Deep Learning Methods for Semantic Parsing and Question Answering over Knowledge Graphs

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

von
Denis Lukovnikov
aus
Irkutsk, USSR

Bonn, 2021

1. Gutachter:	Prof. Dr. Jens Lehmann
2. Gutachterin:	Prof. Dr. Asja Fischer
Tag der Promotion:	2 May 2022
Erscheinungsjahr:	2022

I dedicate this work to my parents, my friends and my future wife and children.
For my mother who helped me in all things great and small.
For my maternal grandmother and grandfather, who helped raise me, but are no longer here.

Acknowledgements

First of all, I would like to thank my parents, who made this work possible not just by bringing me into the world, raising me, and teaching me, but also by always being there for me.

I would like to acknowledge the support of the Marie Curie-Skłodowska programme, the University of Bonn, Fraunhofer IAIS and the Ruhr University Bochum for providing me with the means to keep working toward my PhD degree.

Especially important for this work have been my supervisors, Prof. Sören Auer, Prof. Jens Lehmann and Prof. Asja Fischer. Without their support and guidance, this work would not have been possible. I would also like to thank all others who advised me in my work: Dr. Ioanna Lytra, Dr. Christoph Lange-Bever, Prof. María-Esther Vidal, and Prof. Saeedeh Shekarpour.

A special thanks goes to my awesome colleagues and co-authors Gaurav Maheshwari, Priyansh Trivedi, Nilesh Chakraborty, Sina Däubener, Dr. Henning Petzka, Agustinus Kristiadi and Asif Khan. Another special thanks goes to all my other wonderful colleagues and friends, who made life inside and outside the office fun.

I would also like to thank the wonderful people I met during conferences, summer schools, project meetings and other events, who made the journey much more interesting.

Finally, I would like to thank all my teachers and professors for sharing their passion and knowledge.

Abstract

Recently, the advances in deep learning have lead to a surge in research on semantic parsing and question answering over knowledge graphs (KGQA). Significant improvements in these fields and in natural language processing (NLP) in general have been achieved thanks to the use and better understanding of training neural-networks-based models. Particularly important in training any model for any task is their generalization ability. While the generalization ability of machine learning models can be improved with general techniques (e.g. dropout), semantic parsing and KGQA present unique generalization challenges that have been a focal point of research in the field. Other important aspects when using machine learning are its computational efficiency and response time, as well as the ability to measure the reliability of the predictions on given inputs.

In this thesis, we explore some questions regarding the generalization challenges in semantic parsing and KGQA. We also explore the tasks of out-of-distribution (OOD) detection for semantic parsing models, as well as the challenge of reducing the number of decoding steps. In particular, we investigate zero-shot or out-of-vocabulary generalization in KGQA with simple questions, which require only a single triple pattern to find the answers. Here, we are concerned with the ability to generalize to entities and relations that were not observed during training. Another question we investigate is the ability to detect compositionally OOD examples. Recent work has shown that standard neural semantic parsers fail to generalize to novel combinations of observed elements, which humans can easily do. While different works have investigated specialized inductive biases and training techniques, to the best of our knowledge, none have focused on detecting whether the inputs are compositionally OOD, which is the focus of our work. The third question we focus on is transfer learning in the context of KGQA, where we investigate its impact on both simple questions and more complex questions. Since the emergence of large-scale pre-trained language models (PLM), transfer learning from PLM's has been shown to significantly improve accuracy on various NLP tasks. In this thesis, we look at transfer learning from PLM's, additionally providing a qualitative analysis of the model and an investigation of data efficiency. We also look at transfer learning between KGQA tasks. A unique aspect that can be present in tasks requiring the generation of formal languages is order-invariance in the queries, which is the fourth point we focus on in this thesis. For example, in SQL, the order in which the conditions appear in the WHERE clause does not matter: the meaning of the query remains the same. Nevertheless, when training, typically only one possible linearization of the query is used, which can lead to the learning of spurious patterns that do not generalize. In this thesis, we investigate whether the standard training can be problematic and also explore an order-invariant training method. Finally, we also explore insertion-based decoding in semantic parsing. Usually, semantic parsing is performed in an auto-regressive left-to-right manner, which requires as many decoding steps as there are tokens in the decoded sequence. In this thesis, we explore alternative decoders that rely on insertion, and can thus achieve a sub-linear number of decoding steps and have different independence assumptions between output variables. In addition, we propose a novel insertion-based decoder for trees.

Contents

1 Introduction **1**

1.1 Research Directions . . . 2

1.1.1 Question Answering over Knowledge Graphs and Semantic Parsing . . . 2

1.1.2 Deep Learning . . . 3

1.2 Challenges . . . 4

1.3 Research Questions . . . 9

1.4 Contributions . . . 10

1.5 List of Publications . . . 13

1.6 Thesis Structure . . . 15

2 Background and Preliminaries **17**

2.1 Semantic Parsing and Question Answering over Knowledge Graphs . . . 17

2.1.1 Semantic Parsing . . . 17

2.1.2 Knowledge Graphs . . . 19

2.1.3 Question Answering over Knowledge Graphs . . . 19

2.2 Neural networks for sequence processing . . . 21

2.2.1 What's in a sequence . . . 21

2.2.2 Neural Network Components . . . 22

2.2.3 Neural Architectures for NLP . . . 26

2.3 Training . . . 30

2.3.1 Maximizing Likelihood . . . 31

2.3.2 Stochastic Gradient Descent . . . 32

2.4 Pretraining and transfer learning . . . 33

2.4.1 Pre-trained word embeddings . . . 33

2.4.2 Pre-trained Language Models . . . 35

3 Related Work **37**

3.1 Semantic Parsing . . . 37

3.1.1 Structured decoding . . . 37

3.1.2 Exploiting grammar in semantic parsing . . . 39

3.1.3 Exploiting semantic constraints . . . 39

3.1.4 Copying Mechanisms . . . 40

3.1.5 Other Structured Decoders . . . 40

3.1.6 Non-autoregressive decoding and semantic parsing . . . 41

3.1.7 Weak supervision for semantic parsing . . . 42

3.2 Question Answering over Knowledge Graphs 44
3.2.1 Simple Questions 45
3.2.2 Complex Questions 47
3.2.3 Datasets 51
3.3 Generalization 52
3.3.1 Symbol representations 53
3.3.2 The effect of order and linearization 53
3.3.3 Compositional Generalization 55

4 Word- and character-level representations in question answering over knowledge graphs 61
4.1 Introduction 62
4.2 Approach 63
4.2.1 Model description 64
4.2.2 Prediction 69
4.2.3 Training 70
4.2.4 Technical details 72
4.3 Evaluation 72
4.3.1 Influence of candidate generation 72
4.3.2 Comparison 73
4.4 Error analysis 74
4.5 Discussion 76
4.6 Conclusion 76

5 Transfer learning for question answering over knowledge graphs 77
5.1 Transfer Learning from Pre-trained Language Models for Simple Questions 78
5.2 Approach for Simple Questions 79
5.2.1 Entity span prediction 80
5.2.2 Relation prediction 80
5.2.3 Logical Form Selection 81
5.3 Experimental Setup for PLMs for Simple Questions 81
5.3.1 Metrics 82
5.3.2 Baseline 82
5.3.3 Effect of Limited Training Data 82
5.4 Results and Analysis for PLMs for Simple Questions 83
5.4.1 Full data results 84
5.4.2 Effect of Limited Training Data 84
5.4.3 Performance on test set 85
5.4.4 Final results 85
5.4.5 Attention analysis 85
5.5 Complex Questions, Ranking Models and Transfer Learning 88
5.6 Background: Query Graphs 89
5.7 Approach for Complex Questions 91
5.7.1 Core Chain Candidate Generation 91
5.7.2 Slot Matching Model 92
5.7.3 Predicting Auxiliary Functions 94

5.8 Experiments on Complex Questions . . . 95
5.8.1 Approach Evaluation . . . 95
5.8.2 Transfer Learning across KGQA datasets . . . 99
5.8.3 Transfer Learning with pre-trained Transformers . . . 100
5.9 Conclusion . . . 100

6 Insertion-based decoding for semantic parsing 103
6.1 Insertion Transformer . . . 104
6.2 Tree-based Insertion Transformer . . . 105
6.2.1 Decoding . . . 106
6.2.2 Model . . . 107
6.2.3 Decoder Training . . . 109
6.3 Experiments . . . 112
6.3.1 Results . . . 114
6.4 Discussion . . . 116
6.5 Conclusion . . . 116

7 Linearization order when training semantic parsers. 119
7.1 Introduction . . . 120
7.2 Queries, Trees and Linearizations . . . 120
7.3 Model . . . 122
7.3.1 The Seq2Seq Model . . . 122
7.3.2 Embedding Function of the Decoder . . . 123
7.3.3 Output Layer of the Decoder . . . 123
7.3.4 Pretrained Embeddings and Rare Words . . . 125
7.3.5 Coherence of decoded logical forms . . . 125
7.4 Training . . . 125
7.4.1 Non-deterministic Oracle . . . 126
7.4.2 REINFORCE . . . 126
7.5 Evaluation . . . 127
7.5.1 Experimental Setup . . . 127
7.5.2 Results . . . 127
7.6 Discussion and previous work . . . 130
7.7 Conclusion . . . 131

8 Detecting compositionally out-of-distribution examples. 133
8.1 Detecting OOD examples . . . 134
8.1.1 Posterior predictive distribution . . . 134
8.1.2 Heterogeneous ensemble . . . 135
8.2 Experiments . . . 135
8.2.1 OOD Detection Performance . . . 137
8.3 Conclusion . . . 138

9 Conclusion and future directions 141
9.1 Research Questions . . . 142

9.2 Limitations and Possible Improvements 144
9.3 Future Research Directions 145
9.3.1 Compositional Generalization 145
9.3.2 Transfer Learning 146
9.3.3 Alternative execution and supervision 146
9.3.4 Graph Representation Learning 147
9.4 Closing Remarks 147

Bibliography **149**

A List of Publications **167**

B An example of insertion-based decoding for a real example. **169**

List of Figures **171**

List of Tables **173**

CHAPTER 1

Introduction

"A beginning is the time for taking the most delicate care that the balances are correct."

– Princess Irulan, from *Dune*, by Frank Herbert

Realizing the age-old dream of Artificial Intelligence (AI) remains a distant prospect. Perhaps one of the most important desired aspects of "the AI" would be the ability for it to communicate naturally with humans in a human language. As opposed to formal languages that are used to unambiguously instruct machines to perform certain actions, human language expressions can be difficult to really understand without a vast body of knowledge about the physical world and human perception of it. And while this world knowledge can help resolve some ambiguity, or at least make certain interpretations more probable, natural language remains ambiguous, occasionally fooling even humans.

Despite the vast fundamental difficulties we still have to overcome, several research communities have focused on developing systems that possess a certain degree of semantic understanding of human language. This is one of the most important problems in natural language processing (NLP) as well as information retrieval (IR). Understanding human languages is especially interesting for the purpose of question answering, where the goal is to compute a direct answer to a question. However, language understanding also has applications beyond IR, for example in robot instruction, program generation, and conversational agents.

Question answering enables us to go beyond a simple keyword-based search that returns a ranked list of relevant pages (e.g. Google Search) and transition to a more focused and effective search paradigm where the system reasons over the available knowledge to provide a direct answer. In fact, Google Search already integrates question answering in its search results when the answer is clear. Similarly, Google Assistant, Apple's Siri, Amazon's Alexa and other virtual personal assistants also integrate question answering to answer user questions directly. Question answering is especially useful in such applications because of the limitations of the voice interface: it is not possible or not very user-friendly to communicate the usual search result list.

Different types of question answering tasks exist, depending on the way the questions and the knowledge are provided to the system. The question is usually provided in the form of a sentence in natural language (e.g. English). Sometimes, the question is presented in the context of a conversation, which requires the understanding of the coreference structure. On the other hand, the knowledge can be provided in various different ways, like text, databases, knowledge graphs and even images. When structured data sources are used, such as relational databases or knowledge graphs, question answering is usually performed by relying on a *semantic parser* that translates the natural language question into a formal query in a compatible query language. This query can then be executed by a database to retrieve the answer to the question in the form of a narrow subset of its data (e.g. a set of entities).

While many ways have been proposed to develop semantic parsers and question answering systems, recently, there has been increased interest in neural network based methods from both academic and industrial communities. As with other application areas in NLP and beyond, deep learning automatically learns to extract relevant features, and thus enables us to avoid manual feature engineering, which was characteristic for earlier machine learning methods[1].

In this thesis, we focus on deep learning for semantic parsing and question answering over knowledge graphs (KGQA), which is a broad research field. The selection of concrete questions we focus on will be elaborated later in this introduction. Most of these questions concern generalization while others also focus on methods to detect when predictions may be unreliable, as well as potential methods to improve response times.

Next, in Section 1.1, we elaborate more on semantic parsing, question answering over knowledge graphs and deep learning in their context. After that, we will discuss some challenges in their context in Section 1.2. In Sections 1.3 and 1.4, we discuss respectively the research questions and contributions of this thesis. Subsequently, in Section 1.5, we list the publications of the author of this thesis. We conclude the introduction with a presentation of this thesis's structure in Section 1.6.

1.1 Research Directions

In this section, we elaborate more on the research directions of this thesis, first discussing semantic parsing and question answering over knowledge graphs and then the application of deep learning in this context. See also Chapter 2 for more concrete details.

1.1.1 Question Answering over Knowledge Graphs and Semantic Parsing

Question answering can be performed using different sources of knowledge: relational databases, graph databases and knowledge graphs, text, images, other multimedia, and any combination of these. These different sources of knowledge can be categorized as structured and unstructured. With *structured* sources, we refer to those sources where the meaning of the stored information is unambiguously defined in a formal logical system. Structured sources include knowledge graphs and relational databases. With *unstructured* sources, we refer to sources where the meaning of the stored information is not clearly indicated and where the data must be interpreted in their raw form in order to compute an answer. Text and images are examples of such sources.

Note that structured sources of knowledge usually provide some automatic way of querying the contained information, for example, relational databases typically support retrieval using SQL queries and RDF knowledge graphs can be queried using SPARQL. While the underlying way of storing data may be different, as well as the languages used to query them, question answering by building a query is fundamentally the same task with minor differences.The task of building a query given a question is

[1] Features are used by machine learning methods to describe examples numerically in order to be able to distinguish between them and make decisions.

a form of semantic parsing, and the different underlying storage methods give rise to various flavours of semantic parsing, for example text-to-SQL for relational databases and text-to-SPARQL for question answering over knowledge graphs (KGQA).

Relational databases are well-established in the industry, and have been an essential component in various applications. While relational databases store data in multiple tables, each with its own schema, graph databases store data in a graph, where every fact is stored as an individual triple, rather than being part of a database row with other facts. The advantage of graph databases is higher flexibility of data storage, which allows to easily adapt the schema without affecting other data, and easily insert additional facts. Several large-scale knowledge graphs have been developed, which contain general factual knowledge about the world. Some of the most prominently known examples are Freebase [1], DBpedia [2, 3] and WikiData [4]. These knowledge graphs contain millions of facts about millions of entities over various topics and thus form an enticing source of knowledge for general-domain question answering applications and research. In addition, some standardization efforts like RDF enable a better publishing and interoperability of data, resulting in the easier re-use of existing knowledge graphs and the growth of an interconnected "cloud" of data. In fact, the Semantic Web community fosters the growth of Linked Open Data, which has as goal to go beyond a Web of just interlinked pages to a Web of interlinked facts. The LOD principles enable a new knowledge graph to be connected to existing graphs and be easily reused by others.

Semantic parsing is the task of translating a natural language utterance into a machine-understandable form that retains the meaning of the original utterance. This includes, for example, converting a question into an SQL query or a SPARQL query. The semantic parse is considered correct if it accurately conveys the full meaning of the input utterance, in the context of the used formal language and knowledge source. When executed over a source of knowledge (e.g. a relational or a graph database), the correct semantic parse should yield the expected answer. Semantic parsing is useful for various applications, including question answering, robot instruction and conversational systems.

In this work we focus mostly on question answering over knowledge graphs, although we also investigate some aspects of the semantic parsing task using other structured knowledge sources. Note that semantic parsing for question answering over knowledge graphs is similar to semantic parsing for other sources, although it has its own unique characteristics, such as a large number of unique entities and predicates.

1.1.2 Deep Learning

Question answering systems and semantic parsers can be developed using different approaches. While manual rules can be used to create a system for question answering, such systems are likely to fail on a new domain or even new data, requiring continuous manual (re-)engineering efforts. The field of machine learning offers a solution by providing methods that automatically infer rules or patterns from a collection of examples of a task (this is called "training") such that they can be applied to solve a new, previously unseen example. In the recent decades, and most notably in the recent years, incredible progress has been made in the field of deep learning. Compared to traditional machine learning methods (e.g. Support Vector Machines [5]), where features must be manually predefined and are then used in a classifier or regression model, deep learning methods aim to automatically learn to extract useful features. This paradigm shift, combined with the rapid increase in computing power and availability of data, resulted in the development of novel learning techniques for various applications. Notable success stories include Generative Adversarial Networks (GAN) [6], which for example can be trained to generate realistic looking faces [2], cats [3], anime characters [4], and more. Another breakthrough, in the field of NLP, was the development of BERT [7] and other pre-trained language models, which enabled transfer learning from more powerful pre-trained models. One of these pre-trained models is the recently released OpenAI GPT3 [8], which was impressive in its ability to generate coherent looking and eloquently written text and even other things (e.g. HTML code).

Like other NLP tasks, semantic parsing and KGQA can similarly benefit from deep learning, and it is important to investigate related questions. The ability to learn and pre-train representations rather than engineer features can make it easier to train and adapt semantic parsers for new domains. However, as recent works pointed out, we need to take special care of compositional generalization (see Section 3.3.3). Additionally, semantic parsing and KGQA have certain characteristics that make them a unique structured prediction problem, compared to the well-studied sequence generation methods used for machine translation. In particular, semantic parsing can often be formulated as a sequence-to-tree (or more generally sequence-to-graph) task, where the tree-structured output can inspire certain modeling choices. Another aspect is that semantic parsing for question answering over databases produces executable programs, which, when executed produce answers that can be used for supervision. This is called weak supervision and is an important question in semantic parsing that separates it from machine translation. In the following sections, we more thoroughly describe the challenges and research questions we focus on and state our contributions.

1.2 Challenges

Perhaps the most important challenge in machine learning is the ability of the trained models to generalize to new inputs. Developing a model that perfectly fits the training data is easy, especially with the usually over-parameterized models used in deep learning. However, performance on training data is not indicative of future performance on inputs that have not been observed during training. Regularization techniques such as dropout [9], weight decay, and early stopping are the most commonly known general-purpose techniques to reduce the degree to which (neural network) models over-fit the training data. However, in semantic parsing and question answering over knowledge graphs, we are confronted with a set of specific generalization challenges, such as generalization to unseen tokens and generalization to unseen combinations of tokens[5]. In fact, in the work proposing GrailQA [10], the authors point out three different types of generalization: (1) i.i.d. generalization, (2) compositional generalization, and (3) zero-shot generalization. While the i.i.d. setting is the typical generalization problem of generalizing to examples from the same distribution as the training distribution, the other two types are more challenging. Compositional generalization concerns generalization to novel combinations of already observed elements while the zero-shot generalization concerns generalizing to novel tokens and novel domains. Another important aspect of machine learning based methods is the ability to detect when the model's predictions are unreliable. This is especially important in critical applications such as autonomous vehicles. Another important challenge in deep learning is working with finite amounts of training data, which can be small for some tasks, including some

[2] `https://thispersondoesnotexist.com`
[3] `https://thiscatdoesnotexist.com`
[4] `https://make.girls.moe`
[5] Given a natural language expression, we first have to split it up into a sequence of tokens to process the sentence. The tokens can be words, characters or sub-word tokens, depending on how we split the sentence.

semantic parsing tasks. Techniques such as data augmentation [11] and transfer learning can be used here. Beyond these generalization-related challenges, other challenges exist, including (1) reducing the prediction and training time, (2) using weak supervision for training, and (3) conversational question answering. In the following the challenges related to this thesis are discussed in more detail.

Challenge 1: Generalization to unseen tokens

Semantic parsing is essentially a task that requires the translation of a sequence of natural language tokens to a structure (e.g. a tree) consisting of formal language tokens. This first challenge relates to the ability of the used models to generalize to novel tokens at test time that have not been observed during training. Essentially, this is generalization to out-of-vocabulary (OOV) and rare tokens, which can also be a problem in machine translation (MT). This challenge is also related to *zero-shot* generalization described in the work of Gu et al. [10].

The challenge of generalization to unseen tokens arises when at test time we need to generate (and interpret) tokens that we have not observed during training. Usually, in neural network-based models for NLP, every token that can be used in the input or that can be generated in the output is represented using a unique vector of parameters. So if a token is not observed or observed very infrequently, its representations are not or poorly trained. This challenge can be especially important in KGQA because there are thousands of relations and millions of entities so creating natural examples covering all entities (and even all relations) can be too expensive. A possible solution is data augmentation, for example, by simply generating novel examples with different entities inserted both in the input sentence as well as in the output query. Typically, most KGQA systems employ a dedicated *entity linking* component that determines which spans in the input sentence correspond to which entity in the underlying KG. However, this approach might not work well for all relations and the ways in which they can be expressed in the natural language.

Alternatively, it is also possible to generalize to novel symbols within the main semantic parsing model by instead learning to build token representations from additional information about that token. For example, a word can also be represented as a sequence of the characters constituting the word, or its vocabulary definition. When this additional information itself is conveyed using a set of tokens that is always observed during training, we prevent using untrained or poorly trained token representations, although it is still not a guarantee that the resulting representations generalize well (for example, Schick and Schütze [12] found that BERT, which uses a WordPiece vocabulary and is trained on vast amounts of natural language text, still handles many rare words poorly).

Note that this challenge can also be viewed as an out-of-distribution (OOD) generalization challenge: the distribution of training data is different from the distribution of test data because test data contains symbols not observed during training.

Challenge 2: Compositional generalization

Even when all necessary symbols have been observed during training, it can be challenging for machine translation and semantic parsing models to generalize well to previously unseen examples.

The challenge of generalization to novel combinations concerns the ability to understand and generate novel patterns/combinations of tokens at test time. It is referred to as *compositional generalization* in the literature since it concerns the generalization regarding compositionality of language, the principle that the meaning of a phrase is determined by (1) the meaning of its parts and (2) its syntactic structure (which dictates how the parts are combined). We can view this as another type of OOD generalization: the distribution of the compounds seen during training is different from the test data and thus we could call this type of OOD generalization problem *compositional* OOD generalization. Note that it is possible for the test data to follow the same distribution over tokens and thus not suffer from OOV problems, but have different co-occurrence probabilities of tokens between train and test data, and thus be at risk of learning spurious patterns that do not generalize well. Recent work [13, 14] found that baseline sequence-to-sequence models generalize poorly when confronted with a compositionally challenging train-test split of the data. In such splits, the models appear to learn both useful (general) patterns as well as spurious patterns (arbitrary correlations) in the training data.

Various approaches have been introduced to improve the compositional generalization of sequence-to-sequence and semantic parsing models, which include compositional inductive biases that better implement some useful independence assumptions for semantic parsing, but also data augmentation, iterative back-translation, and using pre-trained models. See Section 3.3.3 for a more detailed overview.

While usually not explicitly targeting the challenge of compositional generalization, we believe that many works investigating better inductive biases for semantic parsing, such as tree decoders, might also result in improved compositional generalization. In fact, such works investigate different ways to condition some output variables on other output variables, in order to improve generalization. For example, the tree decoder of Dong and Lapata [15] conditions the generation of a node directly on the parent state, as well as the previous sibling, in contrast to a sequence decoder that always conditions on only the previous state of the generated sequence. Such improvements may have a measurable effect on challenging compositional generalization datasets. For example, Guo et al. [16] found that hierarchical decoding (similar to the Coarse2Fine decoder of Dong and Lapata [17]) already improves results on the CFQ dataset.

Please note that we do not address this challenge directly in this work but rather consider this as background for Challenge 3.

Challenge 3: Measuring the reliability of model predictions

A model developed for semantic parsing may work well on the dataset used to test and compare methods, especially if the test set has been drawn from the same data generating distribution as the training set. However, what happens when the model encounters an example that is from a different domain or an input that is not even a question or a coherent sentence? The model trained to predict logical forms will nevertheless try to predict some logical form, and that logical form may even turn out syntactically correct and produce an answer when executed. However, in such cases it is unlikely that the query correctly captures the meaning of the input.

Even when we are not faced with a different domain or other OOD cases, and the input is similar to those seen during training, the model may still make mistakes because it did not learn or generalize well enough to correctly process the new input. It is important to have well-calibrated models, whose predictive uncertainty is indicative of their accuracy. When models fail and they are miscalibrated, the probability of failure, computed from the output distribution, may be lower than in reality. This over-confidence may lead to more dangerous situations in critical real-world applications.

In both in-distribution and out-of-distribution settings, we would like to be able to detect when the output of the model is not trustworthy. This could be used, for example, to decide to run a more powerful but more expensive model, to convey to the user that the model was not able to reliably process the request, and/or to mark the new data point for annotation for further training of the model.

In critical application areas, such as autonomous vehicles, uncertainty quantification is crucial for safety and robustness to unexpected and OOD inputs and is an important direction of research and development. Semantic parsing and question answering may not always be employed in a critical application, but nevertheless has several potential application areas where the cost of failure is high, for example in medical expert systems. In addition, measurable reliability can greatly improve user experience and efficiency in deployment, and may effectuate safe deployment in more critical applications.

Neural network models are usually trained within a probabilistic framework, where the output of a classifier that uses a softmax or sigmoid output layer can be interpreted as a probability distribution over the possible classes. This output distribution can be used to quantify the uncertainty of the prediction: a more entropic distribution should indicate higher uncertainty. Another possible measure is the probability or log-likelihood of the most probable (predicted) class. However, models may still fail while producing high-confidence predictions. To better model uncertainty over model parameters, Bayesian Neural Networks (BNN) can be used. Rather than training to find a point estimate of the parameters, BNN are trained to find a *distribution* over parameter values. The predictive posterior (which integrates over the possible parameter values) of a BNN then also captures the parameter uncertainty.

In this thesis, we focus on out-of-distribution detection in semantic parsing. OOD detection is an important problem that focuses on deciding whether the input is from the data distribution that was used for training and thus may help decide whether the model is likely to fail or if the input is similar to the training data and thus that the model prediction is probably reliable. What makes this problem different from simply training a classifier is the fact that out-of-distribution data are usually not observed during training.

Challenge 4: Training with (relatively) little task- and domain-specific data

Another aspect affecting the performance of machine learning methods is the amount of available training data. More data usually leads to models that generalize better. However, in semantic parsing and KGQA, data collection can be an expensive process. With the limited amount of data available, networks become more prone to over-fitting.

To mitigate this, data augmentation (for example, see the approach described by Jia and Liang [11]) can be used. Data augmentation automatically generates additional training points in order to improve robustness to irrelevant data variations. Another approach to mitigate this is simply to collect more training data. Thus, research into how to make data collection simpler and cheaper would be practically useful.

Another useful family of techniques is transfer learning. In transfer learning, the goal is to learn knowledge from one task and re-use the knowledge in another task. Even though no new data for the task is generated, it essentially aims to re-use data from one task with more data for another task with less data. If the initial task has much more training data, and some of the knowledge learned from that task is relevant for a new task, then the model trained on the initial task can be seen as providing a prior over the model parameters. Subsequent training on another task (this is called fine-tuning) starts from these parameter values and finds more suitable values for the concrete task. In practice it appears that based on this procedure often a good optimum for the new task is achieved that generalizes better to new examples, which is usually attributed to the pre-training phase acquiring knowledge over multiple domains as well as domain-agnostic knowledge, and the fine-tuning phase reusing that knowledge (and thus also the corresponding data). In this thesis, we investigate transfer learning for KGQA.

Challenge 5: Reducing decoding time and other computational requirements

Another aspect of systems using machine learning is the computational efficiency both in training and inference. While both are important, efficiency in inference is arguably most important for most practical uses since it dictates infrastructure requirements (and thus cost) in deployment. Unless the requirement is that the models keeps continually learning from all user interactions, training usually happens much less frequently than inference.

We distinguish three types of efficiency here. Firstly, there is response time, which is the amount of time necessary to process a request. Some practical applications require fast inference while others are less time-critical. Secondly and thirdly, there are the computational and memory efficiency, or how much computations (FLOPs) and memory running a certain model takes. On hardware with computational and memory limitations, response time can be impacted by the available memory and computational power. When not enough computational power is available, the parallelism of a model can not be fully exploited and response time increases. Similarly, models requiring too much memory may make it impossible to fully exploit the computational power of the hardware. Reducing response time, computational and memory requirements can improve user experience and can be economically and ecologically interesting.

Response time: When decoding sequences, typically a left-to-right autoregressive decoding approach is used that decomposes the probability $p(y|x)$ of the entire sequence y of length T in the product of probabilities that take into account all previously generated tokens at every time step: $\prod_{i=1}^{T} p(y_i|y_{<i}, x)$[6]. This decoding approach leads to linear time complexity and response time: the next token can not be predicted before we know all the previous tokens. Within every decoding step, there might also be steps that have to be taken sequentially, for example when we have multiple layers. Recurrent neural networks implement this factorization naturally by conditioning the next state h_t directly on the previous output y_{t-1} and previous state h_{t-1}. The state h_t is the used to compute a distribution for the next token y_t.

Even during training with teacher forcing, when both x and y are given, RNNs have response time linear in the length of y because they have to compute the previous state h_{t-1} before computing the state to produce y_t. In contrast, in transformers [18] the feature vector $h_i^{(l)}$ for word y_i in the l-th layer only depends on the outputs of layer $l - 1$ (previous layer) for all preceding tokens ($y_{<i}$). Thus, the feature vectors can be computed simultaneously for all positions i in the same layer. While at inference time, transformers still depend on the previously generated token and have linear response time, increasing the parallelism (constant response time given sufficient computational power) during training allows for faster training of large language model transformers (e.g. BERT [7]).

To decrease response time, aside from using shallower and less heavy models, we can also look into alternative, not fully auto-regressive decoding methods that propose sub-linear or even constant number of decoding steps for sequence generation.

Number of operations: The number of operations (FLOPs) is defined by the architecture and input data. With larger model dimensions, the matrix-matrix multiplications and other tensor manipulations

[6] $y_{<i}$ is a short-hand notation we use to refer to the i-length prefix of sequence y.

that implement various neural network layers, require more computations. Some of these operations (e.g. matrix multiplication, convolutions) can be parallelized on specialized hardware (GPUs, TPUs). More parallelized architectures and implementations can reduce the response time but would not necessarily affect the total computation. While with current hardware advancements, computation becomes more and more accessible, it is still important to investigate more efficient architectures for better cost efficiency in deployment or to make deployment in less powerful environments (e.g. smartphones, Raspberry Pi, Potato GLaDOS) possible.

Memory requirements: While modern hardware usually has sufficient memory, it is still limited and thus it is important to develop methods without prohibitively large memory requirements. The memory efficiency of a method can also affect response time and hardware requirements in deployment as it could decrease the amount and/or sizes of examples being processed on the available hardware.

In this thesis, we focus on decreasing the number of decoding steps, which may help improve response time, but may also increase or decrease accuracy.

1.3 Research Questions

In this work, we focus on a subset of the challenges presented above. The concrete research questions we focus on are listed and discussed here. They naturally tie in with the contributions presented next.

RQ1: Does combining word- and character-level representations improve accuracy in KGQA?

One of the challenges associated with semantic parsing and question answering over knowledge graphs is the generalization to novel tokens that have not been observed during training (Challenge 1). External entity and relation linking tools can be used. However, it is also interesting to investigate alternatives that enable end-to-end systems for KGQA with better generalization for out-of-vocabulary and rare output tokens. With this research question, we aim to investigate the combination of word- and character-level token representations for KGQA.

RQ2: Does transfer learning improve KGQA accuracy?

Relating to the challenge of limited training data (Challenge 4), transfer learning from other domains and other tasks can be seen as indirect data augmentation. The specific questions we consider here are (1) how much does using a pre-trained language model increase performance for simple questions as well as for complex questions, (2) how does performance degrade when the number of training examples is decreased, (3) how much does transfer learning from other KGQA tasks increase performance?

RQ3: Does insertion-based decoding improve accuracy and how much does it decrease the number of decoding steps?

Normally, left-to-right auto-regressive decoding is used in semantic parsing, where one token is decoded in one decoding step, which implements a left-to-right factorization of the joint probability of the predicted sequences. While reducing the number of layers and using smaller layers should decrease the response time, a reduction in the number of decoding steps can also be interesting for reducing response times and possibly overall computation.

With this research question, we aim to investigate insertion-based decoding in semantic parsing. This can help with the challenge of reducing response time of neural semantic parsers (Challenge 5) and is also a design choice that is interesting to investigate for generalization. When doing insertion-based decoding, it is possible to insert many tokens in parallel in a single decoding step, where the generated tokens are not conditioned on each other. This leads to different dependencies between the elements of the output structure, compared to left-to-right decoding and may affect accuracy.

RQ4: Does the query linearization order used during training affect accuracy?

When training neural semantic parsing models, typically left-to-right autoregressive sequence-to-sequence/tree models are trained using a fixed linearization of a query tree. However, queries may contain parts where changing the order does not change their meaning. It has been shown that the order matters when encoding or decoding sets [19]. In addition, using only a single linearization of a tree may lead to the learning of *spurious* patterns, especially with low numbers of training examples. Thus, with this research question, we aim to investigate how linearization order affects training. This question also relates to the challenge of compositional generalization: learning spurious patterns that arise due to training with only one possible linearization of every query may hurt generalization to novel combinations. In fact, Guo et al. [16] show that their order-agnostic *poset decoder* achieves improvements on the challenging splits of the recently proposed CFQ dataset [14] for measuring the compositional generalization for semantic parsing for KGQA.

RQ5: Can we detect compositionally out-of-distribution inputs in semantic parsing?

It has been shown that standard sequence-to-sequence models are not able to generalize well to novel combinations of observed elements. Recently, several works have explored different ways to improve the compositional generalization in semantic parsing. In this work, we instead ask the question of how easy it is to *detect* when the model receives a compositionally out-of-distribution input. It is possible for neural network models to fail in its predictions on novel input while also being confident. Thus, it is important to assess and improve the ability to detect compositionally OOD inputs, which could help to inform the decision on whether the model's predictions on that input can be trusted. This relates to the challenges of compositional generalization (Challenge 2) as well as measuring the reliability of a semantic parser's predictions (Challenge 3).

1.4 Contributions

This thesis covers different topics within deep learning applied to semantic parsing and question answering over knowledge graphs. More specifically, we identify the following contributions:

Contribution 1: *An investigation of word- and character-level representations for answering simple questions over knowledge graphs.* We propose an end-to-end neural-network–based approach for answering simple questions over knowledge graphs that considers entities and relations on word- and character-level. Simple questions are those that require only a set of single facts to be answered. Questions that require the integration of multiple facts are not considered simple. We follow an approach where we rank entity-relation pairs against the question, filtering for valid entity-relation combinations. Within this general approach, we investigate the representation of entities and relations on word- and character-level, in order to improve the generalization to previously unseen relations and entities. We perform experiments on the SimpleQuestions [20] dataset. This contribution corresponds to research question **RQ1** and is presented in Chapter 4.

Contribution 2: *An investigation of transfer learning for question answering over knowledge graphs.* We mainly investigate transfer from pre-trained language models but also provide some experiments on transfer from another question answering task. Pre-trained language models (PLM) are first trained on a large collection of text with an objective that trains the models to predict the next (left-to-right language model) or missing (masked language model) words. Using this form of pre-training with subsequent fine-tuning on downstream tasks, for example sentiment classification, has shown significant improvements. Currently, PLM's are as ubiquitous in NLP as pre-trained word embeddings were before them (see also Section 2.4).

We investigate the use of PLM in the context of answering simple questions as well as more complex questions. For simple questions, we perform an analysis where we vary the number of examples to show the improvement that fine-tuning a PLM has over training from scratch when less training data is available. We also investigate pre-training on one KGQA task with complex questions and fine-tuning on another KGQA task with complex questions. We perform experiments on the SimpleQuestions [20] dataset, and also use LC-QuAD [21] and QALD [22]. Additional contributions as part of this work is the development of architectures for answering simple questions over knowledge graphs as well as complex questions. This contribution corresponds to research question **RQ2** and is presented in Chapter 5.

Contribution 3: *An investigation of insertion-based decoding for semantic parsing.* Usually sequences and other structures are decoded in a left-to-right autoregressive manner, where the next predicted token or action is predicted based on all previously predicted elements. With this approach, n decoding steps are needed to decode a sequence of n elements and every token is conditioned on all preceding tokens.

In our work, we investigate insertion-based methods that can produce multiple tokens in a single decoding step. In particular, we investigate sequence-based insertion decoders for semantic parsing, and propose a *tree-based insertion decoder*. Trees are very important structures for formal languages: abstract syntax trees capture the syntactic structure of programming languages and some languages commonly used in semantic parsing directly express the tree of function applications and their arguments. While sequence-based insertion decoding [23] needs only $\log_2 n$ decoding steps to decode a sequence of n elements, the proposed tree-based insertion decoder can achieve a number of decoding steps below $\log_2 n$, where the exact number of steps heavily depends on the structure of the tree. Moreover, explicit tree-based decoding allows us to omit structure tokens that are needed in the insertion-based sequence decoding approach to indicate the tree structure, which results in smaller structures.

Additionally, it is interesting to investigate what effect maximally parallel insertion-based decoding has on generalization, both in the sequence-based and tree-based approaches. In the investigated approaches, the tokens that are generated in parallel are predicted independently, and tree-based insertion decoding decodes in a different order. This may increase or decrease accuracy on the task. In addition, the model design choices (our tree-based approach models the outputs differently than the sequence-based approach) and resulting inductive bias may affect accuracy.

We perform experiments to compare normal left-to-right autoregressive sequence decoding, autoregressive tree decoding (we use an approach similar to [15]), semi-autoregressive sequence-based insertion decoding [23], and semi-autoregressive tree-based insertion decoding. For experiments, we use the Overnight dataset, a well-known semantic parsing dataset that consists of examples with complex questions for different domains. This contribution corresponds to research question **RQ3** and is presented in Chapter 6.

Contribution 4: *An investigation of the effect of order of linearization of trees during the training of semantic parsers.* As elaborated before, the order of linearization can affect training and generalization performance. We investigate the effect of order on a text-to-SQL semantic parsing task using the WikiSQL dataset. In an effort to make the training of neural semantic parsers more order-agnostic, we develop a novel training approach based on dynamic oracles [24]. When training using teacher forcing, the model is only exposed to the given decoding path. This introduces exposure bias and may lead to the learning of unwanted correlations, especially when training data is limited. In contrast, a dynamic oracle allows the model to make suboptimal decisions and is able to provide the sequence of actions to achieve the best possible final state starting from any state, even suboptimal ones. In our experiments, since we are only concerned with order, we only define a dynamic oracle that works as long as the gold output is reachable. We experimentally show that this approach can be beneficial when the training data are not ordered consistently. Another contribution of our work on linearization order is a simple neural architecture that outperformed the state-of-the-art on the WikiSQL dataset at the time the work was initially performed. This contribution corresponds to research question **RQ4** and is presented in Chapter 7.

Contribution 5: *An investigation of detection of compositionally out-of-distribution examples in semantic parsing and question answering over knowledge graphs.* To investigate whether we can detect compositionally OOD examples, we use the SCAN [13] and CFQ [14] datasets, which propose data splits that are challenging the compositional generalization, such as length-based splits (where generalization to longer outputs is tested) and *maximum compound divergence* (MCD) splits, which follow a data splitting approach based on maximizing the divergence between compound distributions of the training and test sets.

In this context, we investigate uncertainty quantification methods that use the output distribution of the (discriminative) model trained for the task. We assess how well different measures derived from the output distributions (entropy, likelihoods of the highest scoring tokens) can be used to separate in-distribution (ID) data from OOD data. We also investigate the use of pseudo-Bayesian neural networks by experimenting with MC dropout [25]. From this analysis, it appears that different architectures are better at detecting different types of OOD examples. Thus, we also develop a heterogeneous ensembled method that combines the best of RNN and transformer-based models for compositional OOD detection. This contribution corresponds to research question **RQ5** and is related to Challenges 2 and 3. The contribution is presented in detail in Chapter 8.

1.5 List of Publications

Most of the work for this thesis has been published and/or presented at international conferences, relevant high quality workshops or journals. Listed below are all the works that have been written based on studies conducted during the author's PhD studies. Note that we do not use all these works in this thesis because some are not specifically connected with the main topic of this thesis, semantic parsing and KGQA. Nevertheless, there exist connections, as mentioned below. More details regarding publication venue, individual contributions as part of a joint publication are given for every publication. In the list, the asterisk * indicates authors with equal contribution.

1. **Denis Lukovnikov**, Asja Fischer, Jens Lehmann, Sören Auer. *Neural network-based question answering over knowledge graphs on word and character level.* In Proceedings of the 26th International Conference on World Wide Web (WWW 2017). 2017. DOI: 10.1145/3038912.3052675

   I conducted most of the study. The other authors helped in the writing and acted as supervisors.

2. Henning Petzka*, Asja Fischer* and **Denis Lukovnikov**. *On the regularization of Wasserstein GANs.* In the Conference Track Proceedings of the 6th International Conference on Learning Representations, (ICLR 2018). 2018. An earlier version of this work was also presented at the 2nd OTML workshop at NIPS 2017.

   While my co-authors developed the theory in this work, I implemented and ran most of the experiments and helped in the writing. This work is not used in this thesis since it concerns Generative Adversarial Network (GAN) in general and is thus not explicitly aimed at semantic parsing or question answering, although there exist connections with generative models (GANs are a well-known type of generative models).

3. **Denis Lukovnikov**, Nilesh Chakraborty, Jens Lehmann and Asja Fischer. *Translating Natural Language to SQL using Pointer-Generator Networks and How Decoding Order Matters.* In the AAAI 2019 Reasoning for Complex Question Answering Workshop. 2019.

   I conducted most of the study. Nilesh Chakraborty helped with the final experiments and writing. Other co-authors acted as supervisors and helped with writing.

4. Agustinus Kristiadi, Mohammad Asif Khan, **Denis Lukovnikov**, Jens Lehmann and Asja Fischer. *Incorporating Literals into Knowledge Graph Embeddings.* In Proceedings of the 18th International Semantic Web Conference (ISWC 2019). 2019. DOI: 10.1007/978-3-030-30793-6_20

   The first two authors, Agustinus Kristiadi and Mohammad Asif Khan conducted most of the work. I contributed in discussions and writing. The other co-authors acted as supervisors and helped writing. This work is not used in this thesis since it concerns knowledge graph embeddings in general and is thus not explicitly aimed at semantic parsing or question answering. However, there exist connections, in particular, some works have directly used knowledge graph embeddings for question answering over knowledge graphs.

5. **Denis Lukovnikov**, Asja Fischer and Jens Lehmann. *Pretrained Transformers for Simple Question Answering over Knowledge Graphs.* In Proceedings of the 18th International Semantic Web Conference (ISWC 2019). 2019.

I conducted most of the study. Other co-authors acted as supervisors and helped with writing.

6. Gaurav Maheshwari*, Priyansh Trivedi*, **Denis Lukovnikov***, Nilesh Chakraborty*, Asja Fischer and Jens Lehmann. *Learning to Rank Query Graphs for Complex Question Answering over Knowledge Graphs.* In Proceedings of the 18th International Semantic Web Conference (ISWC 2019). 2019. DOI: 10.1007/978-3-030-30793-6_28

   This work was done in close collaboration among the joint first authors (indicated using the asterisk *). The remaining co-authors acted as supervisors and helped with writing. I contributed in the development of ideas, discussion, implementation, experiments and writing.

7. Nilesh Chakraborty*, **Denis Lukovnikov***, Gaurav Maheshwari*, Priyansh Trivedi*, Jens Lehmann and Asja Fischer. *Introduction to neural network-based question answering over knowledge graphs.* Published in the journal Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, Volume 11. 2021. Journal article. DOI: 10.1002/widm.1389

   This material was written in close collaboration with the joint first authors (indicated using the asterisk *). The remaining co-authors acted as supervisors and helped with writing. My main contributions include parts on semantic parsing, contributions to other parts and final editing before publication.

8. **Denis Lukovnikov**, Asja Fischer. *Improving Breadth-Wise Backpropagation in GNNs Helps Learning Long-Range Dependencies.* To appear in the Proceedings of the 38th International Conference on Machine Learning (ICML 2021). 2021.

   I conducted most of the study. The other co-author acted as supervisor and helped with writing. This work is not used in this thesis since it concerns Graph Neural Networks (GNNs) in general and is thus not explicitly aimed at semantic parsing or question answering, although there exist connections. In particular, closely related GNNs have been used by other works on semantic parsing and question answering over knowledge graphs.

9. **Denis Lukovnikov**, Asja Fischer. *Insertion-based tree decoding.* In the Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics: Findings (ACL Findings 2021). 2021. This work was also presented in the 5th Workshop on Structured Prediction for NLP at ACL 2021. DOI: 10.18653/v1/2021.findings-acl.283

   I conducted most of the study. The last co-author acted as supervisor and helped with writing.

10. **Denis Lukovnikov**, Sina Däubener, Asja Fischer. *Detecting Compositionally Out-of-Distribution Examples in Semantic Parsing.* To appear in the Conference on Empirical Methods in Natural Language Processing: Findings (EMNLP Findings 2021). DOI: 10.18653/v1/2021.findings-emnlp.54

    I conducted most of the study. Sina Däubener contributed with discussion and writing. The last co-author acted as supervisor and helped with writing.

1.6 Thesis Structure

This thesis consists of nine chapters, which are as follows:

Chapter 1: Introduction.
This chapter presents the motivation, describes the problems and challenges, and presents the main contributions.

Chapter 2: Background and preliminaries.
This chapter provides the background knowledge necessary to understand the rest of the thesis. It elaborates on semantic parsing, question answering over knowledge graphs, and deep learning. Parts of this chapter have been adapted from Publication 7.

Chapter 3: Related work.
Here, related recent work in semantic parsing and question answering over knowledge graphs is presented and discussed. Parts of this chapter have been adapted from Publication 7.

Chapter 4: Word- and character-level representations in question answering over knowledge graphs.
In this chapter, we present our work on the OOV generalization problem. This corresponds to RQ1, and Contribution 1 listed above. Parts of this chapter re-use Publication 1.

Chapter 5: Transfer learning for question answering over knowledge graphs.
In this chapter, our work on transfer learning is presented, which corresponds to RQ2 and Contribution 2 listed above. This chapter is based on Publications 5 and 6.

Chapter 6: Insertion-based decoding for semantic parsing.
In this chapter, we present our work on the insert-based decoding in the context of semantic parsing, where we investigate both sequence-based and tree-based insertion decoding. This corresponds to RQ3 and Contribution 3 listed above. Parts of this chapter re-use Publication 9.

Chapter 7: Linearization order when training semantic parsers.
This chapter presents our work on the effect of linearization order in training semantic parsers, where we also investigate the use of dynamic oracles to enable more order-agnostic training. This corresponds to RQ4 and Contribution 4 listed above. Parts of this chapter re-use Publication 3.

Chapter 8: Detecting compositionally out-of-distribution data.
This chapter presents our work on detecting compositionally out-of-distribution inputs in semantic parsing and question answering over knowledge graphs. This corresponds to RQ5 and Contribution 5 listed above. Parts of this chapter re-use Publication 10.

Chapter 9: Conclusion and future directions.
Finally, we summarize our contributions and findings, revisiting the research questions. Also, limitations and some directions for future research are discussed.

CHAPTER 2

Background and Preliminaries

"The only true wisdom is in knowing you know nothing."

– Socrates

In this chapter, we will first present several concepts necessary for the work presented below. Parts of this chapter have been adapted from Publication 7. Such preliminaries include the basics of neural network based sequence-to-sequence models, as well as sequence encoders. In addition, it is necessary to discuss the semantic parsing task, the task of question answering over knowledge graphs as well as the connection between them and their connection to other semantic parsing tasks, such as text-to-SQL.

2.1 Semantic Parsing and Question Answering over Knowledge Graphs

In this section, we further elaborate on the concepts of semantic parsing (Section 2.1.1), knowledge graphs (Section 2.1.2) and question answering over knowledge graphs (Section 2.1.3), continuing the discussion from the introduction.

A note on terminology used in the following sections. The *natural language question* (which we abbreviate to *NLQ*), is also referred to as *question* or *utterance*. The *meaning representation* is also referred to as *logical forms* or *formal queries*. The *execution results* or *answers* for a formal query are also referred to as *denotations*. In the QA community, KGQA is often also called Knowledge Base Question Answering (KBQA).

2.1.1 Semantic Parsing

In general, **semantic parsing** is the task of translating a natural language (NL) utterance $x \in \mathcal{X} \subset \mathcal{N}$ into a machine-interpretable representation $q \in \mathcal{Q}$ of its meaning in a given formal language (FL). Here, $\mathcal{X}$ is a subset of the natural language $\mathcal{N}$, which specifies the scope (domain) of some semantic parser. While $\mathcal{N}$ contains all possible utterances that are considered to be part of a natural language (such as English), the $\mathcal{X}$ subset of it may be restricted to those utterances that are in the *domain/scope* of the semantic parser. We use domain to refer to the knowledge or competence area in which a particular semantic parser might be specialized. The set $\mathcal{X}$ is then the set of utterances for which the system using the semantic parser should provide a response that is not an out-of-scope response. For example, one can make a semantic parser to interpret commands for a basic autonomous vacuum cleaner. In this case, only a small fraction of possible NL utterances is meaningful, that is, should result in a change of movement or air flow. Other questions, for example asking about the weather, should be considered out-of-scope.

The produced logical form q is considered **correct** if it most accurately captures the meaning of the NL utterance x under the restrictions of the scope of the parser and the used formal language. However, it is difficult to clearly formalize the concept of meaning. One property of the correct logical form is that its execution in the environment $\mathcal{E}$ (which we denote by $q(\mathcal{E})$) results in the desired change of the environment. For example, executing a query in a computer system to retrieve the results from a database yields the set of results. However, the correctness of execution results is not sufficient because multiple logical forms could yield the expected results but not all of them correctly capture the meaning of the question. We call such logical forms that satisfy the first constraint but not the second *spurious* logical forms. For example, translating "*What is two plus two*" to $2*2$ yields the same result but is not the correct query.

Formal Languages

Depending on the usecase, semantic parsing can use different output languages. For example, under the above definition of semantic parsing, code generation (generating code from a NL description) can be considered a form of semantic parsing, as well as translating questions into SQL queries (text-to-SQL).

Some commonly used formal languages are lambda expressions, λ-DCS [26], FunQL [27], SQL and SPARQL. Such languages have a well defined formal grammar and structure, and allow for complex fact retrieval involving logical operands (like disjunction, conjunction, and negation), aggregation functions (like grouping or counting), filtering based on conditions, and other ranking mechanisms.

The formal nature of the output language, together with the ability to execute a logical form in a (simulated) environment, is the essential difference between semantic parsing and machine translation (MT).

Lambda calculus [28] is a formal language for defining computable functions originally devised by Church [29] and used for his study of the *Entscheidungsproblem* [30]. Logical forms in semantic parsing can be expressed in a similar way. The example question *"Where was the author of Dune born?"* can be expressed as follows:

$$\lambda x.\exists e.\mathsf{birthplace}(e, x) \wedge \mathsf{author}(\mathsf{Dune\_(novel)}, e) \quad . \tag{2.1}$$

This lambda calculus form is rather verbose and may result in longer and syntactically complex expressions than necessary for their use in question answering. λ-DCS [26] and FunQL [27] both provide a more concise query representation than SPARQL or lambda calculus by avoiding variables and making quantifiers from lambda calculus implicit. In λ-DCS, our example NLQ can be simply expressed as $\mathsf{R[birthplace].R[author].Dune\_(novel)}$, where $\mathsf{R[birthplace]}$ and $\mathsf{R[author]}$ are the inverse relations of $\mathsf{birthplace}$ and author, respectively[1]. We refer interested readers to [27] and [26] for a more in-depth explanation about FunQL and λ-DCS, respectively.

[1] R inverts the relation by reversing the order of its arguments in the corresponding lambda expression: $(\llbracket \mathsf{b} \rrbracket = \lambda x.\lambda y.\mathsf{b}(x, y)) \Rightarrow (\llbracket \mathsf{R[b]} \rrbracket = \lambda x.\lambda y.\mathsf{b}(y, x))$

Natural language can also be parsed into abstract meaning representation (AMR) [31]. AMR is a graph-based meaning representation language that is aimed to capture the semantics of sentences using rooted, directed, acyclic, edge-labeled, leaf-labeled graphs. The leaf nodes of AMR graphs represent concepts, such as words and semantic frames, and the edges represent the relations between the nodes, which can indicate the argument structure (for frames) and general semantic relations (e.g. "destination" or "direction"). While the AMR language has been carefully crafted to represent the meaning of English utterances in terms of PropBank framesets and words, these meaning representations are not executable, and can not be used as-is, for example to obtain answers from a KG. However, AMR parsing can be useful as an intermediate annotation step for KGQA, as for example illustrated by Kapanipathi et al. [32]. We do not focus on AMR parsing in this thesis, even though it is an interesting topic with its own interesting characteristics, such as the sequence-to-graph prediction task.

2.1.2 Knowledge Graphs

A knowledge graph (KG) is a formal representation of facts pertaining to a particular domain (including the general domain). It consists of *entities* denoting the subjects of interest in the domain, and *relations*[2] denoting the interactions between these entities. For instance, :Tacoma and :Frank_Herbert can be entities corresponding to the real world city of Tacoma, Washington in the USA, and the famous author Frank Herbert respectively. Further, :birthplaceOf can be a relation between the two denoting that Frank Herbert was born in Tacoma. Instead of linking two entities, a relation can also link an entity to an data value of the entity (such as a date, a number, a string etc.) which is referred to as a *literal*. The entity :Tacoma for example might be connected via a relation :area to a numerical literal with the value of $180km^2$ describing the size of city.

Formally, let $\mathcal{E} = \{e_1 \dots e_{n_e}\}$ be the set of entities, $\mathcal{L}$ be the set of all literal values, and $\mathcal{P} = \{p_1 \dots p_{n_p}\}$ be the set of relations connecting two entities, or an entity with a literal. We define a *triple* $t \in \mathcal{E} \times \mathcal{P} \times (\mathcal{E} \cup \mathcal{L})$ as a statement comprising of a subject entity, a relation, and an object entity or literal, describing a fact. Then, a KG $\mathcal{K}$ is a subset of $\mathcal{E} \times \mathcal{P} \times (\mathcal{E} \cup \mathcal{L})$ of all possible triples representing facts that are assumed to hold.

