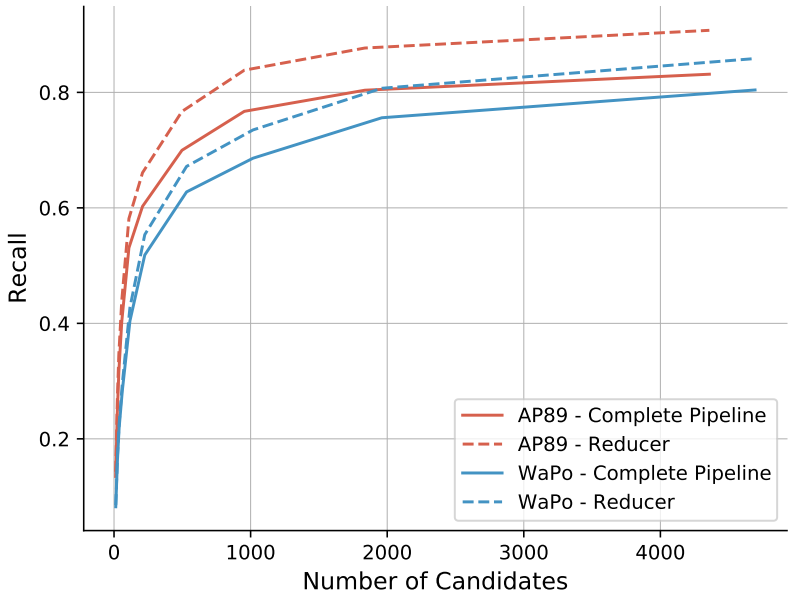
In this section, the recall-candidate curves for the fourth layer are interpreted in detail. Inspecting these curves should create a better understanding of which interval the reducer is losing relevant entities. Therefore, it should be possible to determine the optimum number of candidates.

As mentioned in Section 4.1, not all valid solutions exist in the original corpus. Thus, the pipeline can not reach a recall value of 100%. Therefore, the pipeline, as well as the reducer, are independently analyzed. At first, the recall curve of the reducer is further inspected. Since the recall drops the most for the last thousand entities, the first plot is depicted logarithmic. Figure 4.5a shows this plot for both datasets using three query seeds, while the experiments for using four and five query seeds lead to similar conclusions. The plot illustrates that the reducer can keep higher recall values for the AP89 corpus than the WaPo corpus. This is likely caused since the WaPo dataset consists of about eight times more entities than the smaller AP89 dataset. Therefore, the reducer curve is shifted to the right. Furthermore, the plot also illustrates that the reducer can remove many irrelevant entities without losing a lot of entities for the AP89 dataset. For instance, for the AP89 if the candidates are reduced to 1000 candidates per seed, 99.75% of the candidates are removed. However, only 25% of the relevant information is lost.

The results from the experiment can also be used to determine how many candidates would be favorable considering merely the reducer. An intuitive approach for choosing the number of candidates would be to examine the plot with linear scales of the same results illustrated in Figure 4.5b. Here, the resulting graph can be described as ”elbow-plot”. For such a graph a heuristic approach to determine the best value can be edge before the plot is stagnating. Based on this assessment, for these datasets, the optimal number of candidates would be between 1000 and 2000 candidates.



(a) Logarithmic plot of the recall



(b) Linear plot of the recall

Figure 4.5: Plots of the recall for both datasets. The plots show both the recall curve of considering only the data provided from the NER as well as the actual recall on the actual test dataset. The features are extracted from the fourth layer of the pre-trained distilBERT model.

4.4 Candidate Classifier

In this section, the final element of the pipeline, the fine-tuned classifier, is further investigated. As mentioned earlier, it is not reasonable to apply the classifier to all extracted candidates. To still be able to examine the performance of merely the classifier, the perfect candidates are used instead of all candidates. This makes these results completely independent from the entity reducer. This section, therefore, shows how the different context sizes of the classifier affect the performance. It also elaborates on the question, how much context is already captured by the entity reducer.

4.4.1 Performance Measures

It is more important for the classifier to evaluate the ratio of correct entities in the extracted set, rather than how many entities from the solutions are remaining. Thus, instead of considering a metric like the recall, the precision is calculated instead. Here, the precision for the first k elements of a set is denoted as P@k.

Since the entity set expansion task can also be described as a ranking problem, the Mean Average Precision (MAP) score mentioned in [44] is compared as well. The MAP score measures the precision of different recall steps. Here the average precision of the result can be described as

$$AP = \frac{1}{|Rel|} \sum_{i=1}^{k} relevant(i) \cdot P@i \tag{4.1}$$

where k denotes the size of the set of correct solutions. The function $relevant(i)$ describes if the entity at position i is correct. If the entity is valid, the corresponding value is 1, otherwise, it is 0. As mentioned before, $P@i$ is the precision for the first i entities and $|Rel|$ describes the total number of relevant entities.

