THE UNIVERSITY
of EDINBURGH

Learning Natural Language Interfaces with Neural Models

Li Dong

Doctor of Philosophy
Institute for Language, Cognition and Computation
School of Informatics
University of Edinburgh
2018

Abstract

Language is the primary and most natural means of communication for humans. The learning curve of interacting with various devices and services (e.g., digital assistants, and smart appliances) would be greatly reduced if we could talk to machines using human language. However, in most cases computers can only interpret and execute formal languages. In this thesis, we focus on using neural models to build natural language interfaces which learn to map naturally worded expressions onto machine-interpretable representations. The task is challenging due to (1) structural mismatches between natural language and formal language, (2) the well-formedness of output representations, (3) lack of uncertainty information and interpretability, and (4) the model coverage for language variations. In this thesis, we develop several flexible neural architectures to address these challenges.

We propose a model based on attention-enhanced encoder-decoder neural networks for natural language interfaces. Beyond sequence modeling, we propose a tree decoder to utilize the compositional nature and well-formedness of meaning representations, which recursively generates hierarchical structures in a top-down manner. To model meaning at different levels of granularity, we present a structure-aware neural architecture which decodes semantic representations following a coarse-to-fine procedure.

The proposed neural models remain difficult to interpret, acting in most cases as a black box. We explore ways to estimate and interpret the model's confidence in its predictions, which we argue can provide users with immediate and meaningful feedback regarding uncertain outputs. We estimate confidence scores that indicate whether model predictions are likely to be correct. Moreover, we identify which parts of the input contribute to uncertain predictions allowing users to interpret their model.

Model coverage is one of the major reasons resulting in uncertainty of natural language interfaces. Therefore, we develop a general framework to handle the many different ways natural language expresses the same information need. We leverage external resources to generate felicitous paraphrases for the input, and then feed them to a neural paraphrase scoring model which assigns higher weights to linguistic expressions most likely to yield correct answers. The model components are trained end-to-end using supervision signals provided by the target task.

Experimental results show that the proposed neural models can be easily ported across tasks. Moreover, the robustness of natural language interfaces can be enhanced by considering the output well-formedness, confidence modeling, and improving model coverage.

Acknowledgements

The Ph.D. journey in Edinburgh has been a meaningful and unforgettable experience. First of all, I would like to thank Mirella Lapata for her generous guidance and constant encouragement throughout the years. It is a great pleasure to have meetings with Mirella, because she can always give me some constructive suggestions. Her energy and enthusiasm are contagious, which greatly motivates me in terms of work and study. There are many different types of good supervisors, and Mirella is the best of the best a student could have.

I am also thankful to Shay Cohen and my second supervisor Adam Lopez for their feedback on my first-year review. Their advice helped me refine my research plan.

I would like to thank my examiners, Mark Steedman and Luke Zettlemoyer, for their time to read through the thesis. The detailed comments not only improved this thesis but also will be valuable for my future work.

Special thanks to my ILCC collaborators, Jianpeng Cheng, Jonathan Mallinson, Ratish Puduppully, and Siva Reddy, for their useful input. I learned a great deal from discussions with them. I am grateful to Chris Quirk for mentoring me during the summer internship at MSR. His insightful views inspired my research on confidence modeling. I also would like to acknowledge Adeptmind for the PhD fellowship.

I have had the fortune to conduct my research in the Edinburgh NLP group. Many thanks to Shay Cohen, Sharon Goldwater, Kenneth Heafield, Frank Keller, Alex Lascarides, Adam Lopez, Rico Sennrich, Mark Steedman, Henry Thompson, Ivan Titov, and Bonnie Webber for their comments on my work and presentations in group meetings. I will also miss all the NLP group members: Annie, Bowen, Carina, Chunchuan, Clara, David, Desmond, Diego, Esma, Federico, Javad, Jiangming, Jianpeng, Joana, John, Jonathan, Laura, Lea, Marco, Michael, Naomi, Nathan, Philip, Philippa, Ratish, Rui, Shashi, Sorcha, Spandana, Stefanos, Xingxing, Yang, Yumo, etc.

I would like to thank my office mates Carol, Javad, Jiangming, Jianpeng, John, Nicolas, Pablo, Rui, and Xingxing. We created a pleasant and cheerful working environment together.

Finally, I am grateful to my parents and my wife Shu-Ting for their unconditional love and support. I dedicate the thesis to my loving family.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Li Dong*)

Table of Contents

Chapter 1

Introduction

Language is the primary and most natural means of communication for humans. Besides ease of use and sufficient expressive power, recent development in speech technology makes natural language a very appealing user interface for many applications, such as digital personal assistants and wearable devices. Language provides a convenient way to interact with different services without requiring users to be domain experts. This advantage is of great value especially when there are quite a lot of operations allowed in an application. For example, a smart home usually contains many appliances with various functions, it would be much more convenient to express desired actions in natural language, rather than struggling to find the buttons and enter the execution plans according to user manuals. Moreover, it is hard for users to remember different machine-interpretable languages, while human language could provide a unified interaction experience across domains. For devices (such as smart speakers) that have restricted keyboard usage, dictating in natural language together with a speech recognition module complements other input methods. For instance, voice control when used in the automotive environment improves the driving experience and reduces cognitive load (i.e., the driver focuses on the driving task per se with having to use their hands or direct their gaze for other functions).

Natural language interfaces aim at allowing users to interact with devices in human language (such as English), rather than relying on special-purpose machine-interpretable language. The learned interfaces are transparent layers between users and computers, which can handle commands expressing various intents. It can greatly reduce the learning curve of interacting with various devices and services (e.g., robots, digital assistants, and smart appliances) if we can manipulate machines using human language. Such kind of human-computer interface has a wide range of applications, such

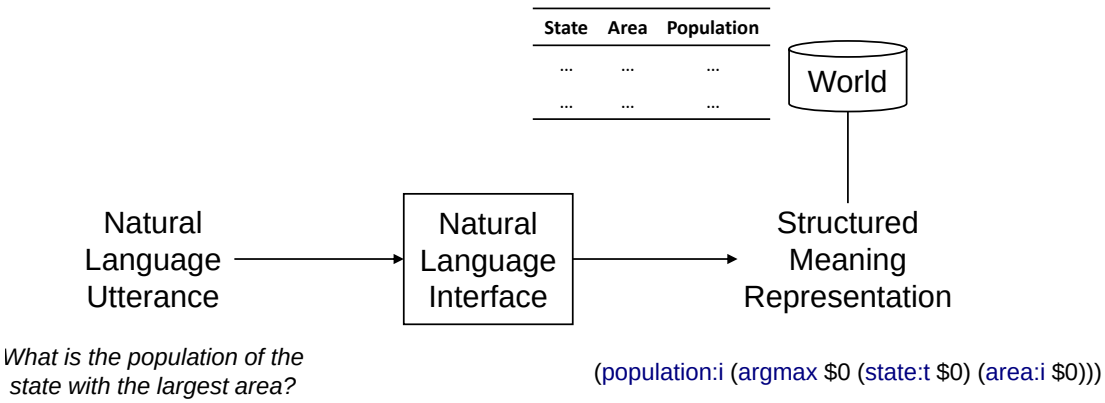


Figure 1.1: The goal of natural language interfaces is allowing users to interact with computers in human language. As shown by the example from the GEO dataset (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005), the model maps the input question to the λ-calculus meaning representation, and then execute it over the database to obtain the answer.

as querying databases, completing tasks, and answering questions. There have been some successful use cases by now: Google answers questions over a knowledge base; Apple Siri, Amazon Alexa, and Microsoft Cortana assist individuals in performing tasks or services on smartphones, computers, and smart speakers; and some cars enable us to adjust climate controls or move seats using natural speech. Although these applications can presently only handle limited commands, they have shown the great potential of natural language interfaces.

Computers can only interpret and execute formal languages that are more machine-friendly but difficult to learn and master for most people. So one of the core challenges of building a natural language interface is *semantic parsing*, i.e., how to map a naturally worded expression onto the machine-interpretable representation of its underlying meaning. Figure 1.1 shows the workflow of a typical natural language interface and an example taken from the GEO dataset (see Chapter 2 for more detail on this dataset). Given a natural language utterance, the system predicts a machine-readable (structured) representation. The prediction is then executed against a read-world environment to perform a task (e.g., query a knowledge base, or instruct a robot). The example question in Figure 1.1 is used to query a database of U.S. geography. The predicted meaning representation is based on lambda calculus. And the logical form can be transformed to a database query in order to obtain the answers. As we can see from this example, a regular user could utilize everyday languages to query a knowledge base without the need to learn a database query language.

What microsoft jobs do not require a bscs? `ans(company(J,'microsoft'),job(J),not((req_deg(J,'bscs'))))`
What is the population of the state with the largest area? `(population:i (argmax $0 (state:t $0) (area:i $0)))`

Table: ‖*Pianist*‖*Conductor*‖*Record Company*‖*Year of Recording*‖*Format*‖
What record company did conductor Mikhail Snitko record for after 1996?
`SELECT Record Company WHERE (Year of Recording > 1996)`
`AND (Conductor = Mikhail Snitko)`

In which office was the patent computer mouse filed? `Law.Us_patent.Patent_office(ENTITY.mouse) ⊓ TYPE.Law.Patent_office`

Turn on heater when temperature drops below 58 degree
`Weather−Current_temperature_drops_below−((Temperature (58)) (Degrees_in (f)))`
`THEN WeMo_Insight_Switch−Turn_on−((Which_switch? ("")))`

if length of bits is lesser than integer 3 or second element of bits is not equal to string 'as', `if len(bits) < 3 or bits[1] != 'as':`

Table 1.1: The examples of natural language descriptions and their meaning representations are taken from (Tang and Mooney, 2001; Zettlemoyer and Collins, 2005; Zhong et al., 2017; Su et al., 2016; Quirk et al., 2015; Oda et al., 2015).