2.1.3 Question Answering over Knowledge Graphs

Using the concepts introduced above, we now formally define the task of KGQA. In simple terms, KGQA can be defined as the task of retrieving answers to NL questions or commands from a KG. While the expected answers could take on complex forms, such as ordered lists (e.g. for the command "*Give me Roger Deakins movies ranked by date.*"), or even a table (e.g. for the question "*When was which Nolan film released?*"), the vast majority of works on KGQA assume the answer to be of a simpler form: a set of entities and/or literals or booleans. We can define the latter more formally as follows.

Let $\mathcal{G}$ be a KG and let q be a natural language question then we define the set of all possible answers $\mathcal{A}$ as the union of (i) the power set $\mathcal{P}(\mathcal{E} \cup \mathcal{L})$ of entities $\mathcal{E}$ and literals $\mathcal{L}$ in $\mathcal{G}$, (ii) the set of the numerical results of all possible aggregation functions $f : \mathcal{P}(\mathcal{E} \cup \mathcal{L}) \mapsto \mathbb{R}$ (such as SUM or COUNT), and (iii) the set $\{\mathsf{True}, \mathsf{False}\}$ of possible boolean outcomes, which is needed for yes/no questions.

[2] The words *predicate* or *property* are often used interchangeably with relations here.

[2] Set of all subsets of a set of elements.

The task of KGQA then is defined as returning the correct answer $a \in \mathcal{A}$, for a given question q. The most common way in which KGQA approaches accomplish this is by creating a formal query representing the semantic structure of the question q, which when executed over $\mathcal{G}$, returns $\mathcal{A}$. Thus, KGQA can be seen as an application area of semantic parsing.

KGQA Subtasks

Regardless of the specific semantic parsing approach taken, the construction of logical forms for KGQA requires taking several decisions regarding their structure and content. Here, we briefly highlight those different tasks the semantic parser will have to perform. In the following, consider the question "*Where was the author of Dune born?*" as an example.

1. **Entity Linking**: Entity linking in the context of KGQA is the task of deciding which KG entity is referred to by (a certain phrase in) the NLQ q. In the case of our example, the entity linker must identify :Dune_(novel) as the entity being referred to by "*Dune*" in the NLQ. The large number (often millions) of entities forms a core challenge in entity linking with large-scale KGs. A particular phrase (e.g. "*Dune*"), if considered without context, can refer to one of possibly many different entities (Dune can refer to the novel, the David Lynch film, the Denis Villeneuve film, the landform, or several other possibilities). It is essential to take the context of the phrase into account to correctly *disambiguate* the phrase to the correct entity. The large number of entities makes it practically unfeasible to create fully annotated training examples to learn lexical mappings for all entities. As a consequence, statistical entity linking systems need to generalise well to unseen entities. Many modern KGQA systems externalise the task of entity linking by employing a standalone entity linking system like DBpedia Spotlight [2] for DBpedia based KGQA systems [33, 34], or S-Mart [35] for Freebase.

2. **Identifying Relations**: It is essential to determine which relation must be used in a certain part of the logical form. Similarly to entity linking, we need to learn to map natural language expressions to the KG - in this case to its relations. However, unlike in entity linking, where entities are expressed by entity-specific noun phrases, relations are typically expressed by noun and verb phrase *patterns* that use less specific words. For instance, in our example NLQ, the relations :author and :birthplace[3] are explicitly referred to by the phrases "*author of*" and "*born*", respectively. However, depending on the KG schema, relations may also need to be inferred, like in "*Which American wrote Dune?*" where "*American*" specifies an additional (SPARQL) constraint ?answer :residentOf :USA without explicitly mentioning the relation :residentOf.

3. **Identifying Logical/Numerical Operators**: Sometimes questions contain additional operators on intermediate variables/sets. For example, "*How many books did Frank Herbert write*?" implies a COUNT operation on the set of books written by Frank Herbert. Other operators can include ARGMAX/MIN, comparative filters (e.g. "*older than 40*"), set inclusion (e.g. "*Did Frank Herbert write Dune?*") etc. Like entities and relations, identifying such operators is primarily a lexicon learning problem. However, compared to entities and relations, there is a fixed small set of operators, which depends on the chosen formal language and not on the KG.

[3] :birthplace is a relation mapping from a person to a place where that person was born.

4. **Determining Logical Form Structure**: In order to arrive at a concrete logical form, a series of decisions must be made regarding the structure of the logical form, i.e. how to arrange the operators, relations and entities such that the resulting object accurately captures the meaning of the question and executes to the intended answer. For example, if FunQL is used as target formal language, a decision must be taken to supply :Dune_(novel) as the argument to the relation function :author, and to pass this subtree as the argument to :birthplace, yielding the final query :birthplace(:author(:Dune_(book))).

Question answering often solve a number of these subtasks in a single process. For instance, translation based systems in principle could generate the whole query in a single sequence decoding process, thus solving all subtasks at once. However, in practice, specialised modules can be employed, for example for entity linking, in order to constrain the search space of the main model.

SPARQL

SPARQL, a recursive acronym for "SPARQL Protocol and RDF Query Language", is one of the most commonly used query languages for KGs and is supported by many publicly available KGs like DBpedia [3], Wikidata [4] and Freebase [1]. For an in-depth understanding of the syntax and semantics of SPARQL, we refer interested readers to the W3C Technical Report[4]. The central part of a SPARQL query is a graph pattern, composed of resources from the KG (i.e. entities and relations) and variables to which multiple KG resources can be mapped. This graph patterns specifies conditions on variables, whereas the SELECT clause determines which variables are returned as the answer. For example, given our question "*Where was the author of Dune born*", the corresponding SPARQL query is `SELECT ?y WHERE {:Dune_(novel) :author ?e.  ?e :birthplace ?y }`.

2.2 Neural networks for sequence processing

In natural language processing (NLP), we very often work with sequences, which usually contain natural language tokens. These sequences arise naturally from the sequential nature of language, where a text, an utterance, a phrase or a sentence are all sequences of words (or other symbols) and punctuation tokens. In this section, we will provide a quick overview of common methods used for deep learning with sequences, which we also use in our work.

2.2.1 What's in a sequence

Before discussing neural networks for sequence processing, we first quickly discuss sequences. In NLP, we often work with sequences of natural language elements. Natural language elements can be words, characters or other sub-word tokens, as used in BPE [36]/WordPiece [37, 38] tokenization.

Informally, a sequence is an ordered collection of elements. More formally, a sequence can be defined as follows:

Definition 2.2.1 (Sequence)**.** A sequence is a function $f : [0, n] \cap \mathbb{Z}^+ \mapsto \mathcal{V}$ that maps an integer from an interval of the positive integers to an element from the codomain $\mathcal{V}$.

[4] `https://www.w3.org/TR/rdf-sparql-query/#introduction`

Since we work with discrete elements, we shall also refer to $\mathcal{V}$ as the *vocabulary* used for the sequence.

While the mathematical notation for sequences uses round parentheses, for example $(1, 2, 3)$ or $(n^2)_{n\in\mathbb{N}}$, throughout this work, we also use the informal notation using square brackets (for example, $[1, 2, 3]$) where it can not be confused with range notation.

Given the definition of a sequence, we can also define a *string* as a sequence of characters. For NLP purposes, the codomain/vocabulary $\mathcal{V}$ of strings is the alphabet of a language, together with additional characters, such as punctuation tokens.

When processing natural language, the text is typically passed[5] as a string. However, a string may not be the best possible way to work with language. Almost always, text is *tokenized* before being passed further into the NLP pipeline. Tokenizers are functions mapping strings to sequences of more meaningful elements, such as words, morphemes or other sub-word tokens.

While a word-level text representation, which represents a given text as a sequence of words, is commonly used in NLP, other representations are also frequently used because they allow suitable methods to generalize better. Character-level text representation (which simply uses the original string) can improve the capacity of NN-based models to process out-of-vocabulary (OOV) words, such as person names. This is interesting because, as we shall see, NN-based methods for sequences of discrete elements typically learn representation for every element of $\mathcal{V}$. While usually, unobserved tokens are replaced with a special `UNK` token, or otherwise replaced with more specific placeholders, it may be necessary or beneficial to use all information contained in the sentence. Character-based representation has the advantage that the vocabulary size is relatively small, and the chance of unobserved elements is very low. Morpheme-based representation is a middle-ground between word- and character-level representation and has the advantage that the vocabulary size is smaller than a word-based representation while every element in the sequence is meaningful. In fact, morphemes are the smallest meaningful units in natural language. However, morphemes are not commonly used in NLP, because of practical considerations.

Practically, especially with the rise of pre-trained transformers, tokenizers using statistically constructed subword vocabularies are frequently used. For example, WordPiece [37] is built using an iterative process that starts with a vocabulary consisting of characters, and adds new subword elements to its vocabulary by merging two elements from the vocabulary, such that the new element maximizes the likelihood of the data. With this approach, out-of-vocabulary words are avoided while keeping more frequently used words as a single token. To illustrate the differences between word-level and a (possible) WordPiece-based tokenization, consider the following example (in Japanese): “Omae wa mou shindeiru”. While the word-level tokenizer will produce the sequence (“Omae”, “wa”, “mou”, “shindeiru”), the WordPiece tokenizer might produce the sequence (“Omae”, “wa”, “mou”, “shin”, “##dei”, “##ru”). Note that some WordPiece tokens are prefixed with “##”, which indicates that whatever follows should be concatenated to the previous token (or in other words, that the preceding word space should be ignored when reconstructing the original sequence).

2.2.2 Neural Network Components

Neural network architectures are constructed using different neural network building blocks. In this section, we discuss various such building blocks that are relevant to NLP and our tasks.

[5] After speech transcription, if necessary.

Token representations

Natural language consists of discrete elements, such as words, which, as previously discussed, can be decomposed into sub-word units. In order to use such discrete elements in a model, we need to find vector representations for the tokens.

A simple vector representation for tokens that has long been in use is the **one-hot** vector representation. This representation can be formalized as a function $f_{\text{one-hot}} : \mathcal{V} \mapsto \mathbb{N}_0^{|\mathcal{V}|}$ that maps any token w from the vocabulary $\mathcal{V}$ onto a vector v of the size of the vocabulary, such that only the element in the vector corresponding to the token w is equal to one and the others are equal to zero:

$$v_i = \begin{cases} 1, & \text{if } i = \text{id}(w) \\ 0, & \text{otherwise} \end{cases}, \tag{2.2}$$

where the function $\text{id}(w) : \mathcal{V} \mapsto \mathbb{N}_0$ maps the word w to its unique integer id.

Since the one-hot representation do not scale well in practice, and the distance between two words is equal for any two words, tokens are usually represented as **dense** vectors of relatively low dimensionality. The token embedding function is the a function $f_{\text{emb}} : \mathcal{V} \mapsto \mathbb{R}^d$, where d is some chosen dimension for the token embedding space. Note that such a dense embedding allows the vector representations of different words to be arbitrarily close to each other, which is useful for handling synonyms. In addition, the dimensionality of the vectors can be kept low and does not directly depend on the vocabulary size. The function f_{emb} can be implemented as:

$$f_{\text{emb}}(w) = A \cdot v_w, \tag{2.3}$$

where $v_w \in \mathbb{N}_0^{|\mathcal{V}|}$ is a one-hot vector for word w and $A \in \mathbb{R}^{d \times |\mathcal{V}|}$ is a trainable matrix containing the vectors for every word in $\mathcal{V}$.

In the earlier days of deep learning for NLP, several fundamental works have investigated different ways of pre-training word vectors using large unsupervised corpora. Among the most commonly used ones are Word2Vec [39] and GloVe [40]. These methods provide initializations for word vector parameters that capture synonymity and other relations between words, induced from statistics on a large text corpus (see 2.4.1). Essentially, similar words obtain similar embeddings, which improves generalization to unobserved words. While recent focus in NLP has shifted towards pretrained transformers (see Section 2.4.2), pretrained word embeddings are still used in applications due to their simplicity and efficiency.

Recurrent Neural Networks

Natural languages, as well as formal languages, are sequential in nature, which means that we require models that can process and generalize for different sequence lengths. Recurrent neural networks (RNN) are a type of neural networks suitable for this case. RNNs typically process the sequence one token at a time, and use a "state" variable/vector that describes the sequence elements "consumed" so far. Thus, the typical RNN is a parameterized function $f : \mathbb{R}^{d_i} \times \mathbb{R}^{d_h} \mapsto \mathbb{R}^{d_h}$ that computes a new state $h_t \in \mathbb{R}^{d_h}$ from the previous state $h_{t-1} \in \mathbb{R}^{d_h}$ and current input $x_t \in \mathbb{R}^{d_i}$:

$$h_t = f(h_{t-1}, x_t; \theta) \tag{2.4}$$

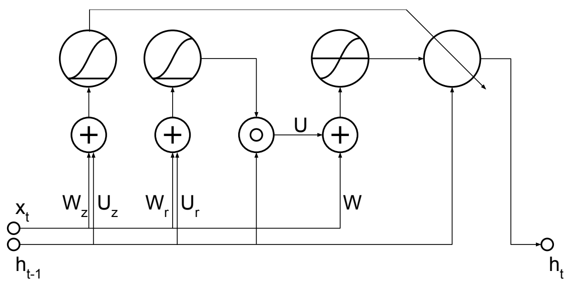


Figure 2.1: Gated Recurrent Unit (GRU)

RNNs usually share their parameters across time steps to improve generalization.

The most basic RNN can be implemented as a single layer with an affine transformation and non-linearity:

$$h_t = \tanh(Wh_{t-1} + Ux_t + b) \ , \tag{2.5}$$

where W and U are trainable linear transformations and b is a trainable bias term.

However, it has been shown that such basic RNNs suffer from gradient problems [41–44] due to multiplicative updates and non-linearities in the backpropagation path to an early state. In practice, gated RNNs are used, such as the Long Short-Term Memory (LSTM [42]) unit or the Gated Recurrent Unit (GRU [45]). Both implement *additive* gated state updates, which prevents the gradients from vanishing or exploding due to linear transformations and repeated non-linearities. The GRU is defined according to the following equations and is illustrated in Figure 2.1:

$$\mathbf{z}_t = \sigma(W_z\mathbf{x}_t + U_z\mathbf{h}_{t-1} + \mathbf{b}_z) \tag{2.6}$$
$$\mathbf{r}_t = \sigma(W_r\mathbf{x}_t + U_r\mathbf{h}_{t-1} + \mathbf{b}_r) \tag{2.7}$$
$$\hat{\mathbf{h}}_t = \tanh(W\mathbf{x}_t + U(\mathbf{h}_{t-1} \odot \mathbf{r}_t) + \mathbf{b}_h) \tag{2.8}$$
$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \hat{\mathbf{h}}_t \ , \tag{2.9}$$

where the $W_\times$ and $U_\times$ are trainable matrices and $b_\times$ are trainable bias terms.

The current hidden state $\mathbf{h}_t$ at time step t of the RNN (which is also its output at time t) is computed by interpolating between the state $\mathbf{h}_{t-1}$ at previous time step and the candidate state $\hat{\mathbf{h}}_t$ (Equation 2.9), with $\mathbf{z}_t$ the update vector and $\cdot$ the element-wise vector product. The *update gate* $\mathbf{z}_t$ for the interpolation is computed using the current input $\mathbf{x}_t$ and the previous state $\mathbf{h}_{t-1}$ (Equation 2.6), where W_z and U_z are parameter matrices to be learned during training and σ is the sigmoid activation function $\sigma(x) = 1/(1 + e^{-x})$ applied element-wise to the vector entries. The current candidate state $\hat{\mathbf{h}}_t$ is computed based on the current input $\mathbf{x}_t$ and previous state $\mathbf{h}_{t-1}$ (Equation 2.8), where $\mathbf{W}$ and $\mathbf{U}$ are parameter matrices, tanh is the hyperbolic tangent activation function and $\mathbf{r}_t$ is the value of the *reset gate*, computed as in Equation 2.7, with parameter matrices $\mathbf{W}_r$ and $\mathbf{U}_r$. A schematic representation of the GRU can be found in Figure 2.1.

The advantage of using gated units such as GRU or long short-term memory (LSTM) [42] is their ability to better process longer sequences, which arises from their additive manipulation of the state vector and explicit filtering using gates. In the case of the GRU, the reset gate $\mathbf{r}_t$ determines which parts of the previous state $\mathbf{h}_{t-1}$ are "ignored" in the computation of the candidate state and the update gate $\mathbf{z}_t$ determines how much of the previous state is "leaked" into the current state $\mathbf{h}_t$. The update gate could decide to forget the previous state altogether or to simply copy the previous state and ignore the current input. Both gates are parameterized (and thus trainable) and their values depend on both the input $\mathbf{x}_t$ and the previous state.

The LSTM is similar but uses two "state" variables $\mathbf{c}_t$ and $\mathbf{y}_t$, where $\mathbf{c}_t$ is the "cell state" and $\mathbf{y}_t$ is the previous output state. The LSTM is defined according to the following equations:

$$\mathbf{i}_t = \sigma(W_i\mathbf{x}_t + U_i\mathbf{y}_{t-1} + \mathbf{b}_i) \tag{2.10}$$
$$\mathbf{f}_t = \sigma(W_f\mathbf{x}_t + U_f\mathbf{y}_{t-1} + \mathbf{b}_f) \tag{2.11}$$
$$\mathbf{o}_t = \sigma(W_o\mathbf{x}_t + U_o\mathbf{y}_{t-1} + \mathbf{b}_o) \tag{2.12}$$
$$\hat{\mathbf{c}}_t = \tanh(W_c\mathbf{x}_t + U_c\mathbf{y}_{t-1} + \mathbf{b}_c) \tag{2.13}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t \tag{2.14}$$
$$\mathbf{y}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \ , \tag{2.15}$$

where the $W_\times$ and $U_\times$ are trainable matrices and $\mathbf{b}_\times$ are trainable bias terms.

Note that in both, there is a mechanism for the network to "forget" parts of the previous state. Also, in both cases, state updates are additive, which allows the gradient to be backpropagated while being only element-wise multiplied with the forget gate vectors.

Convolutional Neural Networks

While typically, RNNs and Transformers are used, convolutional neural networks (CNNs) can also be applied on text. A 1D CNN is parameterized using a tensor $W \in \mathbb{R}^{l \times d \times h}$, where l is the window size or kernel size, d is the input dimension and h the output dimension. The output is computed by sliding the weights W over the length dimension of the input (the input is shaped as $\mathbb{R}^{s \times d}$, multiplying every $l \times d$ window from the input with every h-slice of size $l \times d$ of W using element-wise multiplication, and summing up. The result is in $\mathbb{R}^{s \times h}$. Additionally, CNNs use non-linearities, where a ReLU is a standard choice, as well as local (and global) pooling.

Attention

Originally, attention was proposed as a way to produce a latent alignment for neural machine translation [46, 47]. Since then, it has been a crucial component in many NLP applications and is the primary mechanism of context integration in transformers.

Most generally, given some input sequence X of length L of vectors $\mathbf{x}_i \in \mathbb{R}^d$ and a *query* vector $\mathbf{q} \in \mathbb{R}^h$, attention is a function that compares $\mathbf{q}$ to every $\mathbf{x}_i$ and builds a weighted sum of $\mathbf{x}_i$ that can be trained to contain most relevant information from X pertaining to the given query $\mathbf{q}$. We can describe the attention mechanism using the following equations:

$$a_i = \text{comp}(\mathbf{x}_i, \mathbf{q}) \tag{2.16}$$

$$\alpha_i = \frac{e^{a_i}}{\sum_{i=0}^{L} e^{a_i}} \tag{2.17}$$

$$\mathbf{v} = \sum_{i=0}^{L} \alpha_i \cdot \mathbf{x}_i \ , \tag{2.18}$$

where $\text{comp}(\cdot) : \mathbb{R}^d \times \mathbb{R}^h \mapsto \mathbb{R}$ is a function comparing two vectors.

One variant of attention uses a feedforward network on the concatenation of $\mathbf{x}_i$ and $\mathbf{q}$ [46]:

$$\text{comp}(\mathbf{x}, \mathbf{q}) = \mathbf{w}^{\mathsf{T}} \tanh(W\mathbf{x} + U\mathbf{q}) \tag{2.19}$$

Another commonly used variant, usually called *multiplicative* attention, simply uses the dot product as $\text{comp}(\cdot)$:

$$\text{comp}(\mathbf{x}, \mathbf{q}) = \mathbf{q}^{\mathsf{T}}\mathbf{x} \ . \tag{2.20}$$

The query q and key vectors x can also be projected before the dot product:

$$\text{comp}(\mathbf{x}, \mathbf{q}) = (W\mathbf{q})^{\mathsf{T}}(U\mathbf{x}) \ , \tag{2.21}$$

where W and U are trainable matrices.

2.2.3 Neural Architectures for NLP

Different architectures have been proposed for different tasks in NLP. Here, we will briefly overview the most standard ones that are most relevant to our work: RNN-based encoders and sequence-to-sequence models and transformers.

Types of tasks

First, we will briefly highlight five main types of tasks we might want to do with sequences: (1) classification, (2) regression, (3) tagging, (4) generation, and (5) translation. Out of these tasks, we are mostly interested in translation and classification.

Sequence classification and regression: As an example of a classification task with sequences, let us consider the task of relation classification. Given a sentence q, and a KG $\mathcal{K}$, the relation classification task consists of predicting which of the n_r relations $r_1, \ldots, r_{n_r} \in P_{\mathcal{K}}$ is referred to in q. In the first step, an *encoder network* can be used to map the variable-length input sequence of q onto a fixed-length vector $\mathbf{q} \in \mathbb{R}^d$, which is called *the latent representation*, or the *encoded representation* of q. The encoder network can be comprised of different neural architectures, including recurrent neural networks (RNN), convolutional neural networks (CNN), or transformers [18]. The encoded question is subsequently fed through an affine transformation to calculate a *score vector* $s(q) = (s_1(q) \ldots s_{n_r}(q))$, that is

$$s(q) = W_o\mathbf{q} + \mathbf{b_o} \ . \tag{2.22}$$

Here, $W_o \in \mathbb{R}^{n_r \times d}$ and $\mathbf{b_0} \in \mathbb{R}^{n_r}$, in conjunction with the parameters of the encoder network, constitute the trainable parameters of the classification model. In the output layer the classification model typically turns the score vector into a conditional probability distribution $p(r_k|q)$ over the n_r relations based on a softmax function, given by

$$p(r_k|q) = \frac{e^{s_k(q)}}{\sum_{j=0}^{n_r} e^{s_j(q)}} \quad , \tag{2.23}$$

for $k = 1, \dots n_r$. Classification is then performed by picking the relation with the highest probability given q.

A regression model can be obtained simply by replacing the softmax output layer of the classifier with an MLP whose last layer outputs a single scalar value.

Sequence tagging: A prime example of a sequence tagging task is part-of-speech (POS) tagging, where every word in a given sequence has to be annotated with a label from the predefined set of POS tags (e.g. noun, verb, ...). Similarly to sequence classification, we can solve this task by using an encoder, but this time, we generate a representation vector for every token rather than one vector the entire sequence. We write the representation vector for the token at position t in the sequence as h_t. It is desireable to use an encoder because we want to take into account the context of the words when building their representations in order to improve prediction accuracy. Then, similarly to classification, we can use a classifier (similar to Eq. 2.22), but this time at every position rather than just once for the entire sequence. The result is a sequence of tags y_t of the same length as the input sequence, where y_t corresponds to the input token x_t.

Sequence generation and translation: In machine translation and semantic parsing, we need to map the input sequence $x = (x_1, x_2 \dots x_T)$ $(x_t \in \mathcal{V}_{\text{in}})$ to an output sequence (or tree/graph) $y = (y_1, y_2 \dots y_L)$ $(y_l \in \mathcal{V}_{\text{out}})$. Language modeling and translation are both sequence prediction tasks, with the difference that language modeling usually is not conditioned on some input, while translation takes another sequence as the input. Sequence generation is typically done in a left-to-right fashion, where the next prediction takes into account all previously predicted tokens.[6] This factorizes the joint probability of the output sequence $p(y|x;\theta)$ into the product $\prod_{l=0}^{L} p(y_l|y_{<l}, x; \theta)$.

Translation can be accomplished by first encoding the input sequence using some encoder, similar to the sequence classification and tagging settings. We then take the encoding of the entire sequence, like in sequence classification, and feed it to the decoder to condition the decoder on the input. If the decoder uses attention (see below), we also take the encoder's representations for every input position and provide them to the decoder. Different from the previous tasks, we need a decoder model that can be conditioned on some input vectors, and generates a sequence (or another structure). Usually, a decoder model is used that at every step produces the next token in the output sequence. At every step, it takes information about the input, as well as information about what has been decoded so far, and generates the distribution for the next token using a softmax output layer (Eqs. 2.22, 2.23). In greedy decoding, simply the most probable token from this distribution is taken and appended to the output. Subsequently, the process is repeated until termination. The decoding loop can be terminated by using

[6] As we shall see in this work however, this is not the only viable method for decoding sequences (see Section 7).

special end-of-sequence (EOS) tokens. When the decoder produces an EOS token, the recurrence is stopped and the sequence produced up until the EOS token is returned.

RNN-based architectures

RNN-based architectures were most commonly used before transformers emerged. RNNs can be used both in encoder-only and sequence-to-sequence settings.

Encoding sequences using RNN's: First, the input sequence of length L of symbols from $\mathcal{V}$ is projected to a sequence X of vectors $\mathbf{x}_t \in \mathbb{R}^{d_{\text{emb}}}$ using an embedding layer.

Subsequently, the sequence X is encoded using an RNN layer:

$$h_t = \text{RNN}(x_t, h_{t-1}) \ . \tag{2.24}$$

This yields a sequence H of vectors $h_t \in \mathbb{R}^{d_{\text{RNN}}}$. However, since this way of encode X only takes into account the tokens *preceding* any token x_t, usually, bidirectional RNN encoders are used that encode the sequence in parallel in both forward and reverse directions:

$$\mathbf{h}_t^{\text{FWD}} = \text{RNN}_{\text{FWD}}(\mathbf{x}_t, \mathbf{h}_{t-1}) \tag{2.25}$$

$$\mathbf{h}_t^{\text{REV}} = \text{RNN}_{\text{REV}}(\mathbf{x}_t, \mathbf{h}_{t+1}) \tag{2.26}$$

$$\mathbf{h}_t = [\mathbf{h}_t^{\text{FWD}}, \mathbf{h}_t^{\text{REV}}] \ , \tag{2.27}$$

where RNN_{FWD} and RNN_{REV} are two different RNNs, the latter consuming the sequence of vectors $\mathbf{x}_t$ in a right-to-left order. The output $\mathbf{h}_t$ is simply the concatenation of the outputs of the two RNNs for a certain position in the input.

Such encoders are commonly used for sequence tagging and classification in various NLP tasks.

Translating sequences using RNNs: Recall that the sequence-to-sequence model only has to learn to predict the correct token at every decoding step to generate the output sequence, i.e. it learns the distribution $p(y_l|y_{<l}, x)$. A simple way to accomplish this using RNNs is to first encode the input sequence (see above), initialize the decoder RNN with the input representation, and run the decoder RNN.

$$H = (\mathbf{h}_t)_{t\in[0,T]\cap\mathbb{N}_0} = \text{Encoder}(x) \tag{2.28}$$

$$\mathbf{g}_0 = \mathbf{h}_T \tag{2.29}$$

$$\mathbf{u}_l = \text{Embed}(y_{l-1}) \tag{2.30}$$

$$\mathbf{g}_l = \text{RNN}(\mathbf{u}_l, \mathbf{g}_{l-1}) \tag{2.31}$$

$$p(y_l|y_{<l}, x) = \text{Classifier}(\mathbf{g}_l) \tag{2.32}$$

$$\hat{y}_l = \text{argmax}(p(y_l|y_{<l}, x)) \ , \tag{2.33}$$

where $\text{Embed}(\cdot)$ is token embedding layer, $\text{RNN}(\cdot)$ is a (multi-layer) RNN and $\text{Classifier}(\cdot)$ is an MLP followed by a softmax that produces the probabilities over the output vocabulary $\mathcal{V}_{\text{out}}$.

A big problem with the naive sequence-to-sequence model described in Eqs. 2.28-2.33 is that all the information about the input sequence must be summarized in a single vector ($\mathbf{g}_0 = \mathbf{h}_T$), which introduces a bottleneck. Using attention solves this problem by allowing every $\mathbf{g}_l$ to be directly informed by the most relevant $\mathbf{h}_t$. An attention-based decoder can be described as follows:

$$\mathbf{s}_l = \text{Attend}(H, \mathbf{g}_l) \tag{2.34}$$

$$\hat{\mathbf{g}}_l = [\mathbf{g}_l, \mathbf{s}_l] \tag{2.35}$$

$$p(y_l | y_{<l}, x) = \text{Classifier}(\hat{\mathbf{g}}_l) \ , \tag{2.36}$$

where Attend$(\cdot)$ is an attention mechanism that takes a sequence of input vectors (H here) and a query vector ($\mathbf{g}_l$ here) and produces a summary of the input vectors, as described above.

Transformers

Transformers [18] are an alternative to RNNs for sequence processing. In contrast to RNNs, transformers allow for more parallel training, and offer better interpretability. Especially with the emergence of a wide variety of pre-trained language models, transformers are now ubiquitous in NLP.

Transformer Encoder Layer: A single transformer layer for an encoder consists of two parts: (1) a self-attention layer and (2) a two-layer MLP. The self-attention layer collects information from neighbouring tokens using a multi-head attention mechanism that attends from every position to every position. With $h_t^{(l-1)}$ being the outputs of the previous layer for position t, the multi-head self-attention mechanism computes M different attention distributions (one for every head) over the t positions. For every head m, self-attention is computed as follows:

$$\mathbf{q}_t = W_Q \mathbf{h}_t \qquad \mathbf{k}_t = W_K \mathbf{h}_t \qquad \mathbf{v}_t = W_V \mathbf{h}_t \tag{2.37}$$

$$a_{t,j} = \frac{\mathbf{q}_t^{\mathsf{T}} \mathbf{k}_j}{\sqrt{d_k}} \qquad \alpha_{t,j} = \frac{e^{a_{t,j}}}{\sum_{i=0}^{T} e^{a_{t,i}}} \qquad \mathbf{s}_t = \sum_{j=0}^{T} \alpha_{t,j} \mathbf{v}_j \ , \tag{2.38}$$

where W_Q, W_K and W_V are trainable matrices. Note that attention is normalized by the square root of the dimension d_k of the $\mathbf{k}_t$ vectors.

The summaries $\mathbf{s}_t^{(m)}$ for head m, as computed above, are concatenated and passed through a linear transformation to get the final self-attention vector $\mathbf{u}_t$:

$$\mathbf{u}_t = W_O \bigoplus_{m=0}^{M} \mathbf{s}_t^{(k)} \ , \tag{2.39}$$

where W_O is a trainable matrix and $\oplus$ denotes concatenation.

The whole transformer layer then becomes:

$$\begin{aligned}
(\mathbf{u}_t)_{t\in[0,T]\cap\mathbb{N}_0} &= \text{Attention}((\mathbf{h}_t^{(l-1)})_{t\in[0,T]\cap\mathbb{N}_0}) && (2.40)\\
\hat{\mathbf{h}}_t^{(l)} &= \mathbf{h}_t^{(l-1)} + \mathbf{u}_t && (2.41)\\
\mathbf{v}_t &= W_A \text{ReLU}(W_B \hat{\mathbf{h}}_t^{(l)}) && (2.42)\\
\mathbf{h}_t^{(l)} &= \hat{\mathbf{h}}_t^{(l)} + \mathbf{v}_t \quad , && (2.43)
\end{aligned}$$

where W_A and W_B are trainable matrices. Note that we did not discuss normalization layers and dropout. Different placements of these layers are possible, for example, placing the layer normalization in the beginning of the residual branch and placing dropout at the end of the residual branch.

Transformer Decoder Layer: The decoder should take into account the previously decoded tokens, as well as the encoded input. To do so, two changes are made to the presented transformer encoder layers: (1) a causal self-attention mask is used in self-attention and (2) a cross-attention layer is added.

The **causal self-attention mask** is necessary to prevent the decoder from attending to future tokens, which are available during training but not during test. As such, this is more of a practical change.

The **cross-attention layer** enables the decoder to attend to the encoded input. Denoting the encoded input sequence vectors as $\mathbf{x}_i$, cross-attention is implemented in the same way as self-attention, except the vectors used for computing the key and value vectors are the $\mathbf{x}_i$ vectors. The decoder layer then becomes:

$$\begin{aligned}
(\mathbf{u}_t)_{t\in[0,T]\cap\mathbb{N}_0} &= \text{Attention}((\mathbf{h}_t^{(l-1)})_{t\in[0,T]\cap\mathbb{N}_0}) && (2.44)\\
\hat{\mathbf{h}}_t^{(l)} &= \mathbf{h}_t^{(l-1)} + \mathbf{u}_t && (2.45)\\
(\mathbf{z}_t)_{t\in[0,T]\cap\mathbb{N}_0} &= \text{Attention}((\mathbf{x}_i)_{i\in[0,T_x]\cap\mathbb{N}_0}), (\hat{\mathbf{h}}_t^{(l-1)})_{t\in[0,T]\cap\mathbb{N}_0}) && (2.46)\\
\tilde{\mathbf{h}}_t^{(l)} &= \hat{\mathbf{h}}_t^{(l-1)} + \mathbf{z}_t && (2.47)\\
\mathbf{v}_t &= W_A \text{ReLU}(W_B \tilde{\mathbf{h}}_t^{(l)}) && (2.48)\\
\mathbf{h}_t^{(l)} &= \tilde{\mathbf{h}}_t^{(l)} + \mathbf{v}_t \quad , && (2.49)
\end{aligned}$$

Position information: The self-attention layers are oblivious to the ordering of the key vectors. For this reason, it is essential to explicitly add position information to the model. One option is to include non-trainable sinusoid vectors to the token embeddings before feeding them into transformer layers [18]. Another option is to learn position embeddings and add these to the token embeddings before feeding them into the transformer layers. This is done in BERT [7], for example. These two options use absolute position information. In contrast to this, Shaw et al. [48] propose a relative position encoding method that is invariant to the absolute position in the sequence.

2.3 Training

Neural networks are usually trained using maximum likelihood. Here, we briefly elaborate on maximum likelihood for sequences and trees.

2.3.1 Maximizing Likelihood

Maximum Likelihood Estimation (MLE) is a training method that maximizes the likelihood function $\mathcal{L}(\theta|\mathcal{D})$ where θ are the parameters and $\mathcal{D}$ the data. For discrete data, the likelihood of the parameters given data is equal to the probability of the data given the parameters: $\mathcal{L}(\theta|\mathcal{D}) = p(\mathcal{D}|\theta)$, with the difference that for the likelihood, the parameters are varied while for the probability the data are varied. Essentially, MLE methods find the parameters for a given model such that the probability of the observed data under this model are maximized:

$$\theta^* = \text{argmax}_{\theta \in \Theta} \mathcal{L}(\theta|\mathcal{D}) = \text{argmax}_{\theta \in \Theta} p(\mathcal{D}|\theta) \ . \tag{2.50}$$

In the case of discriminative models, which map some input x (e.g. a sentence) to an output y (a e.g. a sentiment label), MLE can be written as follows:

$$\theta^* = \text{argmax}_{\theta \in \Theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \mathcal{L}(\theta|x, y) = \text{argmax}_{\theta \in \Theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} p(y|x;\theta) \ . \tag{2.51}$$

Consider a simple multi-class classification network. In that case, $p(y|x;\theta)$ is the probability distribution over all possible classes produced by the model for a given input x and parameters θ. Maximizing the likelihood function is then maximizing the value of $p(y|x;\theta)$ for the correct class. In practice, the log-likelihood is maximized due to better numerical stability. Maximizing the log-likelihood corresponds to minimizing the cross-entropy between $p(y|x;\theta)$ and the “true” distribution where 100% probability is concentrated in the correct class. Minimizing the cross-entropy is also equivalent to minimizing the Kullback-Leibler (KL) divergence between the two distributions.

Training classifiers

Given a data set of N pairs of NLQs and single fact based formal queries $\mathcal{D} = \left\{\left(q^{(i)}, f^{(i)}\right)\right\}_{i=1}^{N}$, (here, $f^{(i)}$ is an entity-relation tuple: $f^{(i)} = (e^{(i)}, r^{(i)})$), and a relation classification model can be trained by maximizing the log-likelihood of the model parameters θ, which is given by

$$\sum_{i=1}^{N} \log p_\theta(r^{(i)}|q^{(i)}) \ , \tag{2.52}$$

where $r^{(i)}$ is the predicate used in the formal query $f^{(i)}$.

Training taggers

A common type of task in NLP requires labeling every token in the input sequence. This type of model has many applications in NLP, such as POS tagging and NER, but can also be used in semantic parsing.

Training a tagger using MLE is equivalent to maximizing $p(y|x)$ of (x, y) pairs from the given data $\mathcal{D}$, where x is a sequence of words of length L and y is a sequence of tags of the same length. The most basic method for training the tagger assumes independence between the different tags and factorizes $p(y|x)$ as follows:

$$p(y|x) = \prod_{i=1}^{L} p(y_i|x) \quad . \tag{2.53}$$

Relations between tags can be taken into account by using a conditional random field (CRF) on top of the encoder.

Training sequence generators

When the output structure is a sequence, the joint probability of the output sequence under a given model can be decomposed as the product of probabilities of each token in the sequence given all preceding tokens:

$$p(y|x) = \prod_{i=0}^{L} p(y_i|y_{<i}, x) = p(y_1|x)p(y_2|y_1, x)p(y_3|y_2, y_1, x) \dots p(y_L|y_{<L}, x) \quad , \tag{2.54}$$

where L is the number of elements in y.

Similarly to training a tagger, we maximize $p(y|x)$ for (x, y) pairs from the given data $\mathcal{D}$. Different from training a tagger, the input sequence x and the predicted output y may have different lengths. The simplest way to train a sequence generator is using *teacher forcing*, where we take the ground truth y to feed as input to the decoder (in the $y_{<i}$ variable) when training it. Note that we can not easily use the model's own predictions during training since it might not always be possible to determine what should happen if the generated token is different than the ground truth token. However, in some cases, it may be possible to define dynamic oracles that can generate the best action sequences for all possible states the decoded structure migh be in [7] A possible disadvantage of teacher forcing is that the model only gets exposed to sequences given in the data during training, while during generation, entirely new combinations may be generated, on which the network might be more likely to fail.

Note that while this left-to-right autoregressive model is standard in NLP for machine translation, semantic parsing and other sequence generating tasks, other factorizations and independence assumptions in the modeling of the joint probability $p(y|x)$ are possible [23].

2.3.2 Stochastic Gradient Descent

Deep neural networks are trained by minimizing a loss w.r.t. the parameters using Stochastic Gradient Descent (SGD). SGD-based training iterates over the training data, processing a subset of the examples at a time, and updating the parameter values at every such step. The subset of the training data used in an update step is called a **batch**, and a whole pass over the training data is called an **epoch**.

SGD minimizes the loss by computing the gradient of the loss w.r.t. the parameters and taking a step against the gradient (in the direction that would lower the loss value) in every update step. Computation of the gradient is achieved using **backpropagation**, which can be seen as a special case of reverse mode auto-differentiation, applied on neural networks. Where symbolic differentiation

[7] Imagine for example, a decoder that in addition to generating tokens also can produce actions to erase its previously generated token. In this case, an oracle can be constructed that would train the decoder to use eraser actions when the generated tokens deviate from the ground-truth

would compute a (symbolic) *expression* for the gradient, backpropagation only computes the *value* of the gradient for the given input values.

With the gradient computed, different methods for updating the parameter values can be used. The most basic is the vanilla SGD update:

$$\theta := \theta - \alpha \nabla_{\theta} L(x, y, \theta) \ , \tag{2.55}$$

where θ are the parameters, x and y the inputs and outputs and L the function that runs the neural network on the inputs and returns the loss w.r.t. the desired outputs. More advanced update methods have been proposed, with Adam [49] being one of the most commonly used ones.

2.4 Pretraining and transfer learning

When training neural networks, the amount of training data can heavily influence the quality of the trained model. Models trained on small amounts of data require more extensive regularization to minimize over-fitting[8]. Acquiring more fully annotated training data in many cases should improve the results but may not always be cheap. In semantic parsing and question answering, some commonly used datasets contain relatively little examples. For example, the Geo880 [50] (a.k.a. GeoQuery) dataset contains only 880 sentence-query pairs, Free917 [51], an early QA dataset, contains 917 examples. Later datasets, such as LCQuAD [21] contain more examples and a larger variety of used entities. However, collecting useful manually annotated data for semantic parsing and KGQA is not an easy task. It may require significant effort in designing an intuitive data collection interface and still requires more mental effort from annotators than for example classifying images.

On the other hand, there is a large amount of data available in the form of unannotated text in a particular language. While such data can not be used directly to train a model for a task, it has been found that pre-training the representations of words using these unsupervised data can significantly improve performance of downstream NLP tasks. One way to think of pre-training a model or a part of it is as finding a better prior over the parameter values of the model. In this section, we first focus on pre-trained word embeddings (Section 2.4.1) and then on pre-trained encoders (Section 2.4.2).

2.4.1 Pre-trained word embeddings

As discussed in Section 2.2.2, before using words in a neural network, they have to be mapped to a vector. Typically, a dense vector representation of relatively low dimension (e.g. 100) is used. Normally, the initial values of the parameters for every word are chosen randomly, and subsequently trained with the rest of the model to arrive at better values. However, with a relatively small amount of training data, vanilla initialization and task training might not be able to build good representations for all words. For example, some words may be observed only in a few examples, which makes them "rare". Many words might not be observed during training at all, leading to out-of-vocabulary (OOV) problems. These problems are usually addressed by replacing all words that have a low frequency with a special RARE token, which is also used for OOV tokens during inference. A problem with this approach might be that the identity of rare words is actually important for the task at hand and would

[8] Over-fitting is the phenomenon where a neural network with a relatively large number of parameters to some extent relies on memorizing the training data, as opposed to learning the meaningful patterns present in the data.

ideally not be anonymized (as the rare-token technique does). Another problem is that even for words that have been observed, the learned semantic structure (synonyms, correlations) between words is limited and biased.

The initial values for the word embeddings can also be determined using the vast amounts of unsupervised text available. If we can somehow capture useful information provided in unsupervised data, this can serve as a better starting point for subsequent training. While various methods have been proposed to pre-train word-embeddings, the two most commonly known and used ones are Word2Vec [39] and GloVe [40]. FastText [52] follows Word2Vec and improves its OOV performance. Here, we briefly discuss Word2Vec.

In summary, Word2Vec trains a simple language model using the provided data. Two variants have been proposed by the authors: (1) Continuous Bag-of-Words (CBOW) and (2) Skip-gram. The **CBOW** method takes a context consisting of k words to the left and to the right of some position j in a corpus and predicts the word at that position. Formally, the model simply sums the surrounding word vectors:

$$\mathbf{h}_j = \sum_{i=j-k}^{j+k} \mathbf{w}_j \ , \tag{2.56}$$

where $\mathbf{w}_j$ is the vector for the word v at position j. This can also be thought of as simply matrix multiplication of a "few-hot" vector $x_j^{(\text{one-hot})}$[9] with the weight matrix $W \in \mathbb{R}^{|\mathcal{V}|\times D}$, where D is a chosen dimensionality for the word vectors (e.g. 100). Then, the probability distribution over all possible words is computed using a softmax output layer with its own parameters:

$$p(v_i|v_{j-k}, \ldots v_{j+k}) = \frac{\exp \mathbf{u}_{v_i}^{\mathsf{T}} \mathbf{h}_j}{\sum_{l\in\mathcal{V}} \exp \mathbf{u}_l^{\mathsf{T}} \mathbf{h}_j} \ , \tag{2.57}$$

where v_i is a random variable specifying which word from the vocabulary $\mathcal{V}$ occurs at position i and $\mathbf{u}_l$ is a vector associated with word l. Training is done by maximizing the likelihood of the parameters given the text by iterating over all positions and maximizing the probability given by the model for the word given in the data.

In contrast, the **Skip-gram** method predicts the *context* of a word given the word itself. The objective is then to maximize the likelihood of the model for the given text and is computed by maximizing the following simplified joint probability of the context:

$$p(v_{j-k}, \ldots v_{j+k}|v_i) = \prod_{a=j-k}^{j+k} p(v_a|v_i) \tag{2.58}$$

$$p(v_a|v_i) = \frac{\exp \mathbf{w}_{v_a}^{\mathsf{T}} \mathbf{u}_i}{\sum_{l\in\mathcal{V}} \exp \mathbf{w}_l^{\mathsf{T}} \mathbf{u}_i} \ . \tag{2.59}$$

Practically, negative sampling is used to accelerate training.

It has been shown that word embeddings trained like this capture some semantic structure between words. For example, the cosine similarity between two semantically similar words (e.g. "cat" and "kitten") is higher than between unrelated words. In addition, the word embeddings appear to capture

[9] This vector is $\in \{0, 1\}^{\mathcal{V}}$, and contains a one only at the positions corresponding to words in the context of position j.

some relations between words as translations in the embedding space. The distributional hypothesis [53, 54], which is the idea that words that occur in similar context have similar meanings, is important in this context. Since the word embedding pre-training methods predict the context based on the word or vice versa, and hypothesis states that capturing meaning and context is the same thing, we can say that the pre-training in a way captures word meaning.

Using such pre-trained embeddings can be beneficial because of the larger amount of information captured in the representations. Every word observed during pre-training has a meaningful representation, reducing the chance for out-of-vocabulary words. If the word vector similarities have not been destroyed by subsequent task-specific fine-tuning, the fine-tuned model can also generalize better to unobserved words that are synonyms of the words observed during training (because of the high similarity of the representations of similar words).

2.4.2 Pre-trained Language Models

Word embeddings methods like GloVe that are discussed in previous section only pre-train word embeddings. When using the word embeddings, the model (e.g. the RNNs) are still initialized from scratch. While the pre-trained word vectors usually provide a good starting point compared to random initializations, their language modeling ability is severely limited. For example, the LM in Word2Vec disregards all order information and takes into account only a relatively small context window of a word occurrence. In addition, the representations for the words are *context-independent* in the sense that the same vector will be used for the same word, regardless of context.

In recent years, it has been shown that pre-training an entire network as a language model results in better performance on downstream tasks [7, 8, 55–60]. Since the entire network is pre-trained, all weights are set to maximize the ability of the model to predict other words in the input. Thus, the network is pre-trained such that it selects the most relevant pieces of context for representing its words. This results in *context-dependent* word representations with the possibility to capture long-range dependencies.

While several methods have recently been proposed for pre-training models for NLP, we will focus here on the seminal work of Devlin et al. [7] to provide a concrete reference for the background of the rest of this work. BERT is a transformer-based bidirectional masked language model. It is pre-trained on a large corpus of text (BookCorpus (800M words) and English Wikipedia (2500M words)) using an objective that consists of two parts: (1) the masked LM task and (2) the next sentence prediction task.

Rather than training a left-to-right language model (like OpenAI's GPT [57]), BERT has been pre-trained as a masked language model. This allows the model to use tokens both to the left (preceding) as well as right (after) to build representations of tokens. In contrast, a left-to-right language model would only be allowed to use tokens to the left of the current position.

The **masked LM** (MLM) task trains the model to predict words that have been omitted from the input. To do this, 15% of all tokens in a sentence are replaced with (1) a special `[MASK]` token (in 80% of the cases) or (2) a random token (in 10% of the cases). In the remaining 10% of the cases, the selected token is not replaced. This is done to reduce the discrepancy between training and test, because during testing, no special mask tokens are used. The objective of the masked LM task is to predict the (20%) masked words in the sentence. The next **sentence prediction** (NSP) task consists of predicting whether two sentences follow each other in the corpus or not. To train for this task, in 50% of the cases, two adjacent sentences are taken from the corpus, and in the other 50%, two random sentences are taken. The model is trained to distinguish between the two cases. The NSP task is meant to improve performance on downstream tasks that compare two sentences.

The model used in BERT is simply a transformer encoder with learned positional embeddings rather than the original fixed sinusoid ones. Before the input is fed to the model, it is first tokenized using a WordPiece [37] vocabulary of 30000 tokens. A special `[CLS]` token is added at the beginning of the input sequence. If the input is a single sentence, it is ended using the special `[SEP]` token. If the input is a sentence pair, they are separated using another `[SEP]` token. Thus, the sentence pair "The dog ate." and "Man bites dog" would be fed into the model as "[CLS] The dog ate . [SEP] Man bites dog [SEP]". In addition to positional embeddings, a special segment embedding is added to the WordPiece embeddings. This segment embedding simply specifies whether the token is part of the first sentence or the second.

Once pre-trained, the model can be fine-tuned to other tasks. In the author's experiments, the pre-trained BERT model achieved significant improvements across different text classification tasks as well as text-based question answering.

Follow-up work by Liu et al. [56] (RoBERTa) examines BERT's training choices and finds that the next sentence prediction task is unnecessary and proposes to use dynamic word masking rather than the static word masking implementation that was used in BERT. They also train with more data and further improve upon BERT's performance on downstream tasks.

While BERT and RoBERTa are examples of pre-trained transformer encoders, other work has proposed to pre-train an entire sequence-to-sequence transformer, such as BART [60] and the "text-to-text transfer transformer" (T5) [61]. Both essentially perform some form of denoising auto-encoding during pre-training, where noise is introduced to the sequence that serves as input to the encoder and the goal is to reconstruct the original sequence by the decoder. Pre-training is performed on massive amounts of text.

CHAPTER 3

Related Work

"The only constant in life is change."

– Heraclitus

In this chapter, we present the already existing research in the research directions relevant for this thesis. Parts of this chapter have been adapted from Publication 7. We first focus on neural semantic parsing in Section 3.1 and then continue to question answering over knowledge graphs in Section 3.2. In Section 3.3, we focus explicitly on generalization in semantic parsing and question answering.

3.1 Semantic Parsing

As mentioned before, semantic parsing is the task of translating a natural language utterance x into an executable meaning representation y, such that y conveys the full meaning of x in the context of the formal language and the underlying resources (e.g. knowledge graph, database, set of instructions for an environment). Even though question answering over knowledge graphs is possible without relying on an explicit semantic parse of the input questions, semantic parsing is commonly used for question answering.

Various approaches have been proposed in earlier work on semantic parsing, such as [62–67], which relied on Combinatory Categorial Grammars (CCG), rule-based systems or other techniques. In this work, we focus on neural network based approaches and refer the reader to Kamath and Das [68], Kumar and Bedathur [69], and Zhu et al. [70] for an overview of earlier work.

Unlike SimpleQuestions, where the outputs have a fixed structure, in the general case, semantic parsers need to be adaptive to different query structures, and ideally, generalize beyond the exact patterns observed during training. Standard neural sequence-to-sequence models present a possible approach for generating variable sized outputs given variable sized inputs. Some early works on neural semantic parsing have investigated sequence-to-sequence models [15] and proposed extensions that provide better inductive biases exploiting the structure of the outputs.

3.1.1 Structured decoding

In order to be valid (and executable), logical forms have to obey a certain set of grammatical rules. In many cases, logical forms can be represented as trees, for example, the syntax tree for Python programs or the predicate-argument structure denoted by s-expressions in LISP. These trees provides more structural information about the query than the sequence of tokens representation usually assumed by sequence decoders. Several works proposed decoders that explicitly exploit the structure of queries to better model dependencies within and between different query clauses.

Semantic parsers with structured decoders use the same sequence encoders to encode the NLQ but induce additional structure on top of the normal attention-based sequence decoder that exploits the hierarchical tree structure of the query.

Dong and Lapata [15] propose a tree decoding model for neural semantic parsing that decodes a query tree in a top-down breadth-first order. For example, instead of decoding the lambda-calculus logical form `(argmin $0 (state:t $0) (size:i $0))` (corresponding to the question "*Which state is the smallest?*") as a flat sequence of tokens, the decoder decodes the query in several steps. Internally, the tree decoder uses a normal RNN and manipulates the inputs and states of the sequence decoder according to the tree structure. For our example, the tree decoder proceeds as follows:

1. First, the topmost level of the query tree is decoded: `(argmin $0` <n><n>`</s>)`. Here, <n> and `</s >` are artificial topological tokens introduced to indicate the tree structure: <n> is a non-terminal token indicating that a subtree is expected at its position and `</s>` is the end-of-sequence token indicating the end of a sequence of siblings. For this first step, the decoder RNN is used like in a normal sequence decoder.

2. As the second step, after decoding the top-level sequence, the first child subtree, `(state:t $0)`, is decoded and inserted at the position of the first non-terminal <n>. This yields the query `(argmin $0 (state:t $0)` <n>`</s>)`. While decoding this subtree, the RNN is conditioned on the state corresponding to the first non-terminal produced in the first step (top level query).

3. Finally, `(size:i $0)` is decoded and inserted at the position of the last remaining non-terminal. While decoding the second subtree, the RNN is conditioned on the state corresponding to the second non-terminal produced in the first step (top level query).

Decoding terminates after these three steps because no non-terminals remain in the generated output. From experimental results on the sematic parsing data sets Geo880, ATIS, JobQueries, and IFTTT, it appears that the inductive bias introduced by the tree decoder improves generalisation.

Alvarez-Melis and Jaakkola [71] propose an improved tree decoder for semantic parsing, where the parent-to-child and sibling-to-sibling information flows are modelled with two separate RNNs. With this model, each node has a parent state and a previous-sibling state, both of which are used to predict the node symbol and topological decisions. Instead of modelling topological decisions (i.e. whether a node has children or further siblings) through artificial topological tokens like Dong and Lapata [15], they use auxiliary classifiers at every time step that predict whether a node has children and whether the node is the last of its siblings. The elimination of artificial topological tokens reduces the length of the generated sequences, which should lead to fewer errors. The decoding proceeds in a top-down breadth-first fashion, similarly to Dong and Lapata [15]. Experimental results on the IFTTT semantic parsing dataset show improvement obtained by the introduced changes.

Cheng et al. (2017, 2018, 2019) develop a transition-based neural semantic parser that adapts the Stack-LSTM proposed by Dyer et al. [75]. The Stack-LSTM decodes the logical forms in a depth-first order, decoding a subtree completely before moving on to its siblings. The Stack-LSTM of Cheng and Lapata [73] uses an adapted LSTM update when a subtree has been completed: it backtracks the state of the LSTM to the parent state of the completed subtree, and computes a summary encoding of the completed subtree which then serves as input (the entire completed subtree is thus treated like a single token) in the next LSTM update. Cheng et al. [74] show how to perform bottom-up transition-based semantic parsing using the Stack-LSTM. Cheng et al. [72] obtain state-of-the-art results on GraphQuestions and Spades and results competitive with previous works on WebQuestions and GeoQueries. Cheng and Lapata [73] further improve performance of their Stack-LSTM model on the weakly supervised GraphQuestions, Spades, and WebQuestions datasets by using a generative ranker.