Entities that do not appear in the solution set are considered a precision of 0. Therefore, the MAP score increases if the number of elements grows since the set might capture more entities. For multiple queries, the mean of the average precision is calculated as

$$MAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{4.2}$$

where N denotes the number of queries.

In the previous work like SetExpand [5] or in [38], solely the MAP score is used to analyze the performance of the system. But in these papers, the MAP score decreases from a very high score continuously to lower values, if the size N increases. This indicates that in these papers, entities that are outside the range of the set are ignored. If that is the case, the described scores would be higher as they actually are. Thus, the results can hardly be interpreted. For instance, if the system is only able to expand only one value correct, but classifies it correctly on the first position, it seemingly performs better than a system that recognizes all values but places an incorrect solution on the first position. Furthermore, as results in [9] show that the results of SetExpan [5] and SetExpander [3] are much lower than in the original papers.

4.4.2 Classifier on the Perfect Candidates

As already mentioned, the classifier can not be used on all the extracted candidates on its own. However, it is still important to investigate the effect of the reducer on the final classification. Furthermore, it is also relevant to show how the behavior of the classifier changes if it is applied to a vast number of candidates. Therefore, this section analyzes the classifier's results if it ranks the perfect candidates from both datasets. Here, two models are trained with different context sizes. One model is trained on the biggest context possible for three seeds and one solution. The second employs a smaller context of the size of 40 tokens.

Metrics	MAP@100		P@5		P@10	
Query Seeds	3	5	3	5	3	5
Classifier Small Context	0.264	0.324	0.404	0.387	0.465	0.438
Classifier Big Context	0.184	0.227	0.281	0.268	0.330	0.308

Table 4.4: Results for the fine-tuned classifier trained on three seeds, which is applied to the perfect candidates for the AP89 dataset.

Compared to other approaches, the results shown in Table 4.4 seem to be already quite promising. In comparison, in [9], SetExpander, SetExpan and CaSE have been tested on the same dataset and lead to noticeable lower results than the proposed classifier.

Further experiments showed that combining the two different contexts can increase the precision of the classifier. The reason behind this might be that while the bigger context captures the context well, it also gets a lot of unrelated context information. The smaller

context is not affected by the misleading context information, but on the other hand, might lack information for the broader context.

4.4.3 Classifier with the Reducer on the Perfect Candidates

After showing the precision of the entity classifier based on different context sizes, the effect of the reducer on the subsequent classifier is further investigated. Table 4.5 illustrates the precision for the same two models, as in Table 4.4, after the perfect candidates are reduced to 100 candidates.

Interestingly, although the perfect candidates set consists only of a few candidates, adding the reducer seems to increase the performance noticeably. This behavior indicates that the reducer already captures information of the bigger context, which is beneficial for the classifier that can only process the smaller context.

Metrics	MAP@100		P@5		P@10	
Query Seeds	3	5	3	5	3	5
Classifier Small Context	0.351	0.387	0.496	0.522	0.465	0.475
Classifier Big Context	0.319	0.340	0.432	0.427	0.421	0.418

Table 4.5: Results for the fine-tuned classifier and the entity reducer applied to the perfect candidates for the AP89 dataset.

4.5 KNN Baseline

The following section describes a baseline, which is essentially an abstraction of the proposed system. Therefore, the baseline is more lightweight but still exceeds many of the previous methods.

As described in Chapter 3, utilizing merely the features of BERT can already lead to reasonable results for entity reducer. Therefore, this work compares the results of a simple KNN classifier as a baseline to the more sophisticated pipeline. Since the baseline is less complex than the complete pipeline, the proposed method should at least slightly exceed the results of the baseline.

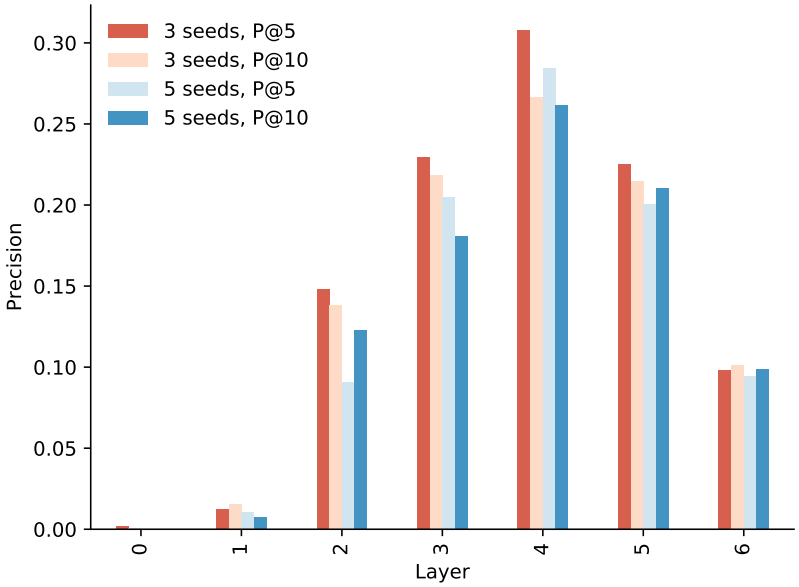
The algorithm of the baseline is similar to the one of the reducer described in Section 3.3 except that instead of considering the nearest neighbors of each seed individually, the solutions are the nearest candidates to all query seeds. All seeds are combined to a

