[1. Theoretical Foundations 2](#_Toc119517869)

[1.1. Relation Extraction 2](#_Toc119517870)

[1.1.1. Definition 2](#_Toc119517871)

[1.1.2. Pipelines or End-to-end 3](#_Toc119517872)

[1.1.3. Sequence-based or Dependency-based 6](#_Toc119517873)

[1.2. Transformers 7](#_Toc119517874)

[1.2.1. Architecture 7](#_Toc119517875)

[1.2.2. Transfer learning 10](#_Toc119517876)

[1.2.3. Types of transformers 10](#_Toc119517877)

# Theoretical Foundations

## Relation Extraction

### Definition

[Maybe add graph of the respective tasks as diagram?]

Relation extraction is the task of extracting a semantic relationship between two entities in natural language. Linguistically, these relationships require a subject, an object, and the given relation, indicated by the verb connecting these entities. To visualize this, one example would be “The Berlin School of Economics and Law is situated in Berlin Schöneberg” where “The Berlin School of Economics and Law” would be the leading subject, “Berlin Schöneberg” the object. Their relationship is defined by “situated in”.

Relation extraction presents a way to create structured data, such as actors, targets, and their relations, from unstructured data, such as news articles, medical descriptions, or any other text. This data can be used to enable knowledge bases and information that is focused on interactions and co-dependency. The task of relation extraction can also be combined in other use cases such as question answering (e.g., Li et al., 2019; Zhao et al., 2021).

To accomplish this task of relation extraction, both the relevant entities must be correctly identified and the relationship between them must be correctly classified. The first part, identifying relevant entities, is referred to as named entity recognition. The task of relation extraction is inseparable from named entity recognition and in result most commonly used benchmark datasets for relation extraction are also used to evaluate named entity recognition performance. The challenge within the task of named entity recognition does not lie in finding candidate words, as these are mostly nouns or groups of words resolving around nouns, but in identifying which words are relevant entities. In result, this task was originally handled by utilizing gazetteers and similar knowledge resources (Mikheev, Moens, & Grover, 1999). However, as the use for named entity recognition model is spread across several industrial domains, the entities that are relevant also differ highly. The second part of relation extraction, relation classification, is a classification task for the relation between two known named entities from a list of defined relations (Hendrickx et al., 2010). Much like the type of named entities this list of relation is often domain or problem related.

* Maybe write about coref here.
  + A related task, often incorporated in relation extraction is coreference resolution. This becomes particularly critical when extraction not only relations but the knowledge behind the relation. Coreference resolution aims at finding entities with different names that describe the same entity (SOURCE). In a sentence, “The Berlin School of Economics and Law was founded in 1971. It produced many qualified professionals” the entities “The Berlin School of Economics and Law” and “It” refer both to the same organization. However, when splitting the text into separate sentences, this information will be lost in the second sentence. As such coreference resolution is commonly used in pre-processing for relation extraction tasks (SOURCE).

### Pipelines or End-to-end

As relation extraction tasks needs to accomplish both named entity recognition and relation classification, one approach is to use separate models for both tasks. The necessary chaining of at least two different models into a pipeline gives this approach its name. This approach was commonly used with support vector machines and tree-based approaches (Bach & Badaskar, 2007) as these models gave state-of-the-art performance on separated tasks.

The first papers describing a joint training of both tasks used linear programming (Roth & Yih, 2007), a graph-based approach (Kate & Mooney, 2010) and a table-based approach (Miwa & Sasaki, 2014), respectively. This joint training is also referred to as end-to-end approach, since one model performs all tasks from start to finish. Arguably, models trained in a joint manner should outperform pipeline models due to their ability to incorporate the similarities in the tasks of extracting entities and the relationships between them (Yan et al., 2021). While Roth and Yih (2007) found the joint training to fully outperform the pipeline approach, the other two papers found mixed results when comparing the approaches. The end-to-end model of Miwa and Sasaki (2014) outperforms on overall scores for relation extraction but underperforms the pipeline approach on the named entity recognition subtask. Similarly, Kate and Mooney (2010) showed that the pipeline models performed better on specific relation types, while the end-to-end model performed better on others. End-to-end models have since been used more commonly, typically in combination with neural networks.

Several studies showed that the inherent benefit of joint training, being shared features and parameters during training can improve overall performance (e.g., Yan et al., 2021, Miwa & Bansal, 2016). However, Zhong & Chen (2021) have found better performance with a pipeline approach than with the end-to-end approach. They argue that a shared encoder for both tasks may reduce performance either due to different features being important for the separate task or if different input formats for the respective tasks are required. The first argument hints at a general problem, that some data may require learning of nuances in either task which are possibly less focused in joint training. In recent papers, this shortcoming was addressed by utilizing both unshared features for named entities and relations while also using shared features separately (e.g., Crone, 2020; Yan et al., 2021; Basu et al., 2022), achieving new state-of-the-art performances. Basu et al. (2022) also highlights a performance gain of combining both shared and unshared features compared to only using unshared features, however, the performance of shared features without unshared features was not recorded.