3.1.2 Exploiting grammar in semantic parsing

The formal grammar underlying the output language in semantic parsing can be exploited to improve parsing performance. In the simplest case, the grammatical rules can allow us to apply some restrictions on the output tokens at a certain time step, depending on the sequence decoded so far. For example, in the FunQL language definition used by [72, 74], the $\text{argmax}(x, r)$ function is restricted to have a relation symbol r as second argument. Constrained decoding can be trivially implemented by just considering the set of allowed tokens as possible outputs at a certain time step.

Many semantic parsing approaches rely on a decoder that **generates production rules** or more generally, actions, rather than logical form tokens [73, 76–83]. For example, suppose we would like to decode a query $\text{argmax}(\text{Astronauts}, \text{height})$. We could decode the sequence of tokens ["argmax", "(", "Astronauts", ",", "height", ")"]. However, if we could define a CFG that contains the rules $X \longrightarrow \text{argmax}(X, R)$, $X \longrightarrow$ Astronauts and $R \longrightarrow$ height, we could produce the same expression as a sequence of these three rules (rather than a sequence of 6 tokens). In addition, with such a grammar, the constraints on the decoding can be trivially implemented by looking up rules applicable for the selected non-terminal according to the predefined grammar. This would ensure that only valid logical forms are being generated.

In [79], the authors discuss the difficulties associated with grammar-based semantic parsing, which include the use of variables and the need to take into account the schema's to make more restrictive yet shallow grammars. They propose a grammar-based decoder for the text-to-SQL semantic parsing task that decodes production rules at every decoding step. The production rule generated at a certain time step are applied to the left-most non-terminal in the intermediate output. The grammar consists of a global production rules, which is shared between examples and allows to describe the general structure of the SQL queries. This part includes rules such as $\underline{\text{query}} \rightarrow$SELECT $\underline{\text{distinct}}$ $\underline{\text{select\_results}}$ FROM $\underline{\text{table\_refs}}$ $\underline{\text{where\_clause}}$ $\underline{\text{group\_by}}$, where the non-terminals are underlined. Additional local production rules are created for every example using the schema for the database associated with that example, as well as the words from the input. This allows to restrict the allowed table names to those from the used database, using a rule like $\underline{\text{table_name}} \rightarrow$city | flights | ..., and to restrict values to existing values from the table, or from the input.

3.1.3 Exploiting semantic constraints

Some work in semantic parsing and question answering over knowledge graphs pointed out that exploiting data during decoding might help to produce more accurate results. For example, STAGG [84] (see also Section 3.2) actively exploits the knowledge graph structure to reduce the number of candidate actions at every decoding step. This is done by starting from one entity (called topic entity) and generating the core inferential chain of relations from that entity. The resulting method is a staged query graph construction method that combines bottom up and top-down decoding and despite limitations covers a sufficient range of queries.

Chen et al. [85] investigate query graph construction for semantic parsing. Rather than decoding logical forms as given in the dataset (e.g. SPARQL or s-expressions), similarly to [63, 84], they decode an intermediate representation called a query graph. To do so, they employ a sequence-to-action network that is implemented using an LSTM-based sequence-to-sequence attention-based model that at every decoding step generates query graph building actions. These actions include adding variable, entity and type nodes and adding edges and arguments for edges and types. These actions allow to easily exploit semantic constraints during decoding. The resulting approach is comparable to previous work on GeoQuery and ATIS and yields improvements on the Overnight dataset.

Note that similar semantic constraints could also be implemented as part of a normal depth-first top-down decoding procedure relying on logical forms with s-expressions.

3.1.4 Copying Mechanisms

The architectures proposed by Vinyals et al. [86], See et al. [87], Gu et al. [88], and Jia and Liang [11] are an augmentation of the attention-based sequence-to-sequence neural architecture which enables a direct copy of tokens, sub-sequences or other elements from the input to the output. Although it is not generally required for semantic parsing and KGQA, it can be useful for certain tasks or datasets, and could help to integrate neural decoding with external entity linking tools [89]. In WikiSQL, for example, a copying mechanism is required to copy SQL condition values into the query [90–92]. It has also been used for semantic parsing in general [11, 93], when the query may contain NL strings. In general, it can be seen as a more general alternative to replacing specific entities with placeholders.

Among the works using copying for semantic parsing, the work of Shaw et al. [89] is particularly interesting. There, the natural language question is converted to a graph representation that contains both the words from the question as well as candidate entities for phrases in the question. Candidate entities are generated using an external entity linker and are integrated into the input graph by adding edges that connect the candidate entity nodes to the nodes representing words to which the entities were linked. The decoder can generate tokens from the query language vocabulary, or use a copy mechanism to produce one of the linked entity candidates from the input graph. Even though the authors did not test this approach for QA over KGs, we believe this approach could be useful for KGQA.

3.1.5 Other Structured Decoders

The two-stage Coarse2Fine decoder of Dong and Lapata [17] can be seen as a middle-ground between a sequence decoder and a tree decoder. The decoder consists of two decoding stages: (1) decoding a query template and (2) filling in the specific details. Compared to other tree decoders, the Coarse2Fine decoder also proceeds in a top-down breadth-first manner, but is restricted to have only two levels. For cases when there is a limited number of query templates, Dong and Lapata [17] also investigate the use of a template classifier (instead of decoding the template) in the first decoding stage and evaluate on WikiSQL. An additional improvement to the two-step decoding scheme is obtained by encoding the generated template using a bidirectional RNN, before using the output states to decode the details.

This allows to condition the generation of specific arguments of the template on the whole structure of the template. As will be discussed later, the two-stage decoding approach appears to be useful to improve compositional generalization as well.

The work of Cheng and Lapata [73] trains and uses a translation model for semantic parsing. Using beam search, several logical forms are decoded from the translation model and additional ranking models are used to re-rank the logical forms in the beam.

3.1.6 Non-autoregressive decoding and semantic parsing

Several methods have recently been proposed for non-autoregressive decoding and insertion-based decoding. Stern et al. [23], Gu et al. [94], and Gu et al. [95] experiment with insertion-based decoding where Gu et al. [95] also support deletion operations. Ma et al. [96], on the other hand, develop a non-autoregressive sequence generation model using normalizing flows [97]. Some other examples of non-autoregressive decoding for NMT are [98–100].

In this thesis, we focus on insertion-based decoding. Rather than decoding a sequence left-to-right, as is usually done by decoders, the Insertion Transformer [23] decodes sequences using insertion operations. A notable property of the Insertion Transformer is its ability to generate multiple tokens in one decoding step, which enables $O(log_2(n))$ number of decoding steps for a sequence of n tokens where a standard left-to-right decoder needs n steps. The Insertion Transformer operates simply by inserting a token between any of the already decoded tokens and can be trained using different methods. Stern et al. [23] explore a few training methods, where the "Binary" method trains the insertion model to insert tokens from the middle of the still-to-decode span (hence the name), which minimizes the number of decoding steps in maximally parallel decoding. See Section 6.1 for more details.

To the best of our knowledge, to this date, only two other works have investigated non-autoregressive methods for trees and semantic parsing. Zhu et al. [101] apply the Insertion Transformer to semantic parsing, albeit on SNIPS [102], ATIS [103] and TOP [104] datasets and focuses on cross-lingual performance.

Particularly interesting is the work of Rubin and Berant [105] who propose a bottom-up tree decoder for semantic parsing. Even though their decoder is limited to being linear in depth and thus less parallelizable for deeper and narrower trees (in contrast, the Insertion Transformer [23] can be sub-linear both in depth and breadth), it is nevertheless interesting for several reasons and in practice, reduces the number of decoding steps similarly to insertion-based decoding. The authors develops a bottom-up parser (in contrast to most other work in semantic parsing, which is top-down). The parser uses a beam, which holds candidate subtrees for a given input. In every decoding step, (1) representations for the subtrees are updated taking into account the input, (2) the possible combinations of trees that are one level deeper are generated and scored based on the contextualized representations from the first step, (3) top-K of these tree are retained, and their representations computed. The algorithm starts out with trees of zero depth (single node), which contain entities and predicates that will be used in the query. One of the potential advantages of this parser, is that the intermediate trees on the beam can be executed, which can help improve parser performance. We believe that the approach of Rubin and Berant [105] can further be extended for KGQA, where the bottom-up decoding can help filter relations based on the used entities (similarly to STAGG, which partially, also decodes bottom-up). In addition, it could be possible to extend the parser actions such that they support attaching subtrees contained in the beam to each other, which would resolve the issue that the parser requires at least the same number of steps as the depth of the decoded tree.

3.1.7 Weak supervision for semantic parsing

In the *weakly supervised* setting, the training data $\mathcal{D} = \{\left(q^{(i)}, a^{(i)}\right)\}_{i=1}^{N}$ consists only of pairs of NLQs and corresponding execution results (answers). Different methods have been proposed for training translation models for semantic parsers using only weak supervision. Even though the proposed methods vary in the objective function and training procedure, they all operate on the general principle of maximizing the probability of producing the correct execution results, generally marginalizing out the latent logical forms in some way.

The following training methods are commonly used for weakly supervising translation-based semantic parsing models.

Maximum Marginal Likelihood (MML): Because the correct logical form is not given, rather than using maximum likelihood over question-query pairs, MML "integrates" over all possible logical forms that produce a certain answer, and maximizes the probability of the correct answer:

$$\sum_{(q,a)\in\mathcal{D}} \log p(a|q) = \sum_{(q,a)\in\mathcal{D}} \log \sum_{f\in\mathcal{F}} p(a|f) p_\theta(f|q) \ , \tag{3.1}$$

where a is the correct answer to the natural language question q and the sum is computed over all f's in the space of all possible logical forms $\mathcal{F}$. Since semantic parsing environments are usually deterministic (i.e. the same query always executes to the same answer when the KG is kept fixed), the $p(a|f)$ term is either 1 (if f's execution produced the correct results a, i.e. $f(\mathcal{K}) = a$) or 0, which leads to the following simplified form:

$$\sum_{(q,a)\in\mathcal{D}} \log \sum_{f\in\mathcal{F}^*} p_\theta(f|q) \ , \tag{3.2}$$

where $\mathcal{F}^*$ is the set of consistent logical forms that execute to the correct answer $a^{(i)}$. The set $\mathcal{F}^*$ is usually approximated using online beam search. However, an approximation of $\mathcal{F}^*$ can also be computed beforehand [106] and kept fixed throughout training.

Hard Expectation Maximization (hard EM): As Min et al. [107] point out, the MML objective can have the following disadvantages: (1) the objective can learn to assign high probabilities to any logical forms that execute to the correct answer, whereas usually, only one or a few of them are not spurious and the spurious ones should have a probability close to zero and (2) during training, MML maximizes the sum over correctly executing logical forms while during test, only the maximum is taken, which creates a discrepancy between training and prediction modes. Subsequently, Min et al. [107] propose a simple alternative to MML that conforms the assumption that there is only one correct logical form. Among all logical forms that execute correctly, the one with the highest likelihood is taken:

$$\hat{f} = \text{argmax}_{f\in\mathcal{F}*} p(\hat{f}|q,\theta) \ , \tag{3.3}$$

and used to update the parameters by taking an SGD step to maximize its probability:

$$\sum_{(q,a)\in\mathcal{D}} -\log p(\hat{f}|q,\theta) \tag{3.4}$$

This approach appears to work well for training span-based semantic parsers with improved compositional generalization [108].

Reinforcement Learning using Expected Rewards (ER): The problem of weakly supervised semantic parsing can also be approached as a reinforcement learning (RL) problem. In fact, we can view the translation model as a parameterised policy that given a state, which in our case consists of the KG $\mathcal{K}$, input q and decoding history $f_{<t}$, must decide what action to take in order to maximise a reward. In our case, the reward function $R(f, a)$ can be a binary reward at the end of an episode (end of decoding phase), that is 1 if the produced trajectory (logical form) f executes to the correct answer a and 0 otherwise. The RL setup for semantic parsing is characterised by (i) a deterministic environment (the next state produced by executing the action on the current state is always the same) and (ii) sparse rewards (the reward is given only at the end of the episode and, given the huge number of possible logical forms, is likely to be 0 most of the time). In addition, weakly supervised training of semantic parsing is characterised by *underspecified* rewards [109] which could lead to learning *spurious* logical forms. However, the reward function based on execution results only cannot take this into account.

Policy gradient methods are trained by optimising the *expected reward*:

$$\sum_{(q,a)\in\mathcal{D}} \sum_{f\in\mathcal{F}} R(f,a)p_\theta(f|q) \quad , \tag{3.5}$$

where the sum is over all possible logical forms, which, in practice, is estimated using an approximation strategy such as *Monte Carlo integration*, i.e. based on trajectories sampled from the policy.

Iterative Maximum Likelihood (IML): The Iterative Maximum Likelihood (IML) objective [110] uniformly maximizes the probability of decoding *all* consistent logical forms across all examples:

$$\sum_{(q,a)\in\mathcal{D}} \sum_{f\in\mathcal{F}^*} \log\ p_\theta(f|q) \quad , \tag{3.6}$$

When training policy gradient methods using ER, Liang et al. [110] demonstrate that better exploration can be achieved by employing IML to populate an initial buffer of diverse trajectories (i.e. logical forms) which are then used to pretrain the RL model.

Memory augmented policy optimisation (MAPO): In order to reduce the variance of the ER estimator, Liang et al. [111] proposed a novel method that integrates a memory buffer. They reformulated the ER objective as a sum of action sequences in a memory buffer $C(q)$ and outside the buffer:

$$\sum_{(q,a)\in\mathcal{D}} \sum_{f\in C(q)} R(f,a)p_\theta(f|q) + \sum_{f\notin C(q)} R(f,a)p_\theta(f|q) \quad . \tag{3.7}$$

The memory buffers $C(q)$ for each example i are populated using *systematic exploration*, which prevents revisiting sequences that have already been explored. When the two terms are explored further (see [111]), we can see that the weight assigned to trajectories from the memory buffer is low in the beginning of training, when the policy still assigns low probabilities to the buffered trajectories.

In order to speed up training, Liang et al. [111] propose *memory weight clipping*, which amounts to assigning the buffered trajectories an importance weight of at least a certain value. Experimental results show significant improvement of the proposed MAPO procedure against common baselines and show that both systematic exploration and memory weight clipping are essential to achieve high performance.

Agarwal et al. [109] argue that apart from an optional entropy term, MAPO does not encourage exploration, which can be problematic. They propose a MAPO variant called MAPOX where the initial buffer is additionally populated by sequences found using IML.

Discussion: A disadvantage of the RL and MML approaches, noted by Guu et al. [112] and Agarwal et al. [109], is that the exploration of trajectories is guided by the current model policy, which means that logical forms that have high probability under the current model are more likely to be explored. Consequently, logical forms with low probability may be overlooked by exploration. In the presence of many spurious logical forms and only a few correct logical forms, it becomes more likely that spurious ones are explored first. Once the policy settles on these high-reward (but spurious) logical forms, the exploration gets increasingly biased towards them during training, leading to poor generalisation. One common solution to this is to perform ϵ-greedy exploration, which Guu et al. [112] extend from RL to MML. Guu et al. [112] also propose a *meritocratic update rule* that updates parameters such that probability is spread more evenly across consistent logical forms. For a more detailed discussion of the MML and ER objectives, we refer the reader to the work of Guu et al. [112], Norouzi et al. [113], Roux [114] and Misra et al. [115].

Agarwal et al. [109] propose a meta-learning approach for learning an auxiliary reward function that is used to decrease the effect of spurious logical forms in the MAPOX objective. The auxiliary reward function is trained such that the update to the policy under the reward augmented by the auxiliary reward function results in better generalisation over a held-out batch of data.

Cheng and Lapata [73] propose a neural parser-ranker based approach for weakly supervised semantic parsing, where a sequence-to-sequence *parsing* model (trained with beam search and IML) is coupled with a basic ranking model (trained using MML) as well as an *inverse parsing* model, which is a generative model that reconstructs the original question from the logical form generated by the *parsing model*. The reconstruction loss is used to further refine the parsing model in order to tackle the problem of spurious logical forms. Note that here the logical form is treated as a latent variable.

3.2 Question Answering over Knowledge Graphs

Question answering systems over knowledge graphs (and other structured data sources, like relational databases) typically use semantic parsing to construct a logical form that can then be executed over the data source by a database management system. Note that it is also possible to follow an approach where the model learns to select the right answer in the knowledge graph using retrieval and ranking, as for example in Bordes et al. [116]. However, with such direct approaches, it is not clear how to efficiently perform certain operations, like aggregations, that can easily be handled by semantic parsing approaches. In addition, semantic parsing can be done independently of the actual values contained in the database and is thus by default easily re-usable when the data changes.

3.2.1 Simple Questions

In the most general case, the semantic parser should be able to generate a structured output (i.e. the corresponding formal query) of arbitrary size and complexity given the input NLQ. In some cases, however, we can assume a fixed structure for the formal query. This holds in the case of single fact based questions (e.g. SimpleQuestions), where only a single subject entity and a relation need to be predicted. For instance, the question *"Where was Frank Herbert born?"* is represented in the KG by a single triple pattern based SPARQL query with the triple pattern being ⟨?x, :birthplaceOf, :Frank_Herbert⟩. The logical form corresponding to this type of question thus always consists of one subject entity and a relation, and the answer is given by the missing object entity. More formally, given a knowledge graph $\mathcal{G}$ that contains triples of the form (s, p, o), where s is a subject entity, p a predicate (also denoted as relation), and o an object entity. The task of Simple QA is then: Given a natural language question represented as a sequence of words $q = \{w_1, \ldots, w_T\}$, find the set of triples $(\hat{s}, \hat{p}, \hat{o}) \in \mathcal{G}$ such that their objects $\hat{o}$ is the intended answer for question q. This task can be reformulated as finding the right subject $\hat{s}$ and predicate $\hat{p}$ that question q refers to and which characterize the set of triples in $\mathcal{G}$ that contain the answer to q.

For such fixed-structure prediction tasks, we can make use of simple text classification and ranking methods to predict the different parts of the target formal query given the natural language question as input (see also [117] for a recent overview of deep learning architectures for simple questions)[1].

For single-fact based prediction tasks, systems relying on standard classification models, can achieve state-of-the-art performance [118, 119]. Since the target formal queries consist only of one subject entity and one relation, they can, in principle, be predicted using two separate classifiers, receiving the NLQ as input and producing an output distribution over all entities and all relations in the KG, respectively. This approach can be successfully applied to predict the KG relation mentioned or implied in the question. However, large KGs like Freebase contain a huge amount of entities and training datasets can only cover a small fraction of these (and therefore many classes remain unseen after training). This makes it difficult to apply the approach described above for relation classification to entity prediction.[2]

Therefore, several works on SimpleQuestions initially only perform relation classification and rely on a two-step approach for entity linking instead. In such an two-step approach, first an entity span detector[3] is applied to identify the entity mention in the NLQ. Secondly, simple text based retrieval methods can be used to find a limited number of suitable candidate entities. The best fitting entity, given the question, can then be chosen from this candidate set based on simple string similarity and graph-based features. The entity span detector can be implemented based on classifiers as well, either as a pair of classifiers (one for predicting the start position and one for predicting the end position of the span) or as a collection of independent classifiers[4] (one for every position in the input, akin to a token classification network), where each classifier predicts whether the token at the corresponding

[1] One may argue that such simple methods that only output an entity and relation, are not really semantic parsing methods. Since for simple questions these are sufficient to form a formal query returning the correct answer, they nevertheless fit our definition of semantic parsing in this context.

[2] Relation prediction might suffer from the same problem as entity prediction. However, usually there are much fewer relations (thousands) than entities (millions) in a KG, such that a dataset of substantial size can provide several instances of each (important) relation.

[3] The entity span detector does not need to learn representations for entities (like encoder networks used in classifiers), avoiding the need for data covering all entities.

[4] representing a sequence tagger

position belongs to the entity span or not.

To give a concrete example of a full KGQA semantic parsing algorithm relying on classification models, we will describe the approach proposed by Mohammed et al. [118], chosen due to its simplicity and competitive performance on SimpleQuestions. This QA system follows the following steps for predicting the entity-relation pair constituting the formal query:

1. Entity span detection: a bidirectional Long-Short Term Memory network (BiLSTM) [42] is used for sequence tagging, i.e. to identify the span a of the NLQ qthat mentions the entity. In the example "*Where was Frank Herbert born?*", the entity span could be "*Frank Herbert*".

2. Entity candidate generation: Given the entity span, all KG entities are selected whose labels (almost) exactly match the predicted span.

3. Relation classification: A bidirectional gated recurrent unit (BiGRU) [120] encoder is used to encode q. The final state of the BiGRU is used to compute probabilities over all relations using a softmax output layer, as described in Equations (2.22) and (2.23).

4. Logical form construction: The top-scoring entities and relations from previous steps are taken, and all their possible combinations are ranked based on their component scores and other heuristics. The top-scoring pair is then returned as the predicted logical form of the question.

For training the full system, the relations given in the formal queries of the dataset are used as correct output classes. For the span detector, we first need to extract "pseudo-gold" entity spans since they are not given as part of the dataset. This is done by automatically aligning NLQs with the entity label(s) of the correct entity provided in the training data, i.e. the part of the NLQ that best matches the entity label is taken as the correct span to train the sequence tagging BiLSTM.

Mohammed et al. [118] investigate different RNN and convolutional neural network (CNN) based relation detectors as well as bidirectional long-short-time-memory (LSTM) [42] and CRF-based entity mention detectors. They show that a simple BiLSTM-based sequence tagger can be used for entity span prediction, leading to results competitive to the state-of-the-art.

Both Mohammed et al. [118] and Petrochuk and Zettlemoyer [119] first identify the entity span, similarly to previous works, but disambiguate the entity without using neural networks. Petrochuk and Zettlemoyer [119] employed a BiLSTM-CRF model for span prediction, which can no longer be considered a simple collection of independent classifiers and instead forms a structured prediction model that captures dependencies between different parts of the output formal query (in this case the binary labels for every position).

Bordes et al. [121] propose a memory network (MemNN)-based solution to SimpleQuestions. They use bag-of-words representations for triples and question and train a model that predicts the right triple by minimizing a margin-based ranking loss as defined in the section above. They compute scores between questions and whole triples, including the triple objects. However, triple objects are answers and thus might not be present in the question, which may affect the performance adversely. The same work introduces the SimpleQuestions dataset, consisting of approximately 100,000 question-answer pairs. Other works on SimpleQuestions typically predict only the subject entity and predicate separately, (unlike [121], which ranks whole triples).

He and Golub [122] explore fully character-level encoding of questions, entities and predicates, and use an attention-based decoder [46]. Their model consists of a character-level RNN-based question encoder and an attention-enhanced RNN decoder coupled with two separate character-level CNN-based entity label and predicate URI encoders. Given that their model works purely on character-level, which results in much longer input sequences, the effect of an attention mechanism could be more pronounced than with word-level models. From the output size perspective, it is reasonable to assume that compared to machine translation an attention mechanism is of lesser importance in Simple QA where only two predictions need to be made per input sequence.

Yu et al. [123] explore relation detection in-depth and propose a hierarchical word-level and symbol-level residual representation. They employ CNNs and propose an attentive pooling approach to improve the model. First, the question is split into entity mention and relation pattern. Then, the entity mention is encoded using a character-level CNN and matched against a character-level encoded subject label; the relation pattern is encoded using a separate word-level CNN with attentive max-pooling and matched against a word-level encoded predicate URI. Instead of using an attention mechanism to implicitly perform segmentation, in this approach an attention mechanism is used to obtain better matches for predicates.

Dai et al. [124] investigate a word-level RNN-based approach. They encode the question on word-level only and train/use Freebase-specific predicate and entity representation (such as pretrained TransE embeddings and type vectors). Thus, the approach of Dai et al. is highly specific for Freebase and can less easily be transferred to other KGs, such as DBpedia without re-training embeddings. Similarly to [125], they use a BiLSTM+CRF tagger to improve the performance. The tagger is trained to predict which parts of the input correspond to the entity mention.

3.2.2 Complex Questions

One of the early works on question answering over knowledge graphs (KGQA) is the work of Berant et al. [64], who create the WebQuestions dataset and propose a semantic parser that scales to Freebase [1]. The WebQuestions dataset contains 5810 examples which consist of a question and the corresponding answer in the Freebase KG. The dataset was constructed by prompting the Google Suggest API for questions involving one entity and asking Amazon Mechanical Turk workers to find the answer on that entity's Freebase page. The authors also propose Sempre, a semantic parser. First, a lexicon mapping between natural language phrases and KG predicates is extracted, which will be used by the parser to generate predicates from the question. The authors note that the lexicon may not always produce all the necessary predicates and propose to circumvent this issue by allowing a "bridging" operation, which essentially produces a predicate even when it was not detected using the lexicon. The queries are constructed in a bottom-up fashion, where every span is mapped to a logical form using either the lexical rules, the bridging operation or the composition rules. The composition rules include intersection and union. All possible compositions can be generated for any pair of non-overlapping spans, which leads the approach to heavily over-generate the candidates. The authors use a log-linear model to learn to select the right query tree. Because the dataset is not assumed to provide the full query annotation, which is considered to be expensive to obtain, the authors train the model by marginalizing over all the derivations that produce the correct answer. In their follow-up work on WebQuestions, Berant and Liang [126] propose a semantic parsing approach based on paraphrasing: (1) a number of candidate logical forms is first constructed and then (2) translated back to natural language, thus paraphrasing the original question. Finally, (3) the logical forms are ranked by comparing their corresponding paraphrases. Interestingly, a vector space model is used, where the authors rely on pre-trained word embeddings [39] to represent natural language utterances.

Bordes et al. [116] propose an approach for KGQA that compares the embeddings of the question with the embedding of the subgraph surrounding a candidate answer entity. The approach uses the encodings $f(q)$ of the question q and $g(a)$ of the answer a and scores the answer using the simple scoring function $f(q)^{\mathsf{T}} g(a)$. The question encoding is simply a bag of word vectors while the authors experiment with different representations of the subgraph around a: (1) entity embedding of the answer entity a, (2) embedding of path from entity in question q to answer entity a and (3) embedding of the entire subgraph around answer entity a, which is also combined with (2). The scoring function is optimized using a margin-based ranking loss with negative sampling due to a large number of candidates. At prediction time, the answer a is returned by finding the best-scoring entity. This approach is similar to Bordes' earlier work [20], where a similar scoring function is used but the answer a is represented by the entity embedding only.

The works of Bordes et al. [116] and Berant and Liang [126] are perhaps the first attempts at question answering using neural network based models, even though they only relied on embeddings.

The WebQuestions dataset spawned other approaches [63, 84, 127, 128]

Yih et al. [84] proposed the Staged Query Graph Generation (STAGG) approach for semantic parsing for question answering over knowledge graphs. They formulate semantic parsing as a staged search problem, where each stage extends the parse with specific information, and use the KG structure more aggressively to prune the search space of possible queries. Similarly to Reddy et al. [63], the query is represented using a graph. While STAGG can generate only a limited number of query structures (for example, it's limited to two-hop inferential chains with variables only on the this chain), it appears to provide sufficient coverage for the WebQuestions dataset. The building process of a query proceeds through the following fixed stages: (1) the *topic* entity is generated, (2) the *core inferential chain* is predicted and (3) the core chain is augmented with additional constraints and aggregations. For (1) entity linking, an existing system is re-used. For (2) core inferential chain prediction, the authors propose a neural network based model that encodes the chain and the question using a CNN-based encoder. The CNN maps a sequence to a fixed dense vector representation and subsequently, the candidate inferential chains are scored by cosine similarity with the vector representation of the question. Finally, to make predictions, a log-linear model is used that uses various predefined features, among which is the similarity between question and core chain encodings. The log-linear model is trained as a ranking model using the F1 scores of candidate queries. This way, partially correct queries are still encouraged more than completely incorrect queries.

The previously discussed works used the WebQuestions dataset, which is annotated only with answers rather than logical forms. This was done because answer annotation was believed to be cheaper to obtain than full SPARQL queries, like in Free917 [51], which would require expert annotation. Because only answers, i.e. query execution results, are available, we can not directly train a decoder for structured prediction and instead have to use training methods suitable for this weak supervision (see Section 3.1.7). However, training under weak supervision could suffer from issues due to the presence of *spurious* logical forms, which are output structures that produce the correct answer but do *not* correctly convey the same meaning (the simplest example is perhaps "what is two plus two" →"2 × 2" while the correct logical form should be "2 + 2").

The work of Yih et al. [129] challenges the assumption that fully annotated semantic parses are significantly more expensive to obtain and provide the WebQuestionsSP dataset, in which a large subset of WebQuestions questions is annotated with SPARQL queries, rather than only answers. They show that using full semantic parses for training can yield large gains. The author perform a small annotation cost analysis and conclude that it can actually be faster and cheaper to obtain full semantic parses given that the annotation interface is well-designed. They also note that the annotation process can be faster and more complete because when annotating with query execution results only, ideally all answer entities should be enumerated. The resulting dataset contains 4737, which is smaller than the 5810 in WebQuestions because a fraction of the original questions could not be correctly annotated. Finally, the authors train STAGG [84] on WebQuestionsSP, comparing the setting where it is trained using only answers to when it is trained using full queries. Training with full queries proceeds similarly to the original procedure for training with answers, with the difference that the ranking score for the candidate queries is set to 0 if the query is not contained in the gold standard query. This effectively filters out spurious logical forms that are also trained to obtain a higher probability with weak supervision. The results show that using full semantic parses for training results in an absolute 5 point gain in F1 on WebQuestionsSP.

Yu et al. [123] focus on improving relation detection for KGQA over both SimpleQuestions and WebQuestionsSP. The authors note that relation detection in questions for KGQA can be more challenging than the general relation detection task in NLP. In comparison to general relation detection, KGQA-specific relation detection (1) has more possible output classes (there are thousands of unique relations in real-world KGs), (2) may require zero-shot generalization to new relation types due to limited training data coverage and large number of relations and (3) may require predicting a chain of relations rather than one relation. The authors address these issues by proposing a hierarchical residual BiLSTM model for relation detection. To enable zero-shot generalization, they represent relations both at word- and relation-level (2) and allow for multiple relations in a sequence (3). For example, the relational path "birthplace_of writer_of" for the question "Where was the writer of Pet Sematary born?" would be represented as a sequence of relations [birthplace_of, writer_of], as well as a sequence of words [birthplace, of, writer, of]. The two sequences are fed into a BiLSTM, the word sequence first. Finally, the representation for the relation path is obtained by maxpooling between the word-level and relation-level encodings. Correspondingly, the authors use residual connections on the question side to better represent the question at different levels of granularity and use maxpooling to obtain the final question representation. The scorer then compares the representations of the question with the relation using cosine similarity. A margin-based ranking loss with negative samples is used to train the scorer. The improved relation detector is then used in a STAGG-like parser and evaluated on SimpleQuestions and WebQuestions.

Talmor and Berant [130] generate a dataset with more complex questions, called ComplexWebQuestions. The dataset is generated starting from WebQuestions, which, as the authors note, mostly contains simple questions querying a particular property of some entity. Dataset generation is performed by generating SPARQL queries for the new more complex queries, automatically generating natural language versions, which are subsequently paraphrased by AMT workers. The final dataset contains nearly 35k examples. Another interesting aspect of the dataset is that it also provides text snippets from the web that may contain useful information to answer the question using reading comprehension (RC) systems. This makes the dataset useful for both the semantic parsing as well as reading comprehension communities. The authors use this dataset to investigate an approach for answering complex questions that could support the integration of facts expressed both in text as well as as knowledge graph facts.

Lan and Jiang [131] note that simply trying to apply STAGG [84] on WebQuestions would lead to a combinatorial explosion of candidate core chains when longer core chains are allowed. Recall that STAGG generates a set of candidate core inferential chains and subsequently ranks them. While beam search in this context [132] would help to contain the explosion, Lan and Jiang [131] propose to further restrict the candidate space by *early merging* of constraints into the main core chain. Recall that STAGG first generates the core chain and subsequently attaches constraints. However, as Lan and Jiang [131] note, attaching constraints before the core chain is fully decoded could significantly reduce the number of core chain continuations to consider, in other words, it enables much more aggressive pruning. They modify the STAGG approach accordingly, and achieve state-of-the-art results on WebQuestionsSP [129], ComplexWebQuestions [130] and ComplexQuestions [133].

Sun et al. [134](PullNet) and Sun et al. [135](Graft-Net) target a question answering setting where both knowledge graphs and text are used to answer questions. This can be practically more useful as KG's can be incomplete while text generally provides higher fact coverage at the expense of being unstructured. PullNet [134] extends Graft-Net [135] by better guided exploration of facts, learning which facts to "pull" while exploring the subgraph surrounding the question. Both methods rely on generating a subgraph (that can use both KG facts as well as text snippets) around the question that contains the answer entities. Once such a sugbraph is constructed, the answers can be retrieved simply by node classification in the resulting graph. PullNet reuses the model of Graft-Net for answer selection, which is a highly advanced graph neural network for the heterogeneous KG-and-text graph that combines graph convolution [136, 137] and LSTM [42] and also uses graph attention [138] and directional propagation. While Graft-Net retrieves a subgraph using some heuristics, PullNet notes that this method frequently results in subgraphs that are too large and may not even contain the answer. Instead, PullNet proposes to learn how to expand the subgraph, guiding the exploration towards more useful regions of the combined KG-text graph. This results in an iterative approach where in every iteration, first the nodes are classified for which information is to be retrieved, then this information is retrieved and added to the graph. Finally, when all relevant information is retrieved, the node classification model is used for answer selection.

The work of Das et al. [139] takes a prototype-based approach for building queries for knowledge graphs: rather than generating queries from scratch, like normal sequence-to-sequence models would do, their approach instead looks up similar questions and builds the new query by copying from the retrieved queries. The approach stores previously seen examples consisting of question-query pairs in a non-parametric memory. ROBERTA [56] is used to encode the natural language questions, which will be used as keys in the retrieval process. Retrieval is done by comparing the encoding of the new example with that of the stored question-query pairs. In the next step, the best matching questions and their queries are concatenated to the original question and fed into BigBird [140]. The output query can then be generated by copying from the retrieved queries. Finally, the authors note that the desired relations might not be present in the retrieved queries and propose a simple rewriting step that can replace a relation from the output with a relation in the neighbourhood of the used entities. Two ways are explored for this: (1) using TransE [141] embeddings for relations and (2) using similarity in surface forms of the relations. The resulting method shows large gains w.r.t. previous state-of-the-art on KG-only ComplexWebQuestions and exhibits compositional generalization comparable to a T5-11B model on CFQ [14] (see also Section 3.3.3).

Saxena et al. [142] propose a method that uses knowledge graph embeddings and question embeddings to find the answer entity for a question. Thus, it does not need to explicitly build a semantic parse. Their approach works as follows: (1) a knowledge graph embedding model is learned, (2) a question embedding module is learned. Answer selection is performed by comparing the question embedding to the embedding of the candidate answer entity. The authors use the ComplEx [143] knowledge graph embedding approach for building entity and relation representation, as well as for learning compatible question encoders. Using this embedding based approach enables the question

Dataset	KG	Size	FQ	CQ
Free917 [51]	Freebase	917	Yes	Yes
WebQuestions [64]	Freebase	5810	No	Yes
WebQuestionsSP [129]	Freebase	4737	Yes	Yes
ComplexWebQuestions [130]	Freebase	35k	Yes	Yes
ComplexQuestions [133]	Freebase	2100	Yes	Yes
GraphQuestions [155]	Freebase	5166	Yes	Yes
SimpleQuestions [121]	Freebase	100K	Yes	No
QALD	DBpedia	50-500	Yes	Yes
LC-QuAD [21]	DBpedia	5000	Yes	Yes
LC-QuAD 2 [156]	DBpedia, Wikidata	30 000	Yes	Yes
CFQ [14]	Freebase	239357	Yes	Yes
GrailQA [10]	Freebase	64331	Yes	Yes

Table 3.1: The table lists the most commonly known KGQA datasets and their characteristics namely: (i) The name of the dataset (ii) the underlying knowledge graph on which the dataset is based (iii) the size of the dataset in terms of number of questions (iv) availability of **F**ormal **Q**uery (FQ) (v) presence of **C**omplex **Q**uestions (CQ) i.e. simple or complex.

answering system to directly benefit from the knowledge graph completion capability of knowledge graph embedding models. The results on WebQuestionsSP indicate improvements in Hits@1 in incomplete KG settings compared to previously proposed approaches that also rely on text snippets.

Some other works we have not discussed here but that are worth mentioning include those of Dubey et al. [34], Guo et al. [82], Shen et al. [83], Cui et al. [144], Fader et al. [145], Reddy et al. [146], Xu et al. [147], He et al. [148], Unger et al. [149], Unger and Cimiano [150], Kacupaj et al. [151], and Saha et al. [152]. Some of these works are not discussed in detail because they concern conversational QA [82, 83, 151, 152]. Even though conversational QA and semantic parsing in dialogue context are very important and interesting research directions, they are not the main focus of this thesis. Some of the other discussed works concern earlier methods, such as rule-based QA systems. Overall, these works are relevant but not as closely related as those discussed before.

3.2.3 Datasets

In this subsection, we summarize the datasets mentioned above and other datasets that have been proposed for KGQA. Research in the field of KGQA has seen a shift from manual feature engineering based solutions [63, 64, 153, 154] to neural network based, data driven approaches. One of the pivotal requirements for these data-driven approaches is the availability of large datasets comprising of a wide variety of NLQs-label pairs. As a consequence, one can observe a shift in the scale, and more recently, also in the complexity of questions in KGQA datasets. The properties of the most popular KGQA datasets are summarised in Table 3.1.

One of the first attempts to create a large scale KGQA dataset was by Cai and Yates [51] who developed the Free917 dataset, consisting out of 917 question / formal query pairs covering over 600 Freebase relations. Along the same lines, Berant et al. [64] developed another dataset called WebQuestions by finding questions using the Google Suggest API and answering them with the help of Amazon Mechanical Turk (AMT) workers using Freebase. Although this dataset is much larger than Free917, it suffers from two disadvantages. First, it does not provide formal queries as targets for NLQs but only question-answer pairs, inhibiting supervised training of KGQA approaches which rely on a logical form. Second, it is primarily composed of simple questions, and relatively few require complex reasoning. In order to alleviate the first issue Yih et al. [129] introduced WebQuestionSP, a subset of WebQuestions, having both answers as well as formal queries corresponding to each question. In order to provide more structural variance and expressiveness in questions, Bao et al. [133] and Su et al. [155] have released ComplexQuestions and GraphQuestions, respectively, consisting of pairs of questions and their formal queries, where they augment a subset of WebquestionSP with type constraints, implicit and explicit temporal constraints, aggregation operations etc. Trivedi et al. [21] released LC-QuAD, a dataset of complex questions for the DBpedia KG. They started by generating formal queries for DBpedia and semi-automatically verbalised them using question templates, and then leveraged crowd-sourcing to convert these template-based questions to NLQs, performing paraphrasing and grammar correction in the process. Dubey et al. [156] used a similar mechanism to create a significantly larger and varied dataset - LC-QuAD 2. QALD[5] [157] is another important KGQA dataset based on DBpedia. Although it is substantially smaller than LC-QuAD, it has more complex and colloquial questions as they have been created directly by domain experts. SQA and CSQA are datasets for sequential question-answering on KGs, where each data sample consists of a sequence of QA pairs with a shared context. In these datasets, while the individual questions in a sequence are generally short with contextual dependencies.

Most datasets discussed so far, due to their relatively small size, do not provide a thorough coverage of the entities and relations that exists in a KG. In order to partially compensate for this and support a larger variety of relations, Bordes et al. [121] created the SimpleQuestions dataset with more that 100k questions. These questions can be answered by a single triple in the KG, and thus the corresponding formal queries can be thought to simply consist of the subject entity and predicate of the triple. However, as Gu et al. [10] note, the relation coverage in SimpleQuestions is still not adequate and many relations are repeated many times.

The Compositional Freebase Questions (CFQ) [14] dataset is an automatically generated dataset specifically designed to measure compositional generalization in question answering over knowledge graphs. The dataset comes with different *maximum compound divergence* (MCD) splits that are generated such that the distributions over the combinations of items and rules (atoms) are maximally different between the training and test sets. This challenges the ability of systems to generalize to a test set uses different combinations than seen during training. See Section 3.3.3 for more on this. GrailQA [10] is a crowdsourced dataset aimed at measuring different types of generalization in KGQA: (1) i.i.d. generalization, (2) compositional generalization and (3) zero-shot generalization, concerns generalization to previously unseen output symbols and even domains.

3.3 Generalization

In this section, we will focus on related work that focuses on the generalization of neural network based models for semantic parsing and question answering over knowledge graphs. While generalization is a central topic in machine learning research, here we will especially focus on important topics that arise in semantic parsing. These include (1) compositional generalization and (2) the effect of order of linearization of query trees, as well as (3) generalization to new output symbols.

[5] QALD is an ongoing KGQA challenge with multiple tracks - see `http://qald.aksw.org/`.

The recent work of Gu et al. [10] points out three types of generalization in KGQA: (1) i.i.d. generalization, (2) compositional generalization and (3) zero-shot generalization. While most work has been done in the i.i.d. setting, where the training and test data are random samples from the same set of examples, the authors note that this setting is insufficient for generalization in KGQA. The next level of generalization is compositional generalization, which requires the model to process novel combinations of already observed schema items. Finally, zero-shot generalization concerns the ability of a model to handle entirely new schema items or even entirely new domains. The authors create a novel dataset for KGQA, called GrailQA which allows to measure all three types of generalization.

3.3.1 Symbol representations

When answering questions over large scale knowledge graphs, we are confronted with a large number of entities and relations not present in the training data. An important challenge, thus, is to find a way to learn representations for entities and relations that generalise well to unseen ones. Representing both entities and predicates as a sequence of words, characters or sub-word units (BPE/WordPiece) in their *surface forms*, as opposed to arbitrary symbols unique to each entity or predicate, can offer some generalisability to such unseen or rare tokens. By default, we would learn a unique representation vector for every token, which would leave unseen and rare tokens untrained or under-trained. However, we can take additional information describing these tokens (such as the letters or words in their labels), and learn the representations for these new sub-tokens instead. If we describe our tokens using sub-tokens in such a way that the sub-tokens used for unseen tokens are themselves observed sufficiently during training, at least the representations of the sub-tokens will be better trained. The representations of the tokens that are then built based on the representations of the sub-tokens prevent relying on poorly trained parameters. However, it is still not guaranteed that the representations for tokens resulting from the composition of sub-token representations is well-constructed, especially for unobserved combinations. In the case of relations, if the representations are learned on sub-token levels, upon encountering sub-words from unseen relations, the parameters of the representation building network (encoder) can leverage sub-words shared between seen and unseen relations and thus, unseen relations will no longer use uninformed random vector representations.

Several works on question answering and semantic parsing (and other NLP tasks) have used such computed representations. For example, Several works on WikiSQL also encode column names on word level to yield vectors representing columns [81, 90, 158]. Some works on KGQA [10, 122, 123, 159, 160] also decompose KG relations and/or entities to word and/or sub-word level. GrailQA [10] uses sub-word based decompositing of Freebase relation ids using BERT. In addition to sub-token level encoding, additional information about the tokens can be encoded and added to its representation. For example, Yu et al. [81] also encodes table names and column data types together with column name words for their model for the Spider dataset. Gu et al. [10] encode Freebase relation URI's on word level using BERT [7]. Note that BERT uses a sub-word tokenizer, and processes common words on word level while less common words are split into multiple possibly multi-characters parts.

3.3.2 The effect of order and linearization

For many query representation languages, such as SPARQL, s-expressions, lambda expressions and SQL, there are elements in the queries whose order does not play a role in the meaning of the query. For example, the different triples in the triple pattern of a SPARQL query can be interchanged without changing the output or the meaning of the SPARQL query. Under the usual sequence-to-sequence training method, *teacher forcing* is used, which feeds the ground truth sequence as the input to the decoder during training. This is different from test time, where the decoder is ran autoregressively, consuming its previous output as the input to the next time step. Due to this difference between train and test, teacher forcing is believed to lead to exposure bias [161].

However, training using teacher forcing for semantic parsing, as is common practice, can lead to problems due to the presence of multiple valid "decoding trajectories" for the same query. For example, as Zhong et al. [92] point out, the order of the conditions in an SQL query does not play a role: "SELECT name WHERE occupation = 'author' AND age > 50" is equivalent to "SELECT name WHERE age > 50 AND occupation = 'author' ". Zhong et al. [92] argue that while both queries are correct, only one of them will be used in training, penalizing the other possible query orders. To address this, Zhong et al. [92] use policy gradient-based reinforcement learning.

The phenomenon of order sensitivity is one instance of the more general issue of decoding sets rather than sequences. Vinyals et al. [19] investigates the effect of order both in the input and the output. They note that the order matters when encoding or decoding sets and propose methods for order-invariant encoding and decoding of sets. Interestingly, they showed that decoding depth-first linearization of constituency parse trees performs better than breadth-first linearizations. Finally, they propose a method that determines the best ordering by first training uniformly with all orders and then allowing the model to settle on an efficient ordering.

Another work on WikiSQL is that of Xu et al. [91], who also address the issue of order. In their method, rather than decoding the WHERE clause as a sequence, they propose a sequence-to-set model that predicts the set of columns that are used in the WHERE clause. The columns are predicted independently using a set inclusion network with a sigmoid (rather than softmax) activation function. While this allows to remove possible conditioning on decoding order, it also introduces some possibly unwanted independence assumptions, specifically that the probability of decoding a column is independent of the other decoded columns in the WHERE clause. In addition, the approach relies on some assumptions that make it difficult to extend the approach to other tasks than WikiSQL (which has little variety in structure and generally has small queries).

Shi et al. [90] propose a sequence-to-action model that alleviates the effect of order during training by using a method with dynamic oracles [24]. The sequence-to-action model produces a sequence *actions* rather than tokens, where an action advances the parser state. Normally, under teacher forcing, the sequence of actions to reach the desired parser state corresponding to the desired query would be determined before training and kept fixed for every example. The idea behind dynamic oracles, which were first investigated by Goldberg and Nivre [24] for syntactic parsing, is to instead use a supervision method that encourages all valid actions for a certain state. This allows the exploration of alternative correct action sequences. Shi et al. [90] also extends the non-deterministic oracle with the `ANYCOL` token, which adds the option to produce a wildcard column token that matches any column. During training, the wildcard column token is provided as an alternative to the true column token in the supervision sequence if it can be unambiguously resolved to the true column using the condition value.

Closely related to these works is the work of Vlachos and Clark [162], who investigate semantic parsing in dialogue context. They propose a new dataset and investigate the use of DAgger [163] for training. The concrete algorithm they use proceeds as follows: (1) a trajectory is sampled using a mixture of expert policy and current policy, (2) for each step in the trajectory, all possible actions are scored and aggregated into a cost that is used adapt model parameters. In the first step, using a mixture of expert and current policy enables the method to explore sub-optimal states. In the second step, the possible actions are scored by first obtaining a complete output, which is done by using the mixed policy, and then scoring the completed output by comparing it with the gold standard output.

Finally, related to the following discussion on compositional generalization is the work of Guo et al. [16]. In this work, the authors point to the observation that for many query languages, such as SPARQL (used in CFQ), the order of certain subtrees in their query parse tree does not matter. For example, the order of triple patterns in SPARQL queries does not affect the meaning or the result of the queries. Thus, many linearizations of the same query are possible. Yet, normal training trains with only one linearization, which may not be the best possible for better generalization. The authors argue that "imposing additional ordering constraints increases learning complexity which makes the learning likely to overfit to bias ordering information in training distribution". To reduce overfitting to the spurious patterns due to a particular linearization order, they propose the hierarchical poset decoder (HPD), which exploits permutational invariances during training. The main idea is to decode posets (partially ordered subsets) rather than sequences. To decode posets, the authors propose a decoding method that learns to decode different traversal paths in the poset of a query. For their final model, the authors use a hierarchical poset decoder, which first decodes a poset sketch, then proposes different way to fill the placeholders in the sketch and finally, scores the many different traversal paths produced by the previous two steps. The final results on CFQ indicate that both the hierarchical decoding as well as poset decoding leads to improvements. This highlights that order in linearization can be an important aspect when training semantic parsers.

It should also be noted that training with multiple valid linearization orders is related to the issue of weak supervision discussed in Section 3.1.7: even though the output structure (the query) is given, there are multiple valid ways (linearizations) to achieve the same output. However, compared to weak supervision, there are no "spurious" linearizations like there are "spurious logical forms" that mean a different thing but produce the same output: all the linearizations mean exactly the same thing, just implemented in a different order. Nevertheless, the output structure can be treated as the output and the different possible linearizations as latent variables. Thus, methods like MML, hard EM and policy gradients can and have been applied.

3.3.3 Compositional Generalization

As most work in machine learning, semantic parsing and QA methods have predominantly been evaluated with test data that follows the same distribution as the training data. More formally, it is assumed that both the training examples $(x, y) \in \mathcal{D}_{\text{train}}$ and test examples $(x', y') \in \mathcal{D}_{\text{test}}$ are drawn from the same underlying joint distribution:

$$(x, y) \sim p_{\text{train}}(X, Y) \tag{3.8}$$

$$(x', y') \sim p_{\text{test}}(X, Y) \tag{3.9}$$

$$p_{\text{train}}(X, Y) \approx p_{\text{test}}(X, Y) \ . \tag{3.10}$$

Usually, after a data collection phase, test sets are created by simply randomly holding out a fraction of all available examples. Note that the underlying joint distributions $p_{\text{train}}(X, Y)$ and $p_{\text{train}}(X, Y)$ are unknown and can only be sampled from by collecting examples. Sometimes, more systematic splits are used that mimic a more realistic usage scenario. For example, WikiSQL [92] and Spider [164] split the data such that the tables used during testing have not been used during training.

The commonly used random split could be problematic in practice, since the data collection process may be biased and not produce data that are fully representative of the entire underlying data distribution. For example, the inputs can change over time (e.g. as the knowledge graph grows and new relations and entities get introduced), or use different underlying data (e.g. a new table for text2sql). If the data collection process is biased, the proposed methods may not work as well in practice as when evaluated on a test set that is distributed similarly to the training distribution. While it can be argued that the difference between deployment and test distributions is natural and should be solved by collecting more data and retraining, it is not a sustainable model because data collection can be expensive and would have to be repeated regularly. It would thus be more advantageous to develop methods (for modeling, training and/or data augmentation) that maximize performance under a dataset shift during testing. Different types of dataset shift can be identified, with covariate shift and prior probability shift being the most well-known. Covariate shift is the change in the distribution of independent variables (features), or more formally, when $p_{\text{train}}(y|x) = p_{\text{test}}(y|x)$ but $p_{\text{train}}(x) \neq p_{\text{test}}(x)$. Prior probability shift (a.k.a. label shift) is the change in the distribution of labels, or more formally, when $p_{\text{train}}(x|y) = p_{\text{test}}(x|y)$ but $p_{\text{train}}(y) \neq p_{\text{test}}(y)$.

One until recently overlooked, yet extremely important source of dataset shift in the context of semantic parsing is a **shift in the distributions of compounds**. Here, the concept of compounds refers to combinations of “atomic” elements (e.g. entities and relations) and can be generalized to many tasks. In semantic parsing, the ability to generalize to novel combinations is extremely important. Given only a few examples using a particular relation (e.g. “writerOf”), we would like it to generalize to other uses of that relation, regardless of the context it occurs in. Unfortunately, several recent works [13, 14, 165–167] arrived at the alarming finding that commonly used sequence-to-sequence models, such as RNN-based sequence-to-sequence models with attention, and transformers, generalize extremely poorly in such scenarios.

One of the first such works is the work of Lake and Baroni [13], who investigate compositional generalization using a new dataset (SCAN) for robot instruction. The dataset contains thousands[6] of automatically generated pairs oif natural langauge instructions and corresponding executable action sequence. The work investigates model performance on different splits aimed at testing the ability of networks to (1) generalize to novel combinations of tokens observed only in isolation during training (the JUMP and TURN LEFT settings) and (2) generalize to longer sequence lengths (the LENGTH setting). For the JUMP setting, the training set consists of the primitive example “jump” as well as all other primitive and composed examples (e.g. “run twice”) while the test set contains all composed examples with “jump” (e.g. “jump twice”). It has been observed that standard sequence-to-sequence models fail on the JUMP and LENGTH generalization challenges (accuracy below 10%) while it performs well (near 100%) on a random split of the data.

The recent work of [14] proposes a different dataset, **CFQ**, which contains tens of thousands of automatically generated question-SPARQL query pairs and provides maximum compound divergence (MCD) splits. The MCD splits are generated such that the distributions over compounds (phrases combining atomic elements) are maximally different between the train and test sets while the distributions over the atomic elements (entities, relations, question patterns) are kept similar. The authors also provide MCD splits for the SCAN dataset. Experiments using standard neural seq2seq models reveal that while the random splits achieve near-perfect accuracy, the MCD splits suffer dramatic losses in performance ($< 20\%$ accuracy for CFQ’s MCD splits and $< 10\%$ for SCAN’s MCD splits).

[6] Exact number differs per split.

Improving Compositional Generalization

Several methods [16, 108, 167–179] have recently been proposed that aim to improve the compositional generalization w.r.t. normal sequence-to-sequence models.

Perhaps one of the simplest methods is the work of [168], nicknamed GECA [7] who proposes an extremely simple data augmentation method that nevertheless obtains impressive improvements on SCAN and some improvements on query-based splits of standard semantic parsing datasets. The main idea behind the approach is to identify a template and generate new additional examples by recombining the fragments used across different examples with the same template in another template. For example, suppose the training set contains the following three examples:

1. The cat followed the dog.
2. The cat taunted the dog.
3. The dog followed the horse.

We can then extract the following template: "... cat ... dog" and the following fragments: "The ... followed the ...." and "The ... taunted the ....". Because the first fragment also occurs in the third example, we can extract a new template "... dog ... horse" and use it with the other fragment to produce a new example "The dog taunted the horse". Despite being extremely simple and not necessarily syntactically or semantically correct, their approach raises the accuracy of the JUMP task on SCAN from nearly 0% to 87% and appears to provide some improvements on the compositional splits on the GeoQuery dataset.

Another data augmentation method is that of Akyürek et al. [174], who propose a *learned* data augmentation technique. In contrast to GECA [168], which is a fixed data augmentation scheme, Akyürek et al. [174] train a model that generates new examples and use these examples to train the desired sequence-to-sequence model. The data generation model is a prototype-based data recombination model. It takes one or more existing examples and is trained to generate another example by reusing parts of the given example. Thus, it models the probability $p(x^*|x, \theta)$ of a new example x^* given some real example x. While previous work on prototype-based decoding used similar methods for producing outputs for the task, Akyürek et al. [174] postulate that this method could be even more useful for data augmentation to improve the compositional generalization when training semantic parsers. Akyürek et al. [174] also find that improvement can be obtained by using a multi-prototype model that takes multiple prototypes, and models, for example for a two-prototype case the probability $p(x^*|x_1, x_2, \theta)$. The authors generate new data by sampling such examples that contain rare phenomena. The improvements on SCAN are comparable to GECA.