single embedding by averaging the embedding of each query seed. Calculating neighbors like this reduces the run-time of the classifier substantially since it is only necessary to calculate the required K candidates. In contrast, considering the nearest entities to all seeds requires calculating the distances to all candidates for each seed. The baseline algorithm leads to better results if it is applied as a classifier. However, the approach mentioned in Section 3.3 still has better recall values, especially if a larger number of candidates are considered.

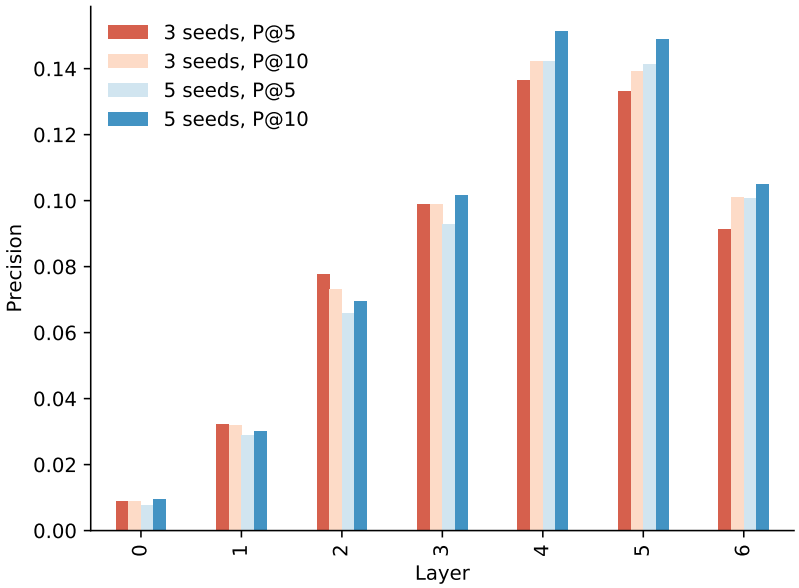
The parameters of the baseline are optimized as well, to make the baseline results better comparable to the final pipeline. However, the only adjustable parameter for the baseline is the used embedding layer, from which the features are extracted. For that, the precision for each layer are further analyzed and are shown in Figure 4.6. Both plots show that for the AP89 as well as for the WaPo dataset, the features of the fourth layer lead to the best performances considering the precision for the classifier. Therefore, the embeddings of this layer are employed for the final classification task.

Besides that, the plots also reveal that the baseline results improve if fewer query seeds are provided. This is surprising because it seems more natural that the ESE task becomes easier if more seeds are provided. A possible explanation for this is that with a higher number of seeds, it is more likely that at least one embedding of a seed differs too much and distorts the averaged embedding. Also, since there are a finite number of solutions for each seed set, a bigger query set leads to less possible valid answers, making the expansion task harder.

The baseline classifier is also applied to only the perfect candidates, which results are illustrated in Figure 4.7. The plots show that the baseline classifier has the marginally best performance if the fifth layer's embeddings are used. This indicates that the lower layers of such a model can better separate between noise and the perfect candidates. In contrast, if the given candidates only consist of valid candidates, the layer does not have a big effect on the classifier anymore.