The formal language is usually chosen by considering the convenience for end applications and used models. Table 1.1 shows some examples of natural language expressions and their meaning representations. The first four blocks are question answering examples which retrieve answers from databases. Their semantic representations are based on Prolog (Zelle and Mooney, 1996; Tang and Mooney, 2000), lambda calculus (Zettlemoyer and Collins, 2005; Kwiatkowksi et al., 2010), SQL (Giordani and Moschitti, 2010; Zhong et al., 2017; Iyer et al., 2017), and knowledge graph query (Reddy et al., 2014; Yih et al., 2015; Su et al., 2016), respectively. The fifth example uses a tree-structured program (Quirk et al., 2015; Beltagy and Quirk, 2016) to represent the automation command. The last example is based on the Python programming language (Yin and Neubig, 2017). In thesis we do not assume that a specific formal language is used in order to enable the proposed models to be general and portable across domains and meaning representations.

Recently, models based on neural networks have achieved promising results in the field of natural language processing (Goldberg, 2017). Typically, discrete words are mapped to a continuous vector space and then processed by nonlinear transformations.

The modeling flexibility allows us to design various neural components to construct larger networks, which can be jointly trained with gradient-based optimization algorithms. The end-to-end learned representations also reduce the requirement of domain knowledge and feature engineering. Additionally, neural models are good at handling compositionality, which is critical for both natural language and formal representations. The composition of dense vectors in neural networks alleviates the problems of data sparsity compared with using symbolic n-gram features. Because of these advantages, in thesis we will explore how to use neural networks to build and improve natural language interfaces.

1.1 History of Natural Language Interfaces

Due to the complexity of language, building natural language interfaces has been a long-standing research goal of artificial intelligence. In the early days, systems were mainly built with manually defined lexicons and rules. Along with the development of machine learning, statistical systems became mainstream. We summarize the development of research on natural language interfaces as follows.

Rule-Based Systems One of the earliest natural language interfaces is THE CONVERSATION MACHINE (Green et al., 1959). The system was built to carry out a conversation about the weather. A dictionary was used to match words which were categorized into *time* (e.g., "*July*"), *operator* (e.g., "*not*"), and *ordinary* (e.g., "*rain*"). Each matched word was assigned with an attribute-value pair to store its meaning. For example, for the time word "*July*", the attribute is the type of month and the value the specific month. The assigned functions of operator words were then executed to change the parsed values. And a reply frame was selected according to the results. Another early system was BASEBALL (Green et al., 1961), which answers questions about baseball games. Similarly, a dictionary was used to match the words and constituent phrases of the given question with the database, which built an executable specification list to obtain the answers. The algebra problem solver STUDENT (Bobrow, 1964) transformed natural-language statements into a set of equations. Heuristic dictionary lookup was employed to identify the equation operators and function terms. Another noteworthy system is PICTURE LANGUAGE MACHINE (Kirsch, 1964), which verified whether natural-language statements were correct for a given image. It is one of the earliest systems that translate sentences into a formal language. A sentence was parsed

to syntactic trees with a constituent grammar. And a rule-based formalizer would translate the trees to first-order functional calculus in a top-down manner. The logical form was executed together with the picture to obtain the result.

The linguistically rich systems LUNAR (Woods et al., 1972) and SHRDLU (Winograd, 1972) achieved significant success in the early 1970s. The system LUNAR can answer questions against a database of isotope, chemical, and age analysis of the Apollo 11 samples. The application was designed to help geologists access the data records without learning a database querying language. The system first performs syntax analysis to obtain the sentence's grammatical structure. Then the syntactic fragments are transformed into structured meaning representations using rules. SHRDLU has a syntax-driven semantic parser built with rules. Users can use natural language to manipulate a block world. Although the application is artificial, the system shows the ability to handle context information and interact with users. The language understanding process and action execution are performed jointly, which connects semantics with the real world.

Some other rule-based systems (Hendrix et al., 1978; Damerau, 1981; Warren and Pereira, 1982; Thompson and Thompson, 1983; Templeton and Burger, 1983; Hafner, 1984; Ballard, 1984; Ballard and Stumberger, 1986; Grosz et al., 1987; Alshawi and van Eijck, 1989; Bobrow et al., 1990) also obtained promising performance for different applications and meaning representations. Typically, syntax analysis is performed first to obtain the chunks or syntactic structures for the given input utterance. Next, a set of manual lexicons and translation rules are applied, which transforms the natural language input to a formal representation. Priority values are often predefined for rules to resolve the ambiguity if multiple rules are matched. However, rule-based systems are often limited to a specialized subset of natural language that can be covered by the predefined lexicons and templates. Substantial engineering efforts and domain expertise are required to build such systems and extend them manually to new tasks. The complexity of rules increases exponentially with the complexity of linguistic phenomena being handled.

Statistical Systems Advances in statistical techniques have led to the development of natural language understanding models which can learn from data, in lieu of hand-coding rules. In the statistical paradigm, a training corpus is first collected to provide supervision signals. Once we define the model and the training objective, we can use optimization algorithms to learn the model's unknown parameters. During testing, the

model with learned parameters is employed to predict the results for input examples. These systems typically learn lexicalized mapping rules to construct a candidate set of meaning representations for a given input. Then a scoring component assigns scores to these candidates and uses the best results (i.e., with largest scores) as the final predictions.

Various models have been proposed over the years to learn natural language interfaces from natural language expressions paired with their meaning representations. Examples include the use of inductive logic programming (Zelle and Mooney, 1996; Tang and Mooney, 2000; Thomspon and Mooney, 2003), parsing models (Miller et al., 1996; Ge and Mooney, 2005; Lu et al., 2008; Zhao and Huang, 2015), probabilistic automata (He and Young, 2006), string/tree-to-tree transformation rules (Kate et al., 2005), classifiers based on string kernels (Kate and Mooney, 2006), machine translation (Wong and Mooney, 2006, 2007; Andreas et al., 2013), and combinatory categorial grammar induction techniques (Zettlemoyer and Collins, 2005, 2007; Kwiatkowksi et al., 2010; Kwiatkowski et al., 2011). CHILL (Zelle and Mooney, 1996) is one of the first statistical systems. CHILL provided a natural language interface to a database about US geography. The second block of Table 1.1 shows an example from the created dataset. The model used inductive logic programming to learn definite-clause logic descriptions for a shift-reduce parser expressed in Prolog. The training examples were used to formulate an overly-general parser. Control rules were then learned to characterize whether the parsing operators should be employed in the context of training examples. The final model was the overly-general shift-reduce parser and the learned rules. Another model worth mentioning was developed by Zettlemoyer and Collins (2005), who employed combinatory categorial grammar (CCG; Steedman (2000)) as a syntax-semantics interface. A log-linear model was used to induce probabilistic CCGs and lexicons from training examples.

Other work learns natural language interfaces without relying on logical-from annotations, e.g., from sentences paired with conversational logs (Artzi and Zettlemoyer, 2011), system demonstrations (Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013; Goldwasser and Roth, 2014), question-answer pairs (Clarke et al., 2010; Liang et al., 2013; Yih et al., 2015; Pasupat and Liang, 2015; Berant and Liang, 2015), and distant supervision (Krishnamurthy and Mitchell, 2012; Cai and Yates, 2013; Reddy et al., 2014).

Neural Systems However, most previous systems rely on manually defined features, which greatly affect model performance. It is time-consuming and expensive to develop features which adequately capture the relation between natural language and semantics. Lexical features are often sparse due to the size of the training data, rendering the task of generalizing to unseen examples difficult. The reliance on predefined templates or lexicons also limits these models from scaling to different domains, languages, or meaning representations. Moreover, the choice of intermediate representation make the systems representation-specific. It is usually nontrivial to design features applicable to various semantic representations (e.g., Python code, and SQL query). In addition, errors propagate, e.g., if the system is based on a pipeline or syntactic parsers are used during the parsing process.

More recently, neural sequence-to-sequence models have been applied to semantic parsing with promising results (Dong and Lapata, 2016; Jia and Liang, 2016; Ling et al., 2016), eschewing the need for extensive feature engineering. There are also efforts to develop structured decoders that make use of the syntax of meaning representations. Dong and Lapata (2016) and Alvarez-Melis and Jaakkola (2017) develop models which generate tree structures in a top-down fashion. Xiao et al. (2016) and Krishnamurthy et al. (2017) employ a grammar to constrain the decoding process. Yin and Neubig (2017) design a grammar model for the generation of abstract syntax trees (Aho et al., 2007) in depth-first, left-to-right order. Rabinovich et al. (2017) propose a modular decoder whose submodels are dynamically composed according to the generated tree structure. Grammar-specific models are also developed to utilize the syntax of formal languages (Zhong et al., 2017; Xu et al., 2017; Sun et al., 2018; Yu et al., 2018). Cheng et al. (2017) use a transition system to generate variable-free queries. Chen et al. (2018) design a sequence-to-action model to build graph-structure representations. Both structure constraints and semantic constraints are applied to ensure predictions form connected acyclic graphs and follow the domain-specific schema. Suhr et al. (2018) propose a context-dependent neural semantic parser to handle multi-turn conversations. In order to incorporate interaction history, previous requests and predictions are encoded and used as context for the current utterance. Moreover, several ideas have been explored to enhance the performance of these models such as data augmentation (Kočiský et al., 2016; Jia and Liang, 2016), transfer learning (Fan et al., 2017), active learning (Duong et al., 2018), sharing parameters for multiple meaning representations (Herzig and Berant, 2017), leveraging cross-lingual data (Susanto and Lu, 2017; Duong et al., 2017; Zou and Lu, 2018; Richardson et al., 2018), handling