Guo et al. [179] investigate the effect of iterative back-translation (IBT) in the context of compositional generalization in semantic parsing. The approach consists of training both NL2FL and FL2NL models on the original paired data, and iteratively produce more synthetic data from monolingual data in both directions to train the models with. IBT obtains significant improvements using standard seq2seq models on both SCAN and CFQ.

Furrer et al. [173] investigate how pre-trained sequence-to-sequence transformers affect compositional generalization when fine-tuned. They use T5 [61], a pre-trained text-to-text transformer with relative positions, and fine-tune it on the SCAN and CFQ tasks. They show that the use of

[7] Good Enough Compositional Augmentation

the pre-trained transformer is beneficial for compositional generalization, and that larger pre-trained models appear to bring more improvement. This is an interesting finding, especially considering that the model or the training are not specialized.

The works of Li et al. [175] and Russin et al. [176] follow a similar approach. The usual sequence-to-sequence model (they use RNNs) is changed such that there are two encodings of the input: (1) a normal bidirection RNN encoding of the input that is used as a "syntactic" model and (2) a simple embedding, which serves as a "semantic" model. The decoder then uses the first to compute attention weights and uses the second to compute the attention-based summary that is used to generate output tokens. Li et al. [175] also propose to use entropy regularization.

Liu et al. [177] propose a method that solves SCAN's challenges very well, including the length generalization challenge, which was largely ignored by other works. The proposed method relies on two collaborating components, (1) the "Composer", which has to recognize which spans of the inputs should be "solved" and the (2) "Solver" that translates the recognized solvable spans into expressions. After the Solver translated the expression, the corresponding recognized span in the input is replaced with a variable and the variable-expression mapping is also stored in a symbolic memory. These variables can be decoded by the Solver in later steps and will be replaced with the corresponding expressions in the memory. The decoding process is thus essentially an iterative bottom-up recognition-translation process that implicitly builds a phrase parse tree. The process repeats until the entire input is replaced with a single variable and the return value is the value of the memory for that variable. For example, the phrase "jump after walk twice" might be resolved in the following way:

1. **jump** after walk twice, []
2. $x after **walk** twice, [$x = JUMP]
3. $x after **$y twice**, [$x = JUMP, $y = WALK]
4. **$x after $z**, [$x = JUMP, $z = $y $y = WALK WALK]
5. $o, [$o = $z $x = WALK WALK JUMP]

In the above sequence, bold spans are the spans that are recognized by the Composer and the key-value mappings in [] are the memory contents. **Training** is not trivial in this approach since there are latent discrete variables. The authors use hierarchical reinforcement learning (Reinforce with baseline) using a curriculum. The authors report that "the training time for a single run is about 20-25 hours". A Tesla P100 GPU is used. To put things in context, the dataset contains less than 10 unique words, and training a normal sequence-to-sequence model on the same data takes about 10-20 minutes, depending on the model, on a computer with an RTX 2070 Super GPU.

Herzig and Berant [108] propose a method that learns to produce a span tree over the input sentence in order to parse it into a logical form. Here, every span is annotated with either a constant from the underlying KG, a composition or a null category. During prediction, the distributions over spans are predicted independently for each span, and the most probable tree is found using a CKY [180–182] algorithm. The model used is simply a BERT encoder that represents a span by the representation vectors of the span's start and end positions. During training, since the gold span trees are unknown, a hard EM-like [107] method is used that learns to find the most probable span tree to produce the given logical form (see also Section 3.1.7).

One of the most recent methods is that of Herzig et al. [178]. They investigate the use of intermediate representations that reduce the discrepancy between natural language and logical forms. The authors improve the performance of the pre-trained T5 model by replacing the one-step $x \longrightarrow y$ decoder with a two-step decoder that first translates the input x to an intermediate representation z and then translates z to y. The $x \longrightarrow z$ step is achieved using a standard seq2seq model. The second step depends on whether the intermediate representation is reversible or lossy. In the first case, z can simply be mapped to y deterministically. In the second, another seq2seq model is trained for the mapping. The authors propose the following principles for designing the intermediate representation: (1) minimize mismatch between formal and natural language (e.g. removing tokens from y that can not be aligned to the input x) and (2) increase structural similarities among the z's of different examples (e.g. anonymizing entities).

CHAPTER 4

Word- and character-level representations in question answering over knowledge graphs

"Hodor!"

– Hodor, from *A Song of Ice and Fire*
by G.R.R. Martin

In this chapter, we focus on the problem of out-of-vocabulary (OOV) words in question answering over knowledge graphs, which can also be seen as a particular type of out-of-distribution generalization challenge: because the OOV words have not been observed during training, the test data are sampled from a distribution different from the training data distribution. It is also similar to zero-shot generalization described by Gu et al. [10]. To tackle this challenge, in this chapter, we investigate *word- and character-level* representations, where instead of learning a unique parameter vector for every possible output token, we instead build representations of output tokens based on other information available about them, such as their surface form (how they are expressed in natural language). In this way, we build representations on a "sub-symbolic" level with fewer possible unique tokens, such that these new tokens are more likely to be observed during training. This is similar to character-level and other sub-word [36, 37] representations for words.

We perform our investigation as part of our work on *answering simple factoid questions*, which are questions whose queries contain only a single entity and predicate, using knowledge from the Freebase knowledge graph. To this end, we use the SimpleQuestions dataset.

This chapter is based on the following publication, which has been updated with minor corrections and updated evaluation results:

1. **Denis Lukovnikov**, Asja Fischer, Jens Lehmann, Sören Auer. *Neural network-based question answering over knowledge graphs on word and character level.* In Proceedings of the 26th International World Wide Web Conference (WWW 2017). 2017.

While the work described in this chapter presents an entire approach for answering simple factoid questions over knowledge graphs, the specific focus of its contributions within this thesis lies in investigating word- and character-level representations for KGQA. This corresponds to Contribution 1, which addresses the following research question:

RQ1: Does combining word- and character-level representations improve accuracy in KGQA?

In Chapter 5, we revisit the task of answering simple questions over knowledge graphs and the SimpleQuestions dataset, albeit with a different research question (RQ2: transfer learning).

The rest of the chapter is organized as follows: We further motivate and elaborate on the problem and list specific contributions in Section 4.1. We present the taken approach in Section 4.2. We evaluate our implementation in Section 4.3 and provide a detailed analysis and discussion in Section 4.4. We provide an extensive discussion of related work in Section 3.2.1, discussing it in the context of this work in Section 4.5, and conclude in Section 4.6 with a conclusion and an outlook on future work.

4.1 Introduction

While question answering over knowledge graphs and other data sources has attracted a lot of research attention, it can still be challenging. Some of the challenges faced by QA systems are:

- *Lexical gap* – surface forms for relations expressed in a question can be quite different from those used in the KG,
- *Ambiguity* – the same word is used for different entities, such as president of a company or a country,
- *Unknown knowledge boundaries* – QA systems can often hardly decide whether a certain question is answerable at all give a certain knowledge base.

The traditional paradigm for QA approaches is (1) to construct complex natural language processing (NLP) pipelines (which can result in error propagation along the pipeline) and (2) require manual adaption to new domains. *AskNow* [34], for example, uses a pipeline comprising a POS tagger, template-fitting, relation extraction, token merging and entity mapping. The *Qanary* methodology [183] follows an extreme realization of this approach by enabling a fine-grained integration, cooperation and competition of pipeline components.

In this work, we follow a fundamentally different approach that mitigates the main disadvantages of the pipeline method. We develop an end-to-end neural network approach that generates a solution in a single process and thus avoids (1) complex NLP pipeline constructions, (2) error propagation, and (3) can be retrained or reused for a different domain. All decisions can be handled together in an integrated fashion in order to leave the learning algorithm the freedom to decide how to use the given information.

In this work, we restrict ourselves to simple questions, which only require the retrieval of a single fact to be answered. This is the setup of the SimpleQuestions task presented by Bordes et al. [121]. The task of QA over KG for simple questions (Simple QA) can be put more formally as follows. Let $\mathcal{G} = \{(s_i, p_i, o_i)\}$ be a background knowledge graph represented as a set of triples, where s_i represents a subject entity, p_i a predicate (also denoted as relation), and o_i an object entity. The task of Simple QA is then: Given a natural language question represented as a sequence of words $q = \{w_1, \dots, w_T\}$, find the set of triples $(\hat{s}, \hat{p}, \hat{o}) \in \mathcal{G}$ such that their objects $\hat{o}$ is the intended answer for question q. This task can be reformulated to finding the right subject $\hat{s}$ and predicate $\hat{p}$ that question q refers to and which characterize the set of triples in $\mathcal{G}$ that contain the answer to q. See also Section 3.2.1 for more details on this task and a discussion of existing approaches.

For example, finding the answer to the question "*What cyclone affected Hainan?*" requires finding the FREEBASE entity `m.0166br` representing the Chinese province Hainan and the relation `fb:meteorology/affected_area/cyclone`. The entity representing Hainan has the label "*Hainan*" and a notable type with label "*Chinese province*". In the rest of this chapter, we use this question as a running example to illustrate the proposed approach.

We develop a neural matching model that takes basic textual information about entities and predicates and uses some training and prediction enhancements to produce results competitive to the state-of-the-art. Even though the presented approach is restricted to simple questions, this work can serve as a foundation for the future development of more advanced neural QA approaches that can handle more complex questions. Compared to recent existing approaches [124, 125, 184], our model is relatively simple in that it does not employ attention mechanisms or separate segmentation models and achieves state-of-the-art results when compared to end-to-end models. A particular innovation of our approach in this context is that we use both character- and word-level information from the question for both entity and predicate prediction. Character-level modelling of questions (or entity mentions) and entities has been shown advantageous for this task because of its out-of-vocabulary (OOV) word handling abilities [125, 184]. However, the downside of purely character-level models is their inability to exploit word-level semantics (as captured in word embeddings [40]), which are useful for modelling questions, predicates and types. Our model combines the OOV-related advantages of character-level models and the rich semantics of word-level models. In this respect, our work is close to that of Yin et al. [125], with the difference that our approach employs a single question representation network and is less specific for simple questions since it does not split the question into mention-pattern pairs.

In summary, our main contributions are as follows:

- a simple and portable neural match-based model for answering simple questions, that
  (a) merges word- and character-level representations of words modelling of the question to exploit the advantages of both.
  (b) learns to generate knowledge base-independent representations of entities and relations that are built only from textual information associated with the entities or relations.
- a custom negative sampling procedure.
- an extensive evaluation of our method and a detailed root cause analysis for 250 questions.

4.2 Approach

The problem of single relation question answering could be divided into the following sub-tasks:

1. *segmentation* of q, i.e. finding the sub-sequences of words $\mathcal{M}_s$ and $\mathcal{M}_p \subset \{w_1, \ldots, w_T\}$, which refer to a subject and a predicate respectively,
2. *mapping* the chosen sub-sequences to an entity resp. a relation in the graph, i.e. finding s_g and p_g to which the sub-sequences $\mathcal{M}_s$ and $\mathcal{M}_p$ are referring to. Mapping to entities is known as *entity linking* whereas mapping to relations is referred to *relation prediction*.

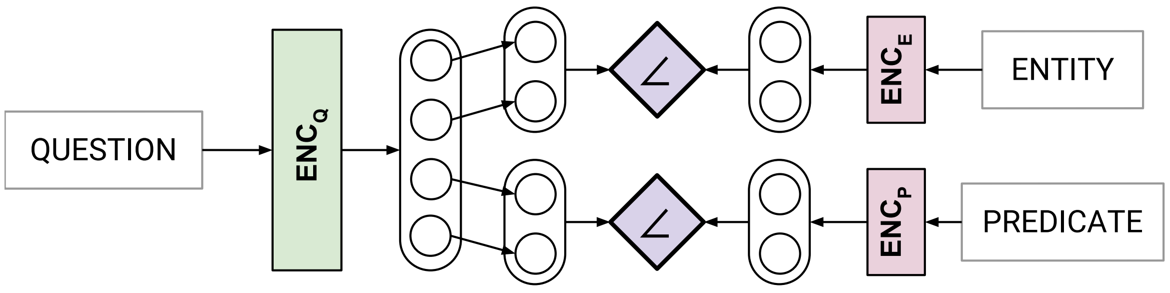


Figure 4.1: Visualization of the whole model. The question is encoded using the $\mathtt{ENC_Q}$ network, producing a question representation vector. Entity and predicate representations are produced by $\mathtt{ENC_E}$ resp. $\mathtt{ENC_P}$. Question-entity and question-predicate pairs are scored using a cosine similarity between their representing vectors.

Instead of explicitly finding solutions for the different sub-tasks separately, our approach considers them together without explicitly looking for a solution to either and instead directly generates a solution for the full problem. We do not employ any explicit relation lexicons (e.g. DBpedia Lemon [185]) or other external information (such as OpenIE extractions), instead letting the network learn the lexicon from examples.

Our approach relies on *recurrent neural networks* (RNN) since we are dealing with sequential data. RNNs take into account the order of words, which is important since the order of words significantly contributes to the meaning of a sentence or phrase.

The model is further elaborated in the next section (Section 4.2.1), followed by a description of the prediction process (Section 4.2.2) and the training process (Section 4.2.3).

4.2.1 Model description

From a broad perspective, our model (see Figure 4.1) is a matching function: given a question q and sets of candidate subject entities and relations, $C_s = \{s_1, \ldots, s_n\}$ and $C_p = \{p_1, \ldots, p_m\}$ respectively, it returns the subject and predicate that matches the question best. To do so, it

(1) maps a question q to vector representation $r_q = (r_q^s, r_q^r)^T$, where r_q^s and r_q^r are subject and the relation sprecific encodings of the question respectively,

(2) maps each candidate subject $s_i \in C_s$ to a vector representation r_{s_i},

(3) maps each candidate predicate $p_j \in C_p$ to a vector representation r_{p_j},

(4) and computes scores $S_s(q, s_i)$ and $S_p(q, p_j)$ for each pair $r_q^s, r_{s_i}, i = 1, \ldots, n$, and $r_q^p, r_{p_j}, j = 1, \ldots, m$.

Based on these scores the final prediction is $(\hat{s}, \hat{p})$, with

$$\hat{s} = \text{argmax}_{s_i \in C_s} S_s(q, s_i) \quad , \tag{4.1}$$

$$\hat{p} = \text{argmax}_{p_j \in C_p} S_p(q, p_j) \quad . \tag{4.2}$$

Steps (1)-(3) heavily rely on RNNs with Gated Recurrent Units (GRUs) [45], which are described in Section 2.2.2. In the following subsections, the four parts of our model are described in detail.

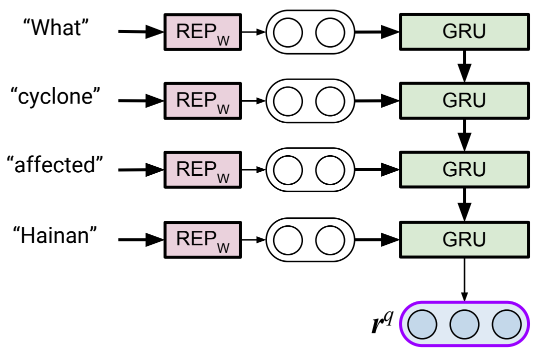


Figure 4.2: Question encoding network $\mathtt{ENC_Q}$. Each word is represented by a vector using $\mathtt{REP_W}$. The sequence of word vectors is encoded using a GRU.

Representing the question

The mapping of a question $q = \{w_1, \ldots, w_T\}$ to its subject and predicate related vector representations $\mathbf{r}_q^s$ and $\mathbf{r}_q^r$, respectively, is done using a single-layered unidirectional GRU based encoder network. We call this part of the model the *question encoder* $\mathtt{ENC_Q}$

$$\mathbf{r}_q = \begin{bmatrix} \mathbf{r}_q^s \\ \mathbf{r}_q^p \end{bmatrix} = \mathtt{ENC_Q}(\{w_1, \ldots, w_T\}) \ . \tag{4.3}$$

The question encoder $\mathtt{ENC_Q}$ first uses the word representation function $\mathtt{REP_W}(w_t)$ to generate vector representations for all words $w_t, t = 1, \ldots, T$ (as described in the next paragraph), which are subsequently fed to the RNN until all words have been seen. Starting with the initial hidden state $\mathbf{h}_0$, the GRU of the question encoder RNN iteratively updates its hidden state $\mathbf{h}_t$ after processing each word according to Equations (2.6) to (2.9), where the word representation vector $\mathtt{REP_W}(w_t)$ is fed as input to the GRU (i.e. $\mathbf{x}_t = \mathtt{REP_W}(w_t)$). The final hidden state $\mathbf{h}_T$ (produced after processing the last word represented by $\mathtt{REP_W}(w_T)$) is returned by $\mathtt{ENC_Q}$ as the representation of question q.

The question encoder is visualized in Figure 4.2.

Word representation In the following we descibe how we generate the vector representations of the words $w_1, \ldots, w_T$. In order to exploit both word- and character-level information of the question, we use a "nested" word- and character-level aproach concatenating the pre-trained embedding of a word with an RNN-based encoding on character level.

As word embeddings, we use *GloVe* [40] vectors provided on the GloVe website[1]. Such pre-trained word embeddings implicitly incorporate word semantics inferred from a large text corpus based on the distributional semantics hypothesis [186]. This hypothesis can be phrased as "words with similar meanings occur in similar contexts", which in our case translates to similar vectors for words with similar meanings. See Section 2.4.1 for more on pre-trained word embeddings. Using such pretrained word embeddings allows us to better handle synonyms and find better matches between words in the question and subject labels or predicate URI's. In addition, during testing, it allows to handle words

[1] http://nlp.stanford.edu/projects/glove/

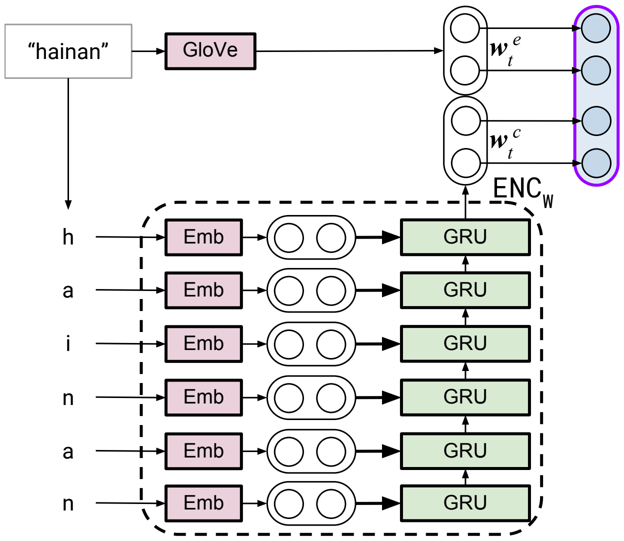


Figure 4.3: Word representation network $\texttt{REP}_\texttt{W}$ with example. The word is considered as a sequence of character and fed to $\texttt{ENC}_\texttt{W}$, where each character is embedded and the sequence of character vectors is encoded using a GRU to produce a character-level encoding of that word. This is concatenated to the word embedding (we use GloVe) to produce the complete word representation.

that have not been seen during training.

The word embedding of w_t resulting in the d_{w^e}-dimensional vector representation $\mathbf{w}_t^e$ can be formally described as follows

$$\mathbf{w}_t^e = \mathbf{W}_g^\top \mathbf{v}_t \quad , \tag{4.4}$$

where $\mathbf{W}_g \in \mathbb{R}^{|V_g| \times d_{w^e}}$ is the provided pretrained word embedding matrix for a vocabulary of size $|V_g|$ (GloVe covers 400k words), and $\mathbf{v}_t$ is the one-hot vector representation of w_t. Since the coverage of word embeddings is limited, many words appearing in the questions (especially those that are part of a reference to a particular subject entity, e.g. the last name "*Golfis*" in the question "*What city was Alex Golfis born in*") are not contained in the vocabulary of the pre-trained embeddings. In such cases (20.8% and 14.5% of unique words in the train resp. test questions), we set the word embedding to the zero vector.

The encoding of the word $w_t = \{w_t^1, \dots w_t^K\}$ on character-level is based on a RNN encoder:

$$w_t^c = \texttt{ENC}_\texttt{W}(\{w_t^1, \dots w_t^K\}) \quad , \tag{4.5}$$

Inside $\texttt{ENC}_\texttt{W}$, the characters $w_t^k, k = 1, \dots, K$ are first embedded by

$$\mathbf{c}_t^k = \mathbf{W}_c^\top \mathbf{v}_t^k \quad , \tag{4.6}$$

with character embedding matrix $\mathbf{W}_c \in \mathbb{R}^{|V_c| \times d_{c^e}}$ (for $|V_c|$ characters) learned during training, and $\mathbf{v}_t^k$ the one-hot vector representation of the character w_t^k. Then we feed the sequence of character vectors $\{\mathbf{c}_t^1, \dots, \mathbf{c}_t^K\}$ to a single-layered unidirectional GRU network and take its final state as the character-level word encoding $\mathbf{w}_c$.

The added character-level encoding provides information necessary for matching question words with entity labels, in addition to providing distinguishable representations for OOV words. This approach is similar to the *char2word* model proposed by Ling et al. [187] with the difference that we use a unidirectional GRU network.

Finally, to get the vector representation of a word w_t, the word embedding $\mathbf{w}_t^e$ and character-level encoding $\mathbf{w}_t^c$ are concatenated:

$$\mathtt{REP_W}(w_t) = \begin{bmatrix} \mathbf{w}_t^e \\ \mathbf{w}_t^c \end{bmatrix} \ . \tag{4.7}$$

The whole word representation network is illustrated in Figure 4.3.

Representing the subject

We use both the entity label and the type label of the entities in the knowledge graph to build subject representations. Entity labels are encoded on the level of characters because of the high prevalence of OOV words and their importance for entity labels. On the other hand, OOV words are rather rare in type labels and thus type labels are encoded on word level.

For Freebase entities, we extract entity labels using the `type.object.name` properties of entities and type labels of entities by first getting the `common.topic.notable_types` of entities[2] and then taking the `type.object.name` value of the types.

The character-level entity label encoding $\mathbf{s}^l$ and word-level type label encoding $\mathbf{s}^t$ are concatenated to produce the subject representation vector

$$\mathbf{s}^l = \mathtt{ENC_{SL}}(\{c_s^1, c_s^2, \ldots\}) \ , \tag{4.8a}$$

$$\mathbf{s}^t = \mathtt{ENC_{ST}}(\{w_t^1, w_t^2, \ldots\}) \ , \tag{4.8b}$$

$$\mathbf{r}_s = \begin{bmatrix} \mathbf{s}^l \\ \mathbf{s}^t \end{bmatrix} \ , \tag{4.8c}$$

where $\mathtt{ENC_{SL}}$ is the character-level encoder of the subject entity label and $\mathtt{ENC_{ST}}$ is the word-level type label encoder. The label characters and type label words, respectively, are first embedded (following Equation 4.6 and Equation 4.4, respectively) and the embedding vectors are fed to the respective encoding RNNs. Both $\mathtt{ENC_{SL}}$ and $\mathtt{ENC_{ST}}$ correspond to single-layer unidirectional GRU-based RNNs and take their final hidden state as the entity label encoding $\mathbf{s}^l$ and type label encoding $\mathbf{s}^t$, respectively. The subject representation network is visualized in Figure 4.4.

This method of building subject representations is similar to the method proposed by Sun et al. [188] who focus on entity linking and CNNs for word-level entity name encoding (instead of using RNNs for character level based encodings like our model, which allows to handle OOV words) and word-level entity type name encoding, followed by an additional layer that merges the two (where we simply concatenate both representations).

Representing the predicate

We use the predicate URI's provided by the KG to build latent vector representations of the predicates. The predicate URI is first split into words $w_p^1, w_p^2, \ldots$, each word is embedded (as described by

[2] The notable types property provide the single, most characteristic type for that entity. However, using all types of the entity (e.g. concatenating their labels) could be interesting as well, which we leave for future work.

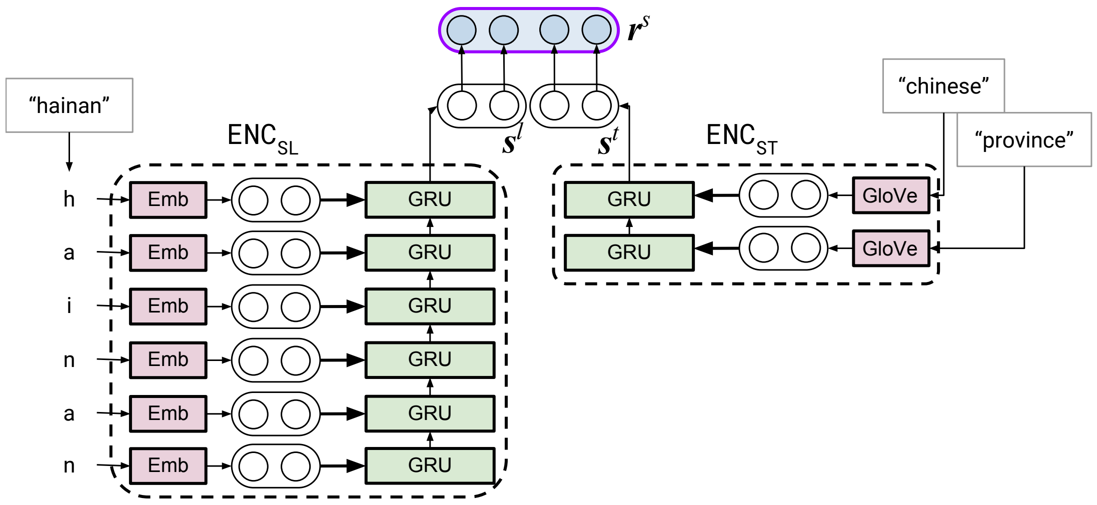


Figure 4.4: Entity encoder with example. The entity label is encoded on character level ($\texttt{ENC}_{\texttt{SL}}$) and the subject type label is encoded on word level ($\texttt{ENC}_{\texttt{ST}}$). The two are concatenated to produce the subject vector.

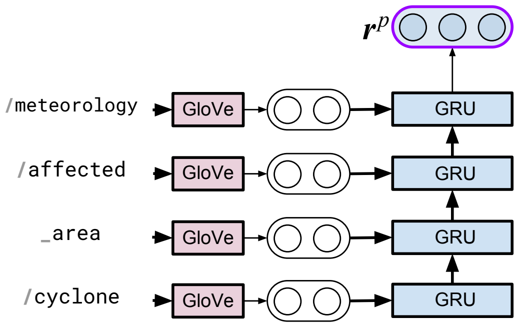


Figure 4.5: Predicate encoder network $\texttt{ENC}_{\texttt{R}}$ with example. The predicate URI is split into words and encoded on word level using GloVe embedding.

Equation 4.4), and then the word embeddings are fed into a single-layer word-level GRU-based encoder $\texttt{ENC}_{\texttt{R}}$ that takes the final state of its RNN as the representation of the predicate URI, that is

$$\mathbf{r}_p = \texttt{ENC}_{\texttt{R}}(\{w_p^1, w_p^2, \ldots\}) \ . \tag{4.9}$$

The relation encoding network is visualized in Figure 4.5.

Matching scores

Given the question encoding vector $\mathbf{r}_q = (\mathbf{r}_q^s, \mathbf{r}_q^p)$, the latent vector representation $\mathbf{r}_p$ of the relation, and the latent representation $\mathbf{r}_s$ of the subject entity, we compute two matching scores: one between the question and subject entity and one between the question and predicate, as follows:

$$S_s(q, s) = \cos(\mathbf{r}_q^s, \mathbf{r}_s) \tag{4.10a}$$
$$S_p(q, p) = \cos(\mathbf{r}_q^p, \mathbf{r}_p) \ , \tag{4.10b}$$

where cos is the cosine similarity given by $\cos(\mathbf{a}, \mathbf{b}) = \frac{(\mathbf{a} \cdot \mathbf{b})}{|\mathbf{a}||\mathbf{b}|}$.

4.2.2 Prediction

With the model described in the previous section we are now able to compute scores for question-subject and question-predicate pairs (Equation (4.10)). Using these scoring functions, we can solve the task of finding the right subject-predicate pair (s_g, p_g) (i.e. , retrieving triples $(s_g, p_g, o_i) \in \mathcal{G}$ such that the set of objects in these triples constitutes the answer to question q) by picking the best scoring subject entity and predicate given a question according to Equations (4.1) and (4.2), respectively. However, computing scores for all entities and predicates in the KG would be very costly and introduces a lot of noise. So we only consider small subsets of entities and predicates which are most likely to be the correct ones for a given question. We refer to these sets as *candidate subjects C_s* and *candidate predicates C_p*, respectively.

We explore two options for generating predictions. The first proceeds in the following four steps:

1. generate subject candidates C_s (as described in Section 4.2.2).
2. score each entity in C_s and take top-scored one as prediction (Equations (4.1) and (4.10a))
3. generate predicate candidates C_p based on the *top subject entity* (see Section 4.2.2)
4. score each predicate in C_p and take top-scored one as prediction (Equations (4.2) and (4.10b))

The second option allows to correct wrong subject predictions by an added pruning step (which we refer to as *subject pruning* in the following):

1. generate subject candidates C_s (see Section 4.2.2).
2. score each entity in C_s (Equation (4.10a))
3. generate predicate candidates C_p based on the *top subject label* (see Section 4.2.2)
4. score each predicate in C_p and take top-scored one (Equations (4.2) and (4.10b))
5. prune C_s based on the predicted predicate as described in Section 4.2.2 and take the top-scoring entity from the filtered C_s as subject prediction

We now describe how the *candidate subjects* and *candidate predicates* are constructed and pruned in detail.

Generation of candidate subjects

First, we collect the lowercased English entity labels from the KG (for Freebase, we use the `type.object.name` property). To generate the set of candidate subjects C_s, all word n-grams of size 1 to L contained in question q are retrieved, filtered and used for searching matching entities using the collected labels according to the following rules:

- An entity whose label exactly matches a n-gram is added to the set of candidates C_s.
- An n-gram that is fully contained inside a bigger n-gram that matched at least one entity label is discarded, unless the bigger n-gram starts with one of the following words: “*the*”, “*a*”, “*an*”, “*of*”, “*on*”, “*at*”, “*by*” .

- In case an exact match is not found for an n-gram, we search for entities whose label has an edit distance with the n-gram of at most 1 and add them to C_s.
- If there are several entities matching a n-gram, the entities are ranked by the number of triples in the KG in which they play the role of the subject. Only the m entities with highest rank are added to the candidate set (in our experiments $m \in \{5, 10, 400\}$.)

Note, that the candidate generation can be efficiently implemented using information retrieval libraries such as Apache Lucene.

Pruning subjects based on predicted relation When generating the predicate candidates based on the *top subject label*, we perform additional pruning of C_s: We remove all entities that *do not* appear as subject in *any* of the triples in $\mathcal{G}$ having the predicted relation as predicate.

Generation of candidate predicates

For generating the candidate set of predicates C_p we explore two approaches, both depending on C_s and the highest scored subject $\hat{s} \in C_s$

- *Top subject entity based*: only predicates occurring in triples from $\mathcal{G}$ which have $\hat{s}$ as subject are added to C_p. This way of generating C_p is highly dependent on subject prediction. If the wrong subject has been chosen, it is highly probable the right predicate is not in C_p.
- *Top subject label based*: all subject candidates from C_s whose label is the same as that of the top-ranking candidate $\hat{s}$ are considered and all predicates that occur in triples having one of those entities as subject are added to C_p.

4.2.3 Training

The model described above is trained with a ranking training approach, which drives the model to output a high score for question-entity and question-predicate pairs contained in the training set while producing a lower score for implausible pairs. The loss function minimized during training is given by

$$\sum_{(q,s^+,p^+)\in\mathcal{D}} \Big(\max\left(0, S_s(q,s^-) - S_s(q,s^+) + \gamma\right) + \max\left(0, S_p(q,p^-) - S_p(q,p^+) + \gamma\right)\Big) \ . \quad (4.11a)$$

That is, both the pair of question and true subject (q, s^+) and the pair of question and true predicate (q, p^+) are forced to have a score of at least margin γ higher than the score of corrupted pairs (q, s^-) and (q, p^-) consisting of the same question sentence and a false subject or relation, respectively. The false subjects s^- and relations p^- are randomly sampled at each epoch as described in the next paragraph. We also experimented with a binary softmax loss, which did not result in improvements and was slightly slower during training.

Negative sampling

At each iteration, a corrupted subject-predicate pair (s^-, p^-) is generated based on the true subject-predicate pair (s^+, p^+) from the training set.

In our experiments, we observed that the way we generate negative samples has a significant influence on the results. We developed the following negative sample generation methodology, which tries to expose the model to conditions close to those it will encounter during prediction.

Subject corruption To corrupt (s^+, p^+), a false subject s^- can be sampled from the space of all entities in the KG $\mathcal{G}$. However, we opt for a "close" corruption sampling scenario that forces the model to learn to distinguish the correct entity from more plausible wrong entities that are collected using the same candidate generation procedure as the one used for prediction.

In this "close" corruption scenario, the corrupted entity s^- for the correct entity s^+ is sampled from the set $\mathcal{X}^-_{s^+}$, which is constructed as

$$\mathcal{X}^-_{s^+} = C^i_s \setminus \{s^+\} \ , \tag{4.12}$$

where C^i_s is the subject candidate set for i-th question q_i in the training set. This way, s^+ can be replaced by any entity that is a subject candidate for the same question.

At each training epoch, we randomly sample from this set $\mathcal{X}^-_{s^+}$ of false close candidates unless its size is smaller than 5, in which case we sample randomly from the whole set of entities in $\mathcal{G}$.

Predicate corruption For predicate corruption, we first take the set $\mathcal{X}^-_{p^+}$ of predicates that occur in triples in $\mathcal{G}$ having s^+ as subject. With probability $P_{\mathcal{X}^-_{p^+}}$ with

$$P_{\mathcal{X}} = \mathsf{tanh}(\mathsf{log}(|\mathcal{X}| + 1)/3) \ , \tag{4.13}$$

(which is higher as larger $\mathcal{X}$ is) we uniformly draw a random sample from this set. Otherwise, we uniformly draw a random sample from the set $\hat{\mathcal{X}}^-_{p^+}$ of predicates whose URI's have at least one word in common with the URI of the true predicate p^+ with probability $P_{\hat{\mathcal{X}}^-_{p^+}}$ or sample uniformly from the set of all predicates in $\mathcal{G}$ with probability $1 - P_{\hat{\mathcal{X}}^-_{p^+}}$. This way, preference is given to outgoing relations of the correct subject but if the set is too small (which would result in the same few predicates being used throughout the whole training process), we draw more samples from the set of similar predicates or the set of all predicates in $\mathcal{G}$.

Training settings

The loss is minimized using the AdaGrad optimizer [189] with learning rate 0.1 in a mini-batch setting with batch size 100. A margin of 0.5 gave the best results in our experiments[3]. We train our model for 50 epochs[4]. We use a 400-dimensional question encoding, 200-dimensional entity and predicate representations and 100-dimensional character-level word representations. (Training with a 600-dimensional question encoding and 300-dimensional entity and predicate representations did not increase performance.) We use 100-dimensional GloVe word embeddings and 50-dimensional

[3] We also tried margins of 0.25 and 1.0 but did not observe improvement.

[4] We observed that after about only 15 epochs our model already reaches decent performance on the validation set. Afterwards the accuracy continued to increase slowly, starting to stagnate around 50 epochs. We did not observe overfitting when training for at least 100 epochs.

character embeddings. A more thorough investigation of the hyperparameter space was not possible due to the long training time and resource constraints.

Training data

We train on SimpleQuestions [121], a dataset consisting of over 100.000 simple questions and the corresponding Freebase triples providing the answer. We use training, validation, and test splits as provided in the dataset, containing 75.910, 10.845, and 21.687 questions, respectively. We work against the FB2M subset of Freebase, as provided with the SimpleQuestions data, which contains approximately 2 million entities and 6701 relations. The number of questions referring to relations that have not been seen during training is negligible (161 questions with 148 unique relations). On the other hand, 78.5% of the 19406 distinct subject entities in the test set are unseen during training.

4.2.4 Technical details

We implemented our approach[5] using Theano[6], Lasagne[7], and NLTK[8]. Training takes approximately two days on a single Titan X GPU for 50 epochs with the reported hyperparameters.

4.3 Evaluation

We evaluate our method on the provided test portion of the SimpleQuestions dataset which contains $N = 21.687$ questions and the corresponding triples. For each question we follow the procedure described in Section 4.2.2 to find the best-scoring subject-predicate pairs, resulting in $(\hat{s}_i, \hat{p}_i), i = 1, \ldots N$. Comparing those to the right subject-predicate pairs $(s_i, p_i), i = 1, \ldots N$ provided by the testset the accuracy is computed as

$$\frac{\sum_{i=1}^{N} \mathbf{1}_{[(\hat{s}_i,\hat{p}_i)=(s_i,p_i)]}}{N} , \tag{4.14}$$

where $\mathbf{1}_{[\cdot]}$ is the indicator function (that is, $\mathbf{1}_{[a=b]} = 0$ if $a \neq b$ and $\mathbf{1}_{[a=b]} = 1$ if $a = b$). In the following subsections, we discuss our obtained results in different settings and compare to existing approaches.

4.3.1 Influence of candidate generation

Table 4.1 shows the recall of the different candidate generation settings for subjects over the test set, where recall is computed as the fraction of questions for which the set of generated subject candidates includes the right subject entity. The numbers indicate that including partial matches (matches of n-grams with edit distance 1) results in a recall increase of up to 2%.

Table 4.2 shows test accuracy for different candidate generation and pruning settings. Test accuracy in the FB2M setting is reported for different numbers of subject entity candidates and with or without

[5] `http://github.com/WDAqua/teafacto`
[6] `http://deeplearning.net/software/theano/`
[7] `https://github.com/Lasagne/Lasagne`
[8] `http://www.nltk.org/`

# Candidates	Partial matches	Recall %
5	no	84.8
10	no	87.3
400	no	91.8
5	yes	85.4
10	yes	88.4
400	yes	93.7

Table 4.1: Candidate generation recall for subjects over the test set. Recall is defined as the percentage of test questions for which the expected entity is retrieved during candidate generation.

# Candidates	Subject pruning	Accuracy %
5	no	72.3
10	no	72.5
400	no	73.5
5	yes	73.2
10	yes	73.5
400	yes	74.8

Table 4.2: Test accuracy for different candidate generation settings.

	5	10	400
Accuracy (%)	5.23	4.72	4.03

Table 4.3: Test accuracy of random choice model, for 5, 10 resp. 400 subject entity candidates.

subject pruning after relation ranking. As can be seen in Table 4.2, pruning subjects after relation prediction provides an improvement of at most 1% for our model by enforcing consistency with the set of available triples. A larger number of candidates translates to a higher accuracy due to higher candidate generation recall (see Table 4.1).

To investigate how much the candidate generation contributes to the results we computed the accuracy resulting from randomly picking an entity from the list of subject candidate entities and subsequently randomly picking a relation predicate from the list of outgoing relations of the chosen subject entity. The results are shown in Table 4.3. The poor performance verifies that the task of finding the right subject predicate pair can not be solved by n-gram based search alone.

4.3.2 Comparison

We compare our results with the four recent state-of-the-art works that developed QA models for the SimpleQuestions dataset. These works include the *Memory Network* approach of Bordes et al. [121], the attentive CNN based approach by Yin et al. [125], the character-level attention-enhanced encoder-decoder approach by Golub and He [184], and the word-level RNN-based approach by Dai et al. [124] (see Section 3.2.1 for a more detailed discussion of these approaches).

Approach	Setting	Test Accuracy %
Bordes et al. [121]	end-to-end	62.7
Yin et al. [125]	end-to-end	68.3
Dai et al. [124]	end-to-end	62.6*
Golub and He [184]	end-to-end	70.9
Our approach	end-to-end	74.8
Dai et al. [124]	active linking	75.7*
Yin et al. [125]	focused pruning	76.4

Table 4.4: Comparison with state-of-the-art systems. When marked with an asterisk (*), accuracy in the FB5M test setting is given, otherwise FB2M test setting results are shown.

We compare the results obtained by the state-of-the-art models when evaluated in an end-to-end setting to the results we obtain with our model (presented in Table 4.4). The second part of Table 4.4 shows the performance of two systems that train and use a separate question segmentation model that finds the entity mention, which improve their results significantly. However, since the segmentation model requires separate training with different data, we deem direct comparison unfair.

As shown in Table 4.4, our approach achieves competitive results in the end-to-end setting. Our approach beats the baseline results of Bordes et al. [121] by a high margin, outperforms the CNN-based approach of Yin et al. [125] and manages to match and surpass the attention-enhanced encoder-decoder approach of Golub and He [184]. However, it is outmatched by approaches that use a separately trained segmentation model [124, 125]. Including a similar segmentation model could improve our results as well by resolving the segmentation errors (see next section) and reducing noise in predicate relation patterns.

4.4 Error analysis

We perform error analysis on our single-layered unidirectional question encoder model with 10 subject entity candidates per n-gram to gain insight into the kind of errors our approach produces.

We distinguish between the following types of errors:

- *Missing candidates:* The correct entity is missing in the candidate set. For example, in the question "*which group recorded tokyo?*", "*tokyo*" has hundreds of possible candidates and by restricting the subject candidate set to a small number (e.g. 10), there is a high chance the right interpretation of "*tokyo*" is not among the considered candidates.
- *Indistinguishability:* The predicted entity is indistinguishable from the correct entity (i.e., has the same label and the same type). For example, the question "*what song was on the recording indiana?*" expects an entity with label "*indiana*" and type label "*musical recording*". However, there exist multiple entities in the KG conforming to these criteria and they all receive the same score.
- *Hard ambiguity:* The top-scored entity has the same label but different type from the lower-scored correct entity and the question does not provide clear indications which entity is expected.

Cause of error	**Counts**
Missing	11
Indistinguishability	21
Hard ambiguity	11
Soft ambiguity	3
Wrong subject mention	2
Wrong predicate	32

Table 4.5: Counts of sources of errors that our model makes on 250 test examples. The numbers have been obtained after prediction with max 400 candidates per n-gram.

For example, for the question "*where was eddie smith born?*", the two top-ranked candidates are *Eddie Smith, baseball player* and *Eddie Smith, film actor*.

- *Soft ambiguity:* The top-scored and correct entities have the same label and different types, as with *Hard ambiguity*, however, the question provides information for correct disambiguation.

- *Wrong subject mention:* The label of the top-scored entity is different from the label of the lower-scored correct entity, indicating that the question encoder made a mistake by (implicitly) taking the wrong subject mention span. For example, for the question "*who directed the show dancin' homer?*", an entity with label "*The Show*" and type label "*film*" is the top-scored entity, with a score slightly higher than the correct subject entity with label "*Dancin' Homer*" and type label "*tv episode*".

- *Wrong predicate:* The subject is predicted correctly, but a wrong relation is chosen.

We manually inspect 250 questions from the test set and present the findings in Table 4.5. Among errors resulting in wrong subject prediction, missing candidates are the most frequent category when only the top-10 candidates per n-gram are used. This is due to the fact that in some cases, there are many more than 10 suitable candidates for some n-gram and the true subject entity was ranked lower than the cutoff (10) during candidate generation (we rank by triple count).

We believe the examples in error categories *Indistinguishability* and *Hard ambiguity* to be unresolvable with the information currently used in our approach (entity labels and entity type labels for matching and using provided knowledge for subject pruning) since there isn't enough information even for humans to pick the right answer. When using larger candidate sets (e.g. 400), we observed that many error cases from the *Missing candidates* category migrated to the *Hard ambiguity* or *Indistinguishability* categories . Other ambiguity errors and segmentation errors, however, offer room for the improvement of our approach.

We did not perform a thorough error analysis of relation prediction, but during inspection, we encountered cases where picking the right relation is very challenging. One such example is "*who produced the film woodstock villa?*", where our approach ranked `/film/film/produced_by` at the top and `/film/film/executive_produced_by`, the correct predicate, only second.

4.5 Discussion

In this work, we focused on better question representation, where we operate on word level but also integrate character-level information for each word. However, when extending our model to more complex questions in future work, we believe the addition of an attention mechanism could be advantageous. We developed a model without an attention mechanism to reduce complexity and improve efficiency. Note that Golub and He [184] explore an attention-based model for SimpleQuestions.

Our work is similar to that of Yin et al. [125] in that it also encodes subject labels on character level. However, our work is different in that we also add type label information in order to improve subject disambiguation and employ a hierarchical word- and character-level question encoder.

In contrast to our work, Dai et al. [124] encode the question on word-level only and train/use Freebase-specific predicate and entity representation (such as pretrained TransE embeddings, which have to be re-trained from scratch for every new KG, and type vectors). Our work fully relies on textual information provided by the KG about the entities and predicates. Thus, our approach is easily transferrable to other KGs.

4.6 Conclusion

In this article, we presented an end-to-end, neural network based approach for answering simple questions over large-scale knowledge graphs, leveraging a hierarchical word- and character-level question encoder. We also investigate the use of type information for better subject disambiguation. While keeping the model simple by refraining from using attention mechanisms or separate segmentation models, the proposed approach achieves competitive results when compared to other end-to-end approaches. In comparison to other character-level approaches for this task [124, 184], we explore hierarchical character-word level representation networks, which allows to exploit richer word-level semantics as captured in pretrained word embeddings. Because the network builds representations for KG symbols based on their labels (and other information), these representations are easily generalizable to novel symbols, as well as to novel KGs, which ties in with RQ1.

We found that the choice of negative samples has a big impact on performance and experimented with a negative sampling generation approach that exposes the model to conditions equivalent to those under which it will be used during prediction.

As our model does not learn KG-specific symbol representations, we believe it to be easily portable to other knowledge graphs and suitable for KG's that change over time (such as DBpedia Live). However, we leave an evaluation of its portability for future work.

In future work, the investigation of implicit [184] or explicit [124, 125] segmentation modelling should improve subject prediction by better candidate pruning and reduce the noise for better learning of predicate patterns. Exploitation of external lexical resources could also prove advantageous as it should help to bridge the lexical gap.

CHAPTER 5

Transfer learning for question answering over knowledge graphs

"Pass on what you have learned."

– Yoda, from *Star Wars: Return of the Jedi*

In this chapter, we focus on transfer learning in the context of question answering over knowledge graphs. We first look into transfer learning from pre-trained language models (PLM) on simple questions and then expand our analysis to transfer learning from PLMs for complex questions, as well as transfer learning between different complex questions datasets.

The first part of this chapter (Sections 5.1-5.4) is part of our work on *answering simple factoid questions* (which is also the task we use in Chapter 4). As already mentioned in the previous chapter and in Section 3.2.1, simple questions are questions whose queries contain only a single entity and predicate. In our work, we use the SimpleQuestions [20] dataset, which uses Freebase [1] as the knowledge graph. We use BERT [7] as our PLM.

In the second part of this chapter (Sections 5.5-5.8), we focus on more complex questions, which, for example, may contain multiple conditions, and may use aggregations. For such queries, the approaches we used for simple questions are no longer applicable because the outputs are larger structures of variable size. See also Section 3.2.2 for more information about answering complex questions over knowledge graphs. In this work, we adopt an approach that relies on ranking candidate logical forms by comparing their encoding to the question encoding. We use the QALD-7 [22] and LC-QuAD [21] datasets in this analysis. Similarly to the first part of this chapter, we investigate BERT [7] as our choice of PLM.

BERT is a transformer [18]-based PLM. For more information about transformers, see Section 2.2.3, and for more information about BERT and other PLMs and transfer learning using such PLMs, see Section 2.4.2.

This chapter is based on the following publications:

5. **Denis Lukovnikov**, Asja Fischer and Jens Lehmann. *Pretrained Transformers for Simple Question Answering over Knowledge Graphs.* In Proceedings of the 18th International Semantic Web Conference (ISWC 2019). 2019.

6. Gaurav Maheshwari*, Priyansh Trivedi*, **Denis Lukovnikov***, Nilesh Chakraborty*, Asja

Fischer and Jens Lehmann. *Learning to Rank Query Graphs for Complex Question Answering over Knowledge Graphs.* In Proceedings of the 18th International Semantic Web Conference (ISWC 2019). 2019.

While the work described in this chapter presents different approaches for answering simple factoid questions and complex questions over knowledge graphs, the focus of the contributions also lies in investigating transfer learning for KGQA. This corresponds to Contribution 2, which addresses the following research question:

RQ2: Does transfer learning improve KGQA accuracy?

The rest of the chapter is organized as follows. Sections 5.1 to 5.4 focus on the case of simple question, while Sections 5.5 to 5.8 focus on complex questions.

Section 5.1 motivates the simple questions task and transfer learning in its context in more detail and lists the specific contributions at the time the work was performed. Section 5.2 presents the approach we take for simple questions. Section 5.3 presents the experimental setup and Section 5.4 presents the results obtained on the SimpleQuestions dataset, where we especially focus on investigating performance when the amount of training data is limited and perform a qualitative analysis of the attention patterns before and after fine-tuning.

For complex questions, Section 5.5 provides a motivation and states the specific contributions in more detail. Section 5.6 provides some definitions and describes query graphs. Section 5.7 presents our approach to tackle KGQA with complex questions. Finally, Section 5.8 presents and discusses experimental results and analysis of the used approach in two transfer learning settings: transfer from PLM (Section 5.8.3) and transfer from a different KGQA task (Section 5.8.2).

See Section 3.2.1 for a general overview of related work on simple questions and Section 3.2.2 for complex questions. We conclude the chapter in Section 5.9 with a conclusion and an outlook on future work.

5.1 Transfer Learning from Pre-trained Language Models for Simple Questions

Recently proposed transfer learning methods [7, 55, 57, 58, 190, 191] show that significant improvement on downstream natural language processing (NLP) tasks can be obtained by finetuning a neural network that has been trained for language modeling (LM) over a large corpus of text data without task-specific annotations. Models leveraging these techniques have also shown faster convergence and encouraging results in a few-shot or data-sparse settings [190]. Owing to their benefit, the use of this family of techniques is an emerging research topic in the NLP community [191]. However, it had received little attention in KGQA research at the time of writing.

The main focus of our work is to investigate transfer learning for question answering over knowledge graphs (KGQA) using models pretrained for language modeling. For our investigation, we choose BERT [7] as our pretrained model used for finetuning, and investigate transfer from BERT using the SimpleQuestions [121] task. BERT is a deep transformer [18] network trained on a masked language modeling (MLM) task as well as a subsequent sentence pair classification task (see also Section 2.4.2). We use SimpleQuestions because it is a very well-studied dataset that characterizes some core challenges of KGQA. The large size of the dataset is particularly appealing for our study, because it allows us to investigate performance for a wider range of amounts of the data used for training. Promising results with BERT for KGQA have been very recently reported on other KGQA datasets [160]. However, we found a thorough investigation of the impact of data availability and an analysis of internal model behavior to be missing, which would help to better understand model behavior in applications of KGQA.

The contributions of the first part of this chapter are as follows:

- We demonstrated for the first time the use of a pretrained transformer network (BERT) for simple KGQA. We also propose a simple change in our models that yields a significant improvement in entity span prediction compared to previous work.
- We provide a thorough evaluation of pretrained transformers on SimpleQuestions for different amounts of data used in training and compare with a strong baseline based on bidirectional Long-Short-Term-Memory [42] (BiLSTM). To the best of our knowledge, our work is the first to provide an analysis of performance degradation with reduced training data sizes for SimpleQuestions and KGQA overall.
- We try to provide an understanding of the internal behavior of transformer-based models by analyzing the changes in internal attention behavior induced in the transformer during finetuning.

We perform our study using the general framework used in recent works [118, 119], where simple question interpretation is decomposed into (1) entity span detection, (2) relation classification, and (3) a heuristic-based post-processing step to produce final predictions. In this work, we particularly focus on the first two subtasks, providing detailed evaluation results and comparison with a baseline as well as with Mohammed et al. [118].

5.2 Approach for Simple Questions

We follow the general approach outlined in BuboQA [118], which decomposes simple question interpretation into two separate learning problems: (1) entity span detection and (2) relation classification. See also Section 3.2.1. Recent works on SimpleQuestions show that this general approach, followed by a heuristic-based entity linking and evidence integration step can achieve state-of-the-art performance [118, 119], compared to earlier works [122–124, 159] that investigated more complicated models.

In summary, our approach follows the following steps at test time:

1. Entity Span Detection and Relation Prediction: The fine-tuned BERT model is used to perform sequence tagging to both (1) identify the span s of the question q that mentions the entity (see Section 5.2.1) and (2) predict the relation r used in q (see Section 5.2.2). In the example "*Where was Frank Herbert born?*", s would be the span "*Frank Herbert*". The tagger is trained using annotations automatically generated from the training data and entity labels in Freebase.
2. Entity Candidate Generation: We retrieve entities whose labels are similar to the predicted entity span s using an inverted index[1] and rank them first by string similarity (using `fuzzywuzzy`)

[1] The inverted index maps words to entities whose labels contain that word

and then by the number of outgoing relations. For our example, the resulting set of entity candidates will contain the true entity for Frank Herbert, the writer (corresponding to Freebase URI http://www.freebase.com/m/02xyl). Note that the true entity does not necessarily rank highest after the retrieval phase.

3. Query Ranking: Given the relations predicted in Step 1, and the set of entities from Step 2, the entity-relation pairs are re-ranked as detailed in Section 5.2.3. After ranking entity-relation pairs, we take the top-scoring pair, from which we can trivially generate a query to retrieve the answer from the KG.

Whereas previous works experimented with recurrent and convolutional neural network (RNN resp. CNN) architectures, we investigate an approach based on transformers. Several existing works train separate models for the two learning tasks, i.e. entity span detection and relation prediction. Instead, we train a single network for both tasks simultaneously.

5.2.1 Entity span prediction

In this step, we intend to identify the span of tokens in the input question referring to the subject entity mentioned in it. Previous works treated this problem as a binary I/O sequence tagging problem, and explored the use of BiLSTM, conditional random fields (CRFs), and combined BiLSTM-CRF taggers. The sequence tagging model is trained to classify each token in the input sequence as belonging to the entity span (I) or not (O). Instead, we treat span prediction as a classification problem, where we predict the start and end positions of the entity span using two classifier heads. This approach assumes that only one entity is mentioned in the question and that its mention is a single contiguous span. We believe this assumption is reasonable for simple questions and holds for the used dataset. Formally, the start-position classifier has the following form:

$$p(i = \mathtt{START}|x_1, \ldots, x_N) \quad = \frac{e^{\mathbf{x}_i^{L+1\top}\mathbf{w}_{\mathtt{START}}}}{\sum_{j=1}^{N} e^{\mathbf{x}_j^{L+1\top}\mathbf{w}_{\mathtt{START}}}} \quad , \tag{5.1}$$

where $\mathbf{x}_i^{L+1}$ is the feature vector produced by BERT's topmost (L-th) layer for the i-th token of the sequence and $\mathbf{w}_{\mathtt{START}}$ is the parameter vector of the start position classifier. End position prediction works analogously, applying a different parameter vector, $\mathbf{w}_{\mathtt{END}}$.