Although a pipeline of two or more models may allow for more specialized training, the biggest methodological drawback is the risk of error propagation. Although performance on named entity recognition is generally higher than on relation classification (e.g., Zhong & Chen, 2021; Zhao et al., 2021), the models are far from perfect. This means that several inputs for the relation classification task in a pipeline are prone to be classified incorrectly and the relation classification model can never outperform the named entity recognition model. Zhong & Chen (2021) discuss strategies to potentially reduce error propagation, such as 10-way jackknifing and adapted entity input markers but did not find any success using them.

Several papers suggested a third approach, using a separate named entity recognition model together with a joint relation extraction model (e.g., Wang & Lu, 2020; Han et al., 2020; Giorgi et al., 2022). This approach benefits from receiving entity hints from the first model which help the joint model to find relevant entities. At the same time, this approach is not subjected to the error propagation of a pipeline since the joint model is still able to identify different entities than given as input. To have the model learn when these entity hints are or are not effective, the use of negative samples, i.e., samples that do not have any relevant relation to report, within the training set may help (Han et al., 2020).

Recent models achieving or improving state-of-the-art performances on relation extraction benchmark datasets utilized mainly the end-to-end approach (Papers with Code, 2022). However when evaluating the approaches, a critical lack in literature exists regarding direct comparisons. While it would be generally possible to use the individual methodology of the models both on pipelines and on end-to-end models, often only one approach is chosen. Table 1 shows the results of papers that incorporated both approaches, either by training both approaches individually or by using an ablation study. Out of seven identified papers that compared pipeline models with end-to-end models, only two of which used the same dataset, two papers found the pipeline to slightly outperform the end-to-end model. Two papers found the end-to-end model to perform slightly ahead of the pipeline while three papers found strongly increased performance when utilizing the joint approach. This small sample size does not allow for definitive results and further research is needed to compare the approaches in a more direct manner.

Table 1

*Direct comparisons of end-to-end and pipeline performance on various datasets*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Authors | Year | Dataset | Approach | RE |
| Miwa & Bansal | 2016 | ACE05 | End-to-end | 51.8† |
| Pipeline | 51.0† |
| Nguyen & Verspoort | 2018 | CoNLL04 | End-to-end | 66.9† |
| Pipeline | 66.3† |
| Han et al. | 2020 | MATRES | End-to-end | 59.6† |
| Pipeline | 57.2† |
| Zhong & Chen | 2021 | ACE05 | End-to-end | 64.4\* |
| Pipeline | 64.8\* |
| Yan et al. | 2021 | SciERC | End-to-end | 38.4† |
| Pipeline | 36.9† |
| Eberts & Ulges | 2021 | DocRED | End-to-end | 59.46† |
| Pipeline | 59.76† |
| Giorgi et al. | 2022 | CDR | End-to-end | 52.4\* |
| Pipeline | 34.1\* |

*Note*. Yan et al. (2021) have not directly compared end-to-end to pipeline but incorporated a pipeline-like model in their ablation study using sequential encoding instead of joint encoding.

\* the type of named entity does not have to be correctly predicted to count as correct prediction, as long as entities and relation were correct

† unspecified whether correct prediction of entity type is relevant for relation to be classified as correct

A final argument to be made for joint training is the reduction of training time (Li et al., 2021). Training time became particularly critical with large transformer models needing a longer time for training and hyperparameter tuning, only allowing hyperparameter tuning on a subset of training data (Cabot & Navigli, 2021). As such, it may be preferable to train one model for both tasks instead of separate models.

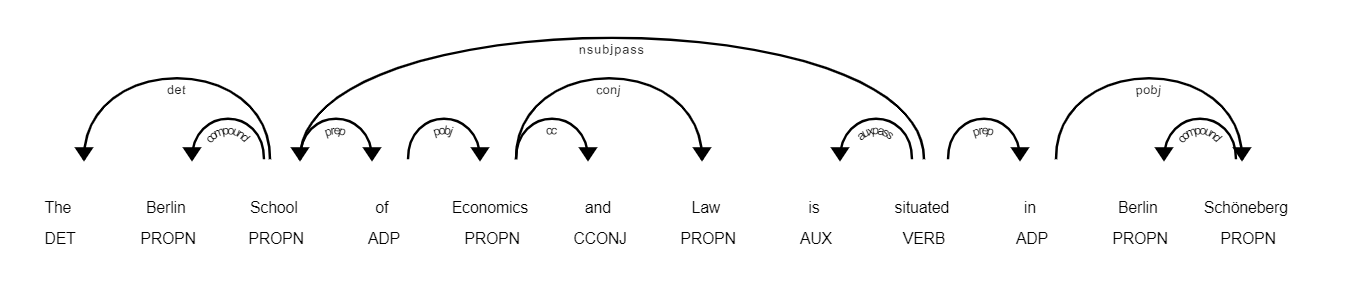
### Sequence-based or Dependency-based

Another differentiation of relation extraction models is separating them into sequence-based and dependency-based models (Guo et al., 2020). While sequence-based relation extraction models are purely based on text, dependency-based models also include features generated from dependencies trees.