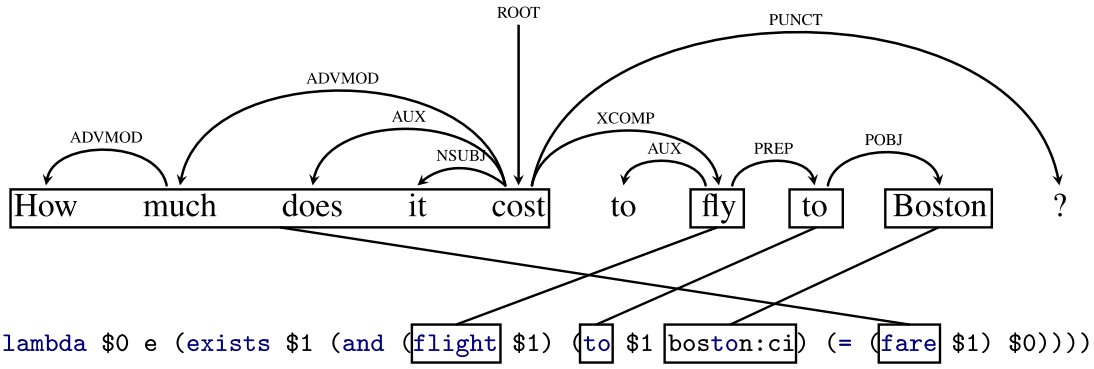


Figure 1.2: Formal meaning representation is structurally different from the natural language string and even its syntactic representation.

out-of-vocabulary words (Ray et al., 2018), utilizing user feedback signals (Iyer et al., 2017; Lawrence and Riezler, 2018), and semi-supervised learning (Yin et al., 2018).

Recently, the weakly-supervised learning paradigm has also been explored to train neural semantic parsers (Neelakantan et al., 2016; Yin et al., 2016; Liang et al., 2017; Iyyer et al., 2017; Guu et al., 2017; Iyer et al., 2017; Zhong et al., 2017; Goldman et al., 2018; Liang et al., 2018a). Apart from utilizing weak supervision as a training signal, various methods have been developed to ease the acquisition of logical-from annotations, such as using paraphrases to populate examples (Wang et al., 2015; Su et al., 2016), automatically generating natural language queries (Serban et al., 2016), and increasing lexical diversity (Ravichander et al., 2017).

1.2 Challenges

Building natural language interfaces is a challenging task which often requires substantial engineering effort. We summarize important challenges that need to be tackled as follows.

Structural Mismatch There is usually a divergence between the grammar of human expression and the syntax of formal language, which introduces challenges to modeling. Apart from capturing the semantics expressed by the input, models need to learn how to transform user intentions into formal language. For instance, in a flight booking system, the input "*How much does it cost to fly to Boston?*" is mapped to `(lambda $0 e (exists $1 (and (flight $1) (to $1 boston:ci) (= (fare $1) $0))))`, which is structurally very different from the natural language string and even its syntactic rep-

resentation (as shown in Figure 1.2). The structural mismatch requires us to handle different types of alignments (e.g., one-to-many, and many-to-many) and reordering during the decoding process.

Output Well-Formedness After obtaining the output meaning representations from natural language interfaces, we usually need to execute them to obtain user intentions as shown in Figure 1.1. Because the downstream executors only accept grammatical programs, it is beneficial to explicitly model the structure of predictions. The structural information of the output should be taken into consideration so that the models can generate well-formed meaning representations. It is still an open problem to add grammatical and semantic constraints into neural models (Xiao et al., 2016; Rabinovich et al., 2017; Yin and Neubig, 2017; Chen et al., 2018).

Uncertainty and Interpretability Natural language interfaces involve interactions with end-task applications, where the actions often need to be executed with high confidence. For example, if smart appliances are uncertain about the user commands, we would like to verify the predicted actions to avoid unwanted behaviors. In addition to the uncertainty caused by models, the vagueness and ambiguity of human language also make confidence estimation necessary. However, most models cannot directly provide uncertainty information, which prevents the deployment of natural language interfaces in some scenarios. It would also be beneficial to obtain fine-grained interpretations of uncertainty for model predictions, so that users can rewrite the input text according to the obtained results.

Model Coverage The word choices of natural language are varied. The same meaning can be expressed in many different ways. Learning lexical variation is nontrivial for natural language interfaces, because we usually only have limited training data and do not have fine-grained annotations of lexicons. Lexical variation makes it difficult to generalize to new examples. For instance, the questions "*who created microsoft*" and "*who started microsoft*" express the same meaning, but the relation is realized by different verbs. Models need to ground different words or phrases that have the same semantics to the same predicate in logical forms. External resources could be helpful to learn such kind of lexical knowledge especially for expressions that are unseen in the training data.

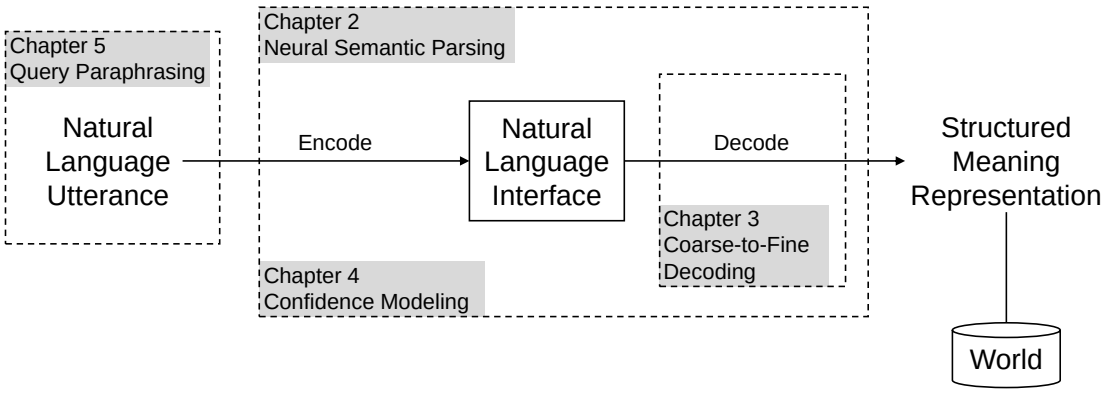


Figure 1.3: Thesis overview.

1.3 Thesis Overview

In this thesis, we aim at developing portable and robust neural models for natural language interfaces, while addressing the challenges outlined in the previous section.

Firstly, our models are designed with the goal of being easily and quickly portable across domains and meaning representations. As shown in Table 1.1, various formal languages are employed for different applications, so the model portability is important for natural language interfaces. Because of structural mismatches between natural language and formal language, previous work often relies on linguistic formalisms to alleviate the learning difficulty. For example, Zettlemoyer and Collins (2005) generate lambda calculus based on CCG parse results, and further improvements (Zettlemoyer and Collins, 2007; Kwiatkowksi et al., 2010; Kwiatkowski et al., 2011) are proposed to handle more linguistic phenomena. However, relying on particular linguistic formalisms often requires substantial engineering effort and domain knowledge, rendering the models domain- or representation-specific.

Our second objective is to enhance the robustness of models that are built for natural language interfaces. Specifically, models should be robust in terms of the well-formedness of predictions. By considering structural information of formal languages, constraints can be added into the decoding process to prune invalid outputs. Moreover, models should be robust in the sense of knowing whether they are uncertain in their predictions, rather than always guessing some results. Confidence modeling helps models to make robust decisions, and provides interpretations for uncertain outputs. Additionally, we would like to make natural language interfaces robust to lexical variation, so that models can handle different input utterances that express the same intention.

We do not assume a syntactic parser is available. Because user inputs are often of spoken and informal, the error propagation of syntactic parsing could harm the final performance. Moreover, if we would like to deploy the models to various languages, we would need a syntactic parser for every language. In this thesis, we regard inputs as sequences of words. The simplification enables us to adapt our models to various domains or languages lacking syntactic parsing resources.

Figure 1.3 shows the overview of the thesis. We propose a neural semantic parsing framework based on encoder-decoder networks, which learns to map natural language inputs onto their meaning representations in an end-to-end fashion. Both encoder and decoder are built on recurrent neural networks. The underlying networks can cope with variable-length examples, and are suitable for handling compositional structures, which is critical for semantic parsing (Liang and Potts, 2015). We also introduce an attention mechanism (Bahdanau et al., 2015; Luong et al., 2015a) allowing the model to learn soft alignments between natural language and formal representations. The models are end-to-end learned from utterances annotated with formal meaning representations, so we can directly optimize the objective rather than using a pipeline system. Moreover, we do not rely on predefined lexicons and annotated alignments between input utterances and output meaning representations. The models acquire such kind of knowledge from training data, and utilize this information to overcome structural mismatches.

Meaning representations are typically structured objects instead of just sequences. In order to guarantee the well-formedness of the output we explicitly model the hierarchical and compositional nature of meaning representations, and thus develop a sequence-to-tree model, and a coarse-to-fine decoding algorithm. In the first solution, the proposed tree decoder defines a placeholder to indicate nonterminal nodes. Tree structures are recursively generated in a top-down, and left-to-right manner. The explicit modeling of hierarchical structures constrains results in the space of well-formed trees. In other words, ill-formed logical forms can be pruned from the candidate set. To model meaning at different levels of granularity, the second solution uses a structure-aware neural architecture to decode semantic representations from coarse to fine. The coarse meaning decoder first generates a rough sketch of the meaning representation, which omits low-level details, such as arguments and variable names. Then, the fine meaning decoder fills in missing details by conditioning on the input utterance and the sketch itself. Particularly, the sketch is encoded into vectors to guide the generation process, where the basic meaning is used as global context to improve fine-grained

decoding.