5.2.2 Relation prediction

Relation prediction can be considered a sequence classification task since the SimpleQuestions task assumes there is only a single relation mentioned in the question. Thus, for relation prediction, we use BERT in the sequence classification setting where we take the feature vector $\mathbf{x}_{\mathsf{CLS}}^{L+1} = \mathbf{x}_1^{L+1}$ produced for the `[CLS]`[2] token at the beginning of the input sequence and feed it through a softmax output layer

[2] Before using BERT, the input sequence is first tokenized into WordPieces, a `[CLS]` token at the beginning and a `[SEP]` token is added at the end.

to get a distribution over possible relations:

$$p(r = R_i | x_1, \ldots, x_N) = \frac{e^{\mathbf{x}_{\text{CLS}}^{L+1\top} \mathbf{w}_{R_i}}}{\sum_{k=1}^{N_R} e^{\mathbf{x}_{\text{CLS}}^{L+1\top} \mathbf{w}_{R_k}}} \quad , \tag{5.2}$$

where $\mathbf{w}_{R_i}$ is the vector representation of relation R_i[3].

Previous works [119, 123, 125] propose using the question *pattern* instead of the full original question in order to reduce noise and overfitting. They do this by replacing the predicted entity span with a placeholder token. Doing this would require training a separate model for relation prediction and introduce dependency on entity span prediction. In our BERT-based approach, we chose to train a single model to perform both entity span prediction and relation prediction in a single pass. Thus, we do *not* replace the entity span for relation prediction. We also experimented with training a separate transformer with (1) setting the attention mask for all self-attention heads such that the entity tokens are ignored and (2) replacing the entity mention with a `[MASK]` token. However, both methods failed to improve relation classification accuracy in our experiments.

Training a separate relation classifier network without entity masking yields results equivalent to simply training a single network for both entity span prediction and relation prediction.

5.2.3 Logical Form Selection

To get the final logical forms, we take the top K (where K=50 in our experiments) entity candidates during entity retrieval and for each, we take the highest-scored relation that is connected to the entity in the knowledge graph. We rank the entity-predicate candidate pairs first based on the string similarity of any of their entity labels/aliases with the identified span, breaking ties by favouring entity-predicate pairs with predicates with higher prediction probability under the BERT model, and the remaining ties are broken by entity in-degree (the number of triples the entity participates in as an object).

5.3 Experimental Setup for PLMs for Simple Questions

We use the small uncased pretrained BERT model from a PyTorch implementation of BERT[4]. The whole transformer network and original embeddings were finetuned during training. For training, the Adam optimizer [49] was employed and we experimented with different learning rate schedules. Most of the final results reported use a cosine annealing learning rate schedule with a short warmup phase of approximately 5% of total training updates. We indicate if reported results rely on a different schedule.

We used PyTorch 1.0.1 and trained on single Titan X GPU's. The source code is provided at `https://github.com/lukovnikov/semparse/tree/master/semparse/simplequestions`.

[3] The vector $\mathbf{w}_{R_i}$ is a trainable parameter vector, unique for relation R_i (and is thus not presented by sub-symbolic encodings as it is for example the case in [123, 159]).

[4] `https://github.com/huggingface/pytorch-pretrained-BERT`

5.3.1 Metrics

To evaluate the entity span prediction model, we compute the average[5] F1 and span accuracy[6] on word level. Since BERT operates on subword-level (WordPiece), we first need to obtain word-level metrics. To do this, we first transform the predictive distributions over subword units to distributions over words by summing the probabilities assigned to subword units of a word. Then, we take the argmax over the resulting distribution over words.

We also compute F1 on word level over the entire dev/test datasets to compare our numbers to BuboQA [118]. Even though the difference between the dataset-wide F1 and the averaged F1 is small, we believe the latter is more informative, since the contribution of every example is equal and independent of the span lengths.[7] (lower entropy of the predictive categorical distribution)

For relation classification, we report classification accuracy.

5.3.2 Baseline

As a baseline, we use a BiLSTM start/end classifier for entity span prediction and a BiLSTM sequence classifier for relation prediction. The BiLSTM start/end classifier works on word level and uses the same Glove [40] embeddings as BuboQA [118]. We use the same output layer form as our BERT-based model, where instead of performing a binary I/O tagging of the input sequence, we simply predict the beginning and end positions of the span using a softmax over sequence length (see also Eq. 5.1). Using this small change significantly improves the performance of our baseline for entity span prediction, as shown in Section 5.4.

For relation classification, we use a different BiLSTM, taking the final state as the question representation vector and using it in a classifier output as in BuboQA [118] – comprising of an additional forward layer, a ReLU, a batch normalization layer and a softmax output layer. We did not replace the entity mentions with an entity placeholder (like [119]), and instead fed the original sequences into the relation classification encoder.

Even though these BiLSTM baselines are quite basic, previous work has shown they can be trained to obtain state-of-the-art results [118, 119].

Both BiLSTMs were trained using a cosine annealing learning rate schedule as the one used to train our BERT-based model.

As shown in Section 5.4, our baselines perform better than or on par with equivalent networks used in BuboQA [118].

5.3.3 Effect of Limited Training Data

In order to further illustrate the usefulness of pretrained models for SimpleQuestions and KGQA, we perform a series of experiments to measure how performance degrades when fewer examples are available for training. SimpleQuestions is a fairly large dataset containing 75k+ training examples. With abundant training data available, a randomly initialized model is likely to learn to generalize

[5] F1, precision and recall are computed separately for each example based on span overlaps and then averaged across all examples in the dev/test set.

[6] Span accuracy is one only for examples where all token memberships are predicted correctly.

[7] The implementation of F1 in BuboQA's evaluation code seems to be computing F1 based on precision and recall computed over the dataset as a whole, thus letting examples with longer spans contribute more towards the final score.

	Accuracy	Avg. F1	F1*
BiLSTM [118]	–	–	93.1
CRF [118]	–	–	90.2
BiLSTM (ours)	93.8	97.0	97.1
BERT (ours)	95.6	97.8	97.9

(a) Entity span prediction.

	Accuracy
BiGRU [118]	82.3
CNN [118]	82.8
BiLSTM (ours)	82.8
BERT (ours)	83.6

(b) Relation prediction.

Table 5.1: Component performance evaluation results, trained on all available training data, measured on validation set. (a) Entity span prediction performance, measured by span accuracy, average span F1 and dataset-wide F1 (F1*), all on word level. (b) Relation prediction performance, measured by accuracy (R@1).

well, which might make the advantage of starting from a pretrained model less pronounced. The large size of SimpleQuestions makes it possible to study a wider range of data-sparse cases than other, smaller datasets.

We run experiments for both BERT and our baseline BiLSTM with different fractions of the original 75k+ training examples retained for training. Examples are retained such that the number of relations not observed during training is minimized, favouring the removal of examples with most frequently occurring relations. We assume that this strategy, compared to random example selection, should not have a big effect on entity span prediction accuracy but should minimize errors in relation prediction due to unseen relation labels, and create more balanced datasets for more informative performance assessment. We report span accuracy and relation accuracy on the validation set of SimpleQuestions as a function of the fraction of data retained in Table 5.3. For relation prediction, we only report experiments where the retained examples cover all relations observed in the full training dataset at least once.

5.4 Results and Analysis for PLMs for Simple Questions

For the two learning tasks, we observe significant improvements from using BERT, as shown in Table 5.1(a) for entity span prediction and Table 5.1(b) for relation prediction (see Section 5.4.1). Section 5.4.2 talks about experiments with fewer training data, Section 5.4.3 shows component performance on the test set. Final results for the whole simple QA task are discussed in Section 5.4.4. Finally, we conclude with an analysis of the attentions in the transformer in Section 5.4.5.

R@N	BiLSTM [118]	BiLSTM (ours)	BERT (ours)
1	67.8	76.45	77.17
5	82.6	87.46	88.18
20	88.7	91.47	92.13
50	91.0	93.07	93.71
150	–	94.88	95.40

Table 5.2: Entity recall on validation set.

		0.03% (22)	0.2% (151)	1% (757)	2.5% (1k9)	5% (3k8)	10% (7k6)	25% (18k9)	50% (37k9)	75% (56k8)	100% (75k7)
Entity Span	BiLSTM	33.1	64.5	74.0	78.1	82.5	85.5	90.1	92.0	93.4	93.8
	BERT	62.5	79.1	85.4	88.9	90.8	92.4	94.2	94.9	95.5	95.6
Relation	BiLSTM	–	–	–	26.5	41.0	56.3	72.4	79.0	81.3	82.8
	BERT	–	–	–	29.6*	48.6	67.5	76.5	80.1	82.6	83.6

Table 5.3: Entity span detection accuracies (top half) and relation prediction accuracies (bottom half) as a function of fraction of training data retained. Evaluated on the entire validation set. (*) indicates a cosine learning rate schedule with restarts — in extremely low data scenarios for relation classification, this seems to yield better results than the cosine learning rate schedule without restarts that is used everywhere else.

5.4.1 Full data results

From Table 5.1(a), we can see that BERT outperforms our BiLSTM baseline by almost 2% accuracy (evaluated on validation set), although the difference in F1 is smaller. Compared to BuboQA [118], we obtain much higher dataset-wide F1 scores, which we attribute to our start/end prediction rather than I/O tagging used by previous works, including BuboQA.

The improvement is less pronounced in relation classification accuracies (see Table 5.1(b)), where our baseline BiLSTM achieves the same results as those reported by BuboQA [118] for a CNN. Our BERT-based classifier beats our BiLSTM by almost 1% accuracy.

Table 5.2 shows entity retrieval performance for different numbers of candidates, compared against the numbers reported in [118]. The recall at 50 is 2.71% higher. Please note that we also use entity popularity during retrieval to break ties that occur when multiple retrieved entities have the same name (and thus the same string similarity—the main sorting criterion).

5.4.2 Effect of Limited Training Data

From the limited-data experiments for entity span prediction shown in Table 5.3 (top part), we can conclude that a pretrained transformer is able to generalize much better with fewer examples. In fact, with only 1% of the original training data used (757 examples), BERT reaches a best span prediction accuracy of 85.4% on the validation set, corresponding to an average F1 of 93.2. In contrast, our BiLSTM baseline achieves only 74.0% span prediction accuracy on the validation set, corresponding to 88.6 F1. In an extremely data-starved scenario, with only 0.03% of the original dataset—corresponding to just 22 training examples—the best validation accuracy we observed for BERT was 62.5%, corresponding to 80.9 F1. In the same setting, we were not able to obtain more than 33.1% accuracy with our BiLSTM baseline. Overall, we can clearly see that the degradation in performance with less data is much stronger for our Glove-based BiLSTM baseline.

Limited-data experiments for relation prediction (shown in Table 5.3) (bottom part) reveals that relation classification is more challenging for both our BiLSTM and BERT-based models. However here too, BERT seems to degrade more gracefully than our Glove+BiLSTM baseline.

5.4.3 Performance on test set

After identifying good hyperparameters for both our BiLSTM baseline and our BERT-based model using the validation set, we evaluated our models using the same evaluation metrics on the test set. Results for both entity span prediction and relation prediction on the test set are reported in Table 5.4.[8] As shown in Table 5.4, the test set results are close to the validation set results for both models.

5.4.4 Final results

In Table 5.5, we compare our final predictions against previous works on SimpleQuestions. With our simple entity linking and logical form selection procedure (see Section 5.2.3), we achieve 77.3% accuracy on the test set of SimpleQuestions, beating all but one of the existing approaches. We suspect that the final score can be further improved by finding better rules for logical form selection, however that is not the goal of this study.

Investigating the entity and relation prediction accuracies separately, we find accuracies of 82.7% for entities and 86.6% for relations. Comparing the 86.6% for relation accuracy after re-ranking (Section 5.2.3) to the 83.5% (Table 5.4) relation accuracy before the re-ranking confirms that re-ranking has helped to reduce errors. By analyzing the 22.7% of test examples that were predicted incorrectly, it turned out that in 35% of those cases both a wrong relation and a wrong entity had been predicted, in 41% only the entity was wrong and 24% had only a wrong relation. Of all the cases where the entity was predicted wrong, in 28.6% cases this resulted from the correct entity missing in the candidate set. Entity retrieval errors are also correlated with relation errors: of the cases where the correct entity was not among the retrieved candidates, 71.2% had a wrongly predicted relation, against 55.7% for cases where the correct entity was among the candidates.

5.4.5 Attention analysis

One of the advantages of using transformers is the ability to inspect the self-attention weights that the model uses to build its representations. Even though this does not completely explain the rules the model learned, it is a step towards explainable decision making in deep learning, and a qualitative improvement upon RNNs. In an attempt to understand how the model works, before and after fine-tuning, we manually inspected the attention scores used by the transformer network internally during the encoding process.

[8] Note that the test set contains "unsolvable" entries, where the correct entity span has not been identified in pre-processing. For these examples, we set the accuracy and F1 to zero.

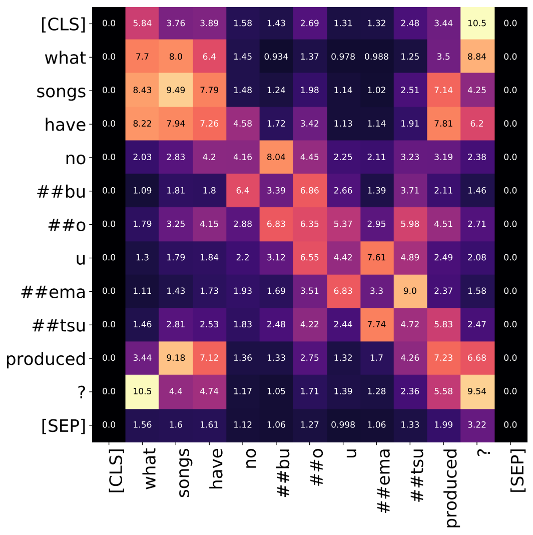
[9] [192] is not included in the comparison because neither [118] or [119] could reproduce the reported results (86.8%).

	Entity Span		Relation
	Accuracy	Avg. F1	Accuracy
BiLSTM	93.2	96.7	82.4
BERT	95.2	97.5	83.5

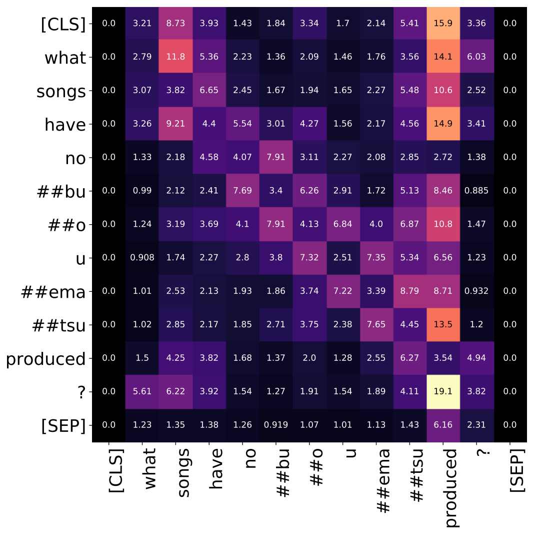
Table 5.4: Component results on test set.

Approach	Accuracy
MemNN [121]	61.6
Attn. LSTM [122]	70.9
GRU [159]	71.2
BuboQA [118]	74.9
BiGRU [124]	75.7
Attn. CNN [125]	76.4
HR-BiLSTM [123]	77.0
BiLSTM-CRF [119]	78.1
BERT (ours)	77.3

Table 5.5: Final accuracy for the full prediction task on the test set of SimpleQuestions. [9]



(a) Before fine-tuning



(b) After fine-tuning

Figure 5.1: Average attention distribution for the example "*What songs have Nobuo Uematsu produced?*", (a) before training on our tasks (vanilla pretrained BERT) , and (b) after training on our tasks (finetuned BERT). The numbers are scaled to values between 0 and 100, and are computed by averaging the normalized attention scores over all heads in all layers, and multiplying the average by 100. We set the scores for `[CLS]` and `[SEP]` tokens to zero in the plots since they always receive a much higher average attention weight than the actual words from the sentence and thus would dominate the plot.

We compute the average of the 144 attention distributions produced by the $M = 12$ different attention heads in each of the $L = 12$ layers of the employed BERT network:

$$\beta_{i,j} = \frac{\sum_{l=1}^{L} \sum_{h=1}^{M} \alpha_{l,h,i,j}}{L \cdot M} \quad , \tag{5.3}$$

where $\alpha_{l,h,i,j}$ is the normalized attention score from position i to position j at l-th layer's h-th head, as

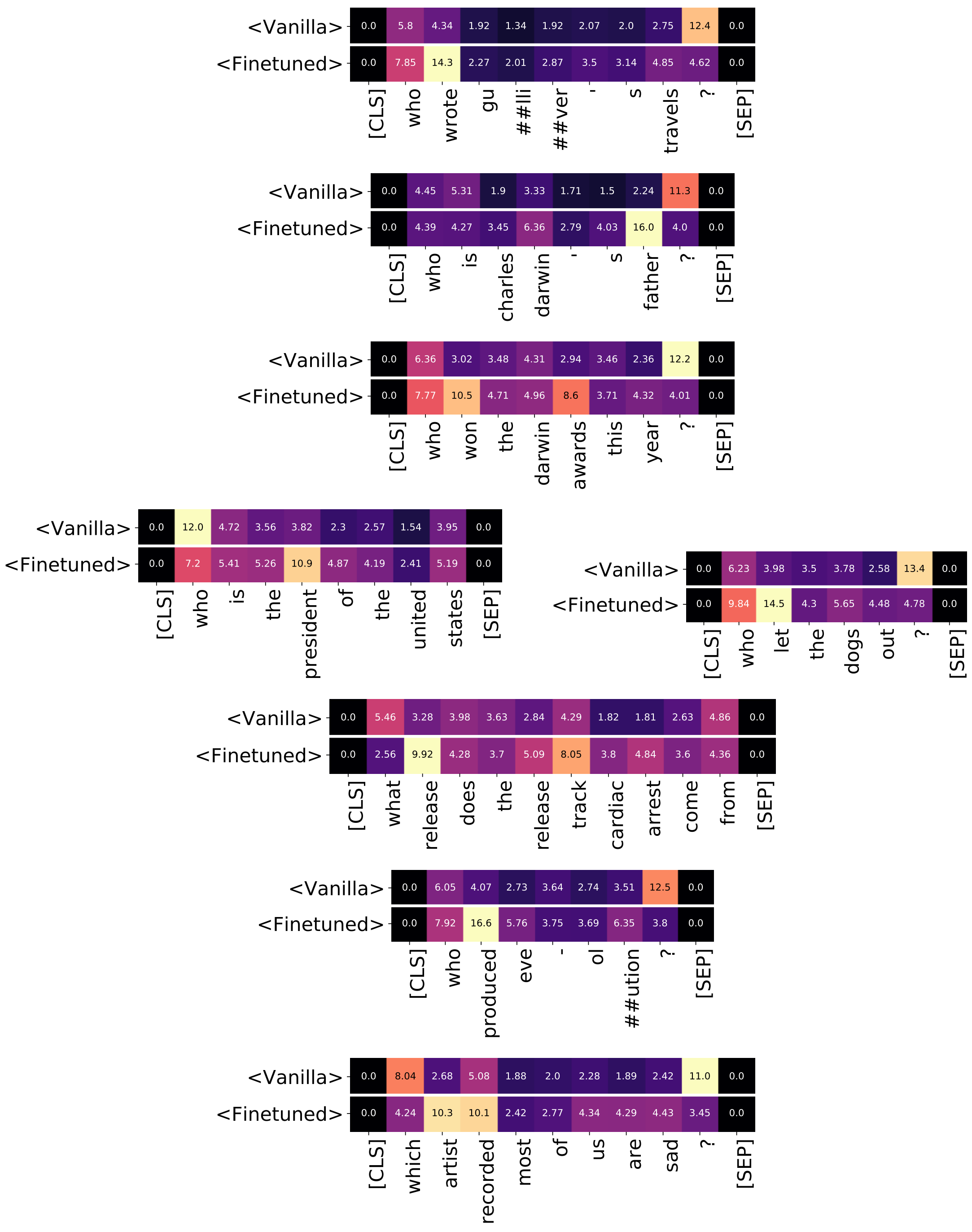


Figure 5.2: Average normalized attention scores for the `[CLS]` token for several examples. <`Vanilla`> is pretrained BERT before finetuning. <`Finetuned`> is BERT finetuned on our tasks. The numbers are scaled to values between 0 and 100, and are computed by averaging of the normalized attention scores over all heads in all layers, and multiplying the average by 100.

computed in the transformer. Here, $\beta_{i,j}$'s are the average attention scores; these values are displayed in Figures 5.1 and 5.2 (multiplied by 100). More concretly, we compare this average attention signature of a (vanilla) BERT network before fine-tuning it with the attention signature of a BERT model fine-tuned for our tasks (recall that we trained a single model to perform both entity span detection and relation classification simultaneously). By comparing the attentions before and after training on our tasks, we can identify differences in internal behavior of the model that arose during training.

In Figure 5.1, the average of all attention distributions is shown for an example question for two versions of the transformer model: pre-trained (vanilla) BERT and BERT fine-tuned for our tasks. While in general, the average attention distribution roughly follows the same patterns after fine-tuning, we can see that the behavior of the attention mechanism responsible for building the representation of the `[CLS]` token is significantly different. We found that, before fine-tuning, the representation building of the `[CLS]` token generally focuses on punctuation and less strongly, on other words. After finetuning, `[CLS]`'s representation is strongly focused on words that characterise the relation conveyed by the sentence. For example, for the question "*Who wrote Gulliver's travels?*" (see Figure 5.2, first example), the attention is shifted towards the word "*wrote*", which specifies the authorship relationship of the intended answer with the subject entity (*Gulliver's Travels*) mentioned in the question. We provide several other examples of this kind of attention change in Figure 5.2.

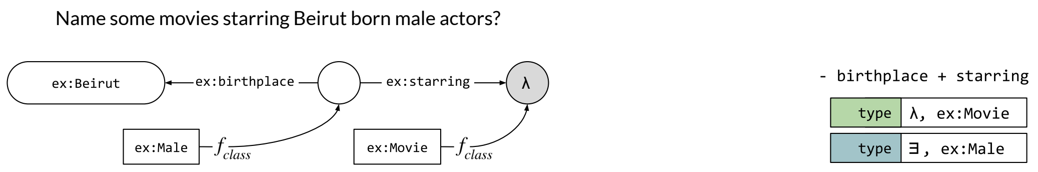
This change in internal attention behavior can be explained by the fact that sequence classification for relation prediction is done based on the representation built for the `[CLS]` token and attending to relation-specifying words more would produce more useful features for the classifier.

5.5 Complex Questions, Ranking Models and Transfer Learning

So far, we only looked at transfer learning from PLMs to simple questions over knowledge graphs. In the remainder of this chapter, we will focus on answering complex questions over knowledge graphs and transfer learning both from PLMs as well as across datasets in this setting. This investigation completes the investigation we presented in the previous section regarding RQ2. Aside from the investigation of transfer learning, we also present a novel approach for KGQA.

Numerous approaches [34, 126, 147] create an *ungrounded* expression of a given natural language question (NLQ), and then ground it w.r.t. a target KG. Here, *grounding* refers to linking elements in the expression with elements (i.e. entities and predicates) in the KG. While this approach suits the non-trivial task of handling wide syntactic and semantic variations of a question during parsing, it needs to handle lexical as well as structural mismatch between the generated expression and the target KG during grounding. For instance, the predicate mother(x_a, x_b), parsed from a question, might be represented as parent$(x_a, x_b) \wedge$ gender$(x_a,$ female$)$ in a KG. Failing to anticipate and tackle these issues can lead to undesirable situations where the system generates expressions which are illegal w.r.t. the given KG.

We focus on an alternate family of approaches [84, 123] which, given an NLQ, first generate a list of formal expressions describing possible candidate queries which are in accordance with the KG structure, and then rank them w.r.t. the NLQ. Note that we also rely on ranking for the approach we used for simple questions in Chapter 4. We use a custom grammar called *query graph* to represent these candidate expressions, comprised of paths in the KG along with some auxiliary constraints. In this work, we propose a novel *slot matching* model for ranking query graphs. The proposed model exploits the structure of query graphs by using attention to compute different representations of the



(a) *Question, and corresponding Query Graph*

(b) *Core-chain corresponding to (a)*

Figure 5.3: A question and its corresponding query graph (a), and core chain (b).

question for each predicate in the query graph. We compare our models against several baseline models by evaluating them over two DBpedia [3] based KGQA datasets namely, LC-QuAD [21] and QALD-7 [22].

Furthermore, we appropriate pre-trained bidirectional transformers (BERT) [7] to be used in the slot matching configuration, thereby enabling the use of large-scale pre-trained language models for the task. To the best of our knowledge, at the time the work was performed, this was the first work that explores their use for KGQA with complex questions.

Finally, we also investigate the potential of transfer learning in KGQA by fine-tuning models trained on LC-QuAD on the much smaller QALD-7 dataset, resulting in a significant improvement in performance on the latter. We thereby demonstrate the efficacy of simple transfer learning techniques for improving performance of KGQA on target domains that lack training data.

The major contributions of this part of our work are summarized as follows:

- A novel ranking model which exploits the structure of query graphs, and uses multiple attention scores to explicitly compare each predicate in a query graph with the natural language question.
- An investigation of fine-tuning based transfer learning across datasets and the use of pre-trained language models (BERT [7]) for the KGQA task.

Our experiments show that the proposed slot-matching model outperforms the baseline models, and that it can be combined with transfer learning techniques to offset the lack of training data in the target domain.

5.6 Background: Query Graphs

In this section we will give a description of the employed query graph language.

We use query graphs as the intermediary query language to express candidates of formal KG queries given an NLQ. They represent a path in K as a directed acyclic labeled graph. We borrow the augmentations made to the query graph grammar in [84], which makes the conversion from query graph expressions to executable queries trivial, and slightly modify it to suit our use case. In the subsequent paragraphs we detail the modified query graph grammar we employ.

Elements of a query graph: A query graph consists of a combination of nodes $n \in \{$*grounded entity*, *existential variable*, *lambda variable*, *auxiliary function*$\}$, connected with labeled, directed edges representing predicates from the set of predicates $\mathcal{P}$ of K. We define each of the aforementioned elements in detail below by the help of a running example, answering the question "*Name some movies starring Beirut born male actors?*":

- **Grounded Entities**: Grounded entities correspond to entities of K mentioned in the NLQ. Each query graph has at least one grounded entity. In the case of our example, ex:Beirut is the grounded entity represented by the rounded rectangle in Fig. 5.3.a.
- **Existential Variables**: Existential variables are ungrounded nodes, i.e. they do not correspond to an explicit entity mentioned in the NLQ but are instead placeholders for intermediate entities in the KG path. They help disambiguate the structure of query graphs by the means of auxiliary functions (described below). In our example, we have one existential variable which is represented by a circle and stands for entities like ex:Keanu_Reeves and ex:Nadine_Labaki.
- **Lambda Variables**: Similar to *existential variables*, they are ungrounded nodes acting as a placeholder for a set of entities which are the potential answer to the query. The are represented by shaded circles in our example, and can have entities like ex:John_Wick (a movie) and ex:Rain (a TV series) mapped to it.
- **Auxiliary Functions**: Auxiliary functions are applied over the set of entities mapped to any ungrounded node (i.e. *grounded* and *existential variables*) in the query graph. In our grammar, they can be of two types, namely, the cardinality function or a class constraint $f_{class} : \{e \in E | (e,$ rdf:type, $class) \in K\}$ where $(class,$ rdf:type, owl:Class$) \in K$. In our example, we can apply the class constraint function over the entities mapped to the existential variable to only include male actors, and to the entities mapped to the lambda variable, represented by rectangle, to only include movies. The cardinality constraint can be used for NLQs like "*How many movies have casted Beirut born male actors?*" by imposing a count constraint over the lambda variable.

Finally, a new flag is defined which determines whether the query graph is used to fetch the value of the lambda variable, or to verify whether the graph is a valid subset of the target KG. The latter is used in the case of Boolean queries like "*Did Keanu Reeves act in John Wick?*" We represent this decision with a flag instead of another node or constraint in the graph as it doesn't affect the execution of the query graph, but in the case of a Boolean query only inquires, post execution, whether the query had a valid solution.

Query Graph Representation: We represent the query graphs in a linear form so as to easily encode them in our ranking models. We linearize the directed graph by starting from one of the grounded entities and using +, − signs to denote the outgoing and incoming edges, respectively. We represent auxiliary functions with another flag along with the linearized chain. Externalizing the auxiliary functions enables us to remove the ungrounded nodes from the core chain, which is now composed of the grounded entity and relations prefixed by +, − signs. Finally, we replace the URIs of the grounded entities and predicates with their corresponding surface forms. Hereafter, we refer to this linearized representation as the *core chain* of a query graph. This representation also ensures that the query graph maintains textual similarity to the source NLQ, enabling us to use a wide variety of text similarity based approaches for ranking them. Fig. 5.3.b illustrates the core chain corresponding to the query graph in our running example. In our preliminary analysis we found that removing mentions of grounded entities in the core chain increased the performance of our approach. We thus exclude the grounded entities from the final core chain representation. Since the grounded entities are the same for a given NLQ, no information is lost from the core chain candidate set upon doing this.

5.7 Approach for Complex Questions

We treat KGQA as the task of generating and ranking query graph candidates w.r.t. a given NLQ. For instance, given the question "What is the population of the capital of Germany?", we would like a ranking model to assign a higher score to "`+ capital + population`" than "`+ capital + mayor`", where "+" indicates that the relation must be followed in the forward direction. More formally, given a question Q and a set of candidate core chains $C^1 \dots C^N$, we select the most plausible core chain as follows:

$$C^* = \text{argmax}_{C^i} \, \mathsf{sim}(Q, C^i) \ , \tag{5.4}$$

where $\mathsf{sim}(\cdot,\cdot)$ is a function assigning a score to a pair of a NLQ and a core chain. We implement $\mathsf{sim}(\cdot,\cdot)$ as the dot product of two vectors produced by the encoder $\mathsf{enc}^\mathsf{q}(Q)$ and the core chain encoder $\mathsf{enc}^\mathsf{c}(C^n)$ respectively, i.e.,

$$\mathsf{sim}(Q, C^n) = \mathsf{enc}^\mathsf{q}(Q) \cdot \mathsf{enc}^\mathsf{c}(C^n) \ . \tag{5.5}$$

We train our ranking model with a pairwise loss function that maximizes the difference between the score of correct (positive) and incorrect (negative) pairs of NLQs and core chains, that is

$$L = \max(0, \gamma - \mathsf{sim}(Q, C^+) + \mathsf{sim}(Q, C^-)) \ , \tag{5.6}$$

where $\mathsf{sim}(Q, C^+)$ and $\mathsf{sim}(Q, C^-)$ are the scores for the correct and incorrect question-core chain pair, respectively.

We assume the entities mentioned in the NLQ to be given (but do not require exact entity spans i.e which tokens in the question correspond to which entity). In the next section (Sec. 5.7.1), we outline a mechanism for generating core chain candidates. Following that, we describe a novel core chain ranking model in Sec. 5.7.2. Furthermore, for a fully functioning QA system, additional auxiliary functions needs to be predicted. We define them, and outline our method of predicting them in Sec. 5.7.3.

5.7.1 Core Chain Candidate Generation

Core chains, as described in the previous section, are linearized subsets of query graphs which represent a path consisting of entities and predicates without the additional constraints. Working under the assumption that the information required to answer the question is present in the target KG, and that we know the entities mentioned in the question, we collect all the plausible paths of up to two hops from an arbitrary grounded entity node[10] to generate the core chain candidate set. Here, we use the term *hop* to collectively refer to a KG relation along with the corresponding $+/-$ sign indicating whether the relation is incoming or outgoing w.r.t. the entity.

We retrieve candidate core chains by collecting all predicates (one-hop chains) and paths of two predicates (two-hop chains) that can be followed from an arbitrary grounded node. In this process,

[10] Entity that has been linked in the question.

predicates are followed in both outgoing and incoming direction (and marked with a + and − in the chain, respectively). We further restrict our candidate set of core chains as follows: if two entities have been identified in the question, we discard the core chains which do not contain both the entities as grounded nodes. When applied, this step substantially decreases the candidate set while retaining all the relevant candidates. Finally, we drop the mention of entities from the core chain since every core chain thus generated will contain the same entities in the same position, and doing so leads to no information loss. Doing so enables our ranking models to retain the focus on comparing the predicates of the core chain to the question.

Although we limit the core chains to a length of two hops for the purposes of this study, this approach can easily be generalized to longer core chains. However, it may result in an additional challenge of handling a larger number of candidate core chains.

5.7.2 Slot Matching Model

To exploit the specific structure of the task, we propose an encoding scheme which partitions core chains into the aforementioned *hops*, and creates multiple, hop-specific representations of the NLQ, which we call *slots*. We then compare the hop (segments of a core-chain) representations with their corresponding slot (an encoded representation of the NLQ) to get the final score.

First, the question $Q = \{q_1 \dots q_T\}$ is encoded using a bidirectional LSTM (LSTM^q) resulting in the question encoding

$$[\hat{\mathbf{q}}_1 \dots \hat{\mathbf{q}}_T] = \mathsf{LSTM}^q(Q) \ . \tag{5.7}$$

Now, consider a core chain consisting of M hops. For each hop $j = 1, \dots, M$, we define a trainable slot attention vector $\mathbf{k_j}$ which is used to compute attention weights $\alpha_{t,j}$, individually for every hop j, over all the words $q_t, t = 1, \dots, T$ of Q. Then, a set of fixed-length question representations $\mathbf{q}_j$ are computed using the corresponding attention weights $\alpha_{t,j}$, that is

$$\alpha_{t,j} = \text{softmax}(\{< \hat{\mathbf{q}}_l, \mathbf{k}_j >\}_{l=1\dots T})_t \quad , \tag{5.8}$$

$$\mathbf{q}_j = \sum_{t=1}^{T} \alpha_{t,j} \cdot \hat{\mathbf{q}}_t \ . \tag{5.9}$$

We represent the core chains by separately encoding each hop by another LSTM (LSTM^c)

$$\mathbf{c}_j = \mathsf{LSTM}^c(C_j) \ , \tag{5.10}$$

where $C_j = [c_{j,1} \dots c_{j,T_j'}]$ is the sequence of words in the surface from of the predicate along with the $+/-$ signs, corresponding to the j^{th} hop of the core chain. Finally, $\mathbf{q}_1, \dots, \mathbf{q}_M$ and $\mathbf{c}_1, \dots, \mathbf{c}_M$ are concatenated to yield our final representation of the NLQ and the query graph ($\mathsf{enc}^\mathsf{q}(Q)$ and $\mathsf{enc}^\mathsf{c}(C)$), respectively, which is used in score function given in Eqn. (5.5), i.e.

$$[\mathbf{q}_1, \dots \mathbf{q}_M] = \mathsf{enc}^\mathsf{q}(Q) \tag{5.11}$$

$$[\mathbf{c}_1, \dots \mathbf{c}_M] = \mathsf{enc}^\mathsf{c}(C) \ . \tag{5.12}$$

Figure 5.4 summarizes the proposed approach.

Note that the model proposed here is not the same as *cross attention* between the input sequences

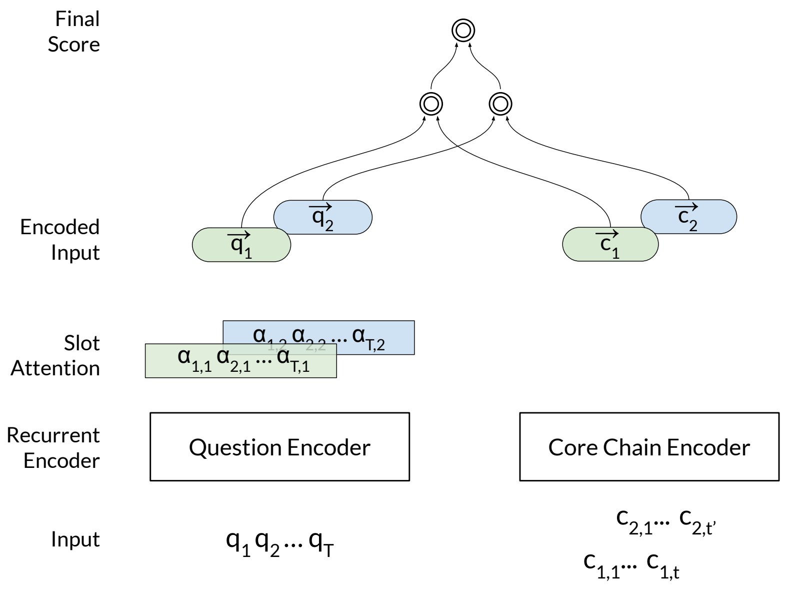


Figure 5.4: The slot matching model uses parameterized attention vectors to create j representations of the question, and compares each of them correspondingly with the j hops in a core chain. Here t and t' represent the number of words in each hop of the core chain, and $c_{t,1}$ is the t^{th} word in the first hop.

(as described by [193] which we also experiment with) as, in our case the attention weights aren't affected by the predicates in the core chain, as the encoder attempts to focus on *where* a predicate is mentioned in Q, and not *which* predicate is mentioned. In Sec 5.8.1, we discuss advantages of *slot based attention* over *cross attention* in further detail.

Using Pre-trained Transformers in the Slot Matching configuration:

[7] demonstrate that the use of pre-trained bidirectional transformers (BERT) can provide improvements for numerous downstream NLP tasks. Motivated by their findings, we investigate whether they can positively impact the performance on our KGQA task as well.

In this subsection, we describe how we use BERT to encode the NLQ and the core chains in the slot matching model. In the simplest approach, we would simply replace the LSTM in Eqn. (5.7) and (5.10) with pre-trained transformers and keep the rest of the model unchanged.

However, [7, 57] prescribe converting structured inputs into an single ordered sequence. We thus concatenate our inputs: (i) a question $Q = \left[q_1 \ldots q_T\right]$ of length T, and (ii) the M hops of a core chain $C = \left[\, [\, c_{1,1} \ldots c_{1,T'_1}\,] \cdots [\, c_{M,1} \ldots c_{M,T'_M}\,]\,\right]$ into a sequence of length $l = T + \sum_{j=1}^{M} T'_j$ (excluding sequence delimiters), and pass it through the transformer. Concretely, we use [7]'s input encoding scheme: we (1) prepend the sequence with a `[CLS]` token, (2) append the `[SEP]` separator token at the end of the question and (3) separate the different predicate surface forms in the appended candidate core chain with the same `[SEP]` token. The input to the transformer corresponding to our previous example then looks like this: "`[CLS]` *Name some movies starring Beirut born male actors* `[SEP]` + *capital* `[SEP]` + *population* `[SEP]`". [7] use the output of the transformer at first position (corresponding to the `[CLS]` token) for classification. Instead, for our slot matching transformer, we replace $[\hat{\mathbf{q}}_1 \dots \hat{\mathbf{q}}_T]$ in eq. (5.7) with the question portion of the transformer's outputs. Applying eq. (5.8) and (5.9) as before yields a set of slot-specific question encodings $\mathbf{q}_1, \dots, \mathbf{q}_M$. Slot-specific hop encodings $\mathbf{c}_1, \dots, \mathbf{c}_M$ are obtained from the same sequence of output vectors of the transformer by taking the representation at the `[SEP]` delimiter preceding the j^{th} hop. Given these encodings, the score for a question-chain pair is computed as before. The model is depicted in Fig 5.5.

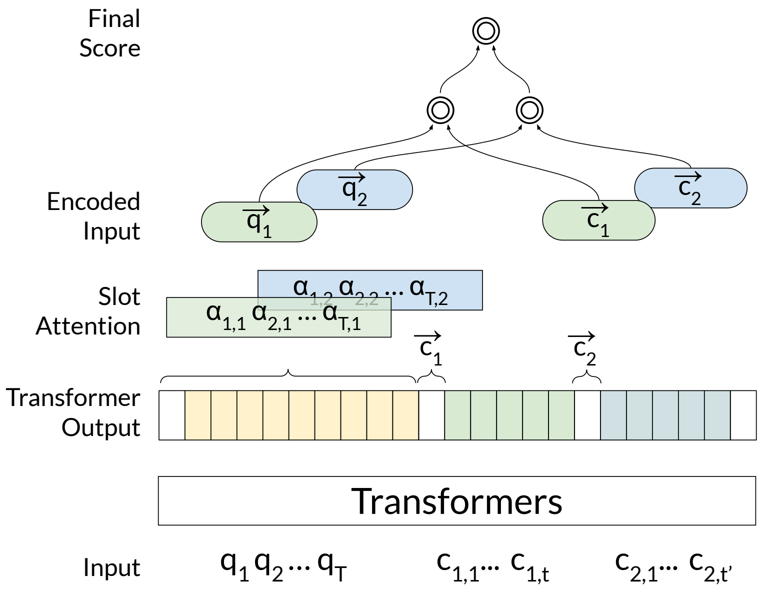

Final Score
Encoded Input
Slot Attention
Transformer Output
Transformers
Input
$\vec{q}_1$ $\vec{q}_2$ $\vec{c}_1$ $\vec{c}_2$
$\alpha_{1,1}\ \alpha_{2,1} \dots \alpha_{T,1}$
$\alpha_{1,2}\ \alpha_{2,2} \dots \alpha_{T,2}$
$q_1\ q_2 \dots q_T$
$c_{1,1} \dots c_{1,t}$
$c_{2,1} \dots c_{2,t'}$

Figure 5.5: Illustration of the transformer model in the slot matching configuration.

5.7.3 Predicting Auxiliary Functions

In this section, we describe our approach towards learning to predict the auxiliary functions used for constructing a complete query graph. We begin by predicting the intent of the question. In both the datasets considered in our experiments, a question can ask for the cardinality of the lambda variable, ask whether a certain fact exists in the KG, or simply ask for the set of values in the lambda variable. Further, this division, hereafter referred to as *count*, *ask* and *set* based questions, is mutually exclusive. We thus use a simple BiLSTM based classifier to predict the intent belonging to one of the three classes.

Next, we focus on detecting class based constraints on the ungrounded nodes of the core chain, as described in Sec. 5.6 as f_{class}. We use two different, separately trained models to predict (i) whether such a constraint exists in the NLQ, and if so, on which variable, and (ii) which *class* is used as a constraint. The former is accomplished with a simple binary BiLSTM classifier (i.e. constraint exist or not), similar to the aforementioned intent classifier. For the latter, we use a BiLSTM based pairwise ranking model trained in a similar setting as described in Eqn. (5.6).

We now have all the information required to construct the query graph, and the corresponding executable query. For brevity's sake, we omit the algorithm to convert query graphs to SPARQL here, but for limited use cases such as ours, simple template matching (based on the $+/-$ signs of the selected core chain, the class constraint, and the result of the intent classifier) shall suffice.

5.8 Experiments on Complex Questions

In a first set of experiments we compare the KGQA performance of the proposed slot matching model with some baseline models as described in the following section. After that, we describe experiments investigating transfer learning across KGQA datasets and from pre-trained transformers.

5.8.1 Approach Evaluation

Our first experiment focuses on investigating the performance of the **Slot Matching (LSTM)** model. As a baseline we use a simple neural ranking model where we replace $\mathsf{enc}^{\mathsf{q}}$ and $\mathsf{enc}^{\mathsf{c}}$ with a single layered bidirectional LSTM (**BiLSTM**).

In recent years, several papers have taken an alternate approach to semantic parsing by treating KGQA as a problem of semantic graph generation and ranking of different candidate graphs. [128] compare a set of manually defined query templates against the NLQ and generate a set of grounded query graph candidates by enriching the templates with potential predicates. STAGG [84] create grounded *query graph* candidates using a staged heuristic search algorithm, and employ a neural ranking model for scoring and finding the optimal semantic graph. The approach we propose in this work is closely related to this. Yu et al. [123] use a hierarchical representation of KG predicates in their neural query graph ranking model. They compare their results against a local sub-sequence alignment model with cross-attention [193] (originally proposed for the natural language inferencing task [194]). We also compare our model to those proposed by Parikh et al. [193] (decomposable attention model, or **DAM**), Yu et al. [123] (hierarchical residual model, or **HRM**), and Kim [195] which uses a multi-channel convolutional neural network (**CNN**).

Datasets

Our models are trained and evaluated over the following two KGQA datasets:
LC-QuAD [21] is a gold standard question answering dataset over the DBpedia 04-2016 release, having 5000 NLQ and SPARQL pairs. The coverage of our grammar covers all kinds of questions in this dataset.
QALD-7 [22] is a long running challenge for KGQA over DBpedia. While currently its 9^{th} version is available, we use QALD-7 (Multilingual) for our purposes, as it is based on the same DBpedia release as that of LC-QuAD. QALD-7 is a gold-standard dataset having 220 and 43 training and test questions respectively along with their corresponding SPARQL queries. Some of the questions in the dataset are outside the scope of our system. We nonetheless consider all the questions in our evaluation.

Evaluation Metrics

We measure the performance of the proposed methods in terms of their ability to find the correct core chain, as well as the execution results of the whole system. For core chain ranking, we report Core Chain Accuracy (CCA) and Mean Reciprocal Rank (MRR). Based on the execution results of the whole system (including auxiliary functions), we also report Precision (P), Recall (R), and F1.

Training

Our models are trained with negative sampling, where we sample 1000 negative core chains per question, along with the correct one, for every iteration. We train our models for a maximum of 300 epochs, using a 70-10-20 split as train, validation and test data over LC-QuAD[11]. QALD-7 has a predefined train-test split, and we use one eighth of the train data for validation. We embed the tokens using Glove embeddings [40], and keep the relevant subset of the embeddings trainable in the model. We use Adam optimizer with an initial learning rate of 0.001 and set the margin (γ) in the pairwise loss function as 1.

We share parameters between $\mathsf{enc}^{\mathsf{q}}$ and $\mathsf{enc}^{\mathsf{c}}$ in the BiLSTM, CNN and DAM models, since in these models, input sequences are processed in the same manner, while the same does not hold for the slot matching, or HRM. We illustrate the impact of this choice on model performance towards the end of this section.

Results

In our experiments, the slot matching model performs the best among the ones compared, suggesting that different attention scores successfully create suitably weighted representations of the question, corresponding to each hop. This is further reinforced upon visualizing attention scores, as presented in Fig 5.6, where we notice that the different attention *slots* focus on different predicate spans in the question. While the decomposable attention model (DAM) proposed by [193] also uses attention, its performance generally lags behind the slot matching model. DAM's cross-attention between question and core chain leads to a summarized question representation that is dependent on the candidate core chain and vice versa. On the other hand, the slot matching approach merely attempts to learn to extract important, relation-specific parts of the NLQ, prior to seeing a specific core chain, which judging by our experiments seems to be a better model bias and might help generalizing. The hierarchical residual model (HRM) [123] is second best in our comparison, suggesting that pooling relation and word level encodings is a promising strategy to form core chain representations. The competitive performance of the BiLSTM model is in coherence with recent findings by [118], that a simple recurrent model can perform almost as well as the best performing alternative.

All models generally exhibit poor performance over QALD-7, which is understandable given the fact that QALD-7 has only 220 examples in the training set, which is 20 times smaller than LC-QuAD. We will show in the next section that transfer learning across datasets is a viable strategy in this case to improve model performance.

Error Analysis:

We now illustrate the effect of different characteristics of the test data on the model performance.

Effect of number of core chain candidates: In our ranking based approach, the difficulty of selecting the correct core chain depends on the number of core chain candidates, which can be disproportionately large for questions about well-connected entities in DBpedia (e.g. `dbr:United_States`). In order to investigate its effect, we plot the core chain accuracy (CCA) vs. the number of core chain candidates,

[11] That is, we use the first 70% of dataset, as made available on `https://figshare.com/projects/LC-QuAD/21812` by [21], to train our models. Next 10% is used to decide the best hyperparamters. The metrics we report in the rest of this section are based on the model's performance on the last 20% of it.

	LC-QuAD					QALD-7				
	CCA	MRR	P	R	F1	CCA	MRR	P	R	F1
BiLSTM [42]	0.61	0.70	0.63	0.75	0.68	0.28	0.41	0.20	0.36	0.26
CNN [195]	0.44	0.55	0.49	0.61	0.54	0.31	**0.45**	0.20	0.33	0.25
DAM [193]	0.57	0.66	0.59	0.72	0.65	0.28	0.40	0.20	0.36	0.26
HRM [123]	0.62	0.71	0.64	0.77	0.70	0.28	0.40	0.15	0.31	0.20
Slot-Matching (LSTM)	**0.63**	**0.72**	**0.65**	**0.78**	**0.71**	**0.31**	0.44	**0.28**	**0.44**	**0.34**

Table 5.6: Performance on LC-Quad and QALD-7. The reported metrics are core chain accuracy (CCA), mean reciprocal rank (MRR) of the core chain rankings, as well as precision (P), recall (R), and the F1 of the execution results of the whole system.

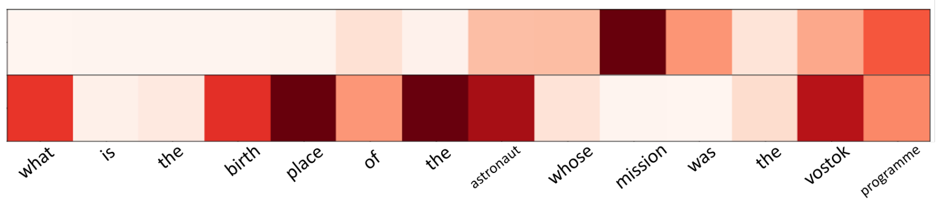


Figure 5.6: Visualized attention weights (darker color corresponds to larger attention weights) of the slot matching question encoder for the question "*What is the birth place of the astronaut whose mission was the vostok programme?*" The two rows represent different slot attention scores. One can see that first puts a higher weight on *mission* while the second (beside others) on *birth place*.

for all the models we experiment with, in Fig. 5.7(a). Upon inspection, we find the core chain accuracy to be inversely correlated to the number of core chain candidates. Specifically, we find that the performance of the BiLSTM, HRM and the slot pointer model (the three best performing ones in Sec. 5.8.1) remain almost the same for as many as 2000 core chain candidates per question. Thereafter, the BiLSTM and HRM models' accuracy declines faster than that of the proposed slot matching model, giving a competitive edge to the latter.

Effect of length of questions: We noticed that it is relatively easier for the models to answer longer questions. To better understand this phenomenon, we plot the core chain accuracy w.r.t. the length of questions for all the models in Fig. 5.7(c), and the frequency of questions w.r.t. their lengths in Fig. 5.7(d).

We find that longer questions are more likely to contain two entity mentions than just one. This hints to the fact that the number of candidate core chains reduces accordingly, as every valid core chain candidate must include all entity mentions of the question, which simplifies the ranking process as detailed in Sec. 5.7.1.

Effect of length of core chains: Inherent biases in the data might make our models more inclined to assign higher ranks to paths of a certain length, at the expense of selecting the correct path. We thus compose a confusion matrix representing the number of hops in the ground-truth and predicted core chains, which we hypothesize can help detect these biases. We find that none of our models suffer from this issue. As an example, we visualize the confusion matrix for the slot matching model's predictions over LC-QuAD's test split in Fig. 5.7(b).

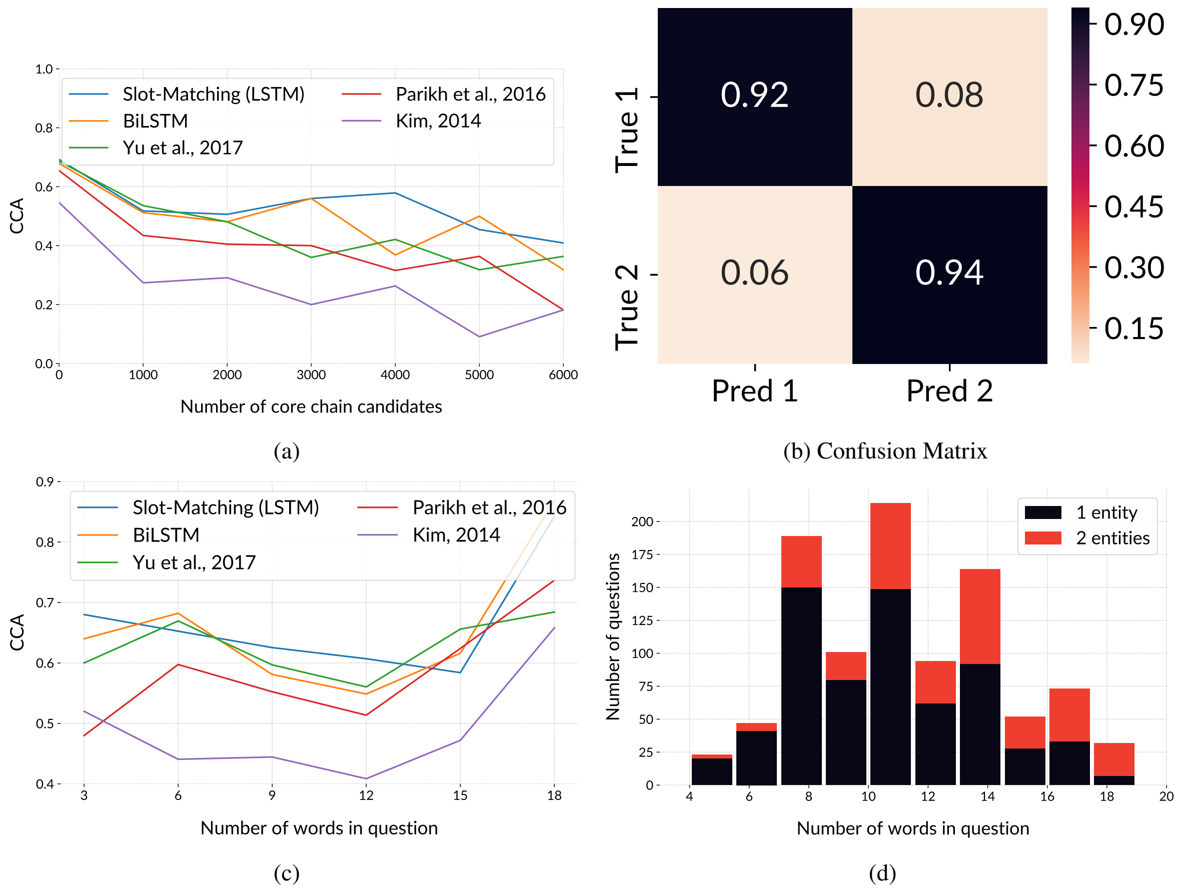


Figure 5.7: Here, (a) shows the decrease in accuracy with increasing number of candidates for all the rankings models in Section 5.8.1; (b) is a confusion matrix representing the number of hops in true and predicted core chains for the test data of LC-QuAD for the slot matching model. (c) the relation of model accuracy w.r.t. question length. And (d) a histogram depicting the distribution of questions in LC-QuAD's test split w.r.t. their question lengths. Here, the proportion of questions with two entity mentions are depicted with red color.