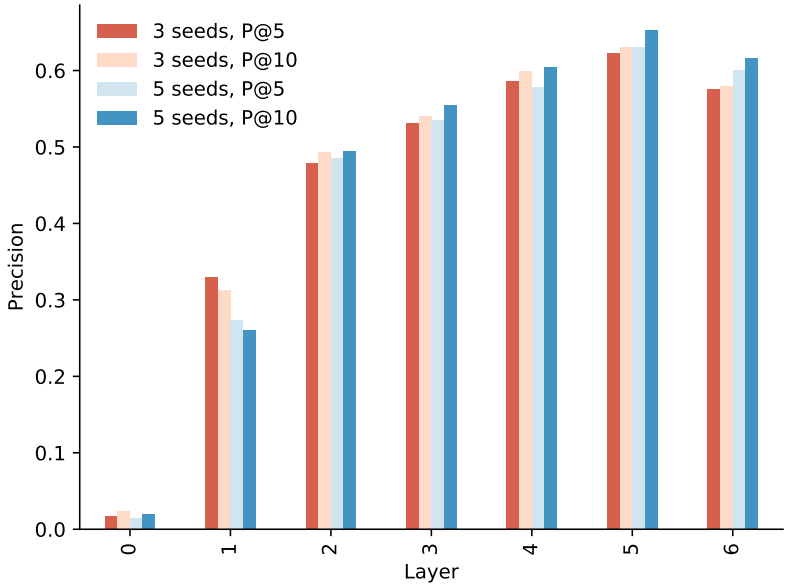


(a) AP89

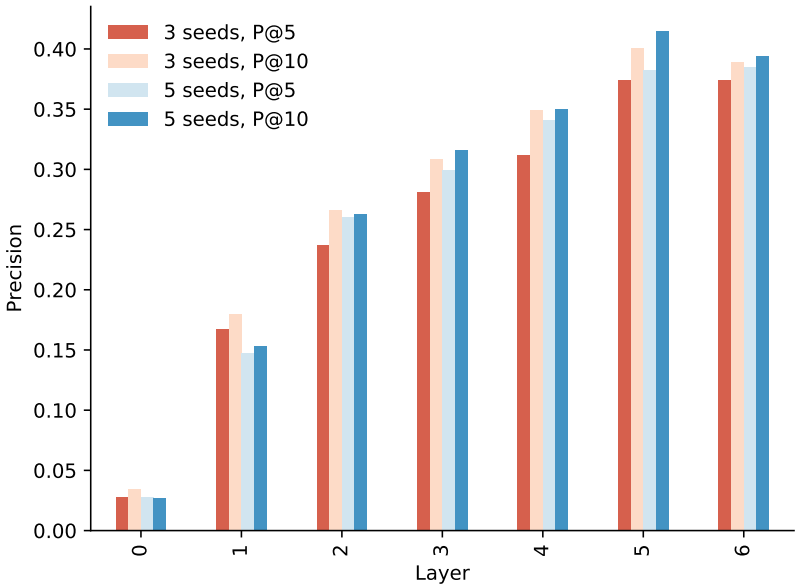


(b) WaPo

Figure 4.6: Precision of the baseline applied on all candidates for both datasets. The bar plots show the results for two different query sizes, with three and five seeds, and the precision values considering five and ten solutions.



(a) AP89



(b) WaPo

Figure 4.7: Precision of the baseline applied on perfect candidates for both datasets. The bar plots show the results for two different query sizes, with three and five seeds, and the precision values considering five and ten solutions.

4.6 Complete Pipeline

After analyzing each element of the pipeline individually, the following section evaluates the system's performance if every component, the entity extractor, the KNN reducer and the classifier are put together. To be able to analyze this pipeline's performance, the system is compared to the KNN BERT baseline described in Section 4.5. For the final evaluation, only the pipeline and the baseline are applied to all candidates.

4.6.1 Choosing the Number of Candidates

For the complete system, the number of remaining candidates after reducing the extracted entities has a direct effect on the classifier performance. As already introduced in Section 4.4.2 and Section 4.4.3, applying the classifier to all candidates can decrease the precision, since it is possible that a correct solution is coincidentally in a rather unsuited context. Also, especially if there are many candidates, it is possible that an incorrect candidate is put in a perfect context. However, selecting only a few candidates might remove too many candidates.

To assess the number of candidates fairly, the system is first tested on some of the training data. These tests indicated that around 100 candidates would be the optimum number of candidates. Running the same experiments on the actual test data support this assumption. As illustrated in Figure 4.8, the pipeline reaches a peak at around 50 candidates and the precision continuously decreases as the number of candidates increases. However, evaluating the system on 100 candidates seems to lead only to slightly worse precision for the optimal number of candidates for these test data. This shows that the approximation of 100 candidates is a valid value.

First observations of the MAP score indicate that the score increases if more candidates are selected. This can be caused since it is more important for the MAP score to capture all entities. However, this work is more interested in a system that can expand a set correctly rather than expanding it with all solutions. Therefore, the system is optimized to the precision, not on the MAP score.