The proposed neural models remain difficult to interpret, acting in most cases as a black box, not providing any information about uncertainty or what made them arrive at a particular decision. For this challenge, we explore ways to estimate and interpret the model's confidence in its predictions, which we argue can provide users with immediate and meaningful feedback regarding uncertain outputs. We categorize the causes of uncertainty into *model uncertainty*, *data uncertainty*, and *input uncertainty*. We then design various metrics to characterize "what the model does not know", and compute confidence scores which indicate how likely the predicted meaning representations are correct. We further backpropagate uncertainty scores from predictions to input words so that we can know the contribution degree of each word to the uncertainty. These scores allow us to interpret model behavior by identifying which parts of the input contribute to uncertain predictions.

Model coverage is one of the major reasons resulting in uncertainty of natural language interfaces. Therefore, we develop a query paraphrasing framework for the challenge. To handle the many different ways natural language expresses the same information need, we leverage external resources to generate paraphrases for the input utterance. We jointly train a neural paraphrase scoring model that assigns higher weights to those which are more likely to yield correct answers. The entire system is end-to-end trained so that the paraphrase model is task-specific. The plug-and-play functionality of the framework allows us to explore various paraphrase generation methods for natural language interfaces.

The main contributions of this work are:

- A general method based on a neural encoder-decoder architecture for mapping natural language expressions to their logical forms. We also demonstrate how to adapt the neural models to natural language interfaces characteristic of different domains and meaning representations.

- A tree decoder and a coarse-to-fine decoding algorithm to better handle hierarchical structures and the well-formedness of meaning representations.

- A proposal of various ways to estimate and interpret the model's confidence in its outputs, which provides users with feedback and interpretations regarding uncertain predictions.

- A query paraphrasing framework to handle the variation of natural language

input. The paraphrase scoring model and the final task are trained end-to-end, which results in learning paraphrases with a purpose.

1.4 Outline

The remainder of this thesis is organized as follows:

- Chapter 2 presents a neural encoder-decoder framework that maps natural language expressions to their logical forms. We encode input utterances into vector representations, and generate their logical forms by conditioning the output sequences or trees on the encoding vectors. Experimental results on four datasets show that our approach performs competitively without using hand-engineered features and is easy to adapt across domains and meaning representations.

- Chapter 3 introduces a structure-aware neural architecture which decomposes the semantic parsing process into two stages. Given an input utterance, we first generate a rough sketch of its meaning, where low-level information (such as variable names and arguments) is glossed over. Then, we fill in missing details by taking into account the natural language input and the sketch itself. Experimental results on four datasets characteristic of different domains and meaning representations show that our approach consistently improves performance, achieving competitive results despite the use of relatively simple decoders.

- Chapter 4 is concerned with confidence modeling for neural semantic parsers which are built upon sequence-to-sequence models. We outline three major causes of uncertainty and design various metrics to quantify these factors. These metrics are then used to estimate confidence scores that indicate whether model predictions are likely to be correct. Beyond confidence estimation, we identify which parts of the input contribute to uncertain predictions allowing users to interpret their model, and verify or refine its input. Experimental results show that our confidence model significantly outperforms a widely used method that relies on posterior probability, and improves the quality of interpretation compared to simply relying on attention scores.

- Chapter 5 describes a general framework which learns felicitous paraphrases for various question answering tasks. Our method is trained end-to-end using question-answer pairs as a supervision signal. A question and its paraphrases

serve as input to a neural scoring model which assigns higher weights to linguistic expressions most likely to yield correct answers. We evaluate our approach on question answering over Freebase and answer sentence selection. Experimental results on three datasets show that our framework consistently improves performance.

- Chapter 6 concludes the thesis, and discusses directions for future work.

Portions of this thesis have been previously published in Dong and Lapata (2016) (Chapter 2), Dong and Lapata (2018) (Chapter 3), Dong et al. (2018) (Chapter 4), and Dong et al. (2017b) (Chapter 5).

Chapter 2

Neural Semantic Parsing

Semantic parsing is the task of translating text to a formal meaning representation such as logical forms or structured queries, which is one of the core components of natural language interfaces (as shown in Figure 1.1). There has recently been a surge of interest in developing machine learning methods for semantic parsing (see the references in Section 2.1), due in part to the availability of corpora containing utterances annotated with formal meaning representations. Figure 2.1 shows an example of a question (left-hand side) and its annotated logical form (right-hand side), taken from JOBS (Tang and Mooney, 2001), a well-known semantic parsing benchmark. In order to predict the correct logical form for a given utterance, most previous systems rely on predefined templates and manually designed features (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Kwiatkowksi et al., 2010), which often render the parsing model domain- or representation-specific. In this chapter, we aim to use a portable method to bridge the gap between natural language and logical form with minimal domain and linguistic knowledge.

Encoder-decoder architectures based on recurrent neural networks have been successfully applied to a variety of NLP tasks ranging from syntactic parsing (Vinyals et al., 2015a), to machine translation (Kalchbrenner and Blunsom, 2013; Cho et al., 2014; Sutskever et al., 2014), and image description generation (Karpathy and Fei-Fei, 2015; Vinyals et al., 2015b). As shown in Figure 2.1, we adapt the general encoder-decoder paradigm to the semantic parsing task. Our model learns from natural language descriptions paired with meaning representations; it encodes sentences and decodes logical forms using recurrent neural networks with long short-term memory (LSTM; Hochreiter and Schmidhuber (1997)) units. We present two model variants, the first one treats semantic parsing as a vanilla sequence transduction task, whereas

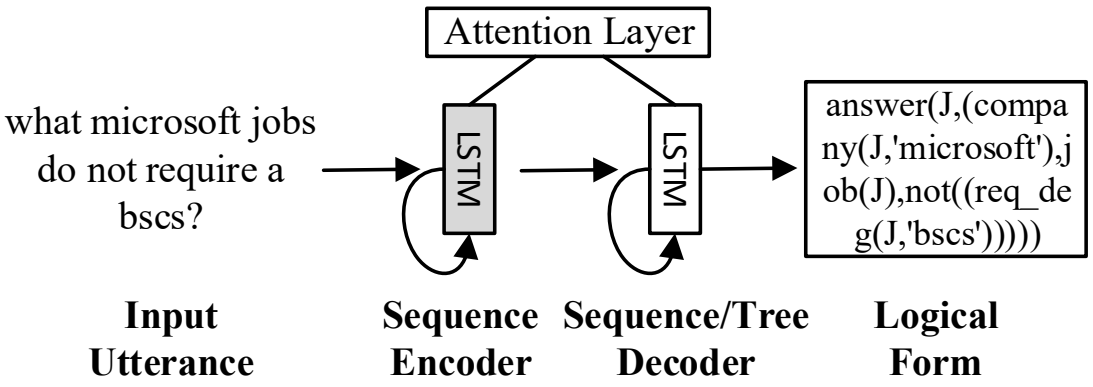


Figure 2.1: Input utterances and their logical forms are encoded and decoded with neural networks. An attention layer is used to learn soft alignments.

our second model is equipped with a hierarchical tree decoder which explicitly captures the compositional structure of logical forms. We also introduce an attention mechanism (Bahdanau et al., 2015; Luong et al., 2015a) allowing the model to learn soft alignments between natural language and logical forms and present an argument identification step to handle rare mentions of entities and numbers.

Evaluation results demonstrate that compared to previous methods our model achieves similar or better performance across datasets and meaning representations, despite using no hand-engineered domain- or representation-specific features.

2.1 Related Work

Our proposed framework synthesizes two strands of research, namely semantic parsing and the neural encoder-decoder architecture. We adopt the general encoder-decoder model based on neural networks which has been recently repurposed for various NLP tasks such as syntactic parsing (Vinyals et al., 2015a), machine translation (Kalchbrenner and Blunsom, 2013; Cho et al., 2014; Sutskever et al., 2014; Bahdanau et al., 2015; Gehring et al., 2017; Vaswani et al., 2017), visual description generation (Karpathy and Fei-Fei, 2015; Vinyals et al., 2015b; Donahue et al., 2015; Venugopalan et al., 2015a,b; Xu et al., 2015), question answering (Hermann et al., 2015), text generation (Wiseman et al., 2017; Dong et al., 2017a; Beck et al., 2018), and summarization (Rush et al., 2015; Chopra et al., 2016; See et al., 2017). The framework typically contains an encoder and a decoder. The input is first encoded into vector representations, which can be viewed as feature extraction. By conditioning on these encoding results, the decoder predicts the final output. Under this flexible framework, we can design different

network architectures for various kinds of data (e.g., sequences, tables, and graphs). The whole model is end-to-end learned from training examples, and thus directly optimizes a given objective. We also describe an attention mechanism (Bahdanau et al., 2015; Xu et al., 2015) to explicitly model alignments between encoding and decoding states.

Compared with the previous models (Sutskever et al., 2014; Karpathy and Fei-Fei, 2015) that are built upon the encoder-decoder framework, the main difference is that decoding results of semantic parsing are structured. Directly using sequence decoders sometimes produces invalid results in terms of the underlying grammar and structure. The nature of semantic parsing task motivates the use of a constrained decoder in order to guarantee the well-formedness of predicted meaning representations. For example, we can leverage a tree decoder to generate well-formed trees rather than using brackets to linearize hierarchical structures. Moreover, structured decoders can model long-term dependencies in meaning representations, and hard constraints can be advantages for model learning because of the reduction of the search space.