Further Analysis:

In order to better assess the impact of different parts of the system we perform a series of analysis over the simplest baseline (BiLSTM), and the best performing model (Slot Matching (LSTM)). For brevity's sake we only report the core chain accuracy in these experiments. Unless specified otherwise, the hyperparameters will be the same as mentioned in the experiment above.

Ablation Study: In order to better understand the effect of slot-specific attention over the question in the slot matching model, we experiment with a simpler model where we use the same attention scores for each slot. Effectively, this transforms our enc^q to a simpler, single-slot attention based encoder.

In our experiments, we find that the model yields similar results to that of the baseline BiLSTM model, i.e. 60.3%, which is considerably worse (- 2.8%) than the regular slot matching model with two slots. Our experiments illustrate several mechanism of using attention for the task, including no attention (BiLSTM, 61.4%), with attention (single slot, 60.3%), with multiple slots of attention (slot matching model, 63.1%), and with cross attention (DAM, 56.8%).

Parameter Sharing between Encoders: In the primary experiment, the BiLSTM model shares parameters between

$\mathsf{enc}^{\mathsf{q}}$ and $\mathsf{enc}^{\mathsf{c}}$, while the slot matching model doesn't. To show the effect of parameter sharing between encoders, we retrain both models in both settings (with and without parameter sharing).

Sharing encoders leads to a *decrease* of 2.9% (60.4% from 63.1%) in CCA of the slot matching model. Conversely, sharing encoders *increases* the performance of the BiLSTM model by 3.1% (61.4% from 58.3%). In the BiLSTM's case, learning a mapping that captures the equivalence of questions and core chains is not hindered by sharing parameters because the model structure is the same on both sides (simple encoders). Sharing parameters in this case could help because of the decrease in the total number of parameters. In the case of the slot matching model, however, sharing the parameters of the encoders would require the encoder to be usable for both the attention-based summary of the question encoder as well as the simple encoder for each hop (where the latter processes much shorter sequences) which leads to a performance bottle neck.

5.8.2 Transfer Learning across KGQA datasets

As mentioned above, all models generally show poor performance when trained solely on QALD-7 due to a more varied and significantly smaller dataset. Therefore, we hypothesize that pre-training the models on the much larger LC-QuAD dataset might lead to a significant increase in performance. To that end, we perform the following fine-tuning experiment: we pre-train our ranking models over LC-QuAD, and then fine-tune and evaluate them over QALD-7. We set the initial learning rate to 0.0001 (which is an order of magnitude less than in the experiments in Section 5.8.1), and experiment with custom learning rate schedules, namely *slanted triangular learning rate* (sltr) proposed in [190], and *cosine annealing* (cos) proposed in [196]. We keep the hyperparameters of *sltr* unchanged, and set the number of cycles for *cos* to 3 based on the performance on the validation set.

Learning Rate	BiLSTM	Slot Matching (LSTM)
constant	0.37	0.37
sltr	**0.39**	**0.42**
cos	0.25	0.28

Table 5.7: CCA for the fine-tuning experiment where we pre-train on LC-QuAD and fine-tune over QALD-7. The initial learning rate is 10^{-3} for all configurations.

We conduct the experiment on the BiLSTM and the Slot Matching (LSTM) ranking model, and only report the *core chain accuracies* (CCA) as the rest of the system remains unchanged for the purposes of this experiment.

We find that *fine-tuning* a ranking model trained over LC-QuAD leads to a substantial (~ 11%) increase in performance on QALD-7 compared to non-pre-trained models. Interestingly, we find that the results of the fine-tuning experiment are sensitive to the learning rate schedule used. While constant learning rate provides a relatively comparable performance w.r.t. *sltr*, using the cosine annealing schedule adversely affects model performance.

We report the results of this experiment in Table 5.7. In summary we conclude that transferring models across KGQA datasets via simple fine-tuning is a viable strategy to compensate for the lack of training samples in the target dataset.

5.8.3 Transfer Learning with pre-trained Transformers

For our transformer based slot matching model, we use a transformer, initialized with the weights of BERT-Small[12], instead of an LSTM, as discussed in Sec 5.7.2. The transformer has 12 layers of hidden size 768 and 12 attention heads per layer. Following [7], we set dropout to 0.1. We train using Adam with initial learning rate 0.00001. Table 5.8 shows the performance of the pre-trained transformer (**BERT**), used as in [7] as well as the pre-trained transformer in the slot matching configuration (**Slot Matching (BERT)**). For BERT, we follow the sequence pair classification approach described by [7].

	QALD-7	LC-QuAD
BERT	**0.23**	0.67
Slot Matching (BERT)	0.18	**0.68**

Table 5.8: CCA for slot matching model, as proposed in Sec 5.7.2 initialized with the weights of BERT-Small, compared with regular transformers initialized with the same weights.

Through this experiment we find that using pre-trained weights immensely improves model performance, as both transformer based models outperform the ones in Section 5.8.1. Additionally, we find that the augmentations we propose in Sec 5.7.2 are beneficial for the task, improving CCA on LC-QuAD by 1.4% relative to regular pre-trained transformers. However, both models exhibit poor performance over QALD-7. The cause of this degradation is unclear to us at this time. An important difference between QALD and LC-QuAD that could help explain this is the extremely small number of training examples (220) in QALD-7, which, coupled with the complexity of the questions could lead to much more challenging training, especially with such an over-parameterized model as BERT.

5.9 Conclusion

In this chapter we demonstrated experimentally that PLMs can lead to a significant improvement over the non-pretrained baselines. Our BERT-based model significantly outperforms a strong BiLSTM baseline on the learning problems of relation classification and entity span prediction for simple questions. Moreover, the pre-trained transformer shows less performance decrease when confronted with less training data, as can be seen in our limited-data study, which to the best of our knowledge is the first ever conducted for the SimpleQuestions dataset. The final results on the full SimpleQuestions task are competitive with the current state-of-the-art.

In this chapter, we also studied the performance of various neural ranking models on the KGQA task. First, we propose a novel task-specific ranking model which outperforms several existing baselines in our experiments over two datasets. An error analysis shows that the model performs especially well on smaller candidate sets and for longer questions which highlights its high potential for answering complicated questions. Second, we present an extensive study of the use of transfer learning for

[12] as provided by the authors at https://github.com/google-research/bert

KGQA. We show that pre-training models over a larger task-specific dataset and fine-tuning them on a smaller target set leads to an increase in model performance. We thereby demonstrate the high potential of these techniques for offsetting the lack of training data in the domain. Finally, we propose mechanisms to effectively employ large-scale pre-trained state of the art language models (BERT [7]) for the KGQA task, leading to an impressive performance gain on the larger dataset.

Our comparison of a fully pre-trained transformer to a BiLSTM-based model where only the word embeddings have been pretrained (Glove) might not yield a fair comparison between the two architectures (transformer vs BiLSTM). Further insights could be gained by analyzing the performance of a BiLSTM, which has also been pretrained as a language model (maybe combined with other tasks) in the future. Here instead, our aim was to provide evidence that the use of neural networks pre-trained as language model is beneficial for knowledge graph-based question answering, in particular for SimpleQuestions, a usecase not included in the original BERT evaluation and, to the best of our knowledge, not yet explored in the literature.

Even though BERT improves upon our BiLSTM baseline on SimpleQuestions, the improvements in the full data scenario might not justify the significantly longer training and inference times and memory requirements. These practical concerns, however, could be mitigated by practical tweaks and future research. Furthermore, with fewer data the performance increases w.r.t. the baseline become more spectacular, indicating that using pre-trained networks like BERT might be essential for achieving reasonable performance in limited data scenarios. Such scenarios are common for datasets with more complex questions. Therefore, we believe pretrained networks like BERT can have a bigger impact for complex KGQA (even when training with all data available). However, from our experiments with the very small QALD-7 dataset, we found that BERT-based models perform worse. While we can not explain this finding, we believe that training difficulties due to the extremely small yet complex dataset and the over-parameterized model could (partially) be to blame.

CHAPTER 6

Insertion-based decoding for semantic parsing

"I always like going south. Somehow it feels like going downhill."

– Treebeard, from *The Lord of the Rings: The Two Towers*

In this chapter, we investigate insertion-based decoding for semantic parsing. Insertion-based decoding may (1) improve the efficiency of decoding by requiring fewer decoding steps and (2) may affect the accuracy because of different independence assumptions (compared to left-to-right decoding) and other inductive biases.

Sequence generation is usually based on a left-to-right autoregressive decoder that decomposes the probability of the entire sequence y conditioned on x (x can be empty) as the product $p(y|x) = \prod_{i=0}^{N} p(y_i|y_{<i}, x)$. At each decoding step, the decoder model predicts the next token y_i based on the previously generated outputs $y_{<i}$ and the input x. This approach to decoding sequences is linear in sequence length: the number of decoding steps necessary to produce the sequence is equal to the length of the sequence. In addition, every generated token y_i is conditioned on all the tokens $y_{<i}$ generated so far, which may encourage learning spurious patterns and thus negatively affect generalization to novel data.

Recently, several works have proposed non-autoregressive decoders that are sub-linear. This allows to decode a sequence using fewer decoding steps than the length of the sequence and can thus greatly speed up inference, especially for longer sequences [23, 96, 98–100]. In particular, the Insertion Transformer of Stern et al. [23] uses insertion operations to iteratively expand the sequence by inserting tokens between existing tokens, thus enabling the generation of multiple tokens in a single decoding step and requiring only $O(\log_2 n)$ decoding steps for a sequence of n tokens.

In this chapter, we investigate how well an insertion-based sequence decoder works for semantic parsing, where we re-implement the method proposed in Stern et al. [23]. Moreover, we also develop a novel insertion-based decoding method that is specifically tailored for trees and compare it against the insertion-based sequence decoder, as well as normal sequence-to-sequence and sequence-to-tree models. The insertion-based tree decoder that we propose here does not need to explicitly decode structure tokens. Moreover, it can achieve a best-case complexity *below* $O(\log_2 N)$[1] in terms of the

[1] The exact best possible speed-up heavily depends on the data.

number of decoding steps and guarantees that all intermediate outputs are valid trees. In addition, in the investigated approaches, the tokens that are generated in parallel are predicted independently. On the one hand, this changes the independence relations between the output variables when compared to left-to-right decoding. Higher degree of independence between output variables may reduce the learning of spurious patterns over the output variables. However, it would not decrease the learning of spurious patterns from the input since the entire input is still observed. On the other hand, the introduced independence relations may negatively affect performance if the "correct" patterns have been broken. Moreover, other design choices and resulting inductive biases may increase or decrease task accuracy. In particular, there may be a difference when treating the trees in semantic parsing as sequences or as trees. Our proposed tree-based insertion transformer decodes and models trees differently than the sequence-based insertion transformer [23], which may affect accuracy. These considerations further motivate the investigation of parallel insertion-based decoding and tree-based insertion decoding.

We run experiments on the Overnight dataset for semantic parsing. Since the formal language queries in semantic parsing can often be represented as trees (e.g. abstract syntax tree), semantic parsing is a particularly interesting task for the evaluation of the presented decoding approach.

The contributions of this work are:

- a transformer-based decoding algorithm that uses insertion operations specifically tailored for decoding trees,
- a novel transformer architecture that uses novel tree-based relative positions,
- and an evaluation of the proposed algorithm and model on the well-known Overnight dataset, and a comparison against a strong non-autoregressive baseline.

This chapter is based on the following publication:

9. **Denis Lukovnikov**, Asja Fischer. *Insertion-based tree decoding.* In the Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics: Findings (ACL Findings 2021). 2021. This work was also presented in the 5th Workshop on Structured Prediction for NLP at ACL 2021.

The investigation presented in this chapter corresponds to Contribution 3, which addresses the following research question:

RQ3: Does insertion-based decoding improve accuracy and how much does it decrease the number of decoding steps?

The rest of the chapter is organized as follows: We elaborate on the insertion-based sequence decoder in Section 6.1. We present the insertion-based *tree* decoder approach in Section 6.2. The experiments are presented in Section 6.3 and a discussion is presented in Section 6.4. We conclude in Section 6.5 with a conclusion and an outlook on future work. Related work regarding this topic is discussed in Section 3.1.6 in Chapter 3.

6.1 Insertion Transformer

Below, we give a very brief overview of the Insertion Transformer, which we use here as a baseline. Due to space constraints, we refer interested readers to the work of Stern et al. [23] for a more elaborate description of their model and training procedure.

Decoding approach. Rather than decoding autoregressively left-to-right (LTR), the insertion transformer decodes sequences by using insertion operations. For example, consider the sequence "*A B C D E F G*". LTR decoding would require at least seven decoding steps, producing some left-aligned subsequence at every step (e.g. "A B C D" at the fourth step). In contrast, decoding using insertions allows to decode the same sequence using just three steps. Starting from the initial empty state " ", we first decode (1) "***D***", then (2) "***B*** *D* ***F***" and finally (3) "***A*** *B* ***C*** *D* ***E*** *F* ***G***", where the bold faced tokens are the ones inserted in each step, respectively.

Model. The model we use in our experiments uses BERT [7] as the encoder and a standard transformer [18] with learned absolute position vectors as the decoder. In contrast to the vanilla transformer decoder however, the causal attention mask is not used. After encoding a subsequence using the transformer, the output layer concatenates the representations of two neighbouring tokens to build a representation for the insertion slot between the tokens. We normalize the probabilities per slot.

Training. Given training data consisting of pairs of input and output sequences (x, y), at every epoch, the training algorithm samples a subsequence $\hat{y}$ for every output sequence y. First, a length for the subsequence is drawn from a uniform distribution on $[0, |y|]$. Then, a subsequence $\hat{y}$ of the given length is randomly drawn from y. For example, for the sequence "A B C D E F G" and a randomly drawn length of 3, a sampled subsequence of length 3 could be "B D E".

The Insertion Transformer is then trained by optimizing a loss of the following form:

$$-\sum_{l}\sum_{i=i_l}^{j_l} w_{i,l} \log p(y_i, l|x, \hat{y}) \ , \tag{6.1}$$

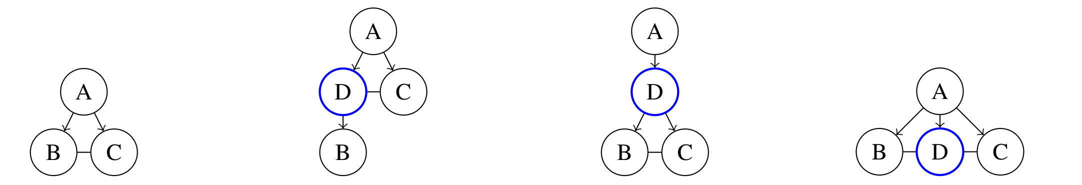
where i_l and j_l are the beginning and end positions of the subsequence to be decoded in slot l. Two variants of this loss function are proposed that differ in the strategy of assigning the weights $w_{i,l}$ to the different tokens and which are referred to as *uniform* and *binary*, respectively. In the uniform case, for a given l the weights $w_{i,l}$, $i = i_l, \ldots, j_l$ are equal and sum up to one. In the binary case, a larger weight is assigned to tokens closer to the center of slot l:

$$w_{i,l} = \frac{e^{-d_l(i)/\tau}}{\sum_{i'=i_l}^{j_l} e^{-d_l(i')/\tau}} \ , \tag{6.2}$$

where $d_l(i)$ is the distance between token i and the center of the span to be decoded in slot l, and τ is the *temperature*.

6.2 Tree-based Insertion Transformer

In this section, we propose a novel transformer-based method for non-autoregressive decoding of trees. The proposed method consists of (1) a novel transformer architecture, and (2) a novel insertion-based decoding procedure.



(a) Simple tree (b) Inserting D at the ancestral slot of B. (c) Inserting D at the descendant slot of A. (d) Inserting D at sibling slot between B and C.

Figure 6.1: Examples of insertions at different insertion slots.

6.2.1 Decoding

The proposed decoding algorithm is similar to the one proposed by Stern et al. [23] in that it uses insertion operations to expand the decoded structure. However, instead of using the implicitly defined insertion slots between two neighbouring tokens in a sequence, here, insertion slots are used that are placed between neighbouring nodes in the graph representing the tree.

As an example, consider the tree *(A B C)*[2], shown in Fig. 6.1(a). For sequence-based insertion decoding, we would take the linearization $(\cdot\, A \cdot B \cdot C \cdot\,)$, where four insertion slots are explicitly denoted by "·". The linearized representation clearly has some disadvantages. For example, performing some insertions at some slots can destroy the tree structure, e.g. inserting ")" between B and C yields *(A B) C)*, which is not a valid tree.

In this work, we propose a tree-based insertion decoding algorithm that defines insertion slots on edges between siblings, as well as on edges between parents and children. Following this approach, the example tree can be described by the following linearization with explicit insertion slots:

$$(\wedge A \vee \mid - \wedge B \vee - \wedge C \vee -) \quad ,$$

where we use three types of insertion slots: (1) ancestor insertion slots denoted by $\wedge$, (2) descendant insertion slots denoted by $\vee$, and (3) sibling insertion slots denoted by –, as described in more detail in the following. Note that the parentheses and the pipe symbol "|" are ignored by the model (when relative positioning is used) and is used mostly for notational and programming convenience. The pipe symbol separates the root portion of the "slotted" subtree string.

Ancestor insertion: If node D is used for the ancestor insertion slot $\wedge$ of B, which corresponds to the red slot in

$$(\wedge A \vee \mid - \wedge B \vee - \wedge C \vee -) \quad ,$$

we obtain the tree *(A (D B) C)* (see Fig. 6.1(b)). In other words, D replaces B and the entire subtree B is attached as a child of D.

Descendant insertion: If node D is used with a descendant insertion slot $\vee$ of A, which corresponds to the red slot in

$$(\wedge A \vee \mid - \wedge B \vee - \wedge C \vee -) \quad ,$$

then the child subtrees of A are moved to node D and the entire subtree D (that now contains the children of A) is attached to A to yield the tree *(A (D B C))*, which is depicted in Fig. 6.1(c).

[2] Trees are given in a Lisp-like notation in this work.

Sibling insertion: If node D is used with a sibling insertion slot "–" between B and C, which corresponds to the red slot in

$$(\wedge\ A \vee \mid - \wedge\ B \vee - \wedge\ C \vee -)\quad ,$$

then D is inserted as a child of A between B and C to yield the tree *(A B D C)* (see Fig. 6.1(d)).

These insertion actions can be used to decode any tree starting from a tree containing only a root node *(ROOT)*. Note that every decoding step is guaranteed to yield a valid tree, unlike in the sequence-based insertion decoder.

Decoder operation

In every step, the decoder takes the previous intermediate tree y_{t-1} (where y_0 = *(ROOT)*) and applies one or more insertion operations to expand the tree to the next intermediate tree y_t. To predict which insertion operations to execute at every available slot, the model described in the following sections encodes the *entire* input tree y_{t-1}. The prediction is then based on this encoding. This procedure is similar to the approach used by Stern et al. [23]. While more efficient methods that re-use computations should be possible, we leave an investigation of those for future work. The decoding process is terminated when all slots predict a slot closing actions indicating that no more nodes should be inserted, and thus $y_t = y_{t-1}$.

6.2.2 Model

We propose a novel architecture that takes advantage of the fact that the intermediate structures generated in all steps of the decoding algorithm are trees. While the encoder stays the same (i.e. BERT), the decoder of our model relies on a transformer-based tagger with tree-based relative positions and a special attention mask. The tree-based relative positions allow us to ignore structure tokens.

Computing Relative Positions in Trees

The proposed architecture uses **tree-based relative positioning** instead of absolute positional embeddings. The relative position is described by the **movements in the tree** that the path from node u to node v defines. For example, in the tree *(A (B (C D E) K (F (G H I J))))* depicted in Fig. 6.2, the relative position of node D to node I is given by "1↑ 2→ 2↓". This describes the following movement in the tree: starting in D go up one hop to reach parent C (1↑), then move right among the siblings two hops to reach F (2→), and finally move down two hops to reach I (2↓). The scheme is *insensitive to the order of children*, e.g., the relative position from B to K is "1↓", the same as for (B, C) and (B, F). To distinguish these cases, we propose to instead use **special relative position relations** "child_X" and "child_X_of" between parent and child nodes (where X denotes the position of the child among all children of its parent, e.g., X is 1 for node K). While this should improve local modeling of parent-child relations, it still does not add sufficient information to other paths, e.g., the relative positions for (D, H) and (D, J) are identical to that for (D, I)[3].

[3] We leave the investigation of better relative position encoding for trees for future work since it is non-trivial to retain constant time complexity of the transformer and have a fully expressive position encoding.

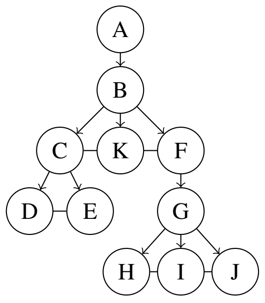


Figure 6.2: An example of a tree.

Using Relative Positions in the Model

With e_{ij} denoting the unnormalized attention scores, x_i the vector corresponding to element i in the input sequence and a^K_{ij} and a^V_{ij} the key (K) and value (V) vector representations of the relative position between the input elements at position i and position j, the following equations describe how the relative position vectors are used in the attention mechanism:

$$\begin{aligned} e_{ij} &= \left(x_i W^Q (x_j W^K)^T + x_i W^Q (a^K_{ij})^T\right) / \sqrt{d_z} \\ z_i &= \textstyle\sum_{j=1}^n \alpha_{ij}(x_j W^V) + \alpha_{ij} a^V_{ij} \quad , \end{aligned}$$

where z_i is the summary of the sequence for token i. This approach to incorporate relative positions is similar to that of Shaw et al. [48] but differs in the computation of a^K_{ij}'s and a^V_{ij}'s. A naive implementation would create an independent position embedding for every possible combination of the elements of a movement pattern. For an efficient implementation that is both faster and has fewer parameters, we separately consider the three parts of a movement pattern, embed them separately and add their embeddings:

$$a^K_{ij} = a^{K,\uparrow}_{ij} E^{K,\uparrow} + a^{K,\leftrightarrow}_{ij} E^{K,\leftrightarrow} + a^{K,\downarrow}_{ij} E^{K,\downarrow} \quad , \tag{6.3}$$

where $a^{K,\downarrow}_{ij}$, $a^{K,\uparrow}_{ij}$, and $a^{K,\leftrightarrow}_{ij}$ are one-hot vectors representing the components of the movement pattern, and $E^{K,\downarrow}$, $E^{K,\uparrow}$, and $E^{K,\leftrightarrow}$ are their corresponding embedding lookup matrices.

In case the relative position is not a movement pattern, as is the case for parent-child relations, we simply look up a single embedding vector. We also use relative positions *from* insertion slot positions to some of the node positions: (1) ancestor ($\wedge$) and descendant ($\vee$) slots use the "ancestors" and "descendant" relations, respectively, and (2) the sibling slots (–) use the "left_sibling" and "right_sibling" as well as a "parent" relation. Since insertion slot positions are not attended *to* (they are not used as keys in attention), we do not need relative positions *to* slot positions.

Note that this approach of encoding relative position in trees is similar to the one proposed by Anonymous [197]. Other work focusing on applying transformers on trees and more specifically position encoding in trees is that of Shiv and Quirk [198]. Other architectures to encode trees exist as well, such as the TreeLSTM [199], however, a transformer-based model enjoys greater parallelism and direct modeling of long-range dependencies.

Attention Mask

We use a custom attention mask pattern that prevents real query tokens from attending to structural tokens and insertion slots. Since we do insertion-based decoding, we don't use a causal attention mask. However, since structural information is already described by the relative positions (see above), we do not need to process the structure-describing tokens, such as parentheses. Thus, these tokens are masked both as keys and as queries and do not participate in the decoding process at all. Additionally, we use a special mask for the insertion slots (defined above) such that (1) other tokens can not attend *to* the slot tokens and (2) the slot tokens can *only* attend to their immediate neighbours.

6.2.3 Decoder Training

The decoder learns to imitate optimal trajectories. Let a tree y be the target output for the current input x. For training the decoder, a **sampling function** is applied to select a partial tree[4] y' from y. A **supervision function** is then used that determines the output distribution for every slot, which serves as the target in the training of the insertion model. This procedure is detailed in the following.

Optimal Trajectories

The insertion operations described earlier in Section 6.2.1 can be used to describe a set of actions that transform one partial tree into another. Given an output tree y and the initial tree y_0 that contains a single root node, i.e., y_0 = *(ROOT)*, a *trajectory* from y_0 to y can be defined as a sequence of in total T partial trees (states) $y_0, \ldots, y_T$ and corresponding actions $a_0, \ldots a_T$, such that, when the actions are applied in succession on y_0, they produce $y = y_T$. Each step thus corresponds to the application of an action a_t to the current partial tree y_t resulting in a new partial tree y_{t+1}, i.e. $y_{t+1} = \text{step}(a_t, y_t)$. Each action a_t is a set of atomic actions that insert a node at one of the insertion slots, which can be either ancestor, descendant, or sibling insertion slots. The atomic actions are described as tuples (k, w), where k is the slot in y_t where a node with label w will be inserted when the action is executed. Note that the type of insertion is characterized by the type of insertion slot.

While many trajectories exists that successfully reach y from y_0, we are interested in *optimal trajectories*, which are trajectories that minimize the number of decoding steps T that need to be performed.

Computing Optimal Trajectories: To practically compute trajectories where we try to minimize the number of steps taken, we rely on the fact that first decoding the most central nodes of a slot's subgraphs enables greater parallelization. For a given tree y_t at decoding step t that is a partial tree of the original tree y, we (1) align y_t and y and determine which nodes are allowed to be inserted in every slot in y_t and (2) compute which of these nodes is the best in order to minimize the number of decoding steps.

Computing allowed insertions: Given a partial tree y_t (e.g. Fig. 6.3) aligned with the original tree y (e.g. Fig. 6.2), we first compute the set of allowed insertions C_k for every slot k.

[4] Note that what we refer to as a partial tree is not the same as a subtree. A subtree retains all the descendants starting from a certain parent node. In contrast, we refer with partial tree to any tree consisting of nodes that can also be found in the original tree, and which can be extended to the full tree y by means of the defined insertion operations.

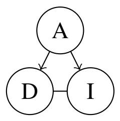


Figure 6.3: A partial tree of the tree in Fig. 6.2.

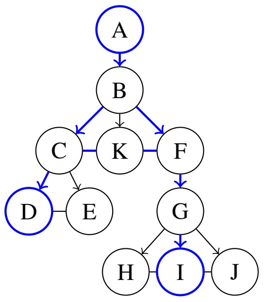


Figure 6.4: The tree from Fig. 6.2 and the nodes and edges covered by the partial tree from Fig. 6.3. Nodes with the same labels in Figures 6.2 and 6.3 are aligned.

For an **ancestor insertion slot** ($\wedge$) in y_t associated with some node n, C_k corresponds to the nodes from y on the path from n up to the *lowest used ancestor* of n. The lowest used ancestor $\text{lua}(n)$ of a node n from the partial tree y_t is the lowest[5] node in the original tree y that is an ancestor of n as well as *any* other node from y_t. That is, for node I in the example tree in Fig. 6.2, the lowest used ancestor of node I is the node B: $\text{lua}(I) = B$ and the set of allowed nodes for I's ancestor slot is $\{F, G\}$.

For a **descendant insertion slot** ($\vee$), the set of allowed nodes C_k is the set of all descendants of n in y , if n doesn't have children in the partial tree y_t. Otherwise, it's the set of all of its descendants in y that are also an ancestor for *all* the children of n in the partial tree y_t.

Finally, for a **sibling insertion slot** (–), to find the set of allowed nodes, we first find the lowest common ancestor in the original tree y of the slot's left node l and right node r. The lowest common ancestor $\text{lca}(n, n')$ of two nodes n and n' in a tree is the lowest node that is an ancestor of both n and n'. We first determine the set C of children of $\text{lca}(l, r)$ that are between the children of $\text{lca}(l, r)$ and that are also the ancestors of nodes l or r. The set of allowed nodes C_k for this slot is then the set C as well as all their descendants in y. In the example, the sibling insertion slot between D and I should accept only the node K since $\text{lca}(D, I)$ is B and K is the only node between C and F, which are the ancestors of D and I, respectively, that are also children of $\text{lca}(D, I)$. [6]

Computing best insertions: Now, for each insertion slot k in y_t, we are given the set C_k of nodes allowed to be inserted. The node $n \in C_k$ to be inserted in slot k in order to minimize the number of decoding steps is the most central node in C_k.

The centrality of a node is computed over the subgraph G_k of the original tree y that contains the nodes in C_k, as well as their descendants that are not separated by the partial tree. The *closeness*

[5] Lowest and highest refer to tree depth, i.e., the root node is the highest node in the tree.

[6] Note that inserting E between D and I will lead to a tree from which we can't recover the original tree since D, E and I now assume the same parent and there is no action defined to separate them under different parents.

centrality is used:

$$H_{G_k}(n) = \frac{1}{\sum_{n' \in G_k} d(n, n')} \ , \tag{6.4}$$

where $d(n, n')$ is the distance between nodes n and n' in the graph of the original tree, which is the minimum number of steps necessary to reach n from n'.

The node picked to be inserted into some slot k is then the one with highest centrality for the slot.

Computing best actions: The best action for a certain partial tree y_t of y then consists of the insertions that are the best for every of its insertion slots. If C_k is empty for some insertion slot, a dummy insertion operation is used that does not insert anything.

Partial Tree Sampling

Rather than sampling all possible partial trees, which would be equivalent to the method described by Stern et al. [23], we use a different method that samples only from the most optimal trajectory. The partial trees that are used for training are only those that occur on one of the optimal trajectories. For efficiency reasons, we precompute a certain number (5 in our experiments) of trajectories, where we randomly sample when ties occur in the centrality measure, and reuse these trajectories throughout training.[7] Sampling more than one trajectory could reduce the exposure bias.

Supervising Partial Trees

To produce the target distribution for a slot k, we take the nodes $n \in C_k$ computed as valid insertions, as well as their centralities. Then, we rank the nodes in C_k by centrality scores, where the most central node is the highest-ranked one, receiving rank value 0. Ties in centrality are broken by favouring nodes that are lower in the tree y and further ties are broken alphabetically (based on node label). The target distribution for a slot k is then computed using a softmax:

$$p_k(n) = \frac{e^{-\text{rank}_k(n)/\tau}}{\sum_{n' \in C_k} e^{-\text{rank}_k(n')/\tau}} \ , \tag{6.5}$$

where τ is a temperature hyperparameter. $p_k(n)$ is zero if n is not in C_k and $\text{rank}_k(n)$ is rank value given to node n.

For each slot k the model outputs a predictive distribution $\pi_k(n)$. Given the target distributions for all slots the training loss is the sum of the Kullback-Leibler (KL) divergences between the target and predictive distributions for all slots:

$$-\sum_k \sum_{n \in \mathcal{G}} p_k(n) \log \frac{\pi_k(n)}{p_k(n)} \ , \tag{6.6}$$

where $\mathcal{G}$ denotes the set of all possible node labels.

[7] Note that only one of the partial trees y_t of some example is used in a single epoch.

6.3 Experiments

We run experiments on the OVERNIGHT [200] dataset and report the results in Tables 6.2 and 6.3.

Evaluation: The metric reported is logical form accuracy, which we compute by (1) taking the predictions of the models, (2) balancing parentheses on the left and on the right and (3) computing whether the trees are the same.[8]

Dataset description and statistics: The OVERNIGHT dataset consists of pairs of natural language questions and corresponding formal language queries from 8 distinct domains, such as "publications" and "restaurants". The dataset was generated as follows: (1) first, a grammar was defined, which consists of a general part applicable for any domain, and a domain-specific part that specifies a *seed lexicon* mapping between predicates and NL. Then, (2) a number of examples was generated, which at this point consist of (i) a canonical utterance and (ii) a formal query. Finally, (3) the canonical utterances are paraphrased by Amazon Mechanical Turk workers to generate more natural examples.

See Table 6.1 for dataset statistics.

Domain	#Train	#Test	Avg. tree size	Avg. tree depth	Avg. seq. length
cal.	669	168	17.5	5.7	39.1
blo.	1596	399	18.8	5.8	42.1
hou.	752	189	17.3	5.6	38.8
res.	1325	332	16.3	5.4	37.1
pub.	640	161	18.3	5.8	41.1
rec.	864	216	16.1	5.7	36.3
soc.	3535	884	25.3	8.7	57.3
bas.	1561	391	19.8	8.2	45.1

Table 6.1: Statistics for the OVERNIGHT dataset after simplification.

An example from the "publications" domain from the OVERNIGHT dataset is the following.

The natural language question is:

"*find an article published in 2004*".

The formal query corresponding to this question is (after preprocessing):

Listing 6.1: Example query.

```
(call_SW_listValue
  (filter
    (call_SW_getProperty
      (call_SW_singleton
```

[8] The trees are considered the same if every node's children occur in both trees in the same order, if the node's children should be ordered (for example argmax expressions) and if every node's children occur in both trees in *any* order for nodes of which the children should not be ordered (for example SW:concat and a collection of filter conditions).

```
        (en.article ))
      (string
        (!type )))
    (condition
      (string
        (publication_date ))
      (string
        (= ))
      (date
        (2004 )
        (-1 )
        (-1 )))))
```

See Appendix B for the intermediate partial trees when decoding this example using tree-based insertion operations described earlier.

Data preprocessing: We notice that many examples in the Overnight dataset contain nested filters. This fact was ignored during training and evaluation of some models in previous work [93, 201]. During evaluation, an example was considered wrong if the filters were decoded in a different nesting order. For this reason, we use slightly simplified logical forms for the Overnight dataset that remove nesting between filters). More specifically, the simplified form is obtained by (1) merging the "call" node with the function of the call (e.g. replacing *(call SW:someFunction ...)* with *(call-SW:someFunction ...)*) and (2) merging nested filters into a multi-conditional expression where the order of conditions doesn't matter. Note that these simplifications do not change the meaning of the query. During training, we sort the different parts of a multi-conditional filter expression alphabetically and during evaluation, we consider the different conditions as unordered and accept them in any decoded order. We use this simplified version for all further experiments.

For the sequence-based decoder, we found that it is necessary to use **numbered tokens** to to successfully train the model. We simply replace every token "X" by a token "X_d", where d specifies how many tokens "X" have been observed before and including the current token "X". For example, this would transform "(A (B C) (B D))" into "$(_1$ A_1 $(_2$ B_1 C_1 $)_1$ $(_3$ B_2 D_1 $)_2$ $)_3$". This ensures that every token in the sequence is unique. We use numbered tokens in all the experiments, both for the sequence-based as well as the tree-based decoders. We leave a deeper investigation of the effect of numbering tokens for future work.

Baselines: We considered the following models as baselines in our experiments: (1) BERT with a transformer decoder (BERT+Transformer), (2) a re-implemented insertion transformer (Seq-Insert) in both binary and uniform supervision settings (BERT is used for encoding the input), and (3) BERT with a regular tree-based decoder similar to the one proposed by Dong and Lapata [15] (BERT+TreeGRU; see also next paragraph). Note that both (1) and (3) are essentially left-to-right decoders and can't decode in parallel. Even though they are not directly comparable, we also include the results reported by Chen et al. [85]. Rather than decoding logical forms (i.e. Lisp-style expressions), they decode an intermediate representation called a query graph using special graph generation actions (e.g. add variable node, add edge). Moreover, the evaluation is based on execution accuracy instead of logical form accuracy [201].

TreeGRU: We implement the tree-based left-to-right baseline decoder similarly to Dong and Lapata [15]. However, in our implementation, we use depth-first instead of breadth-first decoding. We achieve similar conditioning in the different decoding steps by manipulating which of the previous states are used as the previous sibling state and the parent state. If the previously decoded token was an opening parenthesis "(", then the parent state correspond to the state of the GRU after producing the opening parenthesis (this is the previous state), the previous sibling state is set equal to the parent state. Following Dong and Lapata [15], we use the parent state explicitly as part of the GRU input in every step. When a closing parenthesis ")" is decoded, the parent of the previous parent state is used as the parent state, the state accumulated from the closed subtree is discarded, and the previous parent state becomes the previous sibling state. Thus, we make sure that similarly to Dong and Lapata [15]'s breadth-first decoding, the different branches aren't directly conditioned on each other through decoder states.

Training details: We train all models using Adam [49], varying the initial learning rate within $\{0.0001, 0.00005, 0.00001\}$ and experimenting with dropout rates out of $\{0.0, 0.1, 0.2, 0.4\}$. We also experimented with values from $\{1.0, 0.1\}$ for the temperature τ. The validation set for each domain was constructed by taking a random 20% subset of the training examples. This validation set was used for early stopping. We randomly searched the hyperparameter space and took the best performing parameters based on its validation performance on the *publications* domain. These hyperparameters[9] were used for training the models on the other domains. We train each model three times on every domain independently (with the same seed values shared over domains and settings) and also report the average over the domains. For all our experiments, we use a 12-layer BERT model from Huggingface [202] as the encoder and use a transformer with 6 layers, 12 heads, and 768 dimension for the decoder.

6.3.1 Results

Table 6.2 shows the results on all domains of the Overnight dataset. Table 6.3 shows the results on four Overnight domains, and indicates the accuracy and speed-up measured in terms of the number of decoding steps compared to the BERT+Transformer left-to-right baseline.

Baseline equivalence: First, we establish that the results of our left-to-right baseline are on par with previously reported numbers in similar settings. This is shown in the middle part of Table 6.2, where our "BERT+Transformer" beats Xu et al. [201]'s BERT+LSTM baseline by a small margin.

Sequence-based insertion baseline results: In the bottom part of Table 6.2, we report numbers for the slightly simplified logical forms. The performance of the parallel sequence insertion decoder with binary supervision is on par with that of our left-to-right baseline. The uniform variant performs slightly worse. Concerning the speed-up obtained, Table 6.3 shows that the binary supervision is (more than $\times 2$) more effective than uniform supervision. However, even the binary supervised version lags behind the theoretically best possible speed-up ("Seq-Insert Th.B.").

[9] dropout rate=0.2, learning rate=0.00005, τ=0.1, batch size=10/30/50, cosine learning rate scheduler with 20 epoch warm-up, 200 epochs

	cal.	blo.	hou.	res.	pub.	rec.	soc.	bas.	avg (± std)
Seq2Action [85]	81.5	61.4	74.1	80.7	80.7	82.9	82.1	88.2	79.0
Shift-Reduce [93]	43.5	25.1	29.6	37.3	32.9	58.3	51.2	69.6	43.4
BERT+LSTM [201]	58.3	42.6	48.7	55.4	64.6	68.5	70.4	84.1	61.6
BERT+Transformer	61.3	45.4	50.8	58.7	63.4	76.4	69.9	85.4	63.9
BERT+Transformer	80.0	53.9	70.9	83.4	70.4	83.3	73.8	84.0	75.0 ± 1.3
BERT+TreeGRU	77.4	49.7	67.9	82.6	73.3	81.2	73.0	84.4	73.7 ± 1.5
Seq-Insert (Binary)	78.2	50.9	67.5	81.0	72.5	81.2	70.9	84.1	73.3 ± 1.6
Seq-Insert (Uniform)	79.0	47.8	68.3	80.5	70.0	82.3	70.0	83.7	72.7 ± 1.6
Tree-Insert	77.4	51.9	71.8	81.1	72.9	82.9	73.2	84.4	74.4 ± 1.3

Table 6.2: Results on the test set of the different domains of the Overnight dataset. Top part: denotation accuracy. Middle part: logical form accuracy on original trees. Bottom part: logical form accuracy on simplified trees. In the last column, we report the average accuracy over the domains as well as the average (over domains) of standard deviations of accuracies for every domain over different seeds.

	calendar		restaurants		publications		recipes		average	
	Acc.	# Steps	Acc.	# Steps	Acc.	# Steps	Acc.	# Steps	Acc.	# St.
BERT+Transf.	80.0	40.4 (×1.0)	83.4	37.8 (×1.0)	70.4	40.9 (×1.0)	83.3	36.6 (×1.0)	79.3	38.9
BERT+TreeGRU	77.4	41.2 (×1.0)	82.6	38.1 (×1.0)	73.3	41.3 (×1.0)	81.2	36.7 (×1.0)	78.6	39.3
Seq-Insert Th.B.	–	6.7 (×6.0)	–	6.6 (×5.7)	–	6.7 (×6.1)	–	6.5 (×5.6)	–	6.6
Seq-Insert (Bin.)	78.2	7.2 (×5.6)	81.0	6.9 (×5.5)	72.5	7.3 (×5.6)	81.2	6.9 (×5.3)	78.2	7.1
Seq-Insert (Uni.)	79.0	18.3 (×2.2)	80.5	16.9 (×2.2)	70.0	17.6 (×2.3)	82.3	16.8 (×2.2)	77.9	17.4
Tree-Insert Th.B.	–	6.2 (×6.5)	–	6.1 (×6.2)	–	6.3 (×6.5)	–	6.0 (×6.1)	–	6.2
Tree-Insert	77.4	6.4 (×6.4)	81.1	6.2 (×6.2)	72.9	6.6 (×6.3)	82.9	6.2 (×5.8)	78.6	6.4

Table 6.3: Logical form accuracy on simplified logical forms and number of decoding steps used on the test set of the different domains of the Overnight dataset. “Bin” stands for binary supervision, “Uni.” for uniform.

Tree-based insertion results: The tree-based insertion decoding procedure described in Section 6.2.1 enables a decoding complexity that is lower than the theoretically best $O(log_2(N))$ of the sequence-based insertion decoder. However, the best possible speed-up for tree insertion decoding heavily depends on the tree structure. We report this number (“Tree-Insert Th.B.”) for the different domains in Table 6.3. The best possible reduction of the number of decoding steps of the tree-based insertion decoder on our data is slightly higher than that of the sequence-based insertion decoder: an average of 0.48 decoding steps can be gained, which corresponds to a potential reduction of decoding steps of $\times 1.08$ on average. However, one needs to investigate how close an actual tree-based insertion decoder can get to the theoretically best number in practice.

The bottom part of Table 6.3 reports the results for the tree-based insertion decoder. Our insertion-based tree decoder achieves **competitive accuracy** to both the left-to-right baseline as well as the sequence insertion decoder, while requiring **fewer decoding steps** than the strong sequence insertion baseline: an average of 0.7 decoding steps is gained, which corresponds to an average of a $\times 1.11$ reduction in the number of decoding steps.

Ablation	publications	calendar	recipes
τ = 1.	69.8 (-3.1)	74.0 (-3.4)	82.3 (-0.6)
τ = 10.	24.4 (-48.5)	20.0 (-57.4)	28.9 (-54.0)
# traj. = 1	67.3 (-5.6)	71.6 (-5.8)	66.5 (-16.4)
abs. pos	71.0 (-1.9)	77.0 (-0.4)	79.3 (-3.6)
no child. rel.	72.5 (-0.4)	78.8 (+1.4)	81.9 (-1.0)

Table 6.4: Results of an ablation study on the Publications domain.

Ablation study: In Table 6.4, we assess the effects of some design choices and hyperparameters. Decreasing the number of trajectories used during training from 5 to 1 results in a significant decrease in accuracy. The chosen temperature τ also affects training. When τ is set to a high value, the target distribution becomes more uniform, which allows the model to use insertions that are not covered by the trained trajectories making it more likely to fail due to exposure bias, which is reflected in the poor result for $\tau = 10$. Using absolute instead of relative position information leads only to a slight decrease in accuracy. Note that using just absolute positioning requires to process the structure tokens because the structure information contained in relative positions is not being used. The child relations (§6.2.2) appear to not have any significant effect.

6.4 Discussion

To obtain an efficient decoder other considerations next to the number of decoding steps must be taken into account. The actual execution speed and computational load also heavily depend on (1) the size of processed data and (2) efficiency of implementation. In the proposed tree decoder, the *effective* size of model inputs is *smaller* than for the sequence insertion decoder since it does not need to use structure-indicating tokens. Thus, when implemented efficiently, it requires less computation and less memory, which is attractive given the quadratic memory complexity of the transformer's attention.

Another point worth noting is that the best possible speed-up of the proposed insertion-based tree decoder heavily depends on the data. However, considering that the proposed model has between two and three insertion slots per token, and the sequence insertion decoder only one, our method can expand trees much faster.

A limitation of the proposed insertion-based tree decoder is that it, like the Insertion Transformer, is unable to recover from mistakes made during decoding. In contrast, the Levenshtein transformer [95] also defines deletion operations and can thus recover from erroneous predictions. Extending the insertion-based tree decoder to also delete part of the trees is an interesting direction for future work. While we simply sample from a number of optimal trajectories, a more general decoder that allows for deletion would have to be trained using dynamic oracles [24, 162, 163] or reinforcement learning. This is likely to decrease the exposure bias as the model is exposed to many different trajectories.

6.5 Conclusion

Regarding RQ3, we have shown that insertion-based decoding, which uses significantly fewer decoding steps, is competitive to autoregressive decoders such as a normal sequence left-to-right decoder as well as a tree-based decoder. The results indicate that the introduced independence relations between output variables do not adversely affect accuracy but also do not provide significant improvement on the used data splits. We also have shown that it is possible to improve upon sequence-based insertion decoding and use even fewer decoding steps by using a tree-based insertion decoder that defines additional insertion slots.

CHAPTER 7

Linearization order when training semantic parsers.

"All's well as ends better."

– Hamfast Gamgee,
from *The Lord of the Rings*,
by J.R.R. Tolkien

In this chapter, we investigate the effect of linearization order when training semantic parsers. To this end, we focus on the text-to-SQL semantic parsing task, which aims to enable question answering over regular relational databases. We perform our investigation using the WikiSQL dataset.

This chapter is based on the following publication:

3. **Denis Lukovnikov**, Nilesh Chakraborty, Jens Lehmann and Asja Fischer. *Translating Natural Language to SQL using Pointer-Generator Networks and How Decoding Order Matters.* In the AAAI 2019 Reasoning for Complex Question Answering Workshop. 2019.

The work described in this chapter presents a straightforward approach for text-to-SQL that was competitive to the state-of-the-art at the time the work was originally performed. We use this approach to investigate the effect of linearization order and investigate the use of a dynamic oracle in this context. This corresponds to Contribution 4, which addresses the following research question:

RQ4: Does the query linearization order used during training affect accuracy?

The rest of the chapter is organized as follows: We further motivate and elaborate on the problem and list specific contributions in Section 7.1. We elaborate in more detail on linearizations and order in Section 7.2. We discuss the model in Section 7.3 and the training, where we also present the dynamic oracle, in Section 7.4. The experiments are presented in Section 7.5. We discuss our work in the context of specific related work on WikiSQL in Section 7.6 and conclude the chapter in Section 7.7. See also Section 3.3.2 in Chapter 3 for a discussion of related work for the ordering issue.

7.1 Introduction

In this work, we focus on QA over tabular data, which attracted significant research efforts [80, 90–92, 203–209]. In this task, given a NL question and a table, the system must generate a query that will retrieve the correct answers for the question from the given table.

The model we use in this work is a straightforward extension of pointer-generator networks, and yet outperforms early work and compares well against concurrently developed models. Concretely, we add simple LSTM-based column encoders, skip connections and constrained decoding, as elaborated later.

In order to use sequence decoders for generating queries, the queries must first be linearized to sequences which can be used to train a sequence decoder. However, when translating NL questions to SQL queries, as in many semantic parsing tasks, target queries can contain unordered elements, which results in multiple valid decoding paths. The particular ordering of the unordered elements in the decoding path used for supervision can affect the performance of the trained SEQ2SEQ models. We provide a deeper investigation of the potential "order-matters" problem in translating NL to SQL that has been raised by previous work. In this context, we also investigate training with a non-deterministic oracle [24] as well as training with REINFORCE, both of which explore different possible linearizations of the target queries, and show that the use of a non-deterministic oracle can be beneficial when the original supervision sequences are ordered inconsistently.

The contribution of this work are as follows:

- We propose a straightforward sequence-to-sequence architecture with a copy mechanism, improved with special column encoders to combat the OOV problem of column names in unseen tables, and show that it performs competitively to previous work at the time when our work was conducted. Compared to ours, previous methods were designed specifically for text-to-SQL tasks and were not generally applicable.

- We show that using a single fixed ordering during training can affect model performance, especially when that ordering was done arbitarily.

- We investigate the use of a dynamic oracle to train without a fixed decoding order, and show that this leads to improvements, which are significant compared to using arbitrary fixed ordering. Moreover, the dynamic oracle works better than the alternative policy gradient based training similar to the methods used in previous work.

In the following, we first introduce the problem, then describe our model and the training procedure, present an experimental analysis, and conclude with a comparison to related work.

7.2 Queries, Trees and Linearizations

As an illustration of table-based QA, consider the natural language question

> "*How much L1 Cache can we get with an FSB speed of 800MHz and a clock speed of 1.0GHz?*"

This question should be mapped to the following SQL query

```
SELECT  L1_Cache
WHERE   FSB_Speed = 800 (Mhz)
   AND  Clock_Speed = 1.0 (GHz)
```

which will be executed over a table listing processors, the sizes of their caches, their clocks speeds etc. In the query representation format we use, the example SQL query will be represented as the following sequence of output tokens:

```
SELECT L1_Cache AGG0 WHERE COND FSB_Speed OP0 VAL 800 ENDVAL COND Clock_Speed
OP0 VAL 1.0 ENDVAL ,
```

where `AGG0` is a dummy "no aggregator" token that is used to indicate that no real aggregator should be applied and `OP0` is the = (equality) operator. Other aggregators, like `SUM` and `COUNT`, and other operators, like < (less than) are also available.

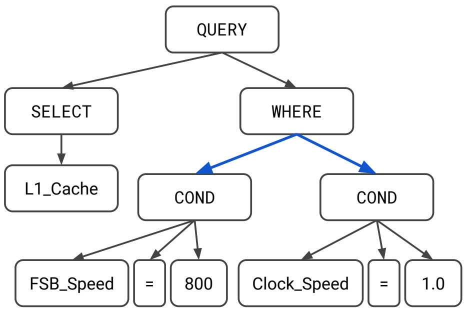


Figure 7.1: Example of a query tree. The blue arrows indicate unordered children.

As illustrated in Figure 7.1, the SQL query can also be represented as a tree where the root node has two children: `SELECT` and `WHERE`. Note that the order of the two conditions appearing in the `WHERE` clause is arbitrary and does not have any impact on the meaning of the query or the execution results. Trees containing such unordered nodes can be linearized into a sequence in different, equally valid, ways ("FSB Speed" first or "Clock Speed" first in the example, as illsustrated in Figure 7.2.). We refer to the linearization where the original order as given in the data set is preserved as the *original linearization*.

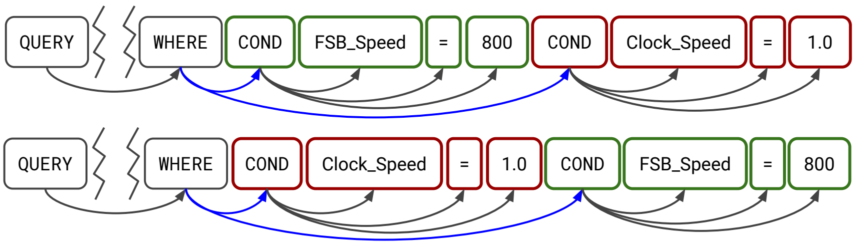


Figure 7.2: Two valid linearizations of the example query tree in Figure 7.1.

7.3 Model

We start from a sequence-to-sequence model with attention and extend the embedding and output layers to better suit the task of QA over tabular data. In particular, we use on-the-fly [210] embeddings and output vectors for column tokens and implement a pointer-based [87, 88] mechanism for copying tokens from the question. The resulting model is a Pointer-Generator [87, 88] with on-the-fly representations for a subset of its vocabulary.

7.3.1 The Seq2Seq Model

The general architecture of our model follows the attention-based sequence-to-sequence (SEQ2SEQ) architecture. The following formally introduces the major parts of our SEQ2SEQ model. Details about the embedding and the output layers are further elaborated in later sections.

The SEQ2SEQ model consists of an encoder, a decoder, and an attention mechanism.

Encoder

We are given a question $Q = [q_0, q_1, \ldots, q_N]$ consisting of NL tokens q_i from the set $\mathcal{V}^E$ (i.e., the encoder vocabulary). The tokens are first passed through an embedding layer that maps every token q_i to its vector representation $\mathbf{q}_i = W^E \cdot \mathsf{one\_hot}(q_i)$ where $W^E \in \mathbb{R}^{|\mathcal{V}^E| \times d^{emb}}$ is a learnable weight matrix and $\mathsf{one\_hot}(\cdot)$ maps a token to its one-hot vector.

Given the token embeddings, a bidirectional multi-layered Long Short-Term Memory (LSTM) [42] encoder produces the hidden state vectors $[\mathbf{h}^*{}_0, \mathbf{h}^*{}_1, \ldots, \mathbf{h}^*{}_N] = \mathsf{BiLSTM}([\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_N])$.

The encoder also contains skip connections that add word embeddings $\mathbf{q}_i$ to the hidden states $\mathbf{h}^*{}_i$.

Decoder

The decoder produces a sequence of output tokens s_t from an output vocabulary $\mathcal{V}^D$ conditioned on the input sequence Q. It is realized by a uni-directional multi-layered LSTM. First, the previous output token s_{t-1} is mapped to its vector representation using the embedding function $\mathrm{EMB}(\cdot)$. The embeddings are fed to a BiLSTM-based decoder and its output states are used to compute the output probabilities over $\mathcal{V}^D$ using the output function $\mathrm{OUT}(\cdot)$. $\mathrm{EMB}(\cdot)$ and $\mathrm{OUT}(\cdot)$ are described in the following sections.

Attention

We use attention [46] to compute the context vector $\hat{\mathbf{h}}_t$, that is

$$a_i^{(t)} = \mathbf{h}_i \cdot \mathbf{y}_t \ , \tag{7.1}$$

$$\alpha_i^{(t)} = \mathsf{softmax}(a_0^{(t)}, \ldots, a_i^{(t)}, \ldots, a_N^{(t)})_i \ , \tag{7.2}$$

$$\hat{\mathbf{h}}_t = \sum_{i=0}^{N} \alpha_i^{(t)} \mathbf{h}_i \ , \tag{7.3}$$

where $\mathsf{softmax}(\cdot)_\mathsf{i}$ denotes the i-th element of the output of the softmax function, $\mathbf{y}_t$ is the output state of the decoder and $\mathbf{h}_1, \ldots, \mathbf{h}_N$ are the embedding vectors returned by the encoder.

7.3.2 Embedding Function of the Decoder

The whole output vocabulary $\mathcal{V}^D$ can be grouped in three parts: (1) SQL tokens from $\mathcal{V}^{\text{SQL}}$, (2) column ids from $\mathcal{V}^{\text{COL}}$, and (3) input words from the encoder vocabulary $\mathcal{V}^E$, that is, $\mathcal{V}^D = \mathcal{V}^{\text{SQL}} \cup \mathcal{V}^{\text{COL}} \cup \mathcal{V}^E$. In the following paragraphs, we describe how each of the three types of tokens is embedded in the decoder.