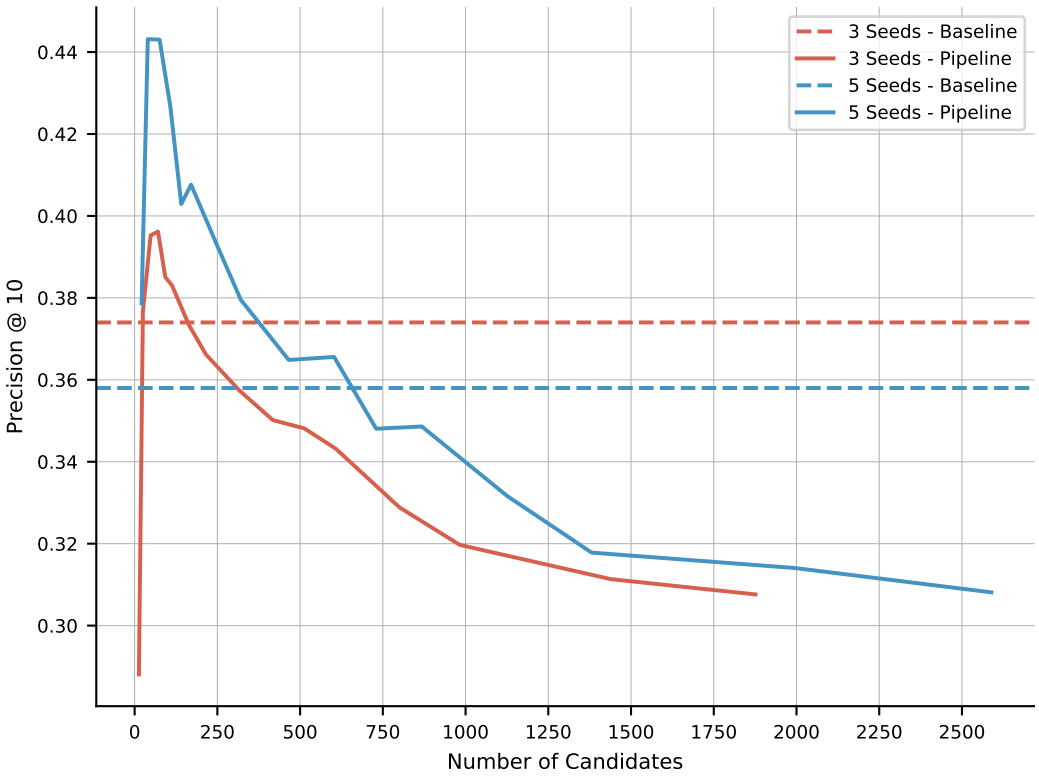


Figure 4.8: Plot of trade-off between precision for the pipeline trained on five seeds considering the top 10 results and the number candidates. The constant baseline precision line is applied on all candidates and not on the reduced number of candidates.

4.6.2 Results of the System on all Candidates

After comparing the pipeline to the previous approaches, the actual pipeline is applied to all candidates. Based on the findings from Section 4.3.2 and Section 4.4.3, the fourth layer of the pre-trained model is employed for the reducer and the classifier is solely trained on a single smaller context window. The model itself is trained on 150 epochs and on three and five seeds as input. The results from the final evaluation of the pipeline are compared in Table 4.6 to the KNN baseline.

Metrics	MAP@100		P@5		P@10	
Query Seeds	3	5	3	5	3	5
KNN Baseline	0.141	0.150	0.364	0.347	0.374	0.358
Pipeline trained on 3 seeds	0.227	0.158	0.347	0.311	0.155	0.176
Pipeline trained on 5 seeds	0.216	0.2680	0.380	0.415	0.383	0.427

(a) AP89

Metrics	MAP@100		P@5		P@10	
Query Seeds	3	5	3	5	3	5
KNN Baseline	0.034	0.045	0.137	0.1442	0.142	0.151
Pipeline trained on 3 seeds	0.049	0.046	0.131	0.090	0.128	0.116
Pipeline trained on 5 seeds	0.044	0.054	0.137	0.141	0.125	0.145

(b) WaPo

Table 4.6: Table shows the final results for the pipeline and the KNN baseline for all extracted candidates. Here, the pipeline results, is trained on either trained on three or five seeds, are described.

The results show that the performances of the pipeline trained on five seeds exceed the precision of all other compared methods for the AP89 dataset. The ESE pipeline with a classifier trained on five query seeds adds almost 7% precision to the results than the sole baseline. Still, compared to the previously published papers, the baseline itself is able to outperform all other, much more complex and cumbersome methods. Reviewing the presented results further, it becomes evident that the model trained on five seeds can also be used with fewer query seeds, while the precision remains adequate. On the contrary, if a model is trained on only three query seeds, classifying more than three seeds heavily impacts the model's performance. An explanation for this might be that a model with more query seeds is trained on more combinations of the contexts with five seeds than a model with three seeds. Also, the model with fewer input seeds is not trained to process more than three seeds.

Among the pipeline results and the baseline, the precision for the WaPo dataset is only a fraction of the AP89 dataset's precision. Two findings indicate why the results for the WaPo dataset are lower. First of all, as described in Section 4.1, the WaPo dataset consists of many more entities. This can subsequently affect the performance of the complete system. Furthermore, observing the test data reveals that the AP89 dataset contains more geographical entities like countries. Completing a set of countries seems to be easier than a set of musicians where there are extensive related answers. Also, these

sets are more impaired by the aforementioned co-reference problem and inaccuracies in the pre-processing.

In addition to that, the table also shows that for the WaPo dataset, the results for the pipeline and the baseline are about the same. However, to reduce the time of the evaluation, the pipeline's parameters, like the optimum number of candidates, epochs and context size, are optimized based on the AP89 training data. In contrast, the baseline is optimized for the WaPo dataset as well. Adapting the parameter for the WaPo dataset might increase the performance of the pipeline remarkably.