Dependencies trees represent grammatical dependencies. Based on a dependency tree it can be inferred which words in a sentence are connected. An example for a dependency tree is visualized in figure X. Taking the sentence “The Berlin School of Economics and Law is situated in Berlin Schöneberg” again, “The Berlin School” and “Berlin Schöneberg” are compound word groups, and that they are indirectly connected by the root word “situated”.

Figure X

*A dependency tree for an example sentence, generated with SpaCy*



These trees offer further features and value when learning relation extraction as linguistic connections between words of the input are defined clearer. Dependency trees have been used successfully in the past with graph convolutional networks (Zhang et al., 2018; Guo et al., 2020) with particular success in document level relation extraction, characterized by long sentences and paragraphs. While several pruning strategies were used to distil the information of a tree into the most relevant parts (e.g. Xu et al., 2015; Miwa & Bansal, 2016; Zhang et al., 2018), Guo et al. (2020) show that using the full unpruned input helps when extracting relations from long inputs, such documents. Transformer-based methods have been proven similarly successful on relation extraction tasks with long inputs both with (Zeng et al., 2020; Xu et al., 2021) and without (Tan et al., 2022) dependency components and are currently achieving state-of-the-art performance.

## Transformers

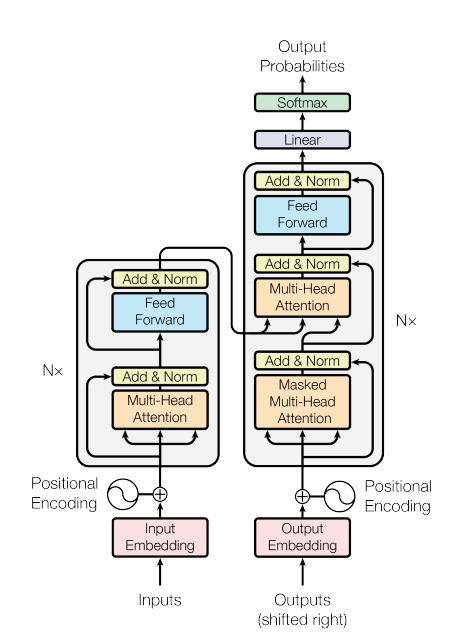
First introduced by Vaswani et al. (2017) as model for machine translation, transformer models revolutionized the performance of NLP models, including relation extraction models (SOURCE). At heart the transformer is based on feed-forward neural networks in combination with attention mechanisms. Additionally, the transformer uses an encoder-decoder structure, which has been proven successful in previous state-of-the-art language models, such as recurrent neural networks and long short-term memory neural networks. However, the attention mechanism is the key difference between the transformer model and these previous state-of-the-art models (Vaswani et al. 2017).

### Architecture

The transformer can be split into three overlapping main features: Embeddings, Encoder-Decoder Structure, and Attention. As the task the transformer was created in machine translation, the original goal was to create a translated output sentence given an English input sentence. The translation is generated word-by-word, each word being generated after a full pass-through the whole transformer architecture. Figure X shows the transformer architecture as designed by Vaswani et al. (2017).

Figure X

*Transformer Architecture, as seen in Vaswani et al. (2017)*



Inputs fed into the model get iteratively encoded over a set number of *N* encoders to receive a more refined word representation, based on the word embeddings, its position in the input sequence and its context to other words in the input, the latter being generated by the encoder. The decoder receives its input auto-regressively and creates similar word representations as the encoder for the so far generated words of the output sequence, based on the generated word itself, its position in the generated sentence and its context to other so far generated words. Importantly, Vaswani et al. (2017) uses one Encoder-Decoder multi-head attention layer, setting the output representation in context to the encoded input, allowing to see which words of the generated output refer to which words of the input and to create new context vectors. These attention vectors represent the decoder output after being adjusted through a feed forward two-layer feed forward neural network and normalization. Based on one additional linear activation and a softmax function, the probabilities for the next output word are calculated and the next output word for the generated sequence is chosen based on highest probability. Following, the attention mechanism and the encoder-decoder structure will be explained in further detail.