2.2 Problem Formulation

Our aim is to learn a model which maps natural language input $q = q_1 \cdots q_{|q|}$ to a logical form representation of its meaning $a = a_1 \cdots a_{|a|}$. The conditional probability $p(a|q)$ is decomposed as:

$$p(a|q) = \prod_{t=1}^{|a|} p(a_t|a_{<t}, q) \tag{2.1}$$

where $a_{<t} = a_1 \cdots a_{t-1}$.

Our method consists of an **encoder** which encodes natural language input q into a vector representation and a **decoder** which learns to generate $a_1, \cdots, a_{|a|}$ conditioned on the encoding vector. In the following we describe two models varying in the way in which $p(a|q)$ is computed.

2.2.1 Sequence-to-Sequence Model

This model regards both input q and output a as sequences. As shown in Figure 2.2, the encoder and decoder are two different L-layer recurrent neural networks with long short-term memory (LSTM) units which recursively process tokens one by one. The first $|q|$ time steps belong to the encoder, while the following $|a|$ time steps belong to

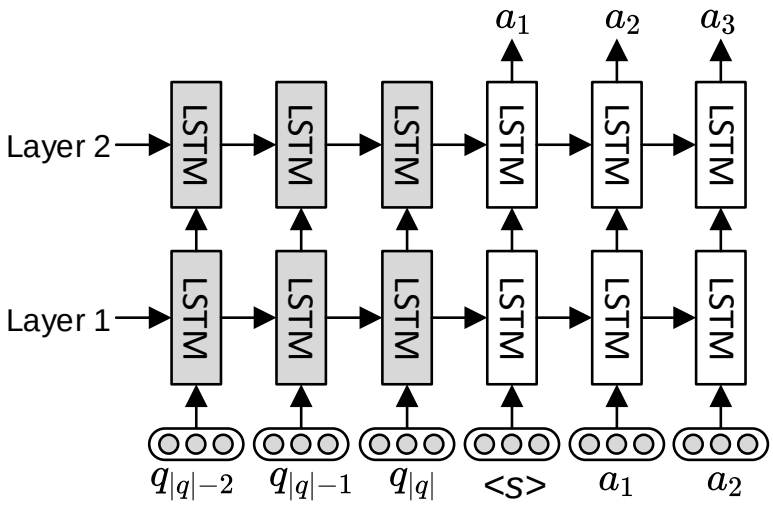


Figure 2.2: Sequence-to-sequence (SEQ2SEQ) model with two-layer recurrent neural networks. LSTM units are shown in Figure 2.3.

the decoder. Let $\mathbf{h}_t^l \in \mathbb{R}^n$ denote the hidden vector at time step t and layer l. $\mathbf{h}_t^l$ is then computed by:

$$\mathbf{h}_t^l = \mathrm{f}_{\mathrm{LSTM}}\left(\mathbf{h}_{t-1}^l, \mathbf{h}_t^{l-1}\right) \tag{2.2}$$

where $\mathrm{f}_{\mathrm{LSTM}}$ refers to the LSTM function being used. In our experiments we follow the architecture described in Zaremba et al. (2015), however other types of gated activation functions are possible (e.g., Cho et al. (2014)). The LSTM unit is given by:

$$\begin{aligned}
\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} &= \begin{pmatrix} \mathrm{sigm} \\ \mathrm{sigm} \\ \mathrm{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} \mathbf{h}_t^{l-1} \\ \mathbf{h}_{t-1}^l \end{pmatrix} \\
\mathbf{p}_t^l &= \mathbf{f} \odot \mathbf{p}_{t-1}^l + \mathbf{i} \odot \mathbf{g} \\
\mathbf{h}_t^l &= \mathbf{o} \odot \tanh(\mathbf{p}_t^l)
\end{aligned} \tag{2.3}$$

where tanh, sigm, and $\odot$ are element-wise operators, and $W^l \in \mathbb{R}^{4n \times 2n}$ is a weight matrix for the l-th layer. As shown in Figure 2.3, the input modulation gate creates vector $\mathbf{g}$ that can be added to the cell state. The input gate controls which values the cell will update. The forget gate determines whether the values of the previous cell state should be kept or not. The output gate masks the cell state vector and obtains the final output.

For the encoder, $\mathbf{h}_t^0 = \mathbf{W}_q \mathbf{e}(q_t)$ is the word vector of the current input token, with $\mathbf{W}_q \in \mathbb{R}^{n \times |V_q|}$ being a parameter matrix, and $\mathbf{e}(\cdot)$ the index of the corresponding

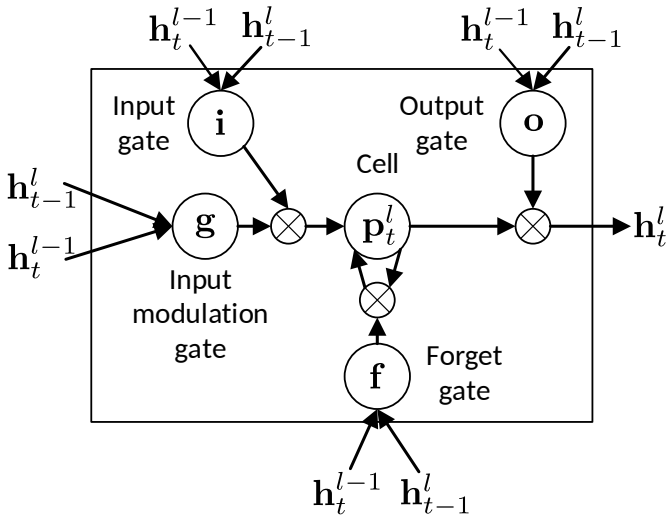


Figure 2.3: Long short-term memory (LSTM) unit.

token. For the decoder, $\mathbf{h}_t^0 = \mathbf{W}_a \mathbf{e}(a_{t-1})$ is the word vector of the previous predicted word, where $\mathbf{W}_a \in \mathbb{R}^{n \times |V_a|}$. Notice that the encoder and decoder have different LSTM parameters.

Once the tokens of the input sequence $q_1, \cdots, q_{|q|}$ are encoded into vectors, they are used to initialize the hidden states of the first time step in the decoder. Next, the hidden vector of the topmost LSTM $\mathbf{h}_t^L$ in the decoder is used to predict the t-th output token as:

$$p(a_t | a_{<t}, q) = \text{softmax}_{a_t}\left(\mathbf{W}_o \mathbf{h}_t^L\right) \tag{2.4}$$

where $\mathbf{W}_o \in \mathbb{R}^{|V_a| \times n}$ is a parameter matrix.

We augment every sequence with a "start-of-sequence" <s> and "end-of-sequence" </s> token. The generation process terminates once </s> is predicted. The conditional probability of generating the whole sequence $p(a|q)$ is then obtained using Equation (2.1).

2.2.2 Sequence-to-Tree Model

The SEQ2SEQ model has a potential drawback in that it ignores the hierarchical structure of logical forms. As a result, it needs to memorize various pieces of auxiliary information (e.g., bracket pairs) to generate well-formed output. In the following we present a hierarchical tree decoder which is more faithful to the compositional nature of meaning representations. A schematic description of the model is shown in Figure 2.4.

The present model shares the same encoder with the sequence-to-sequence model described in Section 2.2.1 (essentially it learns to encode input q as vectors). However,

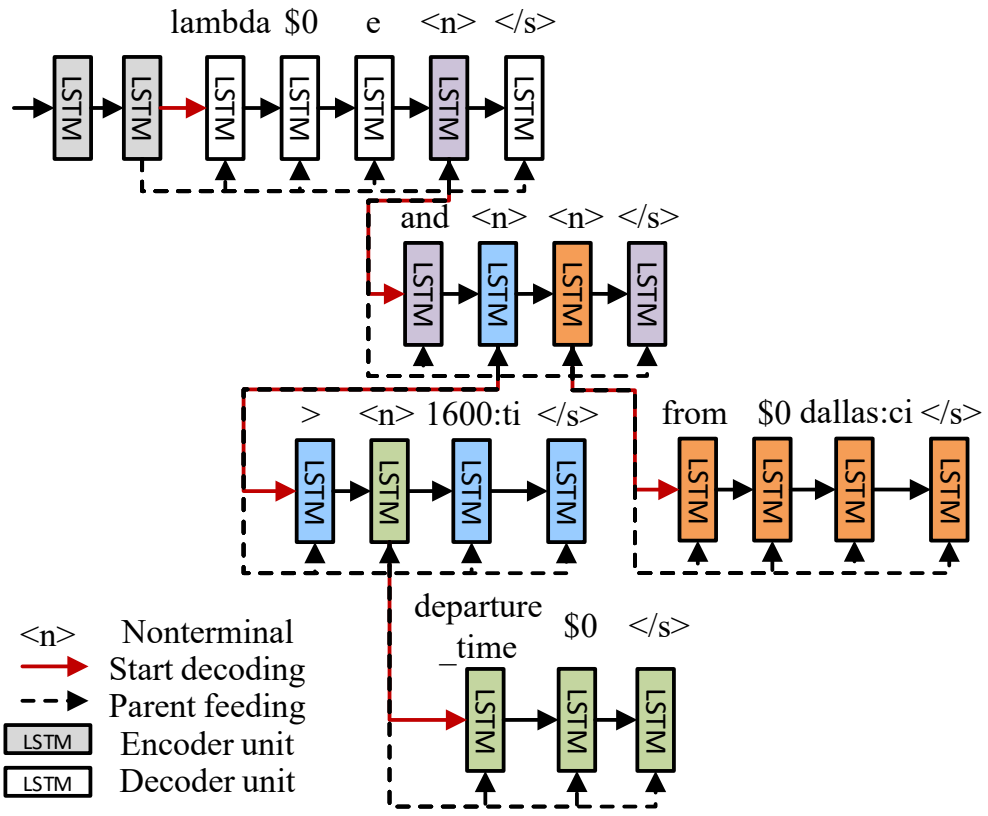


Figure 2.4: Sequence-to-tree (SEQ2TREE) model with a hierarchical tree decoder.

its decoder is fundamentally different as it generates logical forms in a top-down manner. In order to represent tree structure, we define a "nonterminal" <*n*> token which indicates subtrees. As shown in Figure 2.4, we preprocess the logical form "*lambda $0 e (and (>(departure_time $0) 1600:ti) (from $0 dallas:ci))*" to a tree by replacing tokens between pairs of brackets with nonterminals. Special tokens <*s*> and <*(*> denote the beginning of a sequence and nonterminal sequence, respectively (as shown in Figure 2.5). Token <*/s*> represents the end of sequence.