Interestingly, all the other described methods described in Section 2.1 have the worst performance for the AP89 dataset, since these approaches benefit from a larger quantity of contexts. However, the proposed approach, especially the reducer component, can also make strong embeddings even with a few contexts. Also, since the classifier's context is chosen arbitrarily, the current implementation does not benefit from multiple contexts.

In comparison to the results retrieved in Section 4.4.2 and Section 4.4.3, applying the baseline and the complete pipeline on the extracted candidates leads to a stronger discrepancy between the MAP score and the precision. This can be caused by the loss of correct entities after reducing the candidates. For instance, as described before in Section 4.3, the reducer has a recall of 0.25 for the AP89 dataset. This has a stronger effect on the MAP score since it takes all candidates into account.

It is also worth mentioning that if a human expands a given seed set, it should be easier to identify the correct solutions if more query seeds are provided. However, compared to all other models, the proposed model seems to be the only model that can utilize the gained information from several seeds.

5 Discussion

The experiments in this work demonstrate that contextualized embeddings can be used for the ESE task. They show that they exceed all compared methods. Nonetheless, the proposed system can still be further developed.

First of all, the system utilizes the embeddings of the distilBERT model since it only consists of six layers, which makes analyzing the effects of the layers more efficient. However, for future experiments in this field and for productive implementations, it might be reasonable to use a more sophisticated pre-trained model. Besides the original BERT model, a model with the same number of parameters but with better results is introduced in [45]. The authors state that the original BERT model has several weaknesses. First of all, the paper shows that the BERT model is severely under-trained. Further, the authors suggest that the NSP task of BERT does not have any effect on the final results.

Another point of discussion is the processing of the contextualized embeddings. It is elaborated in Section 3.3 as well as in Section 4.3.1, how multiple contextualized embeddings can be utilized for the ESE task. Despite that averaging the embeddings already lead to satisfying results, there are still many weaknesses in this approach. For instance, the co-reference problem strongly affects the final results, since many reduced entities are duplicates of other solutions or even the query seeds. This issue not only leads to wrong candidates, but it also makes it harder for the reducer to select valid candidates. Another issue is that some embeddings of the same word are scattered in multiple clusters, as illustrated in Figure 4.2. Averaging these embeddings can only lead to completely satisfying results if the embeddings are clustered together.

Also, the KNN clustering of the reducer seems quite trivial. Here, the seeds are analyzed independently from each other. However, taking all seed tokens simultaneously into account might be beneficial. Here, it might be necessary to consider other features than only the contextualized embeddings.

Figure 4.8 shows that there is a big gap between the optimum number of candidates for the classifier and the reducer. Mitigating this gap might be an opportunity to increase the system's performance.

5.1 Examples of Entities for Correct and Wrong Results

As already mentioned, the system is able to expand certain sets better than others. Table 5.1 shows some examples for the AP89 dataset. Here, the strengths and weaknesses can be better interpreted. As mentioned earlier, geographic entities like the query in the first row are easier to expand than others since there are fewer alternatives to how a set can be expanded. For example, in the second row of Table 5.1, the seed "Frank Sinatra" is given, who used to be a famous musician, but also an awarded actor. Yet, the system cannot differentiate this ambiguity and, therefore, adds musician to the seed set. The following row reveals two other issues. The first issue leads to the solution "George Clinton", former vice president of the united states. This solution is closely related to the semantic group and also very similar to the input seed "Bill Clinton". The second wrong expansion would be valid. However, it is misclassified because of the co-reference problem. Since the test data only contain "George W. Bush" and "George H. W. Bush", there is no literal match for "George Bush" and the solution is neglected. However, it seems valid to accept this entity as a correct solution. Therefore, the actual precision of each system can be increased if the validation accepts multiple names for the same solution.

Semantic Group	Query Set	Correct	Wrong
Countries in Europe	Romania Bosnia and Herzegovina Iceland	Serbia Croatia Estonia Austria Bulgaria	
Cecil B. Demille Award Golden Globe winners	Frank Sinatra Red Skelton George Clooney		Smokey Robinson Billy Joel Elton John Keith Moon Lee Marvin
Presidents of the United States	Bill Clinton Harry S. Truman George Washington	Jimmy Carter Franklin D. Roosevelt Ulysses S. Grant	George Clinton George Bush

Table 5.1: Examples of expansion on all extracted candidates for the AP89 dataset. The table represents the results for P@5 given three query seeds.

6 Conclusion and Outlook

This report summarizes the research on the effectiveness of contextualized embeddings for the expansion of predefined sets based on a single corpus. Therefore, the work elaborates on the benefits of the recently introduced BERT model compared to the previously implemented approaches. It further describes the basic design for an approach using such a system, so it can not only be applied in a research setup but also for a production scenario.