* + - 1. Attention

The scaled dot-product attention mechanism, previously referred to as “context to other words” describes how close words in a sequence are related to another. Vaswani et al. (2017) calculated the attention for every word in a given sequence as:

(1)

Given *Q*, *K*, *V* being the dot-product of the input embeddings and the weight matrices *WQ*, *WK*, and *WV*. These matrices are the randomly initialized and trained during the encoder step. Vaswani et al. (2017) introduce the scaling of the result by , counteracting the effect of small gradients when giving *K* high dimensionality (*dk*). The resulting attention vector is scaled using softmax, returning a vector with a float between zero and one, higher values indicating higher relation between words.

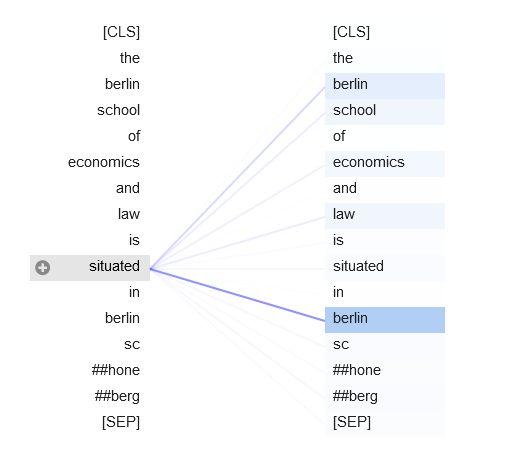
Instead of doing this scaled dot-product attention once, Vaswani et al. (2017) employ multi-headed attention, over a set number of heads *h* simultaneously. The weight matrices *WQ*, *WK*, and *WV* are split into *h* matrices of the same size, being used in different heads. Finally, the attention matrices calculated are concatenated. Vaswani et al. (2017) argue, that splitting this attention task into multi-head attention enables the embeddings to learn different features of the word in its position in the sequence and of its relation to other words within the sequence, potentially creating richer word encodings overall.

The multi-headed attention approach fixes the big struggle of recurrent neural networks, having to carry early token information through every following token, potentially resulting in information being lost and in exploding or vanishing gradients in case of long sequences (Le & Zuidema, 2016). Instead of gathering information from the input sequence sequentially, transformers can calculate the attention matrices in parallel for the whole sequence, as the attention of a word does not require the attention values of a prior word (Vaswani et al., 2017). This parallelization in encoding offers the potential of a speed-up due to the reduction of sequential steps, however coming at the cost of quadratic scaling in complexity for each step due to the attention being calculated between each word of the sentence. To overcome this issue in the case of long sequences, it is possible to restrict the attention into a specified window size, inducing linear instead of quadratic scaling (Vaswani et al., 2017).

Figure X shows the attention of the token “situated” to other tokens in the sentence. The visualization only shows one self-attention attention head generated from a BERT transformer (Devlin et al., 2018) with the BertViz python package. This head visualizes particularly well how transformers hold high potential for relation extraction as the relevant verb pays high attention to relevant noun-tokens.

Figure X

*Attention of verb token to other tokens in a sequence, one head of multi-headed attention*



*Note*. The respective head was chosen out of a selection of 144 potential heads as it visualizes the relationship between the tokens particularly well. Generated with the BertViz python package, using the BERT transformer (Devlin et al., 2018).

#### Encoder-Decoder-Architecture

Before using the encoder-decoder architecture with the transformer, it was used to achieve state-of-the-art performance with recurrent neural networks (Cho et al., 2014; Bahdanau et al., 2016). Encoders are used to create representations of the input sequence, while decoders create representations of the so far generated output sequence. Importantly, the use of an encoder-decoder attention layer allows to set both representations in context to another, as the attention between decoded word and the whole encoded sequence can be calculated at once (Bahdanau et al., 2016). The encoder-decoder attention in recurrent neural networks uses the whole encoded sequence but is limited to the last decoded token (Bahdanau et al., 2016) while the encoder-decoder multi-head attention used in transformers can attend to every decoder input and every encoder input (Vaswani et al., 2017).

This encoder-decoder multi-head attention step is particularly important in machine translation, as it helps learning which words in the encoded language, refer to the which words in decoded language. In the example of a sequence-to-sequence relation extraction where an output sequence “<triplet> subject <subj> object <obj> relation” is generated, the transformer will be able to learn special tokens, such as the <subj> token ends the subject phrase, and after generating both subject and object, the relation indicating verb will show high attention to both the decoded subject and decoded object. With a recurrent neural network using an attention encoder-decoder architecture, the relation indicating verb could not show high attention to specific already decoded tokens, but instead only to the last decoded tokens, being <obj> and indirectly to the prior tokens which were used as input to generate the <obj> token.

* Is there a picture that explains this?
* Parameters & Data Hungry
* Embeddings

### Transfer learning

### Types of transformers

* Seq2Seq
* Zero Shot
* Img2Seq?

1. Related Work