After encoding input q, the hierarchical tree decoder uses recurrent neural networks to generate tokens at depth 1 of the subtree corresponding to parts of logical form a. If the predicted token is <*n*>, we decode the sequence by conditioning on the nonterminal's hidden vector. This process terminates when no more nonterminals are emitted. In other words, a sequence decoder is used to hierarchically generate the tree structure.

In contrast to the sequence decoder described in Section 2.2.1, the current hidden state does not only depend on its previous time step. In order to better utilize the parent nonterminal's information, we introduce a *parent-feeding* connection where the hidden

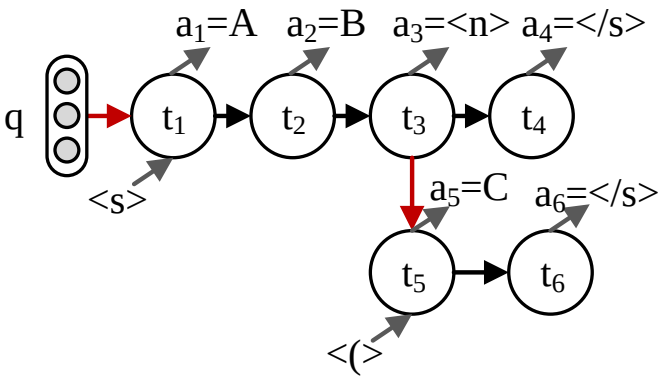


Figure 2.5: A SEQ2TREE decoding example for the logical form "*A B (C)*".

vector of the parent nonterminal is concatenated with the inputs and fed into LSTM.

As an example, Figure 2.5 shows the decoding tree corresponding to the logical form "*A B (C)*", where $a_1 \cdots a_6$ are predicted tokens, and $t_1 \cdots t_6$ denote different time steps. Span "*(C)*" corresponds to a subtree. Decoding in this example has two steps: once input q has been encoded, we first generate $a_1 \cdots a_4$ at depth 1 until token *</s>* is predicted; next, we generate a_5, a_6 by conditioning on nonterminal t_3's hidden vectors. The probability $p(a|q)$ is the product of these two sequence decoding steps:

$$p(a|q) = p(a_1 a_2 a_3 a_4|q)\, p(a_5 a_6|a_{\leq 3}, q) \tag{2.5}$$

where Equation (2.4) is used for the prediction of each output token.

2.2.3 Attention Mechanism

As shown in Equation (2.4), the hidden vectors of the input sequence are not directly used in the decoding process. However, it makes intuitively sense to consider relevant information from the input to better predict the current token. Following this idea, various techniques have been proposed to integrate encoder-side information (in the form of a context vector) at each time step of the decoder (Bahdanau et al., 2015; Luong et al., 2015a; Xu et al., 2015).

As shown in Figure 2.6, in order to find relevant encoder-side context for the current hidden state $\mathbf{h}_t^L$ of decoder, we compute its attention score with the k-th hidden state in the encoder as:

$$r_{t,k} \propto \exp\{\mathbf{h}_t^L \cdot \mathbf{h}_k^L\} \tag{2.6}$$

where $\sum_{j=1}^{|q|} r_{t,j} = 1$, and $\mathbf{h}_1^L, \cdots, \mathbf{h}_{|q|}^L$ are the top-layer hidden vectors of the encoder.

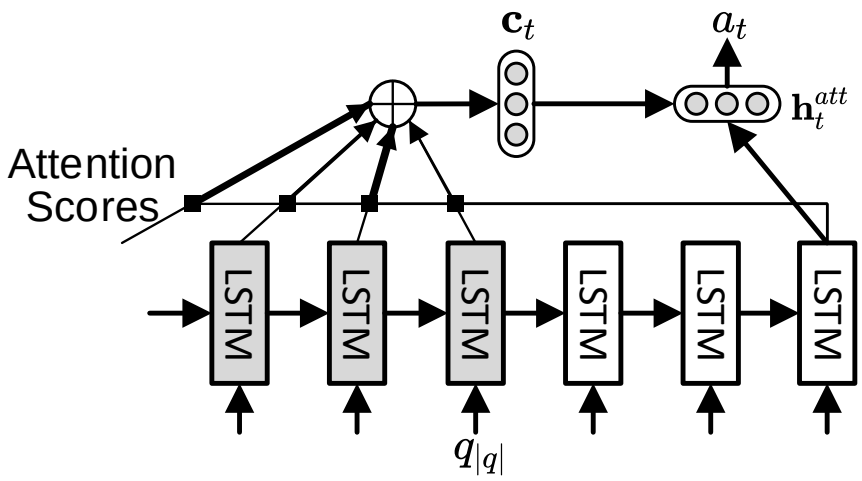


Figure 2.6: Attention scores are computed by the current hidden vector and all the hidden vectors of encoder. Then, the encoder-side context vector $\mathbf{c}_t$ is obtained in the form of a weighted sum, which is further used to predict a_t.

Then, the context vector is the weighted sum of the hidden vectors in the encoder:

$$\mathbf{c}_t = \sum_{k=1}^{|q|} r_{t,k} \mathbf{h}_k^L \tag{2.7}$$

In lieu of Equation (2.4), we further use this context vector which acts as a summary of the encoder to compute the probability of generating a_t as:

$$\mathbf{h}_t^{att} = \tanh\left(\mathbf{W}_1 \mathbf{h}_t^L + \mathbf{W}_2 \mathbf{c}_t\right) \tag{2.8}$$

$$p\left(a_t | a_{<t}, q\right) = \text{softmax}_{a_t}\left(\mathbf{W}_o \mathbf{h}_t^{att}\right) \tag{2.9}$$

where $\mathbf{W}_o \in \mathbb{R}^{|V_a| \times n}$ and $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{n \times n}$ are three parameter matrices.

2.2.4 Model Training

Our goal is to maximize the likelihood of the generated logical forms given natural language utterances as input. So the objective function is:

$$\text{maximize} \sum_{(q,a) \in \mathcal{D}} \log p\left(a|q\right) \tag{2.10}$$

where $\mathcal{D}$ is the set of all natural language-logical form training pairs, and $p\left(a|q\right)$ is computed as shown in Equation (2.1).

The RMSProp algorithm (Tieleman and Hinton, 2012) is employed to solve this non-convex optimization problem. Moreover, dropout is used for regularizing the

Algorithm 1 Decoding for SEQ2TREE

Input: q: Natural language utterance

Output: $\hat{a}$: Decoding result

Function: SeqEnc: Encode sequence to vector

SeqDec: Decode vector to sequence

HidVec: Get hidden vectors

▷ *Push the encoding result to a queue*

$Q.init(\{hid : \mathsf{SeqEnc}(q)\})$

▷ *Decode until no more nonterminals*

while $(c \leftarrow Q.pop()) \neq \varnothing$ **do**

    ▷ *Call sequence decoder*

    $c.children \leftarrow \mathsf{SeqDec}(c.hid)$

    ▷ *Push new nonterminals to queue*

    for $n \leftarrow$ nonterminal in $c.children$ **do**

        $Q.push(\{hid : \mathsf{HidVec}(n)\})$

$\hat{a} \leftarrow$ convert decoding tree to output sequence

model (Zaremba et al., 2015). Specifically, dropout operators are used between different LSTM layers and for the hidden layers before the softmax classifiers. This technique can substantially reduce overfitting, especially on datasets of small size.

2.2.5 Inference

At test time, we predict the logical form for an input utterance q by:

$$\hat{a} = \arg\max_{a'} p\left(a'|q\right) \tag{2.11}$$

where a' represents a candidate output. However, it is impractical to iterate over all possible results to obtain the optimal prediction. According to Equation (2.1), we decompose the probability $p(a|q)$ so that we can use greedy search (or beam search) to generate tokens one by one.

Algorithm 1 describes the decoding process for SEQ2TREE. The time complexity of both decoders is $O(|a|)$, where $|a|$ is the length of output. The extra computation of SEQ2TREE compared with SEQ2SEQ is to maintain the nonterminal queue, which can be negligible because most of the time is spent on matrix operations. We implement the hierarchical tree decoder in a batch mode, so that it can fully utilize GPUs. Specifically,

as shown in Algorithm 1, every time we pop multiple nonterminals from the queue, we decode these nonterminals in one batch.

2.2.6 Argument Identification

The majority of semantic parsing datasets have been developed with question-answering in mind. In the typical application setting, natural language questions are mapped into logical forms and executed on a knowledge base to obtain an answer. Due to the nature of the question-answering task, many natural language utterances contain entities or numbers that are often parsed as arguments in the logical form. Some of them are unavoidably rare or do not appear in the training set at all (this is especially true for small-scale datasets). Conventional sequence encoders simply replace rare words with a special unknown word symbol (Luong et al., 2015b; Jean et al., 2015), which would be detrimental for semantic parsing.