Based on the gathered findings, it has been described how the BERT model can not only be implemented as a classifier but also as an entity reducer. Here, the entity reducer utilizes features of the BERT model to remove unlikely candidates and thus increase the subsequent classifier's performance. The proposed classifier is a fine-tuned BERT model, which processes multiple candidates within its context to identify which of the candidate entities are the most likely solutions.

These findings inspired a combination of all these elements resulting in a new pipeline, which only utilizes these contextualized embeddings. The conducted experiments show that this pipeline exceeds previous results and a simplification of the system as a baseline. Further, since the final expansion tool is lightweight, it seems to be the only approach that is able to classify a vast number of candidates. These findings indicate that future work in this area might benefit if components like the entity reducer or the final classifier are further developed.

6.1 Future Steps

Although the proposed system already creates satisfying results, there are still several potentials to improve the system.

Despite that averaged embeddings are already creating solid representations of the words, they still suffer from several issues. The co-reference problem makes it difficult to use this system for even bigger corpus reliably. This can be counteracted by combining a countermeasure like described in [46] with simple features like the edit-distance. Therefore, several entities, which names and contextualized embeddings are quite similar, are joined to a single entity.

It is also worth thinking about a more sophisticated way to cluster the averaged embeddings than just applying KNN. Also, by averaging the contexts, it is not possible to

process the complete meaning of ambiguous candidates. A possible countermeasure for that would be to create a separate entity of the same word for each of its meaning.

Furthermore, the classifier is only using an arbitrary context for each query seed. Therefore, there is no guarantee that the contexts of the seeds relate to each other. However, as already mentioned, choosing the contexts based on the contextualized embeddings for this certain context does not increase the performance. Here, the aforementioned multiple entities for every word sense might already lead to more suitable contexts. Also, there can be a better way to choose the correct context for the seed, for instance, by making a literal match of the context like in the previous work of SetExpan or CaSE.

List of Figures

List of Tables

List of Algorithms

References

[1] R. C. Wang and W. W. Cohen. “Language-Independent Set Expansion of Named Entities Using the Web”. In: *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. 2007, pp. 342–350.

[2] Zhe Chen, Michael Cafarella, and H. V. Jagadish. “Long-Tail Vocabulary Dictionary Extraction from the Web”. In: *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. WSDM ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 625–634.

[3] Jonathan Mamou, Oren Pereg, Moshe Wasserblat, Alon Eirew, Yael Green, Shira Guskin, Peter Izsak, and Daniel Korat. “Term Set Expansion based NLP Architect by Intel AI Lab”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 19–24.

[4] Jacob Eisenstein. *Introduction to Natural Language Processing*. Adaptive Computation and Machine Learning series. Cambridge, MA, USA: MIT Press, 2019.

[5] Jiaming Shen, Zeqiu Wu, Dongming Lei, Jingbo Shang, Xiang Ren, and Jiawei Han. “SetExpan: Corpus-Based Set Expansion via Context Feature Selection and Rank Ensemble”. In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18-22, 2017, Proceedings, Part I*. Vol. 10534. Lecture Notes in Computer Science. Springer, 2017, pp. 288–304.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient Estimation of Word Representations in Vector Space”. In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2013.

[7] Jeffrey Pennington, Richard Socher, and Christopher Manning. “Glove: Global Vectors for Word Representation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar, Oct. 2014, pp. 1532–1543.

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. “Enriching Word Vectors with Subword Information”. In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146.

[9] Puxuan Yu, Zhiqi Huang, Razieh Rahimi, and James Allan. "Corpus-Based Set Expansion with Lexical Features and Distributed Representations". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR'19. Paris, France: Association for Computing Machinery, 2019, pp. 1153–1156.

[10] Arvind Neelakantan, Jeevan Shankar, Alexandre Passos, and Andrew McCallum. "Efficient Non-parametric Estimation of Multiple Embeddings per Word in Vector Space". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1059–1069.

[11] Yoav Goldberg and Graeme Hirst. *Neural Network Methods in Natural Language Processing*. CA, USA: Morgan & Claypool Publishers, 2017.

[12] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 2227–2237.

[13] S Hochreiter and J Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780.

[14] Oren Melamud, Jacob Goldberger, and Ido Dagan. "context2vec: Learning Generic Context Embedding with Bidirectional LSTM". In: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 51–61.

[15] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. "Learned in Translation: Contextualized Word Vectors". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 6294–6305.

[16] Bengio Yoshua, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166.

[17] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734.

[18] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 3104–3112.

[19] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[20] Jakob Uszkoreit. *Transformer: A Novel Neural Network Architecture for Language Understanding*. `https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html`. Accessed: June 26, 2020.

[21] Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 1412–1421.

[22] Jianpeng Cheng, Li Dong, and Mirella Lapata. "Long Short-Term Memory-Networks for Machine Reading". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 551–561.

[23] 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 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 5998–6008.

[24] Jay Alammar. *The Illustrated Transformer*. `http://jalammar.github.io/illustrated-transformer/`. Accessed: June 26, 2020.