SQL tokens:

These are tokens which are used to represent the structure of the query, inherent to the formal target language of choice, such as SQL-specific tokens like `SELECT` and `WHERE`. Since these tokens have a fixed, example-independent meaning, they can be represented by their respective embedding vectors shared across all examples. Thus, the tokens from $\mathcal{V}^{\text{SQL}}$ are embedded based on a learnable, randomly initialized embedding matrix W^{SQL} which is reused for all examples.

Column id tokens:

These tokens are used to refer to specific columns in the table that the question is being asked against.

Column names may consist of several words, which are first embedded and then fed into a single-layer LSTM. The final hidden state of the LSTM is taken as the embedding vector representing the column. This approach for computing column representations is similar to other work that encode external information to get better representations for rare words [187, 210, 211].

Input words:

To represent input words in the decoder we reuse the vectors from the embedding matrix W^E, which is also used for encoding the question.

7.3.3 Output Layer of the Decoder

The output layer of the decoder takes the current context $\hat{\mathbf{h}}_t$ and the hidden state $\mathbf{y}_t$ of the decoder's LSTM and produces probabilities over the output vocabulary $\mathcal{V}^D$. Probabilities over SQL tokens and column id tokens are calculated based on a dedicated linear transformation, as opposed to the probabilities over input words which rely on a pointer mechanism that enables copying from the input question.

Generating scores for SQL tokens and column id tokens

For the SQL tokens ($\mathcal{V}^{\text{SQL}}$), the output scores are computed by the linear transformation: $\mathbf{o}^{\text{SQL}} = U^{\text{SQL}} \cdot [\mathbf{y}_t, \hat{\mathbf{h}}_t]$, where $U^{\text{SQL}} \in \mathbb{R}^{|\mathcal{V}^{\text{SQL}}| \times d^{out}}$ is a trainable matrix. For the *column id tokens* ($\mathcal{V}^{\text{COL}}$), we compute the output scores based on a transformation matrix U^{COL}, holding dynamically computed encodings of all column ids present in the table of the current example. For every column id token, we encode the corresponding column name using an LSTM, taking its final state as a (preliminary) column name encoding $\mathbf{u}^*$, similarly to what is done in the embedding function. By using skip connections we compute the average of the word embeddings of the tokens in the column name, $\mathbf{c}_i$ for

$i = 1, \ldots, K$, and add them to the preliminary column name encoding $\mathbf{u}^*$ to obtain the final encoding for the column id:

$$\mathbf{u} = \mathbf{u}^* + \begin{bmatrix} \mathbf{0} \\ \frac{1}{K} \sum_i^K \mathbf{c}_i \end{bmatrix} \quad , \tag{7.4}$$

where we pad the word embeddings with zeros to match the dimensions of the encoding vector before adding.

The output scores for all column id tokens are then computed by the linear transformation[1]: $\mathbf{o}^{\text{COL}} = U^{\text{COL}} \cdot [\mathbf{y}_t, \hat{\mathbf{h}}_t]$.

Pointer-based copying from the input

To enable our model to copy tokens from the input question, we follow a pointer-based [87, 88] approach to compute output scores over the words from the question. We explore two different copying mechanism, a *shared softmax* approach inspired Gu et al. [88] and a *point-or-generate* method similar to See et al. [87]. The two copying mechanisms are described in the following.

Point-or-generate: First, the concatenated output scores for SQL and column id tokens are turned into probabilities using a softmax

$$p^{\text{GEN}}(S_t | s_{t-1}, \ldots, s_0, Q) = \mathsf{softmax}([\mathbf{o}^{\text{SQL}}; \mathbf{o}^{\text{COL}}]) \quad . \tag{7.5}$$

Then we obtain the probabilities over the input vocabulary $\mathcal{V}^E$ based on the attention probabilities $\alpha_i^{(t)}$ (Eq. 7.2) over the question sequence $Q = [q_0, \ldots, q_i, \ldots, q_N]$. To obtain the pointer probability for a token q in the question sequence we sum over the attention probabilities corresponding to all the positions of Q where q occurs, that is

$$p^{\text{PTR}}(q | s_{t-1}, \ldots, s_0, Q) = \sum_{i:q_i=q} \alpha_i^{(t)} \quad . \tag{7.6}$$

The pointer probabilities for all input tokens $q \in \mathcal{V}^E$ that do not occur in the question Q are set to 0.

Finally, the two distributions p^{GEN} and p^{PTR} are combined into a mixture distribution:

$$\begin{aligned} p(S_t | s_{t-1}, \ldots, s_0, Q) = {} & \gamma p^{\text{PTR}}(S_t | s_{t-1}, \ldots, s_0, Q) \\ & + (1-\gamma) p^{\text{GEN}}(S_t | s_{t-1}, \ldots, s_0, Q) \quad , \end{aligned} \tag{7.7}$$

where the scalar mixture weight $\gamma \in [0, 1]$ is given by the output of a two-layer feed-forward neural network, that gets $[\mathbf{y}_t, \hat{\mathbf{h}}_t]$ as input.

Shared softmax: In this approach, we re-use the attention scores $a_i^{(t)}$ (Eq. 7.1) and obtain the output scores $\mathbf{o}^{\text{E}}$ over the tokens $q \in \mathcal{V}^E$ from the question as follows: for every token q that occurs in the

[1] Note that the skip connections in both the question encoder and the column name encoder use padding such that the word embeddings are added to the same regions of $[\mathbf{y}_t, \hat{\mathbf{h}}_t]$ and $\mathbf{u}$, respectively, and thus are directly matched.

question sequence Q the output score is given by the maximum attention score over all positions in $Q = [q_0, \dots, q_i, \dots, q_N]$ where q occurs, i.e. it is given by:

$$\max_{i:q_i=q} a_i \quad , \tag{7.8}$$

while the scores for all input tokens $q \in \mathcal{V}^E$ that do not occur in the question Q are set to $-\infty$. The final output probabilities are then computed based on a single softmax function that takes the output scores of the whole output vocabulary as input:

$$p(S_t|s_{t-1}, \dots, s_0, Q) = \mathsf{softmax}([\mathbf{o}^{\mathrm{SQL}}; \mathbf{o}^{\mathrm{COL}}; \mathbf{o}^{\mathrm{E}}]) \quad . \tag{7.9}$$

7.3.4 Pretrained Embeddings and Rare Words

We initialize all NL embedding matrices[2] using GloVe embeddings for words covered by GloVe [40] and use randomly initialized vectors for the remaining words. Whereas randomly initialized word embeddings are trained together with the remaining model parameters, we keep GloVe embeddings fixed, since finetuning them led to worse results in our experiments.

We also replace rare words that do not occur in GloVe with a rare word representation in all embedding matrices.

7.3.5 Coherence of decoded logical forms

The output sequences produced by a unconstrained decoder can be syntactically incorrect and result in execution errors or they can make mistakes against table semantics. We avoid such mistakes by implementing a constrained decoder that exploits task-specific syntactic and semantic rules[3].

The grammar behind the produced sequences is simple and the constraints can be implemented easily by keeping track of the previous token and whether we are in the `SELECT` or `WHERE` clause. In our example discussed earlier (see Figure 7.1), after a `COND` token, only column id tokens (`L1_Cache`, `FSB_Speed`, . . .) can follow, and after a column id token, only an operator token (`OP1`, `OP2`, . . .) is allowed if we are currently decoding the `WHERE` clause.

In addition to such syntactic rules, we take into account the types of columns to restrict the set of aggregators and operators that can follow a column id. In the case of WikiSQL, there are two column types: `text` and `float`. Aggregators like average and operators like greater_than only apply on `float`-typed columns and thus are not allowed after `text` columns. We also enforce span consistency when copying tokens, leaving only the choice of copying the next token from the input or terminating copying, if the previous action was a copy action.

7.4 Training

We train our models by maximizing the likelihood of a correct logical form given the natural language question. We experiment with teacher forcing (TF) and a non-deterministic oracle [24].

[2] W^E simultaneously used for question word embedding in the encoder and input word embedding in the embedding function of the decoder, the embedding matrix W^{CT} for words occurring in column names used in the embedding function of the decoder, and its analogue in the output function.

[3] We use these constraints during prediction only.

Teacher forcing takes the original linearizations of the query trees (as provided in the dataset) and uses it both for supervision and as input to the decoder. However, in the presence of multiple correct decoding paths, teacher forcing can suffer from suboptimal supervision order, as pointed out by previous work on WikiSQL [91, 92] and concurrently explored by [90].

7.4.1 Non-deterministic Oracle

Instead of forcing the model to follow the original decoding sequence, a non-deterministic oracle enables the exploration of alternative linearizations of the query tree and is an adaptation of Goldberg and Nivre's [24] algorithm for a *dynamic oracle with spurious ambiguity* (developed in the context of dependency parsing). It is formally described in Algorithm 1, which is invoked at every decoding step t to get a token g_t (used for supervision) and a token x_{t+1} (used as input to the decoder in the next time step). Essentially, the algorithm always picks the best-scored *correct* token for supervision and uniformly samples one of the correct tokens to be used as decoder input in the next time step, if the *overall* best-scored token (over the whole output vocabulary) does not belong to the correct ones. Thus, the oracle explores alternative decoding orders if the decoder would make a mistake in free-running mode.

Algorithm 1 Non-deterministic oracle

```
1: function GetNextAndGold($p_t$, $t$, $x_{\leq t}$)
2:     $\mathsf{VNT}_t \leftarrow$ get_valid_next($t, x_{\leq t}$)
3:     $x_{t+1} \leftarrow \mathrm{argmax}_{\mathcal{V}^D}\, p_t$
4:     $g_t \leftarrow \mathrm{argmax}_{\mathsf{VNT}_t}\, p_t$
5:     if $x_{t+1} \notin \mathsf{VNT}_t$ then
6:         $x_{t+1} \leftarrow$ random($\mathsf{VNT}_t$)
7:     return $g_t, x_{t+1}$
```

In the algorithm, p_t is the decoder's output distribution over $\mathcal{V}^D$ at time step t. The set of valid next tokens $\mathsf{VNT}_t \subset \mathcal{V}^D$, from which the correct tree can be reached, is returned by the function `get_valid_next(·)`. The query tree can have nodes with either ordered or unordered children (for example, children of the `WHERE` clause are unordered). If we are currently decoding the children of a node with unordered children, all the children that have not been decoded yet are returned as VNT_t. In other cases, VNT_t contains the next token according to the original sequence order.

7.4.2 REINFORCE

The presented oracle is similar to REINFORCE in that it explores alternative paths to generate the same query. In contrast to the oracle, REINFORCE samples the next token (x_{t+1}) according to the predictive distribution p_t and then uses the sampled sequence to compute gradients for policy parameters:

$$\nabla J = \mathbb{E}[\nabla \log(p_t(x_{t+1}))A_t] \tag{7.10}$$

In Alg. 2, we adapt the oracle into an algorithm equivalent to basic REINFORCE with episode reward A_t set to +1 if the sampled sequence produces a correct query and 0 otherwise.

Algorithm 2 Our REINFORCE

```
1: function GetNextAndGold($p_t$, $t$, $x_{\leq t}$)
2:     $\mathsf{VNT}_t \leftarrow$ get_valid_next($t, x_{\leq t}$)
3:     $x_{t+1} \sim p_t$;  $x_{t+1} \in \mathsf{VNT}_t$
4:     $g_t \leftarrow x_{t+1}$
5:     return $g_t, x_{t+1}$
```

7.5 Evaluation

To evaluate our approach, we use the WikiSQL [92] dataset, obtained by following the instructions on the WikiSQL website[4]. The dataset contains a total of 80654 examples. Each example provides a NL question, its SQL equivalent and the table against which the SQL query should be executed. The original training/dev/test splits of WikiSQL use disjoint sets of tables with different schemas.

Similar to the WikiSQL dataset that we used in our experiments are the ATIS [103] and WikiTableQuestions [209] datasets, which also focus on question answering over tables. In contrast to WikiSQL however, both ATIS and WikiTableQuestions are significantly smaller and the latter does not provide logical forms for supervision and thus requires training with execution results as supervision [80, 205, 212]. SQA [208] is a dataset derived from WikiTableQuestions and focuses on question answering in a dialogue context.

7.5.1 Experimental Setup

Evaluation: Similarly to previous work, we report (1) sequence match accuracy (Acc_{LF}), (2) query match accuracy (Acc_{QM}) and (3) query execution accuracy (Acc_{EX}). Note that while Acc_{LF} accepts only the original linearizations of the trees, Acc_{QM} and Acc_{EX} accept all orderings leading to the same query.

Training details: After a hyperparameter search, we obtained the best results by using two layers both in the encoder and decoder LSTMs, with every layer of size 600, and embedding size of 300, and applying time-shared dropouts on the inputs of the recurrent layers (dropout rate 0.2) and recurrent connections (dropout rate 0.1). We trained using Adam, with a learning rate of 0.001 and a batch size of 100, a maximum of 50 epochs and early stopping. We also use label smoothing with a mixture weight $\epsilon = 0.2$, as described in Szegedy et al. [213].

We ran all reported experiments at least three times and report the average of the computed metrics. While the variance of the metrics varies between settings, it generally stays between 0.1 and 0.25 percent for Acc_{QM}.

7.5.2 Results

We present our results, compared to previous and concurrent work in Table 7.1. Our method compares well against previous work, achieving performance similar to Coarse2Fine [17] and close to MQAN [214] which have more complicated architectures. Approaches using execution-guided

[4] `http://github.com/salesforce/WikiSQL`

decoding (EG) show better performance at the expense of access to table content and repeated querying during decoding, and relies on the assumption that the query should not return empty result sets. The concurrently developed oracle-based[5] approach of Shi et al. [90] improves upon our investigation of the oracle using the `ANYCOL` technique (see Related Work section).

In the following sections, we provide an ablation study, an in-depth analysis of the influence of the linearization order of query trees, as well as an error analysis. The analysis reveals that the overall improvement in accuracy obtained from using the oracle can be attributed to improved prediction accuracy of WHERE clauses, which contain unordered elements.

	Dev Accuracies (%)			Test Accuracies (%)		
	Acc_{LF}	Acc_{QM}	Acc_{EX}	Acc_{LF}	Acc_{QM}	Acc_{EX}
Seq2SQL (no RL) [92]	48.2	–	58.1	47.4	–	57.1
Seq2SQL (RL) [92]	49.5	–	60.8	48.3	–	59.4
Pointer-SQL [206]	59.6	–	65.2	59.5	–	65.1
SQLNet [91]	–	63.2	69.8	–	61.3	68.0
PT-MAML [204]*	63.1	–	68.3	62.8	–	68.0
TypeSQL [203]*	–	68.0	74.5	–	66.7	73.5
STAMP [158]*	61.7	–	75.1	61.0	–	74.6
Coarse2Fine [17]	–	–	–	–	71.7	78.5
MQAN [214]	–	–	–	72.4	–	80.4
(ours)						
Seq2Seq+DP+C (*shared softmax*)	70.2	72.6	79.0	69.9	72.1	78.4
Seq2Seq+DP+C (*point-or-generate*)	70.0	72.4	78.5	69.7	71.7	78.0
Seq2Seq+DP+C (*shared softmax*) + oracle	56.2	73.4	79.4	55.0	72.7	78.8
(EG-based or concurrent work)						
Pointer-SQL + EG(5) [207]	67.5	–	78.4	67.9	–	78.3
Coarse2Fine + EG(5) [207]	76.0	–	84.0	75.4	–	83.8
IncSQL + oracle + `ANYCOL` [90]	49.9	–	84.0	49.9	–	83.7
IncSQL + oracle + `ANYCOL` + EG(5) [90]	51.3	–	87.2	51.1	–	87.1

Table 7.1: Evaluation results for our approach (middle section) and comparison with previously reported results (top part) and concurrent work or EG-based systems (bottom part). Entries marked by * are trained and evaluated using a slightly different version of the WikiSQL dataset. Some values in the table, indicated by "–", could not be filled because the authors did not report the metric or the metric was not applicable.

Ablation study

Starting from the best variant of our model (i.e. the *shared softmax* pointer-generator) and standard TF based training, we want to investigate the role of different model components and the different training approaches.

[5] We also investigated non-deterministic oracles in the preprint of this work from May 2018 (`https://openreview.net/forum?id=HJMoLws2z`).

	Dev Accs (%)		Test Accs (%)	
	Acc_{LF}	Acc_{QM}	Acc_{LF}	Acc_{QM}
Seq2Seq+DP+C (*shared softmax*)	70.2	72.6	69.9	72.1
· no constraints	68.6	70.9	68.6	70.5
· using constraints during training	69.8	72.2	69.8	71.9
· no label smoothing	68.3	70.5	68.4	70.1
· no label smoothing (*point-or-generate*)	68.7	70.7	68.5	70.3
· no skip connections	69.6	72.0	69.4	71.6

Table 7.2: Performance of different variations of our approach.

Table 7.2 presents the results of this ablation study. Without constraints enforcing the coherence of the decoded logical rule at test time, the results drop by 1.6% Acc_{QM} on the test set. While also using the constraints during training doesn't deteriorate results much, it results in slower training.

Label smoothing [213] has a significant impact on performance. Label smoothing relaxes the target distribution and thus helps to reduce overfitting. While label smoothing improves the performance of both versions of pointer-generators, it improves the *shared softmax* version by 2% test Acc_{QM}, as opposed to a slightly lower improvement of 1.4% for *point-or-generate*.

Incorporating skip connections into the encoder and decoder of our model improved performance by 0.5% Acc_{QM} on the test set.

Effect of ordering in supervision

To investigate the influence of the ordering in the linearizations of queries, we trained our model with teacher forcing and experimented with (1) reversing the original order of conditions in the WHERE clause and (2) training with target sequences where we assigned a different random order to the conditions in every trial. The results indicate that the order of conditions in the linearization matters for the performance of TF based training to a certain degree. Training with a randomly reassigned order of conditions in the WHERE clause results in a 2.5% drop in query accuracy (Acc_{QM}) on the test set. However, reversing the order of conditions does not affect the results.

Furthermore, we trained our model with REINFORCE as well as with the non-deterministic oracle. In both methods, the originally provided order of the target sequence does not matter. Using REINFORCE (indicated by "RL" in Table 7.3) results in a 1.5% drop in Acc_{QM} on the test set. The

	Dev Accs (%)		Test Accs (%)	
	Acc_{LF}	Acc_{QM}	Acc_{LF}	Acc_{QM}
Original order (TF)	70.2	72.6	69.9	72.1
· Reversed (TF)	–	72.6	–	72.1
· Arbitrary (TF)	–	70.4	–	69.6
· RL	59.9	71.4	59.1	70.6
· Oracle	56.2	73.4	55.0	72.7

Table 7.3: Results for different target tree linearizations.

oracle as described in Alg. 1 results in an improvement of 0.6% query accuracy on the test set. We can also see that $\mathrm{Acc}_{\mathrm{LF}}$ for the oracle is significantly lower compared to TF while $\mathrm{Acc}_{\mathrm{QM}}$ is on par with TF. Given that $\mathrm{Acc}_{\mathrm{LF}}$ is sensitive to the order of arbitrarily ordered clauses and $\mathrm{Acc}_{\mathrm{QM}}$ is not, this means that the oracle-trained models effectively learned to use alternative decoding paths.

Comparing the oracle to TF with arbitrarily reordered conditions in the WHERE clause shows that training with TF can suffer from supervision sequences that are not consistently ordered. When training with the oracle, the order of unordered nodes as provided in supervision sequences does not matter. Thus, it can be beneficial (in this case by 3% query accuracy) to use the oracle if the original linearization is arbitrary and can not be made consistent.

Error analysis

	TF	oracle
Whole Query	72.6	73.4
· SELECT	85.5	85.5
· Aggregator	90.0	90.0
· Column	94.7	94.7
· WHERE	83.4	84.4

Table 7.4: Error Analysis: $\mathrm{Acc}_{\mathrm{QM}}$ of different query parts on the development set for TF and oracle-trained *shared softmax* models.

Table 7.4 shows accuracies of different parts of the query over the development set of WikiSQL. The main cause of a wrongly predicted SELECT clause is an error in the predicted aggregator, while the main cause of error overall is the prediction of the WHERE clause.

Comparison of errors of models trained with TF versus oracle reveals that oracle-trained models make fewer mistakes in the WHERE clause, showing a 1% improvement (84.4% from 83.4%) in WHERE clause accuracy, which is translated to the 0.8% (73.4% from 72.6%) improvement in full query accuracy ($\mathrm{Acc}_{\mathrm{QM}}$) on the validation set.

We find no difference between the accuracies for the SELECT clause between TF and oracle training settings. In both cases, 68.7% of examples with wrongly predicted SELECT clauses had an error in the predicted aggregator, and 36.5% had a wrongly selected column.

7.6 Discussion and previous work

We provide an overview of work related to linearization order during training in Section 3.3.2 in Chapter 3. In this section, we provide a short discussion of the presented approach in the context of previous work on WikiSQL.

Previous work on WikiSQL [91, 92, 203, 204, 206] generally incorporate both slot-filling and sequence decoding, predicting the SELECT clause arguments with separate slot-filling networks, and also include some form of a pointing mechanism. Seq2SQL [92] proposes an *augmented pointer network* that also uses a pointer but encodes the question, column names and SQL tokens together, and completely relies on a pointer to generate the target sequence. To avoid the *order-matters* problem, SQLNet [91] proposes a sequence-to-set model that makes a set inclusion prediction in order to avoid decoding the conditions in any particular order. Both predict the SELECT clause arguments using separate specialized predictors. Zhong et al. [92] also use Dong and Lapata [15]'s Seq2Seq model as a baseline, however, get poor performance due to the lack of pointer and column encoders. Yu et al. [203] build upon SQLNet [91]'s slot filling approach, proposing several improvements such as weight sharing between SQLNet's subnetworks, and incorporate precomputed type information for question tokens in order to obtain a better question encoding. Wang et al. [206] develop a model similar to ours; they propose a Seq2Seq model with copy actions. Similarly to Zhong et al. [92], they encode the concatenation of column names and the question. Similarly to our work, they use a constrained decoder to generate SQL tokens or copy column names or question words from the encoded input sequence. In contrast to Wang et al. [206], we encode column names separately, and independently from the question. Huang et al. [204] experiment with meta-learning (MAML), using Wang et al. [206]'s model. STAMP [158] presents a "multi-channel" decoder that considers three types of tokens (SQL, column, cell), and mixes the distributions for each type using coefficients produced by a trainable subnetwork. Compared to STAMP, we do not encode cells (we assume no knowledge of table contents) and instead use a pointer to copy values of conditions from the input. Coarse2Fine [17] explores a middle ground between purely sequence and tree decoding models [15, 71] and proposes a two-stage decoding process, where first a template (sketch) of the query is decoded and subsequently filled in.

Very recent and concurrent work on WikiSQL explores execution-guided (EG) decoding [207] and non-deterministic oracles [90]. Execution-guided decoding keeps a beam of partially decoded queries, which are filtered based on the execution results, that is, a partially encoded query is discarded if it can not be parsed, produces a runtime error, or returns no results after execution. This requires multiple queries to be executed against the database while decoding. In our work, we try to avoid parsing and semantics-related runtime errors more efficiently by using decoding constraints. We suspect that a significant part of the improvement due to EG in decoding relies on the assumption that execution results should not be empty. However, we believe this assumption does not hold in general, due to the existence of queries for which an empty set is the correct answer. IncSQL [90] also uses EG decoding, as well as a non-deterministic oracle extended with the `ANYCOL` token, which adds the option to produce a wildcard column token that matches any column. During training, the wildcard column token is provided as an alternative to the true column token in the supervision sequence if it can be unambiguously resolved to the true column using the condition value. IncSQL's context encoding approach goes beyond ours by adding cross-serial attention between the question and the column representations and a final inter-column BiLSTM encoder. In contrast to our decoder actions, IncSQL's actions consist of action type (e.g. `CONDCOL`) and action parameters (e.g. "`L1_Cache`"), and they approach copying from input as specifying start and end positions. These choices reduce output sequence length.

7.7 Conclusion

In this work we present a Seq2Seq model adapted to the semantic parsing task of translating natural language questions to queries over tabular data. We investigated how the ordering of supervision sequences during training affects performance, concluding that the order of conditions in the linearization of the query tree matters to a certain degree for WikiSQL. In this context, we also evaluated the use of REINFORCE and a non-deterministic oracle for training the neural network-based semantic parser. Our experiments revealed that REINFORCE in our implementation does not improve results and the oracle provides a small improvement, which can be attributed to improved decoding of the WHERE clause. Furthermore, from the results we can conclude that training with a non-deterministic oracle is advisable if the original linearizations are inconsistently ordered.

CHAPTER 8

Detecting compositionally out-of-distribution examples.

"Use the Force, Harry!"

– Gandalf, from *Star Trek*,
by G.R.R. Martin

In this chapter, we consider the task of **detecting** compositionally out-of-distribution (OOD) examples in semantic parsing, which, to the best of our knowledge has not been investigated before at the time the work was performed.

While basic sequence-to-sequence models have shown impressive results on semantic parsing tasks, recent work [13, 14, 165] have presented the disconcerting finding that these models fail to generalize to novel combinations of elements observed in the training set (see Section 3.3.3 in Chapter 3). Therefore, several models and methods with improved compositional generalization have recently been proposed [16, 108, 168, 173, 175–179, 215]

The ability to detect OOD inputs is important, as it helps us to decide whether the model's predictions on the input can be trusted, which is crucial for safe deployment of the model and could be useful to build more efficient systems.

To this end, we analyse the OOD detection performance of recurrent neural network (RNN) and transformer based models using methods relying on predictive uncertainty. In addition, we propose to use a heterogeneous ensemble of a transformer and an RNN-based model that combines the strengths of both to improve the detection of compositionally OOD examples. We perform our analysis on the well-known SCAN and CFQ datasets, which were developed to test the compositional generalization of semantic parsers.

More information on compositional generalization and the SCAN and CFQ datasets is provided in Section 3.3.3.

This chapter is based on the following paper:

3. **Denis Lukovnikov**, Sina Daubener, Asja Fischer. *Detecting Compositionally Out-of-Distribution Examples in Semantic Parsing.* To appear at Conference on Empirical Methods in Natural Language Processing: Findings (EMNLP Findings 2021).

The work described here also corresponds to Contribution 5, which addresses the following research question:

> **RQ5: Can we detect compositionally out-of-distribution inputs in semantic parsing?**

The rest of the chapter is organized as follows: We elaborate in more detail on the investigated OOD detection methods in Section 8.1. The experiments are presented in Section 8.2. We conclude in Section 8.3.

8.1 Detecting OOD examples

In this work, we focus on OOD detection methods that build on the predictive distributions of discriminative task-specific models (extending the work of Hendrycks and Gimpel [216]). This has the advantage that it is easy to use in existing methods and doesn't require additional models or additional training (unlike for example generative modeling [217, 218]). Previous work has shown that neural models can produce incorrect predictions with high confidence on OOD inputs [219], which is detrimental to detecting such inputs. We aim at verifying whether this happens for the predictions of semantic parsers for compositionally OOD examples.

We compare the following measures quantifying the uncertainty of the prediction based on the output distributions of a trained model: (1) the average negative log-likelihood (NLL) for the generated sequence, (2) the sum of the NLLs, and (3) the average entropy of the output distributions.

More specifically, our approach for measuring uncertainty proceeds as outlined: First, the input x is encoded and an output sequence $\hat{y}$ is generated by the decoder. The model's output probability distributions $p(y_i|\hat{y}_{<i}, x)$ for every decoding step i are then used to compute the average and sum of NLL as

$$\frac{1}{T}\sum_{i=1}^{T} -\log p(\hat{y}_i|\hat{y}_{<i}, x) \quad , \tag{8.1}$$

where $\frac{1}{T}$ is omitted for the sum. The average entropy is given by

$$-\frac{1}{T}\sum_{i=1}^{T}\sum_{y_i\in\mathcal{V}} p(y_i|\hat{y}_{<i}, x)\log p(y_i|\hat{y}_{<i}, x) \quad , \tag{8.2}$$

where $\mathcal{V}$ is the set of all output tokens.

8.1.1 Posterior predictive distribution

To take model uncertainty into account, Bayesian approaches can be used [220–222]. A simple method for approximating the predictive uncertainty under a Bayesian posterior distribution over model parameters, is MC Dropout [25].

In our work, we use MC dropout as follows: First, we encode the input x and run the decoder to generate an output sequence $\hat{y}$. Then, we obtain K different output probability distributions $p_k(y_i|\hat{y}_{<i}, x)$, by feeding x and the precomputed $\hat{y}$ through the model K times while randomly

dropping neurons with the same probability as during training. Finally, the posterior predictive distribution is approximated by $\frac{1}{K}\sum_{k=1}^{K} p_k(y_i|\hat{y}_{<i}, x)$ and is used with the metrics described previously.

8.1.2 Heterogeneous ensemble

In our experiments, we found that different underlying architectures are better at detecting different types of OOD examples. To further improve detection performance, we propose to use a heterogeneous ensemble of different models for predictive uncertainty estimation. In contrast to the homogeneous ensemble proposed by Lakshminarayanan et al. [223], we ensemble over different model architectures, each used with MC dropout.

Concretely, given M different architectures, we train them independently on the ID training data. With each trained model with index $m \in 0 \ldots M$, we predict a sequence $\hat{y}_m$ given x and then obtain the output distributions averaged over all the trained models by feeding x and $\hat{y}_m$ through each of the M models K times with dropout as described above. Then, we compute the metrics based on the average of the $K * M$ distributions. Finally, we take the maximum of the NLL-based metrics computed for each $\hat{y}_m$ such that the most pessimistic score across different predicted outputs (across different m's) is retained.

8.2 Experiments

Datasets: We experiment with the SCAN [13] and CFQ [14] datasets. Table 8.4 provides some statistics on the number of examples in each split. See Section 3.3.3 for more information about how the datasets are built and background on compositional generalization.

The CFQ dataset is preprocessed using a simple reversible transformation of SPARQL queries into LISP-style s-expressions. This includes converting the set of triple patterns of the form "?x :r1 ?y. ?a :r2 ?b" to s-expressions of the form "(AND (COND ?x :r1 ?y) (COND ?a :r2 ?b))", where the order of the arguments of "AND" does not matter during evaluation.

Model and training settings: We consider both a transformer-based [18] and a GRU+attention [45] based sequence-to-sequence model in our experiments. Both are randomly initialized. We use six-layer transformers with 6 heads, hidden size 384, and dropout probability 0.25 and two-layer, 384-dimensional GRUs with dropout probability 0.1. The models are trained using Adam [49], with an initial learning rate of $5 * 10^{-4}$. We ran all experiments with three different seeds and report the average.

Method	SCAN				CFQ
	JUMP	T._LEFT	LEN.	MCD	
Transf.	0.4	90.0	0.0	1.7	17.0
GRU+Att.	0.2	62.8	12.1	20.9	9.4

Table 8.1: Logical form accuracy of the considered models on the original (OOD) test sets of the used datasets.

Method	MC	JUMP			TURN_LEFT			LENGTH			MCD		
		AUROC	AUPRC	FPR90	AUROC	AUPRC	FPR90	AUROC	AUPRC	FPR90	AUROC	AUPRC	FPR90
TM+avgNLL	No	99.5	99.9	1.6	86.6	82.2	41.0	80.2	91.5	60.7	88.4	91.6	41.9
TM+avgENT	No	99.5	99.9	1.4	86.7	82.3	40.8	79.8	91.2	61.5	88.1	91.4	43.5
TM+sumNLL	No	99.5	99.9	1.4	82.5	78.7	58.8	88.4	94.8	32.7	91.5	93.6	29.0
GRU+avgNLL	No	89.0	98.0	36.2	84.7	84.9	51.4	93.2	96.7	21.0	91.0	92.7	25.5
GRU+avgENT	No	89.9	98.2	38.9	84.5	84.6	51.5	93.6	96.9	20.0	91.0	92.8	25.9
GRU+sumNLL	No	85.9	97.0	40.3	81.6	81.3	59.8	**97.7**	**98.9**	**6.6**	91.2	92.4	23.0
TM+avgNLL	5	99.4	99.9	1.1	92.3	88.5	22.4	79.5	90.8	60.8	93.2	94.4	18.7
TM+avgENT	5	99.5	99.9	1.2	92.4	88.8	22.3	78.5	89.2	62.1	92.8	94.0	19.0
TM+sumNLL	5	99.5	99.9	**0.6**	91.1	87.2	26.5	85.3	93.6	43.1	94.4	95.6	16.8
GRU+avgNLL	5	88.8	97.8	35.6	87.4	86.3	46.7	88.4	93.9	29.8	93.3	94.4	18.2
GRU+avgENT	5	90.1	98.2	34.3	87.2	86.5	47.7	88.8	93.8	29.0	93.5	94.5	17.7
GRU+sumNLL	5	85.6	96.6	39.5	84.8	83.1	55.5	93.0	96.4	18.3	92.8	93.9	17.7
HE+avgENT	5	99.2	99.9	2.0	**95.2**	**91.5**	**11.8**	89.3	94.6	29.5	98.6	**98.9**	2.9
HE+sumNLL	5	98.7	99.8	2.8	94.1	89.4	13.5	92.7	96.5	21.8	**98.7**	**98.9**	**2.6**

Table 8.2: OOD Detection performance on SCAN's splits. MCD is the average over the three MCD splits. TM is the transformer. GRU indicates the GRU-based seq2seq model with attention. HE is the heterogeneous ensemble. If "MC" is "No", MC dropout is not used, otherwise the value of "MC" specifies the number of samples (K in §8.1.1).

Method	MC	MCD		
		AUROC	AUPRC	FPR90
TM+avgENT	No	92.9	94.8	18.7
TM+sumNLL	No	91.5	93.5	21.3
GRU+avgENT	No	93.3	95.1	16.4
GRU+sumNLL	No	91.9	93.7	19.6
TM+avgENT	5	95.0	96.5	11.9
TM+sumNLL	5	93.7	95.2	15.6
GRU+avgENT	5	94.6	96.2	11.9
GRU+sumNLL	5	93.3	94.9	15.6
HE+avgENT	5	95.0	96.4	**11.4**
HE+sumNLL	5	93.9	95.3	13.7

Table 8.3: OOD Detection performance on CFQ's MCD splits, averaged over the three provided MCD splits. The table is structured similarly to Table 8.2.

Early stopping: Early stopping is used on all splits except MCD because the OOD validation set makes the training exit too early. Early stopping is performed using the *original* validation set. Note that SCAN's original splits (JUMP, LENGTH and TURN_LEFT), have a validation set that contains ID examples while SCAN's and CFQ's MCD splits have a validation set consisting of OOD examples. In early experiments we found that early stopping based on OOD examples as provided in the MCD splits results in poorer OOD performance with high variance because the resulting model is often obtained from very early training steps. For this reason, we do not use early stopping for MCD splits and instead train for a fixed number of epochs.

Evaluation: To evaluate the ability of the techniques presented in Section 8.1 to detect OOD examples, the following metrics are computed: (1) AUROC↑[1], (2) AUPRC↑[2] and (3) FPR90↓[3]. These metrics are commonly used to measure the performance in OOD detection as well as for binary classifiers in general.

Data splits: The experiments are conducted on a slightly different data split compared to previous work and thus the obtained accuracies might not be directly comparable. To evaluate OOD detection performance, the test set must contain both in-distribution (ID) and out-of-distribution (OOD) examples. The ID test examples must be similar to the training data but must **not** have been seen during training. Due to the lack of a predefined ID test set (that does not overlap with the validation set), to evaluate OOD detection performance, we split off a random 10% of the training examples as the ID test set and train only on the remaining 90%. The reported AUROC, AUPRC and FPR90 metrics are then computed by taking the original test set (which is assumed to be OOD) as examples of the positive class and the ID test examples as the negative class. Note that OOD data are not used in any way during training.

Prediction accuracy: The accuracy of the models is evaluated using a tree-based logical form accuracy. The logical form accuracy considers an example correct if the logical form is equivalent to the target logical form, and which is invariant to the effects of linearization order. In the case of CFQ, which uses SPARQL, the order of conditions for example does not affect the accuracy of the obtained results and is therefore ignored. In the case of SCAN, whose outputs are action sequences, this simply becomes the sequence-level accuracy.

We verify that the query accuracy is similar to previously reported numbers in Table 8.1, which shows that the standard sequence-to-sequence models fail on all compositional generalization scenarios except on the TURN_LEFT split from SCAN. In contrast, the ID test accuracy was near 100% for both datasets.

8.2.1 OOD Detection Performance

The OOD detection performance for the different splits of the SCAN dataset is reported in Table 8.2, and for the CFQ dataset in Table 8.3. SCAN's random split obtains 50% AUROC, which is expected since it does not contain OOD data.[4]

[1] Area under the receiver operating characteristic

[2] Area under the precision-recall curve

[3] FPR90 is the false positive rate at 90% true positive rate.

[4] Note that 50% AUROC corresponds to a random classifier. We leave these results out of the tables because of space constraints.

The effect of MC-dropout: The method described in Section 8.1.1 leads to improvements across different settings and architectures, with the exception of SCAN's length-based split.

The effect of architecture: Different architectures appear to produce markedly different results for different types of splits on SCAN. The transformer performs better than the GRU-based model on the primitive generalization splits (SCAN's JUMP and TURN_LEFT splits), slightly underperforms on the MCD splits of both SCAN and CFQ and is worse on the length-based SCAN split.

How difficult are the different splits? Some of the splits are more challenging to detect than others. The JUMP split appears the easiest to detect using transformers. The TURN_LEFT setting is more challenging. However, the query accuracy on this setting is the highest among all settings, reaching 90%. The high query accuracy might indicate that the test set is closer to the training distribution than in the other settings. Nevertheless, several methods are able to achieve high detection performance for TURN_LEFT. Both MCD splits are solved fairly well, especially when using MC dropout.

The most challenging setting is the length-based split of SCAN. The transformer fails to produce the correct outputs on this split and is also bad at *detecting* when it encounters such examples. On the other hand, the GRU-based model performs much better.

The effect of heterogeneous ensemble: Using the heterogeneous ensemble of a transformer and a GRU-based sequence-to-sequence model to detect OOD examples yields the best overall results, combining the strengths of both architectures. Most notable are the gains on SCAN's MCD task, reaching an FPR90 of less than 5%. It also appears to improve results on the TURN_LEFT split and reaches the detection performance of the GRU-based model on the length-based SCAN split.

8.3 Conclusion

In this work, we investigate how easy it is for neural semantic parsers to detect out-of-distribution examples in the context of compositional generalization. While some recent works [224, 225] investigate some similar methods for structured prediction (for NMT and ASR), to the best of our knowledge, our work is the first to investigate compositional OOD detection for NSP. Our analysis shows that relatively simple uncertainty based methods perform well for RNN-based as well as transformer-based models in most settings. OOD detection can further be improved by using an

Table 8.4: Number of examples per dataset and split.

Dataset	Split	#Train	# Dev	#Test ID	#Test OOD
SCAN	JUMP	12.0k	1.5k	1.3k	7.7k
	TURN_LEFT	18.0k	2.2k	2.0k	1.2k
	LENGTH	14.0k	1.7k	1.5k	3.9k
	mcd-{X}	7.5k	1.0k	0.8k	1.1k
CFQ	mcd-{X}	86.0k	12.0k	9.6k	12.0k

ensemble of both models. Some settings however seem to be especially challenging (e.g. the length-based split of SCAN, where the transformer performs poorly), leaving space for further improvements.

CHAPTER 9

Conclusion and future directions

"I've thought of an ending for my book – 'And he lived happily ever after. . . to the end of his days.' "

– Bilbo, from *The Lord of the Rings*, by J.R.R. Tolkien

In this work, we investigated several questions for semantic parsing and question answering over knowledge graphs using deep learning techniques. In Chapter 1, we introduced and motivated our research, discussed some important challenges, identified more specific research questions targeting some of these challenges, listed our contributions, listed the investigations we conducted in support of this thesis, and provided an outline of this thesis. In Chapter 2, we succinctly cover various topics regarding semantic parsing, question answering over knowledge graphs, and deep learning using neural networks that are necessary to fully understand the remainder of this work. In Chapter 3, we then continue to discuss past and contemporary works that are related to semantic parsing and question answering over knowledge graphs, in particular, but not exclusively, focusing on our research questions, techniques, and datasets we used. The later chapters refer back to these two general chapters for more details on related work or preliminaries in order to help better understand the work and contributions in these later chapters. The following chapters (4-8) contain our contributions. These questions were concerned with the generalization [10] of neural networks in the context of the text-to-query tasks in semantic parsing and with the time complexity of the decoder. We investigated the use of word- and character-level representations for out-of-vocabulary generalization in question answering over knowledge graphs in Chapter 4. In Chapter 5, we investigated transfer learning in the context of question answering over knowledge graphs and found that in general, it improves generalization. In Chapter 6, we looked at insertion-based decoding for trees, which can help to reduce the number of decoding steps when using semantic parsers. Chapter 7 presents our investigation of the effect of linearization order on training sequence-to-sequence semantic parsing models. In Chapter 8, we focus on detecting compositionally out-of-distribution examples.

The rest of this conclusion is structured as follows. In Section 9.1, we look back at the research questions that we covered throughout the chapters and provide concluding remarks. In Section 9.2, we discuss limitations and possible improvements to the presented work. Then, in Section 9.3, we discuss some interesting directions for future work in semantic parsing and question answering over knowledge graphs. Finally, we conclude the thesis in Section 9.4.

9.1 Research Questions

As mentioned in the introduction of this thesis (Chapter 1), we considered five research questions throughout this work. We will now revisit them and provide a concise summary and concluding remarks.

RQ1: Does combining word- and character-level representations improve accuracy in KGQA?

Our first contribution, described in Chapter 4, focuses on this research question. In particular, there we investigated building representation of both entities and relations using their labels and types, where we encode the information on word- and character-level. This avoids the issue that an independent representation vector must be learned for every individual entity and relation, which is problematic when many relations and most entities during testing are not observed during training and whose representations would thus remain untrained. We used the SimpleQuestions data set to evaluate this approach empirically. Exploiting the simple question assumption (query contains only one entity and one relation) of the dataset, we developed a simple ranking model that compares the encoding of the question to the encoding of the query. The results indicate that the proposed approach is effective. At the time the work was performed, the presented approach outperformed other end-to-end networks but it was outperformed by methods that relied on a two-part process where the model instead learned to predict the entity span and then identified the entity in a separate step.

RQ2: Does transfer learning improve KGQA accuracy?

This research question is targeted in Chapter 5, where we investigate transfer learning from pre-trained language models (PLM's) as well as from other question answering datasets. Works proposing models such as ELMO [55], GPT [57], and BERT [7] showed that pre-training language models on the vast amounts of available unsupervised text data results in significant performance increase when subsequently fine-tuning on a specific NLP task. Such pre-training offsets the lack of training data for a specific task by allowing to re-use knowledge learned using data for another task (language modeling) for that specific task. In this way, it improves generalization (also compositional generalization [173]). Specifically, in Chapter 5, we investigate the use of BERT in the context of question answering over knowledge graphs. We investigated BERT for simple questions using the SimpleQuestions dataset, also performing a quantitative analysis with reduced amounts of training data, as well as a qualitative analysis where we inspected the attention patterns inside BERT. We found that using BERT improves accuracy compared to a normal baseline, and that the improvement is larger with less available data. The inspection of attention patterns reveals that during fine-tuning the model learns to identify the relation-specific parts of the question. We also presented a novel approach for answering complex questions, evaluated on the LC-QuAD and QALD datasets. Within this approach, we used BERT to investigate transfer from PLM's for complex questions, where we used the LC-QuAD and QALD datasets. We found that while BERT significantly improves results for LC-QuAD, it does not work well for QALD. We are not sure what are the reasons for these results but suspect the extremely small size of QALD could be part of the reason. In addition, we also investigated transfer between different KGQA tasks, where we pre-trained on LC-QuAD and finetuned on QALD. This resulted in improvement.

RQ3: Does insertion-based decoding improve accuracy and how much does it decrease the number of decoding steps?

In Chapter 6, we investigated insertion-based decoding, which reduces the number of decoding steps and thus has the potential to reduce the decoding time. Where sequences and other structures are typically decoded autoregressively in a left-to-right order, this incurs a number of decoding steps equal to the size of the produced output. In addition, every next generated token is conditioned on all previously generated tokens. In our work, we focused on insertion-based decoding using transformers. In addition to reducing the number of decoding steps, maximally parallel insertion-based decoding using the investigated methods introduces more independence in the output structure, which may have an effect on accuracy. First, we tested sequence-based insertion decoding [23], which is a conceptually simple paradigm that enables us to go down to $\log_2(n)$ decoding steps for a sequence of length n. We also proposed tree-based insertion decoding since trees are important and can be used to better represent queries, for example using the abstract syntax tree of code, or LISP's s-expressions. The developed tree decoder defines additional insertion slots and can go below $\log_2(n)$ decoding steps for a sequence of length n. In addition, it guarantees that intermediate steps produce valid trees.

We ran experiments on the Overnight dataset for semantic parsing and found that we are able to use much fewer decoding steps without sacrificing accuracy. Moreover, the proposed tree-based insertion decoding approach resulted in fewer decoding steps than the sequence-based insertion decoder. However, we were not able to demonstrate significant improvements in accuracy due to the independence relations and the inductive bias of maximally parallel insertion-based decoding, or due to the tree-based insertion decoder.

RQ4: Does the query linearization order used during training affect accuracy?

We focused on this question in Chapter 7. When training neural semantic parsing models, typically left-to-right autoregressive sequence-to-sequence/tree/action models are trained using a fixed linearization of a query tree. However, queries may contain parts where changing the order does not change their meaning and training with just one order may lead to learning more *spurious* patterns. In Chapter 7, we presented a relatively simple and general model that performs competitively with the state-of-the-art on the WikiSQL dataset at the time the work was performed. In addition, we investigated how decoding order during supervision matters for WikiSQL and found that inconsistent order can make results worse on the in-distribution test set. We also proposed to use a training method based on dynamic oracles that is insensitive to the original linearization order in the given data. We found that it leads to a small improvement on in-distribution test data and performs better than REINFORCE. A limitation of our work is that we did not investigate this question in the context of compositional generalization, even though –as it was found later– there may exist a relation: learning spurious patterns that arise due to training with only one possible linearization of every query may hurt generalization to novel combinations. In fact, Guo et al. [16] show that their order-agnostic *poset decoder* achieves improvements on the challenging splits of the recently proposed CFQ dataset [14] for measuring the compositional generalization for semantic parsing for KGQA.

RQ5: Can we detect compositionally out-of-distribution inputs in semantic parsing?

We addressed this research question in Chapter 8. It has been shown that standard sequence-to-sequence models are not able to generalize well to novel combinations of tokens observed during training, as for example evaluated using the CFQ and SCAN datasets. In Chapter 8, we investigate several techniques to detect whether an input is compositionally out-of-distribution. In particular, we focus on metrics that re-use the predictive distribution of the semantic parsing model itself. This makes it extremely simple to use as it does not require model modifications or training additional models. We also investigated the use of MC Dropout, which can be seen as an approximation of Bayesian Neural Networks. We found that the basic metrics can already be used to detect OOD examples of different types and that some standard techniques significantly improve the detection performance. We also found that transformer- and RNN-based models are better at detecting different types of compositionally OOD inputs. Thus, we also proposed a simple extension that combines the predictive distributions of transformer- and RNN-based models for better OOD detection and provides the best results on average. We evaluated our approach using the CFQ and SCAN datasets.

9.2 Limitations and Possible Improvements

The work presented in this thesis has limitations. Regarding the first research question, **RQ1**, there could be approaches further improving OOV generalization. Some examples are using BytePair or WordPiece tokenization rather than going to character-level, and including the relation token itself [123]. In addition, our study for Contribution 1 could further be extended to other datasets and more complex questions. Another limitation of the performed analysis is that we did not measure how portable the approach is to other knowledge bases. This can be done by creating/adapting a dataset of simple questions to another KG like DBpedia or Wikidata and test how well a network trained on the original SimpleQuestions dataset works on the new dataset. Because such adaptations and additional experiments would require significant implementation and evaluation efforts, we leave them for future work. In our work on transfer learning, to answer **RQ2**, we conducted analyses using SimpleQuestions, LC-QuAD, and QALD. The analysis could further benefit from extention to other common dataset, such as WebQuestionsSP (see Section 3.2.3 for more information about the available datasets). In addition, there exist PLM's other than BERT, which can additionally be tested. Some of these were pre-trained differently, for example, T5 [61] and BART [60] were pre-trained as sequence-to-sequence models, and thus might have some advantages on sequence-to-sequence tasks like semantic parsing. It would make for a more complete but very labor- and compute-intensive evaluation study to compare various PLMs. However, we do not expect the results to qualitatively change from our findings on BERT. Another limitation of our work in Chapter 5 is that we were not able to answer the question why fine-tuning BERT leads to a *decrease* in accuracy compared to not using BERT for QALD-7. As mentioned in Section 5.8.3, this might be because of the small size of the dataset but it would be interesting to further investigate the causes of this degradation, especially because we found that pre-trained models should be beneficial for smaller dataset sizes (see first part of Chapter 5). In our Contribution 3 for **RQ3**, as always, the analysis can be improved by evaluating on additional tasks and datasets In addition, we were unable to measure the actual speed-up because of an inefficient implementation of our insertion-based tree decoder, which has a severe CPU bottleneck and inefficient memory usage. Solving this problem would require significant engineering efforts. In particular, we would need an efficient implementation of sparse attention for transformers with relative positions and careful rewriting of the CPU-heavy code. This would bring the implementation closer to the best possible speed-up in deployment. In addition, the computation can further be reduced by investigating ways to compute representations for slots incrementally. Currently, the entire partial tree (or subsequence) is re-encoded from scratch at every decoding step. While this does not necessarily increase response time because the computations are fully parallellizable (at least for transformers), and uses as many layers as the left-to-right decoder (and thus leads to similar amount of time to execute one decoding step), it still requires more computation and memory overall. Avoiding recomputing representations from scratch would make for a more efficient insertion-based decoder. For **RQ4**, it would be nice to extend the analysis to investigate the effect of the proposed dynamic oracle approach on compositional generalization. Guo et al. [16] show that their permutation invariant decoder achieves improvements and it would be interesting to know what effect our proposed method has compared to alternatives like Guo et al. [16]. In addition, evaluation on CFQ would further extend the empirical evidence of our approach. Unfortunately, none of the common compositional generalization datasets for semantic parsing existed at the time we performed our work on RQ4, and performing this analysis would require significant additional efforts. Our investigation of **RQ5** could further be extended with additional datasets that have recently been proposed, like COGS [165] and GrailQA [10]. In addition, for a more complete study of compositional out-of-distribution detection, other existing OOD detection techniques can be evaluated, for example techniques relying on generative models [218]. Some interesting future work regarding measuring the reliability of semantic parsing models that would extend our contributions would be to perform a calibration analysis for both ID and OOD settings.

9.3 Future Research Directions

Among the many interesting possible future research directions, in this section, we elaborate on a few topics that we believe can be especially interesting to explore in future work. These include improving compositional generalization and investigating better detection of compositionally challenging inputs, further investigation into advanced specialized transfer learning techniques and alternative execution methods in semantic parsing.

9.3.1 Compositional Generalization

While a multitude of works recently addressed compositional generalization in semantic parsing, we believe it's still an important and promising direction of research. We believe it is also interesting to investigate techniques to improve the robustness to spurious correlations in training data *without relying on strong compositional inductive biases* that many works targeting compositional generalization exploited. In other words, it would be nice to use existing, more general, models, which were shown to generalize poorly when strong spurious correlations are present. Some works, for example those targeting data augmentation [168] and/or improved training techniques [179] go in this direction. Another interesting avenue with potential practical importance could be further exploration of OOD detection in semantic parsing and question answering over knowledge graphs. While in Chapter 8, we focused on detecting compositionally OOD examples. This analysis can be further extended with additional techniques, and new techniques can be developed focusing specifically on sequence-to-query models. In addition, the analysis can be extended to investigate in detail how different linguistic phenomena impact OOD detection, for example using the COGS [165] dataset.

9.3.2 Transfer Learning

Pre-trained language models are by now ubiquitous in much of NLP research and applications. However, we believe there is still room for improvement, especially in task-specific procedures on top of pre-trained models or in alternative pre-trained models. One interesting research direction that, to the best of our knowledge, is insufficiently investigated at this time, is the use of unsupervised pre-training for tasks involving knowledge graphs and text (of which semantic parsing for question answering is one example). Works like TAPAS [226] and TaBERT [227] recently investigated similar pre-training objectives in the context of text-to-SQL or table-based question answering tasks. In these works, roughly speaking, a joint MLM objective on the mixture of text and related tables has been used to pre-train a model that is subsequently fine-tuned for text-to-SQL tasks. Most recently, JointGT [228] investigated a similar pre-training objective for knowledge graphs and text: their objective includes masked text reconstruction given a subgraph of the knowledge graph and masked subgraph reconstruction given some related text. However, Ke et al. [228] evaluate only on the KG-to-text natural language generation (NLG) task. We believe it would be interesting to investigate the effect of such pre-training for different downstream tasks, such as knowledge graph modeling, relation extraction, weakly supervised question answering over knowledge graphs and semantic parsing for question answering over knowledge graphs. Knowledge graph modeling involves predicting links using a local subgraph. KG relation extraction requires predicting which KG relation is expressed in some text. Note that MLM-like pre-training on the KG subgraph and text can share knowledge with these two tasks. The task of question answering over knowledge graphs can be posed as a fill-in-the-blank NLG task (e.g. “The president of New Zealand is _____” or “Who is the president of New Zealand? _____ ”). Alternatively, it can also be posed as a query generation task, where we need to produce a logical form (e.g. SPARQL query) from a given question. While the latter task corresponds to semantic parsing in the context of question answering over knowledge graphs, the former task, if used without additional supervision in the form of logical forms, is a weakly supervised question answering task.

9.3.3 Alternative execution and supervision

While knowledge graphs are good sources of well-defined knowledge that can be reasoned over explicitly, they can still be incomplete. In fact, knowledge graph modeling tasks, such as link prediction focus on completing the knowledge contained in a KG by inferring new relations between existing (or new) entities. Knowledge graph embedding models such as TransE [141] are an example of such models. Usually, question answering is performed over existing knowledge graphs that might have missing facts. Because some facts might be missing, the results of the question answering system then might not be completely satisfactory. One way to resolve this is to use some techniques to build a more complete version of the knowledge graph and somehow incorporate the uncertainty over the inferred facts into the answer of the system. Another way to resolve it is to incorporate the knowledge model in the question answering system directly, as for example has been recently proposed by Ren et al. [229], who rely on their previous work Query2Box [230]. In Query2Box, the authors investigated the latent execution of knowledge graph queries. Rather than executing a query explicitly on the knowledge graph, in Query2Box, the queries are executed in the embedding space and the result is a region of the embedding space, which specifies a set of returned entities. In LEGO [229], the authors propose a method that does question answering and returns an answer in the latent space using Query2Box. The authors use weak supervision, where the correct logical form is not given, and use learned pruning to limit the search space. We believe the research direction of incorporating inference and reasoning into the question answering system can lead to interesting findings.