We have developed a simple procedure for argument identification. Specifically, we identify entities and numbers in input questions and replace them with their type names and unique IDs. For instance, we pre-process the training example "*jobs with a salary of 40000*" and its logical form `job(ans),salary_greater_than(ans,40000,year)` as "*jobs with a salary of* $\underline{num_0}$" and `job(ans),salary_greater_than(ans,`num_0`,year)`. We use different IDs for multiple arguments that have the same type to distinguish them. For example, the input "*i need to go from boston to dallas*" and its meaning representation `(lambda $0 e (and (from $0 boston:ci) (to $0 dallas:ci)))` are converted to "*i need to go from* $\underline{ci_0}$ *to* $\underline{ci_1}$" and `(lambda $0 e (and (from $0 ci`$_0$`) (to $0 ci`$_1$`)))`, where $\underline{ci_0}$ and $\underline{ci_1}$ represent the cities `boston:ci` and `dallas:ci`, respectively. We use the pre-processed examples as training data. At inference time, we also mask entities and numbers with their types and IDs. Once we obtain the decoding result, a post-processing step recovers all the markers $\underline{type_i}$ to their corresponding logical constants.

2.3 Experiments

We compare our method against multiple previous systems on four datasets. We describe these datasets below and present our experimental settings and results. Finally, we conduct model analysis in order to understand what the model learns. The code and pretrained models are available at https://github.com/donglixp/lang2logic.

Dataset	Length	Example
JOBS	9.8	*what microsoft jobs do not require a bscs?*
	22.9	`ans(company(J,'microsoft'),job(J),not((req_deg(J,'bscs'))))`
GEO	7.6	*what is the population of the state with the largest area?*
	19.1	`(population:i (argmax $0 (state:t $0) (area:i $0)))`
ATIS	11.1	*dallas to san francisco leaving after 4 in the afternoon please*
	28.1	`(lambda $0 e (and (> (departure_time $0) 1600:ti) (from $0 dallas:ci) (to $0 san_francisco:ci)))`
IFTTT	7.0	*Turn on heater when temperature drops below 58 degree*
	21.8	`Weather_Current_temperature_drops_below_((Temperature (58)) (Degrees_in (f))) THEN WeMo_Insight_Switch_Turn_on_((Which_switch? ("")))`

Table 2.1: Examples of natural language descriptions and their meaning representations from four datasets. The average length of input and output sequences is shown in the second column.

2.3.1 Datasets

Our model was trained on the following datasets, covering different domains and using different meaning representations. Examples for each domain are shown in Table 2.1.

JOBS This benchmark dataset contains 640 queries to a database of job listings. Specifically, questions are paired with Prolog-style queries. We used the same training-test split as Zettlemoyer and Collins (2005) which contains 500 training and 140 test instances. Values for the variables company, degree, language, platform, location, job area, and number are identified.

GEO This is a standard semantic parsing benchmark which contains 880 queries to a database of U.S. geography. GEO has 880 instances split into a training set of 600 training examples and 280 test examples (Zettlemoyer and Collins, 2005). We used the same meaning representation based on lambda-calculus as Kwiatkowski et al. (2011). Values for the variables city, state, country, river, and number are identified.

ATIS This dataset has 5,410 queries to a flight booking system. The standard split has 4,480 training instances, 480 development instances, and 450 test instances. Sentences are paired with lambda-calculus expressions. Values for the variables date, time, city,

aircraft code, airport, airline, and number are identified.

IFTTT Quirk et al. (2015) created this dataset by extracting a large number of if-this-then-that recipes from the IFTTT website[1]. Recipes are simple programs with exactly one trigger and one action which users specify on the site. Whenever the conditions of the trigger are satisfied, the action is performed. Actions typically revolve around home security (e.g., "*turn on my lights when I arrive home*"), automation (e.g., "*text me if the door opens*"), well-being (e.g., "*remind me to drink water if I've been at a bar for more than two hours*"), and so on. Triggers and actions are selected from different channels (160 in total) representing various types of services, devices (e.g., Android), and knowledge sources (such as ESPN or Gmail). In the dataset, there are 552 trigger functions from 128 channels, and 229 action functions from 99 channels. We used Quirk et al.'s (2015) original split which contains 77,495 training, 5,171 development, and 4,294 test examples. The IFTTT programs are represented as abstract syntax trees and are paired with natural language descriptions provided by users (see Table 2.1). Here, numbers and URLs are identified.

2.3.2 Settings

Natural language sentences were lowercased; misspellings were corrected using a dictionary based on the Wikipedia list of common misspellings. Words were stemmed using NLTK (Bird et al., 2009). For IFTTT, we filtered tokens, channels, and functions which appeared less than five times in the training set. For the other datasets, we filtered input words which did not occur at least two times in the training set, but kept all tokens in the logical forms. Plain string matching[2] against lexicons of augments was employed for pre-processing as described in Section 2.2.6.

Model hyper-parameters were cross-validated on the training set for JOBS and GEO. We used the standard development sets for ATIS and IFTTT. We used the RMSProp algorithm (with batch size set to 20) to update the parameters. The smoothing constant of RMSProp was 0.95. Gradients were clipped at 5 to alleviate the exploding gradient problem (Pascanu et al., 2013). Parameters were randomly initialized from a uniform distribution $\mathcal{U}(-0.08, 0.08)$. A two-layer LSTM was used for IFTTT,

[1] http://www.ifttt.com

[2] More sophisticated approaches, such as named entity linking (Ling et al., 2015) and typing (Ling and Weld, 2012; Dong et al., 2015a), can be used to improve argument identification and make the pre-processing step language-independent.

Method	Accuracy
COCKTAIL (Tang and Mooney, 2001)	79.4
PRECISE (Popescu et al., 2003)	88.0
ZC05 (Zettlemoyer and Collins, 2005)	79.3
DCS+L (Liang et al., 2013)	90.7
TISP (Zhao and Huang, 2015)	85.0
ASN (Rabinovich et al., 2017)	91.4
ASN+SUPATT (Rabinovich et al., 2017)	92.9
SEQ2SEQ	87.1
− attention	77.9
− argument	70.7
SEQ2TREE	90.0
− attention	83.6

Table 2.2: Evaluation results on JOBS. Methods in the last two blocks are neural models.

while a one-layer LSTM was employed for the other domains. The dropout rate was selected from $\{0.2, 0.3, 0.4, 0.5\}$. Dimensions of hidden vector and word embedding were selected from $\{150, 200, 250\}$. Early stopping was used to determine the number of epochs. Input sentences were reversed before feeding into the encoder (Sutskever et al., 2014). We use greedy search to generate logical forms during inference. Notice that two decoders with shared word embeddings were used to predict triggers and actions for IFTTT, and two softmax classifiers are used to classify channels and functions.

2.3.3 Results

We first discuss the performance of our model on JOBS, GEO, and ATIS, and then examine our results on IFTTT. Tables 2.2–2.4 present comparisons against a variety of systems previously described in the literature. We report results with the full models (SEQ2SEQ, SEQ2TREE) and two ablation variants, i.e., without an attention mechanism (−attention) and without argument identification (−argument). We report accuracy which is defined as the proportion of the input sentences that are correctly parsed to their gold standard logical forms. Notice that DCS+L, KCAZ13 and GUSP out-

Method	Accuracy
SCISSOR (Ge and Mooney, 2005)	72.3
KRISP (Kate and Mooney, 2006)	71.7
WASP (Wong and Mooney, 2006)	74.8
λ-WASP (Wong and Mooney, 2007)	86.6
LNLZ08 (Lu et al., 2008)	81.8
ZC05 (Zettlemoyer and Collins, 2005)	79.3
ZC07 (Zettlemoyer and Collins, 2007)	86.1
UBL (Kwiatkowksi et al., 2010)	87.9
FUBL (Kwiatkowski et al., 2011)	88.6
KCAZ13 (Kwiatkowski et al., 2013)	89.0
DCS+L (Liang et al., 2013)	87.9
WKZ14 (Wang et al., 2014)	90.4
TISP (Zhao and Huang, 2015)	88.9
DATARECOMB (Jia and Liang, 2016)	89.3
ASN (Rabinovich et al., 2017)	85.7
ASN+SUPATT (Rabinovich et al., 2017)	87.1
SEQ2ACT (Chen et al., 2018)	88.9
SEQ2SEQ	84.6
– attention	72.9
– argument	68.6
SEQ2TREE	87.1
– attention	76.8

Table 2.3: Evaluation results on GEO. 10-fold cross-validation is used for the systems shown in the first block of the table. The standard split of ZC05 is used for all other systems. Methods in the last two blocks are neural models.

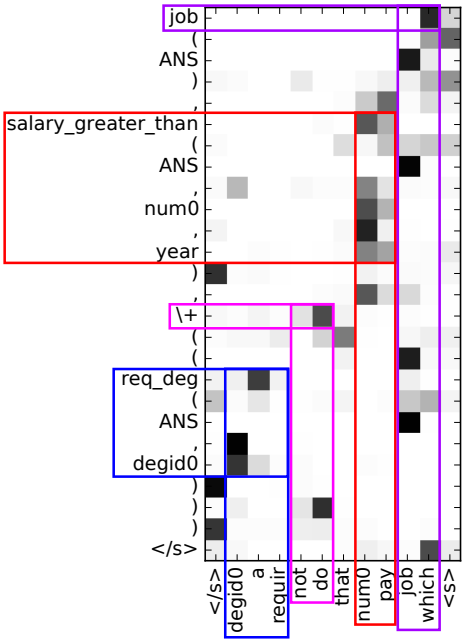
put answers directly, so accuracy in this setting is defined as the percentage of correct answers.