[25] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. Cambridge, MA, USA: MIT Press, 2016.

[26] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. “Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context”. In: *CoRR* abs/1901.02860 (2019).

[27] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. “Improving language understanding by generative pre-training”. In: (2018).

[28] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding”. In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 353–355.

[29] 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)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186.

[30] M. Schuster and K. Nakajima. “Japanese and Korean voice search”. In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012, pp. 5149–5152.

[31] Taku Kudo and John Richardson. “SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 66–71.

[32] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. “What Does BERT Look at? An Analysis of BERT’s Attention”. In: *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 276–286.

[33] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *ArXiv* abs/1910.01108 (2019).

[34] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. “Language Models are Unsupervised Multitask Learners”. In: (2019).

[35] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Neural Machine Translation of Rare Words with Subword Units”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725.

[36] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data”. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML ’01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282–289.

[37] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. “Natural Language Processing (Almost) from Scratch”. In: *J. Mach. Learn. Res.* 12.null (Nov. 2011), pp. 2493–2537.

[38] Jiaxin Huang, Yiqing Xie, Yu Meng, Jiaming Shen, Yunyi Zhang, and Jiawei Han. “Guiding Corpus-Based Set Expansion by Auxiliary Sets Generation and Co-Expansion”. In: *Proceedings of The Web Conference 2020*. WWW ’20. Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 2188–2198.

[39] Joshua Coates and Danushka Bollegala. “Frustratingly Easy Meta-Embedding – Computing Meta-Embeddings by Averaging Source Word Embeddings”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 194–198.

[40] Yijin Liu, Fandong Meng, Jinchao Zhang, Jinan Xu, Yufeng Chen, and Jie Zhou. “GCDT: A Global Context Enhanced Deep Transition Architecture for Sequence Labeling”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 2431–2441.

[41] Stephen M. Omohundro. *Five Balltree Construction Algorithms*. Tech. rep. International Computer Science Institute Berkeley, 1989.

[42] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: The MIT Press, 2012.

[43] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605.

[44] Stefan Büttcher, Charles Clarke, and Gordon V. Cormack. *Information Retrieval: Implementing and Evaluating Search Engines*. Cambridge, MA, USA: The MIT Press, 2010.

[45] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692.

[46] Thibault Févry, Livio Baldini Soares, Nicholas FitzGerald, Eunsol Choi, and Tom Kwiatkowski. *Entities as Experts: Sparse Memory Access with Entity Supervision.* 2020. arXiv: 2004.07202 [cs.CL].

Acronyms

API Application Programming Interface

BCE Binary Cross-Entropy

BERT Bidirectional Encoder Representations from Transformers

biLSTM bidirectional LSTM

BPE Byte Pair Encoding

CBOW Continuous Bag of Words

CE Cross-Entropy

CNN Convolutional Neural Network

CoVe Contextualized Word Vectors

CRF Conditional Random Fields

ELMo Embeddings from Language Models

ESE Entity Set Expansion

FC Fully Connected

GloVe Global Vectors for Word Representation

GLUE General Language Understanding Evaluation

GPT Generative Pretrained Transformer

KNN K-Nearest Neighbors

LM Language Model

LSTM Long Short-Term Memory

MAP Mean Average Precision

ML Machine Learning

MLM Masked Language Model

MLP Multilayer Perceptron

NER Named Entity Recognition

NLP Natural Language Processing

NMT Neural Machine Translation

NSP Next Sentence Prediction

RNN Recurrent Neural Network

TREC Text REtrieval Conference

tSNE t-Distributed Stochastic Neighbor Embedding

Zimbra **amelie.arrer@fh-salzburg.ac.at**

Freigabe MPS Paper Gwechenberger (ITS)

From : Cornelia Ferner <cornelia.ferner@fh-salzburg.ac.at> Wed, Sep 23, 2020 08:14 AM

Subject : Freigabe MPS Paper Gwechenberger (ITS)

To : Amelie Arrer <amelie.arrer@fh-salzburg.ac.at>

Cc : Florian Gwechenberger <fgwechenberger.its-m2018@fh-salzburg.ac.at>

Liebe Frau Arrer,

die Forschungsarbeit wurde von Florian Gwechenberger zur vollsten Zufriedenheit ausgearbeitet und ist für den Upload auf die MPS-Homepage freigegeben.

Vielen Dank und liebe Grüße

Cornelia Ferner

DI Cornelia Ferner, BSc
Lecturer
Informationstechnik & System-Management (its)

Fachhochschule Salzburg GmbH
Salzburg University of Applied Sciences
Urstein Süd 1, 5412 Puch/Salzburg, Austria
Gerichtsstand Salzburg, FN166054y

+43 50 2211-1329
cornelia.ferner@fh-salzburg.ac.at
www.fh-salzburg.ac.at
facebook.com/Fachhochschule.Salzburg
twitter.com/fhsalzburg
instagram.com/fhsalzburg