9.3.4 Graph Representation Learning

Improvements in graph representation learning (GRL) can also benefit semantic parsing and question answering over knowledge graphs. Firstly, the queries can be represented as trees, which is a subtype of graphs, and improved ways of representing queries can lead to better performing question answering systems. In fact, several works already investigated the application of GNNs for semantic parsing and question answering [231, 232]. Graph neural networks (GNN) are also useful for text processing tasks, especially when additional annotation is given in the form of consituency and dependency parses, or other annotations. These annotations can be naturally encoded using GNNs to better exploit these rich features, which might be helpful for semantic parsing and other tasks. GNNs and other GRL methods are of particular interest for representation learning over knowledge graphs as well since they are multi-relational graphs. Note that our Publication 8 concerns GNNs but we did not include it in this thesis because it is not directly evaluated on semantic parsing or question answering tasks.

9.4 Closing Remarks

The question answering task has been around for quite some time and has attracted significant research interest over the last decades. In the last years, most focus in the research community went into deep learning approaches for the task. Despite the significant research efforts, which greatly improved performance on semantic parsing and question answering and our understanding of deep learning in its context, we believe there are still many open questions.

In this thesis, we focused on some of the interesting open questions in the field. Many of them involve the generalization ability of neural networks for semantic parsing for KGQA, which can be regarded as the central challenge when applying machine learning. Specifically, we investigated generalization to new tokens (OOV generalization), transfer learning, detecting compositionally OOD examples, the effect of linearization order when training, and insertionion-based decoding of trees. While these questions were interesting for us, recently some other interesting questions gained more attention, such as improving compositional generalization, which also ties in with the general topic of out-of-distribution generalization that is gaining interest in the general machine learning community. Other questions in neural networks and NLP, for example more efficient fine-tuning techniques, might also inspire important future research in the area.

In the end, we hope that this thesis is helpful for other students and researchers in deep learning and NLP. Thank you for reading.

Bibliography

[1] K. Bollacker et al.,
"Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge,"
Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data,
SIGMOD '08, ACM, 2008 1247, ISBN: 978-1-60558-102-6,
URL: `http://doi.acm.org/10.1145/1376616.1376746` (cit. on pp. 3, 21, 47, 77).

[2] P. N. Mendes et al., "DBpedia spotlight: shedding light on the web of documents,"
Proceedings of the 7th international conference on semantic systems, ACM, 2011 1
(cit. on pp. 3, 20).

[3] J. Lehmann et al.,
DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia,
Semantic Web Journal **6** (2015) 167, Outstanding Paper Award (Best 2014 SWJ Paper)
(cit. on pp. 3, 21, 89).

[4] D. Vrandecic and M. Krötzsch, *Wikidata: a free collaborative knowledgebase*,
Commun. ACM **57** (2014) 78, URL: `https://doi.org/10.1145/2629489`
(cit. on pp. 3, 21).

[5] C. Cortes and V. Vapnik, *Support-vector networks*, Machine learning **20** (1995) 273
(cit. on p. 3).

[6] I. J. Goodfellow et al., *Explaining and harnessing adversarial examples*,
arXiv preprint arXiv:1412.6572 (2014) (cit. on p. 3).

[7] J. Devlin et al.,
"BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,"
Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 2019 4171 (cit. on pp. 4, 8, 30, 35, 53, 77, 78, 89, 93, 100, 101, 105, 142).

[8] T. B. Brown et al., *Language models are few-shot learners*,
arXiv preprint arXiv:2005.14165 (2020) (cit. on pp. 4, 35).

[9] N. Srivastava et al., *Dropout: a simple way to prevent neural networks from overfitting*,
The journal of machine learning research **15** (2014) 1929 (cit. on p. 4).

[10] Y. Gu et al.,
"Beyond IID: three levels of generalization for question answering on knowledge bases,"
Proceedings of the Web Conference 2021, 2021 3477 (cit. on pp. 4, 5, 51–53, 61, 141, 145).

[11] R. Jia and P. Liang, "Data Recombination for Neural Semantic Parsing,"
Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), vol. 1, 2016 12 (cit. on pp. 5, 7, 40).

[12] T. Schick and H. Schütze, "Rare words: A major problem for contextualized embeddings and how to fix it by attentive mimicking," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 05, 2020 8766 (cit. on p. 5).

[13] B. Lake and M. Baroni, "Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks," *International Conference on Machine Learning*, PMLR, 2018 2873 (cit. on pp. 6, 12, 56, 133, 135).

[14] D. Keysers et al., "Measuring Compositional Generalization: A Comprehensive Method on Realistic Data," *International Conference on Learning Representations*, 2019 (cit. on pp. 6, 10, 12, 50–52, 56, 133, 135, 143).

[15] L. Dong and M. Lapata, "Language to Logical Form with Neural Attention.," *ACL (1)*, The Association for Computer Linguistics, 2016, ISBN: 978-1-945626-00-5, URL: `http://dblp.uni-trier.de/db/conf/acl/acl2016-1.html#DongL16` (cit. on pp. 6, 12, 37, 38, 113, 114, 131).

[16] Y. Guo et al., *Hierarchical Poset Decoding for Compositional Generalization in Language*, Advances in Neural Information Processing Systems **33** (2020) (cit. on pp. 6, 10, 55, 57, 133, 143, 145).

[17] L. Dong and M. Lapata, "Coarse-to-Fine Decoding for Neural Semantic Parsing," *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2018 731 (cit. on pp. 6, 40, 127, 128, 131).

[18] A. Vaswani et al., "Attention is All you Need," *NIPS*, 2017 (cit. on pp. 8, 26, 29, 30, 77, 78, 105, 135).

[19] O. Vinyals et al., *Order matters: Sequence to sequence for sets*, arXiv preprint arXiv:1511.06391 (2015) (cit. on pp. 10, 54).

[20] A. Bordes et al., "Open question answering with weakly supervised embedding models," *Proceedings of the 2014th European Conference on Machine Learning and Knowledge Discovery in Databases-Volume Part I*, 2014 165 (cit. on pp. 11, 48, 77).

[21] P. Trivedi et al., "Lc-quad: A corpus for complex question answering over knowledge graphs," *International Semantic Web Conference*, Springer, 2017 210 (cit. on pp. 11, 33, 51, 52, 77, 89, 95, 96).

[22] R. Usbeck et al., "7th Open Challenge on Question Answering over Linked Data (QALD-7)," *Semantic Web Evaluation Challenge*, Springer, 2017 59 (cit. on pp. 11, 77, 89, 95).

[23] M. Stern et al., "Insertion transformer: Flexible sequence generation via insertion operations," *International Conference on Machine Learning*, PMLR, 2019 5976 (cit. on pp. 11, 12, 32, 41, 103, 104, 106, 107, 111, 143).

[24] Y. Goldberg and J. Nivre, "A dynamic oracle for arc-eager dependency parsing," *Proceedings of COLING 2012*, 2012 959 (cit. on pp. 12, 54, 116, 120, 125, 126).

[25] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," *international conference on machine learning*, PMLR, 2016 1050 (cit. on pp. 12, 134).

[26] P. Liang, *Lambda dependency-based compositional semantics*, arXiv preprint arXiv:1309.4408 (2013) (cit. on p. 18).

[27] R. J. Kate and R. J. Mooney, "Using string-kernels for learning semantic parsers," *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, 2006 913 (cit. on p. 18).

[28] H. P. Barendregt, *Introduction to lambda calculus*, (1984) (cit. on p. 18).

[29] A. Church, *An unsolvable problem of elementary number theory*, American journal of mathematics **58** (1936) 345 (cit. on p. 18).

[30] W. Ackermann and D. Hilbert, *Grundzüge der theoretischen Logik*, Berlin, Springcr **1037** (1928) 4 (cit. on p. 18).

[31] L. Banarescu et al., "Abstract meaning representation for sembanking," *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, 2013 178 (cit. on p. 19).

[32] P. Kapanipathi et al., *Leveraging Abstract Meaning Representation for Knowledge Base Question Answering*, Findings of the Association for Computational Linguistics: ACL (2021) (cit. on p. 19).

[33] N. Radoev et al., "A Language Adaptive Method for Question Answering on French and English," *Semantic Web Challenges - 5th SemWebEval Challenge at ESWC 2018, Heraklion, Greece, June 3-7, 2018, Revised Selected Papers*, ed. by D. Buscaldi et al., vol. 927, Communications in Computer and Information Science, Springer, 2018 98, URL: `https://doi.org/10.1007/978-3-030-00072-1%5C_9` (cit. on p. 20).

[34] M. Dubey et al., "AskNow: A Framework for Natural Language Query Formalization in SPARQL.," *ESWC*, ed. by H. Sack et al., vol. 9678, Lecture Notes in Computer Science, Springer, 2016 300, ISBN: 978-3-319-34128-6, URL: `http://dblp.uni-trier.de/db/conf/esws/eswc2016.html#DubeyDSHL16` (cit. on pp. 20, 51, 62, 88).

[35] Y. Yang and M.-W. Chang, *S-mart: Novel tree-based structured learning algorithms applied to tweet entity linking*, arXiv preprint arXiv:1609.08075 (2016) (cit. on p. 20).

[36] R. Sennrich et al., "Neural Machine Translation of Rare Words with Subword Units," *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016 1715 (cit. on pp. 21, 61).

[37] M. Schuster and K. Nakajima, "Japanese and korean voice search," *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2012 5149 (cit. on pp. 21, 22, 36, 61).

[38] Y. Wu et al., *Google's neural machine translation system: Bridging the gap between human and machine translation*, arXiv preprint arXiv:1609.08144 (2016) (cit. on p. 21).

[39] T. Mikolov et al., *Efficient estimation of word representations in vector space*, arXiv preprint arXiv:1301.3781 (2013) (cit. on pp. 23, 34, 47).

[40] J. Pennington et al., “Glove: Global vectors for word representation,” *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014 1532 (cit. on pp. 23, 34, 63, 65, 82, 96, 125).

[41] S. Hochreiter, *Untersuchungen zu dynamischen neuronalen Netzen*, PhD thesis: Institut f. Informatik, Technische Universitaet Munich, 1991 (cit. on p. 24).

[42] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural computation **9** (1997) 1735 (cit. on pp. 24, 46, 50, 79, 97, 122).

[43] R. Pascanu et al., “On the difficulty of training recurrent neural networks,” *International conference on machine learning*, PMLR, 2013 1310 (cit. on p. 24).

[44] Y. Bengio et al., *Learning long-term dependencies with gradient descent is difficult*, IEEE transactions on neural networks **5** (1994) 157 (cit. on p. 24).

[45] K. Cho et al., *On the properties of neural machine translation: Encoder-decoder approaches*, arXiv preprint arXiv:1409.1259 (2014) (cit. on pp. 24, 64, 135).

[46] D. Bahdanau et al., *Neural machine translation by jointly learning to align and translate*, arXiv preprint arXiv:1409.0473 (2014) (cit. on pp. 25, 26, 46, 122).

[47] M.-T. Luong et al., “Effective Approaches to Attention-based Neural Machine Translation,” *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015 1412 (cit. on p. 25).

[48] P. Shaw et al., “Self-Attention with Relative Position Representations,” *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018 464 (cit. on pp. 30, 108).

[49] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980 (2014) (cit. on pp. 33, 81, 114, 135).

[50] J. M. Zelle and R. J. Mooney, “Learning to parse database queries using inductive logic programming.,” *Proceedings of the national conference on artificial intelligence*, 1996 (cit. on p. 33).

[51] Q. Cai and A. Yates, “Large-scale semantic parsing via schema matching and lexicon extension,” *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2013 423 (cit. on pp. 33, 48, 51).

[52] P. Bojanowski et al., *Enriching word vectors with subword information*, Transactions of the Association for Computational Linguistics **5** (2017) 135 (cit. on p. 34).

[53] Z. S. Harris, *Distributional structure*, Word **10** (1954) 146 (cit. on p. 35).

[54] J. R. Firth, *A synopsis of linguistic theory, 1930-1955*, Studies in linguistic analysis (1957) (cit. on p. 35).

[55] M. Peters et al., “Deep Contextualized Word Representations,” *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018 2227 (cit. on pp. 35, 78, 142).

[56] Y. Liu et al., *Roberta: A robustly optimized bert pretraining approach*, arXiv preprint arXiv:1907.11692 (2019) (cit. on pp. 35, 36, 50).

[57] A. Radford et al., *Improving language understanding by generative pre-training*, () (cit. on pp. 35, 78, 93, 142).

[58] A. Radford et al., *Language Models are Unsupervised Multitask Learners*, () (cit. on pp. 35, 78).

[59] Z. Yang et al., *XLNet: Generalized Autoregressive Pretraining for Language Understanding*, Advances in Neural Information Processing Systems **32** (2019) 5753 (cit. on p. 35).

[60] M. Lewis et al., “BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension,” *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020 7871 (cit. on pp. 35, 36, 144).

[61] C. Raffel et al., *Exploring the limits of transfer learning with a unified text-to-text transformer*, arXiv preprint arXiv:1910.10683 (2019) (cit. on pp. 36, 57, 144).

[62] L. Zettlemoyer and M. Collins, “Online learning of relaxed CCG grammars for parsing to logical form,” *Proceedings of EMNLP-CoNLL 2007*, 2007 (cit. on p. 37).

[63] S. Reddy et al., *Large-scale Semantic Parsing without Question-Answer Pairs*, Transactions of the Association of Computational Linguistics **2** (2014) 377 (cit. on pp. 37, 40, 48, 51).

[64] J. Berant et al., “Semantic parsing on freebase from question-answer pairs,” *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013 1533 (cit. on pp. 37, 47, 51).

[65] L. S. Zettlemoyer and M. Collins, “Learning to map sentences to logical form: structured classification with probabilistic categorial grammars,” *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, 2005 658 (cit. on p. 37).

[66] J. Clarke et al., “Driving semantic parsing from the world’s response,” *Proceedings of the fourteenth conference on computational natural language learning*, 2010 18 (cit. on p. 37).

[67] Y. Artzi and L. Zettlemoyer, *Weakly supervised learning of semantic parsers for mapping instructions to actions*, Transactions of the Association for Computational Linguistics **1** (2013) 49 (cit. on p. 37).

[68] A. Kamath and R. Das, “A Survey on Semantic Parsing,” *Automated Knowledge Base Construction (AKBC)*, 2018 (cit. on p. 37).

[69] P. Kumar and S. Bedathur,
A Survey on Semantic Parsing from the perspective of Compositionality,
arXiv preprint arXiv:2009.14116 (2020) (cit. on p. 37).

[70] Q. Zhu et al., *Statistical learning for semantic parsing: A survey*,
Big Data Mining and Analytics **2** (2019) 217 (cit. on p. 37).

[71] D. Alvarez-Melis and T. S. Jaakkola,
Tree-structured decoding with doubly-recurrent neural networks, (2017) (cit. on pp. 38, 131).

[72] J. Cheng et al.,
"Learning Structured Natural Language Representations for Semantic Parsing,"
Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), vol. 1, 2017 44 (cit. on pp. 38, 39).

[73] J. Cheng and M. Lapata,
"Weakly-Supervised Neural Semantic Parsing with a Generative Ranker,"
Proceedings of the 22nd Conference on Computational Natural Language Learning, CoNLL 2018, Brussels, Belgium, October 31 - November 1, 2018, 2018 356,
URL: `https://aclanthology.info/papers/K18-1035/k18-1035`
(cit. on pp. 38, 39, 41, 44).

[74] J. Cheng et al., *Learning an executable neural semantic parser*,
Computational Linguistics **45** (2019) 59 (cit. on pp. 38, 39).

[75] C. Dyer et al., *Transition-based dependency parsing with stack long short-term memory*,
arXiv preprint arXiv:1505.08075 (2015) (cit. on p. 38).

[76] M. Rabinovich et al., "Abstract Syntax Networks for Code Generation and Semantic Parsing,"
Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2017 1139 (cit. on p. 39).

[77] P. Yin and G. Neubig, "A Syntactic Neural Model for General-Purpose Code Generation,"
Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2017 440 (cit. on p. 39).

[78] P. Yin and G. Neubig, "TRANX: A Transition-based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation," *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2018 7 (cit. on p. 39).

[79] K. Lin et al., *Grammar-based neural text-to-sql generation*,
arXiv preprint arXiv:1905.13326 (2019) (cit. on p. 39).

[80] J. Krishnamurthy et al.,
"Neural semantic parsing with type constraints for semi-structured tables,"
Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, 2017 1516 (cit. on pp. 39, 120, 127).

[81] T. Yu et al.,
"SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-Domain Text-to-SQL Task,"
Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018 1653 (cit. on pp. 39, 53).

[82] D. Guo et al.,
"Dialog-to-action: Conversational question answering over a large-scale knowledge base," *Advances in Neural Information Processing Systems*, 2018 2942 (cit. on pp. 39, 51).

[83] T. Shen et al., "Multi-Task Learning for Conversational Question Answering over a Large-Scale Knowledge Base," *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019 2442 (cit. on pp. 39, 51).

[84] W.-t. Yih et al., "Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base," *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, vol. 1, 2015 1321 (cit. on pp. 39, 40, 48, 49, 88, 89, 95).

[85] B. Chen et al.,
"Sequence-to-Action: End-to-End Semantic Graph Generation for Semantic Parsing," *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Association for Computational Linguistics, 2018 766,
URL: `https://www.aclweb.org/anthology/P18-1071` (cit. on pp. 40, 113, 115).

[86] O. Vinyals et al., "Pointer networks," *Advances in Neural Information Processing Systems*, 2015 2692 (cit. on p. 40).

[87] A. See et al., "Get To The Point: Summarization with Pointer-Generator Networks," *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2017 1073 (cit. on pp. 40, 122, 124).

[88] J. Gu et al., "Incorporating Copying Mechanism in Sequence-to-Sequence Learning," *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2016 1631 (cit. on pp. 40, 122, 124).

[89] P. Shaw et al., "Generating Logical Forms from Graph Representations of Text and Entities," *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019 95 (cit. on p. 40).

[90] T. Shi et al.,
IncSQL: Training Incremental Text-to-SQL Parsers with Non-Deterministic Oracles, CoRR **abs/1809.05054** (2018), arXiv: `1809.05054`,
URL: `http://arxiv.org/abs/1809.05054` (cit. on pp. 40, 53, 54, 120, 126, 128, 131).

[91] X. Xu et al.,
Sqlnet: Generating structured queries from natural language without reinforcement learning, arXiv preprint arXiv:1711.04436 (2017) (cit. on pp. 40, 54, 120, 126, 128, 130, 131).

[92] V. Zhong et al.,
Seq2sql: Generating structured queries from natural language using reinforcement learning, arXiv preprint arXiv:1709.00103 (2017) (cit. on pp. 40, 54, 55, 120, 126–128, 130, 131).

[93] M. Damonte et al., *Practical Semantic Parsing for Spoken Language Understanding*, arXiv preprint arXiv:1903.04521 (2019) (cit. on pp. 40, 113, 115).

[94] J. Gu et al., *Insertion-based Decoding with Automatically Inferred Generation Order*, Transactions of the Association for Computational Linguistics **7** (2019) 661 (cit. on p. 41).

[95] J. Gu et al., "Levenshtein Transformer," *Advances in Neural Information Processing Systems*, ed. by H. Wallach et al., vol. 32, Curran Associates, Inc., 2019 11181, URL: `https://proceedings.neurips.cc/paper/2019/file/675f9820626f5bc0afb47b57890b466e-Paper.pdf` (cit. on pp. 41, 116).

[96] X. Ma et al., "FlowSeq: Non-Autoregressive Conditional Sequence Generation with Generative Flow," *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019 4273 (cit. on pp. 41, 103).

[97] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," *International Conference on Machine Learning*, PMLR, 2015 1530 (cit. on p. 41).

[98] M. Ghazvininejad et al., "Mask-Predict: Parallel Decoding of Conditional Masked Language Models," *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019 6114 (cit. on pp. 41, 103).

[99] J. Gu et al., *Non-autoregressive neural machine translation*, arXiv preprint arXiv:1711.02281 (2017) (cit. on pp. 41, 103).

[100] J. Kasai et al., *Parallel machine translation with disentangled context transformer*, arXiv preprint arXiv:2001.05136 (2020) (cit. on pp. 41, 103).

[101] Q. Zhu et al., "Don't Parse, Insert: Multilingual Semantic Parsing with Insertion Based Decoding," *Proceedings of the 24th Conference on Computational Natural Language Learning*, 2020 496 (cit. on p. 41).

[102] A. Coucke et al., *Snips Voice Platform: an embedded Spoken Language Understanding system for private-by-design voice interfaces*, (2018) (cit. on p. 41).

[103] P. Price, "Evaluation of spoken language systems: The ATIS domain," *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*, 1990 (cit. on pp. 41, 127).

[104] S. Gupta et al., "Semantic Parsing for Task Oriented Dialog using Hierarchical Representations," *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018 2787 (cit. on p. 41).

[105] O. Rubin and J. Berant, *SmBoP: Semi-autoregressive Bottom-up Semantic Parsing*, arXiv preprint arXiv:2010.12412 (2020) (cit. on p. 41).

[106] P. Pasupat and P. Liang, "Inferring Logical Forms From Denotations," *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2016 23 (cit. on p. 42).

[107] S. Min et al., "A Discrete Hard EM Approach for Weakly Supervised Question Answering," *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019 2851 (cit. on pp. 42, 58).

[108] J. Herzig and J. Berant, *Span-based semantic parsing for compositional generalization*, arXiv preprint arXiv:2009.06040 (2020) (cit. on pp. 43, 57, 58, 133).

[109] R. Agarwal et al., *Learning to Generalize from Sparse and Underspecified Rewards*, CoRR **abs/1902.07198** (2019) (cit. on pp. 43, 44).

[110] C. Liang et al., "Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision," *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2017 23 (cit. on p. 43).

[111] C. Liang et al., "Memory Augmented Policy Optimization for Program Synthesis and Semantic Parsing," *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.* 2018 10015, URL: `http://papers.nips.cc/paper/8204-memory-augmented-policy-optimization-for-program-synthesis-and-semantic-parsing` (cit. on pp. 43, 44).

[112] K. Guu et al., "From Language to Programs: Bridging Reinforcement Learning and Maximum Marginal Likelihood," *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, 2017 1051, URL: `https://doi.org/10.18653/v1/P17-1097` (cit. on p. 44).

[113] M. Norouzi et al., "Reward Augmented Maximum Likelihood for Neural Structured Prediction," *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016 1723, URL: `http://papers.nips.cc/paper/6547-reward-augmented-maximum-likelihood-for-neural-structured-prediction` (cit. on p. 44).

[114] N. L. Roux, "Tighter bounds lead to improved classifiers," *International Conference on Learning Representations*, 2017 (cit. on p. 44).

[115] D. Misra et al., "Policy Shaping and Generalized Update Equations for Semantic Parsing from Denotations," *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018 2442 (cit. on p. 44).

[116] A. Bordes et al., "Question Answering with Subgraph Embeddings," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014 615 (cit. on pp. 44, 48).

[117] S. Hakimov et al., "Evaluating Architectural Choices for Deep Learning Approaches for Question Answering over Knowledge Bases," *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*, IEEE, 2019 110 (cit. on p. 45).

[118] S. Mohammed et al., "Strong Baselines for Simple Question Answering over Knowledge Graphs with and without Neural Networks," *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, vol. 2, 2018 291 (cit. on pp. 45, 46, 79, 82–86, 96).

[119] M. Petrochuk and L. Zettlemoyer, *SimpleQuestions Nearly Solved: A New Upperbound and Baseline Approach*, arXiv preprint arXiv:1804.08798 (2018) (cit. on pp. 45, 46, 79, 81, 82, 85, 86).

[120] K. Cho et al., "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2014 1724, URL: `http://aclweb.org/anthology/D/D14/D14-1179.pdf` (cit. on p. 46).

[121] A. Bordes et al., *Large-scale simple question answering with memory networks*, arXiv preprint arXiv:1506.02075 (2015) (cit. on pp. 46, 51, 52, 62, 72–74, 78, 86).

[122] X. He and D. Golub, "Character-Level Question Answering with Attention," *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016 1598 (cit. on pp. 46, 53, 79, 86).

[123] M. Yu et al., "Improved Neural Relation Detection for Knowledge Base Question Answering," *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2017 571 (cit. on pp. 47, 49, 53, 79, 81, 86, 88, 95–97, 144).

[124] Z. Dai et al., *CFO: Conditional Focused Neural Question Answering with Large-scale Knowledge Bases*, arXiv preprint arXiv:1606.01994 (2016) (cit. on pp. 47, 63, 73, 74, 76, 79, 86).

[125] W. Yin et al., "Simple Question Answering by Attentive Convolutional Neural Network," *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, 11-16 December 2016, Osaka, Japan, ACL 2016*, 2016 (cit. on pp. 47, 63, 73, 74, 76, 81, 86).

[126] J. Berant and P. Liang, "Semantic Parsing via Paraphrasing," *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, The Association for Computer Linguistics, 2014 1415, URL: `https://doi.org/10.3115/v1/p14-1133` (cit. on pp. 47, 48, 88).

[127] X. Yao and B. Van Durme, "Information extraction over structured data: Question answering with freebase," *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2014 956 (cit. on p. 48).

[128] H. Bast and E. Haussmann, "More accurate question answering on freebase," *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2015 1431 (cit. on pp. 48, 95).

[129] W.-t. Yih et al.,
"The value of semantic parse labeling for knowledge base question answering,"
Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), vol. 2, 2016 201 (cit. on pp. 48, 50–52).

[130] A. Talmor and J. Berant,
"The Web as a Knowledge-Base for Answering Complex Questions,"
Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), 2018 641 (cit. on pp. 49–51).

[131] Y. Lan and J. Jiang, "Query Graph Generation for Answering Multi-hop Complex Questions from Knowledge Bases,"
Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020 969 (cit. on pp. 49, 50).

[132] Z.-Y. Chen et al., "UHop: An Unrestricted-Hop Relation Extraction Framework for Knowledge-Based Question Answering," *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)*, 2019 345 (cit. on p. 49).

[133] J. Bao et al., "Constraint-based question answering with knowledge graph,"
Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, 2016 2503 (cit. on pp. 50–52).

[134] H. Sun et al., "PullNet: Open Domain Question Answering with Iterative Retrieval on Knowledge Bases and Text," *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019 2380 (cit. on p. 50).

[135] H. Sun et al.,
"Open Domain Question Answering Using Early Fusion of Knowledge Bases and Text,"
Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018 4231 (cit. on p. 50).

[136] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, arXiv preprint arXiv:1609.02907 (2016) (cit. on p. 50).

[137] M. Schlichtkrull et al., "Modeling relational data with graph convolutional networks," *European semantic web conference*, Springer, 2018 593 (cit. on p. 50).

[138] P. Veličković et al., *Graph attention networks*, arXiv preprint arXiv:1710.10903 (2017) (cit. on p. 50).

[139] R. Das et al., *Case-based Reasoning for Natural Language Queries over Knowledge Bases*, arXiv preprint arXiv:2104.08762 (2021) (cit. on p. 50).

[140] M. Zaheer et al., *Big bird: Transformers for longer sequences*,
arXiv preprint arXiv:2007.14062 (2020) (cit. on p. 50).

[141] A. Bordes et al., "Translating embeddings for modeling multi-relational data," *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2*, 2013 2787 (cit. on pp. 50, 146).

[142] A. Saxena et al., "Improving multi-hop question answering over knowledge graphs using knowledge base embeddings," *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020 4498 (cit. on p. 50).

[143] T. Trouillon et al., *Knowledge graph completion via complex tensor factorization*, arXiv preprint arXiv:1702.06879 (2017) (cit. on p. 50).

[144] W. Cui et al., *KBQA: Learning Question Answering over QA Corpora and Knowledge Bases*, Proc. VLDB Endow. **10** (2017) 565, ISSN: 2150-8097, URL: `https://doi.org/10.14778/3055540.3055549` (cit. on p. 51).

[145] A. Fader et al., "Open question answering over curated and extracted knowledge bases," *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2014 1156 (cit. on p. 51).

[146] S. Reddy et al., "Universal Semantic Parsing," *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2017 89, URL: `http://aclweb.org/anthology/D17-1009` (cit. on p. 51).

[147] K. Xu et al., "Answering natural language questions via phrasal semantic parsing," *Natural Language Processing and Chinese Computing*, Springer, 2014 333 (cit. on pp. 51, 88).

[148] S. He et al., "CASIA@ V2: A MLN-based Question Answering System over Linked Data.," *CLEF (Working Notes)*, 2014 1249 (cit. on p. 51).

[149] C. Unger et al., "Template-based question answering over RDF data," *Proceedings of the 21st international conference on World Wide Web*, ACM, 2012 639 (cit. on p. 51).

[150] C. Unger and P. Cimiano, "Pythia: Compositional meaning construction for ontology-based question answering on the semantic web," *International Conference on Application of Natural Language to Information Systems*, Springer, 2011 153 (cit. on p. 51).

[151] E. Kacupaj et al., *Conversational Question Answering over Knowledge Graphs with Transformer and Graph Attention Networks*, arXiv preprint arXiv:2104.01569 (2021) (cit. on p. 51).

[152] A. Saha et al., "Complex sequential question answering: Towards learning to converse over linked question answer pairs with a knowledge graph," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 1, 2018 (cit. on p. 51).

[153] C. Unger et al., "Question Answering over Linked Data (QALD-5)," *Working Notes of CLEF 2015 - Conference and Labs of the Evaluation forum, Toulouse, France, September 8-11, 2015.* 2015, URL: `http://ceur-ws.org/Vol-1391/173-CR.pdf` (cit. on p. 51).

[154] K. Höffner et al., *Survey on challenges of question answering in the semantic web*, Semantic Web **8** (2017) 895 (cit. on p. 51).

[155] Y. Su et al., "On Generating Characteristic-Rich Question sets for QA Evaluation," *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016 562 (cit. on pp. 51, 52).

[156] M. Dubey et al., "LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia," *International Semantic Web Conference*, Springer, 2019 (cit. on pp. 51, 52).

[157] R. Usbeck et al., "9th Challenge on Question Answering over Linked Data (QALD-9) (invited paper)," *Joint proceedings of the 4th Workshop on Semantic Deep Learning (SemDeep-4) and NLIWoD4: Natural Language Interfaces for the Web of Data (NLIWOD-4) and 9th Question Answering over Linked Data challenge (QALD-9) co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, California, United States of America, October 8th - 9th, 2018*, ed. by K. Choi et al., vol. 2241, CEUR Workshop Proceedings, CEUR-WS.org, 2018 58, URL: `http://ceur-ws.org/Vol-2241/paper-06.pdf` (cit. on p. 52).

[158] Y. Sun et al., "Semantic Parsing with Syntax-and Table-Aware SQL Generation," *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2018 361 (cit. on pp. 53, 128, 131).

[159] D. Lukovnikov et al., "Neural network-based question answering over knowledge graphs on word and character level," *Proceedings of the 26th international conference on World Wide Web*, International World Wide Web Conferences Steering Committee, 2017 1211 (cit. on pp. 53, 79, 81, 86).

[160] G. Maheshwari et al., "Learning to Rank Query Graphs for Complex Question Answering over Knowledge Graphs," *International Semantic Web Conference*, Springer, 2019 (cit. on pp. 53, 79).

[161] M. Ranzato et al., *Sequence level training with recurrent neural networks*, arXiv preprint arXiv:1511.06732 (2015) (cit. on p. 54).

[162] A. Vlachos and S. Clark, *A new corpus and imitation learning framework for context-dependent semantic parsing*, Transactions of the Association for Computational Linguistics **2** (2014) 547 (cit. on pp. 54, 116).

[163] S. Ross et al., "A reduction of imitation learning and structured prediction to no-regret online learning," *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2011 627 (cit. on pp. 54, 116).

[164] T. Yu et al., "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task," *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018 3911 (cit. on p. 55).

[165] N. Kim and T. Linzen,
"COGS: A Compositional Generalization Challenge Based on Semantic Interpretation," *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020 9087 (cit. on pp. 56, 133, 145, 146).

[166] C. Finegan-Dollak et al., "Improving Text-to-SQL Evaluation Methodology," *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018 351 (cit. on p. 56).

[167] P. Shaw et al., *Compositional Generalization and Natural Language Variation: Can a Semantic Parsing Approach Handle Both?* arXiv preprint arXiv:2010.12725 (2020) (cit. on pp. 56, 57).

[168] J. Andreas, "Good-Enough Compositional Data Augmentation," *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020 7556 (cit. on pp. 57, 133, 145).

[169] B. M. Lake, *Compositional generalization through meta sequence-to-sequence learning*, Advances in Neural Information Processing Systems **32** (2019) 9791 (cit. on p. 57).

[170] M. I. Nye et al., *Learning compositional rules via neural program synthesis*, arXiv preprint arXiv:2003.05562 (2020) (cit. on p. 57).

[171] I. Oren et al., "Improving Compositional Generalization in Semantic Parsing," *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, 2020 2482 (cit. on p. 57).

[172] H. Zheng and M. Lapata, *Compositional generalization via semantic tagging*, arXiv preprint arXiv:2010.11818 (2020) (cit. on p. 57).

[173] D. Furrer et al.,
Compositional generalization in semantic parsing: Pre-training vs. specialized architectures, arXiv preprint arXiv:2007.08970 (2020) (cit. on pp. 57, 133, 142).

[174] E. Akyürek et al.,
Learning to Recombine and Resample Data for Compositional Generalization, arXiv preprint arXiv:2010.03706 (2020) (cit. on p. 57).

[175] Y. Li et al., *Compositional generalization for primitive substitutions*, arXiv preprint arXiv:1910.02612 (2019) (cit. on pp. 57, 58, 133).

[176] J. Russin et al., "Compositional generalization by factorizing alignment and translation," *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, 2020 313 (cit. on pp. 57, 58, 133).

[177] Q. Liu et al., *Compositional Generalization by Learning Analytical Expressions*, Advances in Neural Information Processing Systems **33** (2020) (cit. on pp. 57, 58, 133).

[178] J. Herzig et al., *Unlocking Compositional Generalization in Pre-trained Models Using Intermediate Representations*, arXiv preprint arXiv:2104.07478 (2021) (cit. on pp. 57, 59, 133).

[179] Y. Guo et al.,
Revisiting Iterative Back-Translation from the Perspective of Compositional Generalization, arXiv preprint arXiv:2012.04276 (2020) (cit. on pp. 57, 133, 145).

[180] J. Cocke, *Programming languages and their compilers: Preliminary notes*, New York University, 1969 (cit. on p. 58).

[181] T. Kasami, *An efficient recognition and syntax-analysis algorithm for context-free languages*, Coordinated Science Laboratory Report no. R-257 (1966) (cit. on p. 58).

[182] D. H. Younger, *Recognition and parsing of context-free languages in time n3*, Information and control **10** (1967) 189 (cit. on p. 58).

[183] A. Both et al., "Qanary - A Methodology for Vocabulary-Driven Open Question Answering Systems.," *ESWC*, ed. by H. Sack et al., vol. 9678, Lecture Notes in Computer Science, Springer, 2016 625, ISBN: 978-3-319-34128-6, URL: `http://dblp.uni-trier.de/db/conf/esws/eswc2016.html#BothDSSC016` (cit. on p. 62).

[184] D. Golub and X. He, *Character-Level Question Answering with Attention.*, EMNLP (2016) (cit. on pp. 63, 73, 74, 76).

[185] C. Unger et al., "A lemon lexicon for DBpedia.," *NLP-DBPEDIA@ISWC*, ed. by S. Hellmann et al., vol. 1064, CEUR Workshop Proceedings, CEUR-WS.org, 2013, URL: `http://dblp.uni-trier.de/db/conf/semweb/nlp2013.html#UngerMWWC13` (cit. on p. 64).

[186] M. Sahlgren, *The distributional hypothesis*, Italian Journal of Linguistics **20** (2008) 33 (cit. on p. 65).

[187] W. Ling et al., *Finding function in form: Compositional character models for open vocabulary word representation*, arXiv preprint arXiv:1508.02096 (2015) (cit. on pp. 67, 123).

[188] Y. Sun et al., "Modeling mention, context and entity with neural networks for entity disambiguation," *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2015 1333 (cit. on p. 67).

[189] J. Duchi et al., *Adaptive subgradient methods for online learning and stochastic optimization*, Journal of Machine Learning Research **12** (2011) 2121 (cit. on p. 71).

[190] J. Howard and S. Ruder, "Universal Language Model Fine-tuning for Text Classification," *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018 328 (cit. on pp. 78, 99).

[191] X. Liu et al., *Multi-Task Deep Neural Networks for Natural Language Understanding*, arXiv preprint arXiv:1901.11504 (2019) (cit. on p. 78).

[192] F. Ture and O. Jojic, "No Need to Pay Attention: Simple Recurrent Neural Networks Work!" *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017 2866 (cit. on p. 85).

[193] A. Parikh et al., "A Decomposable Attention Model for Natural Language Inference," *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2016 2249, URL: `http://www.aclweb.org/anthology/D16-1244` (cit. on pp. 93, 95–97).

[194] S. R. Bowman et al., “A large annotated corpus for learning natural language inference,” *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Association for Computational Linguistics, 2015 (cit. on p. 95).

[195] Y. Kim, *Convolutional neural networks for sentence classification*, arXiv preprint arXiv:1408.5882 (2014) (cit. on pp. 95, 97).

[196] I. Loshchilov and F. Hutter, *Sgdr: Stochastic gradient descent with warm restarts*, arXiv preprint arXiv:1608.03983 (2016) (cit. on p. 99).

[197] Anonymous, “A Structural Transformer with Relative Positions in Trees for Code-to-Sequence Tasks,” 2020 (cit. on p. 108).

[198] V. L. Shiv and C. Quirk, “Novel positional encodings to enable tree-based transformers.,” *NeurIPS*, 2019 12058 (cit. on p. 108).

[199] K. S. Tai et al., “Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks,” *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2015 1556 (cit. on p. 108).

[200] Y. Wang et al., “Building a Semantic Parser Overnight,” *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Association for Computational Linguistics, 2015 1332, URL: `https://aclanthology.org/P15-1129` (cit. on p. 112).

[201] S. Xu et al., “AutoQA: From Databases to Q&A Semantic Parsers with Only Synthetic Training Data,” *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020 422 (cit. on pp. 113–115).

[202] T. Wolf et al., “Transformers: State-of-the-Art Natural Language Processing,” *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Association for Computational Linguistics, 2020 38, URL: `https://www.aclweb.org/anthology/2020.emnlp-demos.6` (cit. on p. 114).

[203] T. Yu et al., *TypeSQL: Knowledge-based Type-Aware Neural Text-to-SQL Generation*, arXiv preprint arXiv:1804.09769 (2018) (cit. on pp. 120, 128, 130, 131).

[204] P.-S. Huang et al., *Natural Language to Structured Query Generation via Meta-Learning*, arXiv preprint arXiv:1803.02400 (2018) (cit. on pp. 120, 128, 130, 131).

[205] T. Haug et al., *Neural multi-step reasoning for question answering on semi-structured tables*, (2018) 611 (cit. on pp. 120, 127).

[206] C. Wang et al., *Pointing Out SQL Queries From Text*, (2018) (cit. on pp. 120, 128, 130, 131).

[207] C. Wang et al., *Robust text-to-sql generation with execution-guided decoding*, arXiv preprint arXiv:1807.03100 (2018) (cit. on pp. 120, 128, 131).

[208] M. Iyyer et al., “Search-based neural structured learning for sequential question answering,” *Proceedings of ACL 2017*, vol. 1, 2017 1821 (cit. on pp. 120, 127).

[209] P. Pasupat and P. Liang, *Compositional semantic parsing on semi-structured tables*, arXiv preprint arXiv:1508.00305 (2015) (cit. on pp. 120, 127).

[210] D. Bahdanau et al., *Learning to Compute Word Embeddings On the Fly*, arXiv preprint arXiv:1706.00286 (2017) (cit. on pp. 122, 123).

[211] F. Hill et al., *Learning to Understand Phrases by Embedding the Dictionary*, Transactions of the Association of Computational Linguistics (2016) (cit. on p. 123).

[212] A. Neelakantan et al., "Neural programmer: Inducing latent programs with gradient descent," *ICLR 2016*, 2016 (cit. on p. 127).

[213] C. Szegedy et al., "Rethinking the inception architecture for computer vision," *Proceedings of CVPR 2016*, 2016 2818 (cit. on pp. 127, 129).

[214] B. McCann et al., *The natural language decathlon: Multitask learning as question answering*, arXiv preprint arXiv:1806.08730 (2018) (cit. on pp. 127, 128).

[215] J. Gordon et al., "Permutation equivariant models for compositional generalization in language," *International Conference on Learning Representations*, 2019 (cit. on p. 133).

[216] D. Hendrycks and K. Gimpel, *A baseline for detecting misclassified and out-of-distribution examples in neural networks*, ICLR'2017 (2017) (cit. on p. 134).

[217] E. Nalisnick et al., "Do Deep Generative Models Know What They Don't Know?" *International Conference on Learning Representations*, 2018 (cit. on p. 134).

[218] J. Ren et al., *Likelihood ratios for out-of-distribution detection*, arXiv preprint arXiv:1906.02845 (2019) (cit. on pp. 134, 145).

[219] A. Nguyen et al., "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015 (cit. on p. 134).

[220] C. Louizos and M. Welling, "Multiplicative Normalizing Flows for Variational Bayesian Neural Networks," *International Conference on Machine Learning*, 2017 (cit. on p. 134).

[221] W. J. Maddox et al., "A Simple Baseline for Bayesian Uncertainty in Deep Learning," *Advances in Neural Information Processing Systems*, 2019 (cit. on p. 134).

[222] A. Malinin and M. Gales, "Predictive Uncertainty Estimation via Prior Networks," *Advances in Neural Information Processing Systems*, 2018 (cit. on p. 134).

[223] B. Lakshminarayanan et al., "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles," *Advances in Neural Information Processing Systems*, 2017 (cit. on p. 135).

[224] M. Fomicheva et al., *Unsupervised quality estimation for neural machine translation*, Transactions of the Association for Computational Linguistics **8** (2020) 539 (cit. on p. 138).

[225] A. Malinin and M. Gales, *Uncertainty Estimation in Autoregressive Structured Prediction*, ICLR'21 (2021) (cit. on p. 138).

[226] J. Herzig et al., “TaPas: Weakly Supervised Table Parsing via Pre-training,” *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020 4320 (cit. on p. 146).

[227] P. Yin et al., “TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data,” *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020 8413 (cit. on p. 146).

[228] P. Ke et al., *JointGT: Graph-Text Joint Representation Learning for Text Generation from Knowledge Graphs*, arXiv preprint arXiv:2106.10502 (2021) (cit. on p. 146).

[229] H. Ren et al., “LEGO: Latent Execution-Guided Reasoning for Multi-Hop Question Answering on Knowledge Graphs,” *International Conference on Machine Learning*, PMLR, 2021 8959 (cit. on pp. 146, 147).

[230] H. Ren* et al., “Query2box: Reasoning over Knowledge Graphs in Vector Space Using Box Embeddings,” *International Conference on Learning Representations*, 2020, URL: https://openreview.net/forum?id=BJgr4kSFDS (cit. on p. 146).

[231] B. Bogin et al., “Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing,” *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019 4560 (cit. on p. 147).

[232] M. Yasunaga et al., “QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering,” *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021 535 (cit. on p. 147).

[233] H. Sack et al., eds., *The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Proceedings*, vol. 9678, Lecture Notes in Computer Science, Springer, 2016, ISBN: 978-3-319-34128-6, URL: http://dx.doi.org/10.1007/978-3-319-34129-3.

APPENDIX A

List of Publications

Below, * indicates shared first authorship. See introduction of the thesis for more details. Note that none of the shared first co-authors were PhD students with any of my supervisors.

1. **Denis Lukovnikov**, Asja Fischer, Jens Lehmann, Sören Auer. *Neural network-based question answering over knowledge graphs on word and character level.* In Proceedings of the 26th International Conference on World Wide Web (WWW 2017). 2017. DOI: 10.1145/3038912.3052675
2. **Denis Lukovnikov**, Nilesh Chakraborty, Jens Lehmann and Asja Fischer. *Translating Natural Language to SQL using Pointer-Generator Networks and How Decoding Order Matters.* In the AAAI 2019 Reasoning for Complex Question Answering Workshop. 2019.
3. **Denis Lukovnikov**, Asja Fischer and Jens Lehmann. *Pretrained Transformers for Simple Question Answering over Knowledge Graphs.* In Proceedings of the 18th International Semantic Web Conference (ISWC 2019). 2019. DOI: 10.1007/978-3-030-30793-6_27
4. Gaurav Maheshwari*, Priyansh Trivedi*, **Denis Lukovnikov***, Nilesh Chakraborty*, Asja Fischer and Jens Lehmann. *Learning to Rank Query Graphs for Complex Question Answering over Knowledge Graphs.* In Proceedings of the 18th International Semantic Web Conference (ISWC 2019). 2019. DOI: 10.1007/978-3-030-30793-6_28
5. Nilesh Chakraborty*, **Denis Lukovnikov***, Gaurav Maheshwari*, Priyansh Trivedi*, Jens Lehmann and Asja Fischer. *Introduction to neural network-based question answering over knowledge graphs.* Published in the journal Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, Volume 11. 2021. Journal article. DOI: 10.1002/widm.1389
6. **Denis Lukovnikov**, Asja Fischer. *Insertion-based tree decoding.* In the Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics: Findings (ACL Findings 2021). 2021. This work was also presented in the 5th Workshop on Structured Prediction for NLP at ACL 2021. DOI: 10.18653/v1/2021.findings-acl.283
7. **Denis Lukovnikov**, Sina Däubener, Asja Fischer. *Detecting Compositionally Out-of-Distribution Examples in Semantic Parsing.* In the Proceedings of the Conference on Empirical Methods in Natural Language Processing: Findings (EMNLP Findings 2021). DOI: 10.18653/v1/2021.findings-emnlp.54

APPENDIX **B**

An example of insertion-based decoding for a real example.

Consider the example in Listing 6.1. This example can be decoded in 5 steps using tree-based insertion actions as elaborated in the following. Note that multiple insertion action sequences would work with the same number of steps.

Listing B.1: Example query

```
1.    ( filter )
```

Listing B.2: Example query

```
2. ( call_SW_listValue
      ( filter
          ( string )))
```

Listing B.3: Example query

```
3. ( call_SW_listValue
      ( filter
          ( call_SW_singleton )
        ( condition
          ( string )
          ( string
            (=  ))
          ( date ))))
```

Listing B.4: Example query

```
4. ( call_SW_listValue
      ( filter
        ( call_SW_getProperty
          ( call_SW_singleton
            ( en . article  ))
        ( string ))
    ( condition
        ( string
            ( publication_date  ))
        ( string
            (=  ))
        ( date
            (−1  )))))
```

5. Same as original tree in Listing 6.1.

List of Figures

2.1 Gated Recurrent Unit (GRU) 24

4.1 Visualization of the whole model. The question is encoded using the $\texttt{ENC}_\texttt{Q}$ network, producing a question representation vector. Entity and predicate representations are produced by $\texttt{ENC}_\texttt{E}$ resp. $\texttt{ENC}_\texttt{P}$. Question-entity and question-predicate pairs are scored using a cosine similarity between their representing vectors. 64

4.2 Question encoding network $\texttt{ENC}_\texttt{Q}$. Each word is represented by a vector using $\texttt{REP}_\texttt{W}$. The sequence of word vectors is encoded using a GRU. 65

4.3 Word representation network $\texttt{REP}_\texttt{W}$ with example. The word is considered as a sequence of character and fed to $\texttt{ENC}_\texttt{W}$, where each character is embedded and the sequence of character vectors is encoded using a GRU to produce a character-level encoding of that word. This is concatenated to the word embedding (we use GloVe) to produce the complete word representation. 66

4.4 Entity encoder with example. The entity label is encoded on character level ($\texttt{ENC}_\texttt{SL}$) and the subject type label is encoded on word level ($\texttt{ENC}_\texttt{ST}$). The two are concatenated to produce the subject vector. 68

4.5 Predicate encoder network $\texttt{ENC}_\texttt{R}$ with example. The predicate URI is split into words and encoded on word level using GloVe embedding. 68

5.1 Average attention distribution for the example "*What songs have Nobuo Uematsu produced?*", (a) before training on our tasks (vanilla pretrained BERT) , and (b) after training on our tasks (finetuned BERT). The numbers are scaled to values between 0 and 100, and are computed by averaging the normalized attention scores over all heads in all layers, and multiplying the average by 100. We set the scores for `[CLS]` and `[SEP]` tokens to zero in the plots since they always receive a much higher average attention weight than the actual words from the sentence and thus would dominate the plot. 86

5.2 Average normalized attention scores for the `[CLS]` token for several examples. <`Vanilla`> is pretrained BERT before finetuning. <`Finetuned`> is BERT finetuned on our tasks. The numbers are scaled to values between 0 and 100, and are computed by averaging of the normalized attention scores over all heads in all layers, and multiplying the average by 100. 87

5.3 A question and its corresponding query graph (a), and core chain (b). 89

5.4 The slot matching model uses parameterized attention vectors to create j representations of the question, and compares each of them correspondingly with the j hops in a core chain. Here t and t' represent the number of words in each hop of the core chain, and $c_{t,1}$ is the t^{th} word in the first hop. 93
5.5 Illustration of the transformer model in the slot matching configuration. 94
5.6 Visualized attention weights (darker color corresponds to larger attention weights) of the slot matching question encoder for the question "*What is the birth place of the astronaut whose mission was the vostok programme?*" The two rows represent different slot attention scores. One can see that first puts a higher weight on *mission* while the second (beside others) on *birth place*. 97
5.7 Here, (a) shows the decrease in accuracy with increasing number of candidates for all the rankings models in Section 5.8.1; (b) is a confusion matrix representing the number of hops in true and predicted core chains for the test data of LC-QuAD for the slot matching model. (c) the relation of model accuracy w.r.t. question length. And (d) a histogram depicting the distribution of questions in LC-QuAD's test split w.r.t. their question lengths. Here, the proportion of questions with two entity mentions are depicted with red color. 98

6.1 Examples of insertions at different insertion slots. 106
6.2 An example of a tree. 108
6.3 A partial tree of the tree in Fig. 6.2. 110
6.4 The tree from Fig. 6.2 and the nodes and edges covered by the partial tree from Fig. 6.3. Nodes with the same labels in Figures 6.2 and 6.3 are aligned. 110

7.1 Example of a query tree. The blue arrows indicate unordered children. 121
7.2 Two valid linearizations of the example query tree in Figure 7.1. 121

List of Tables

3.1 The table lists the most commonly known KGQA datasets and their characteristics namely: (i) The name of the dataset (ii) the underlying knowledge graph on which the dataset is based (iii) the size of the dataset in terms of number of questions (iv) availability of **F**ormal **Q**uery (FQ) (v) presence of **C**omplex **Q**uestions (CQ) i.e. simple or complex. 51

4.1 Candidate generation recall for subjects over the test set. Recall is defined as the percentage of test questions for which the expected entity is retrieved during candidate generation. 73
4.2 Test accuracy for different candidate generation settings. 73
4.3 Test accuracy of random choice model, for 5, 10 resp. 400 subject entity candidates. 73
4.4 Comparison with state-of-the-art systems. When marked with an asterisk (*), accuracy in the FB5M test setting is given, otherwise FB2M test setting results are shown. . . 74
4.5 Counts of sources of errors that our model makes on 250 test examples. The numbers have been obtained after prediction with max 400 candidates per n-gram. 75

5.1 Component performance evaluation results, trained on all available training data, measured on validation set. (a) Entity span prediction performance, measured by span accuracy, average span F1 and dataset-wide F1 (F1*), all on word level. (b) Relation prediction performance, measured by accuracy (R@1). 83
5.2 Entity recall on validation set. 83
5.3 Entity span detection accuracies (top half) and relation prediction accuracies (bottom half) as a function of fraction of training data retained. Evaluated on the entire validation set. (*) indicates a cosine learning rate schedule with restarts — in extremely low data scenarios for relation classification, this seems to yield better results than the cosine learning rate schedule without restarts that is used everywhere else. 84
5.4 Component results on test set. 85
5.5 Final accuracy for the full prediction task on the test set of SimpleQuestions. [1] . . . 86
5.6 Performance on LC-Quad and QALD-7. The reported metrics are core chain accuracy (CCA), mean reciprocal rank (MRR) of the core chain rankings, as well as precision (P), recall (R), and the F1 of the execution results of the whole system. 97
5.7 CCA for the fine-tuning experiment where we pre-train on LC-QuAD and fine-tune over QALD-7. The initial learning rate is 10^{-3} for all configurations. 99
5.8 CCA for slot matching model, as proposed in Sec 5.7.2 initialized with the weights of BERT-Small, compared with regular transformers initialized with the same weights. . 100

6.1 Statistics for the Overnight dataset after simplification. 112
6.2 Results on the test set of the different domains of the Overnight dataset. Top part: denotation accuracy. Middle part: logical form accuracy on original trees. Bottom part: logical form accuracy on simplified trees. In the last column, we report the average accuracy over the domains as well as the average (over domains) of standard deviations of accuracies for every domain over different seeds. 115
6.3 Logical form accuracy on simplified logical forms and number of decoding steps used on the test set of the different domains of the Overnight dataset. “Bin” stands for binary supervision, “Uni.” for uniform. 115
6.4 Results of an ablation study on the Publications domain. 116

7.1 Evaluation results for our approach (middle section) and comparison with previously reported results (top part) and concurrent work or EG-based systems (bottom part). Entries marked by * are trained and evaluated using a slightly different version of the WikiSQL dataset. Some values in the table, indicated by “–”, could not be filled because the authors did not report the metric or the metric was not applicable. . . . 128
7.2 Performance of different variations of our approach. 129
7.3 Results for different target tree linearizations. 129
7.4 Error Analysis: Acc_{QM} of different query parts on the development set for TF and oracle-trained *shared softmax* models. 130

8.1 Logical form accuracy of the considered models on the original (OOD) test sets of the used datasets. 135
8.2 OOD Detection performance on SCAN’s splits. MCD is the average over the three MCD splits. TM is the transformer. GRU indicates the GRU-based seq2seq model with attention. HE is the heterogeneous ensemble. If “MC” is “No”, MC dropout is not used, otherwise the value of “MC” specifies the number of samples (K in §8.1.1). 136
8.3 OOD Detection performance on CFQ’s MCD splits, averaged over the three provided MCD splits. The table is structured similarly to Table 8.2. 136
8.4 Number of examples per dataset and split. 138