Overall, SEQ2TREE is superior to SEQ2SEQ. The result is to be expected since SEQ2TREE explicitly models compositional structure. On the JOBS and GEO datasets which contain logical forms with nested structures, SEQ2TREE outperforms SEQ2SEQ by 2.9% and 2.5%, respectively. SEQ2TREE achieves better accuracy over SEQ2SEQ

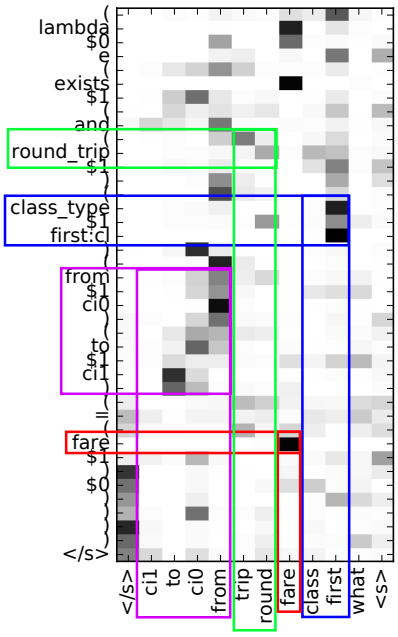
Method	Accuracy
ZC07 (Zettlemoyer and Collins, 2007)	84.6
UBL (Kwiatkowksi et al., 2010)	71.4
FUBL (Kwiatkowski et al., 2011)	82.8
GUSP-FULL (Poon, 2013)	74.8
GUSP++ (Poon, 2013)	83.5
WKZ14 (Wang et al., 2014)	91.3
TISP (Zhao and Huang, 2015)	84.2
DATARECOMB (Jia and Liang, 2016)	84.6
ASN (Rabinovich et al., 2017)	85.3
ASN+SUPATT (Rabinovich et al., 2017)	85.9
SEQ2ACT (Chen et al., 2018)	85.5
SEQ2SEQ	84.2
– attention	75.7
– argument	72.3
SEQ2TREE	84.6
– attention	77.5

Table 2.4: Evaluation results on ATIS. Methods in the last two blocks are neural models.

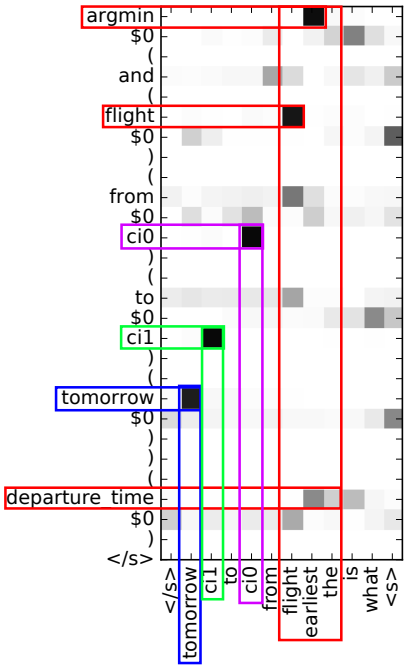
on ATIS too, however, the difference is smaller, since ATIS is a simpler domain without complex nested structures. We further compute significance levels using bootstrap hypothesis testing (Efron and Tibshirani, 1994). The improvements of SEQ2TREE on JOBS and GEO are significant at $p < 0.1$ and $p < 0.05$, respectively, while the gain on ATIS is non-significant. We find that adding attention substantially improves performance on all three datasets. This underlines the importance of utilizing soft alignments between inputs and outputs. We further analyze what the attention layer learns in Figure 2.7. Moreover, our results show that argument identification is critical for small-scale datasets. For example, about 92% of city names appear less than 4 times in the GEO training set, so it is difficult to learn reliable parameters for these words. In relation to previous work, the proposed models achieve comparable or better performance. Importantly, we use the same framework (SEQ2SEQ or SEQ2TREE) across datasets and meaning representations (Prolog-style logical forms in JOBS and lambda calculus in the other two datasets) without modification. Despite this relatively



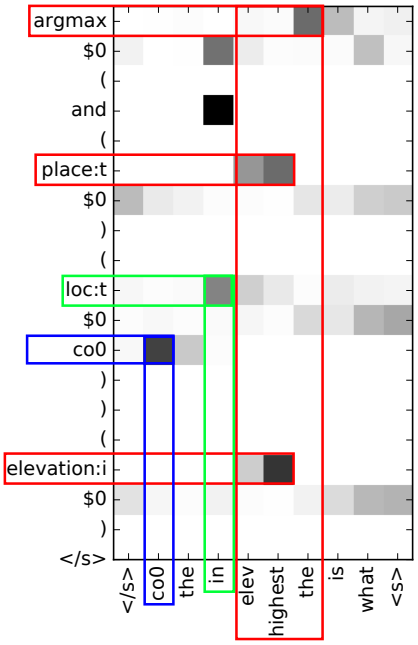
(a) which jobs pay num0 that do not require a degid0



(b) what's first class fare round trip from ci0 to ci1



(c) what is the earliest flight from ci0 to ci1 tomorrow



(d) what is the highest elevation in the co0

Figure 2.7: Alignments (same color rectangles) produced by the attention mechanism (darker color represents higher attention score). Input sentences are reversed and stemmed. Model output is shown for SEQ2SEQ (a, b) and SEQ2TREE (c, d).

simple approach, we observe that SEQ2TREE ranks second on JOBS, and is tied for first place with ZC07 on ATIS.

We show examples of alignments produced by SEQ2SEQ in Figures 2.7a and 2.7b. Alignments produced by SEQ2TREE are shown in Figures 2.7c and 2.7d. Matrices of attention scores are computed using Equation (2.6) and are represented in grayscale. Aligned input words and logical form predicates are enclosed in (same color) rectangles whose overlapping areas contain the attention scores. Also notice that attention scores are computed by LSTM hidden vectors which encode context information rather than just the words in their current positions. The examples demonstrate that the attention mechanism can successfully model the correspondence between sentences and logical forms, capturing reordering (Figure 2.7b), many-to-many (Figure 2.7a), and many-to-one alignments (Figures 2.7c,d).

For IFTTT, we follow the same evaluation protocol introduced in Quirk et al. (2015). The dataset is extremely noisy and measuring accuracy is problematic since predicted abstract syntax trees (ASTs) almost never exactly match the gold standard. Quirk et al. view an AST as a set of productions and compute balanced F1 instead which we also adopt. The first column in Table 2.5 shows the percentage of channels selected correctly for both triggers and actions. The second column measures accuracy for *both* channels and functions. The last column shows balanced F1 against the gold tree over all productions in the proposed derivation. We compare our model against posclass, the method introduced in Quirk et al. and several of their baselines. posclass is reminiscent of KRISP (Kate and Mooney, 2006), it learns distributions over productions given input sentences represented as a bag of linguistic features. The retrieval baseline finds the closest description in the training data based on character string-edit-distance and returns the recipe for that training program. The phrasal method uses phrase-based machine translation to generate the recipe, whereas sync extracts synchronous grammar rules from the data, essentially recreating WASP (Wong and Mooney, 2006). Finally, they use a binary classifier to predict whether a production should be present in the derivation tree corresponding to the description.

Quirk et al. (2015) report results on the full test data and smaller subsets after noise filtering, e.g., when non-English and unintelligible descriptions are removed (Tables 2.5a and 2.5b). They also ran their system on a high-quality subset of description-program pairs which were found in the gold standard and at least three humans managed to independently reproduce (Table 2.5c). Across all subsets our models outperforms posclass and related baselines. Again we observe that SEQ2TREE consistently

Method	Channel	+Func	F1
retrieval	28.9	20.2	41.7
phrasal	19.3	11.3	35.3
sync	18.1	10.6	35.1
classifier	48.8	35.2	48.4
posclass	50.0	36.9	49.3
LR	56.0	44.3	–
NN	55.1	41.2	–
SEQ2SEQ	54.3	39.2	50.1
– attention	54.0	37.9	49.8
– argument	53.9	38.6	49.7
SEQ2TREE	55.2	40.1	50.4
– attention	54.3	38.2	50.0

(a) Omit non-English.

Method	Channel	+Func	F1
retrieval	36.8	25.4	49.0
phrasal	27.8	16.4	39.9
sync	26.7	15.5	37.6
classifier	64.8	47.2	56.5
posclass	67.2	50.4	57.7
LR	71.9	56.6	–
NN	71.3	53.7	–
DOUBLYRNN	74.9	54.3	65.2
SEQ2SEQ	68.8	50.5	60.3
– attention	68.7	48.9	59.5
– argument	68.8	50.4	59.7
SEQ2TREE	69.6	51.4	60.4
– attention	68.7	49.5	60.2

(b) Omit non-English & unintelligible.

Method	Channel	+Func	F1
retrieval	43.3	32.3	56.2
phrasal	37.2	23.5	45.5
sync	36.5	24.1	42.8
classifier	79.3	66.2	65.0
posclass	81.4	71.0	66.5
LR	88.8	82.5	–
NN	88.0	74.3	–
DOUBLYRNN	90.1	78.2	77.4
SNM	90.0	82.0	–
SEQ2SEQ	87.8	75.2	73.7
– attention	88.3	73.8	72.9
– argument	86.8	74.9	70.8
SEQ2TREE	89.7	78.4	74.2
– attention	87.6	74.9	73.5

(c) ≥ 3 turkers agree with gold.

Table 2.5: Evaluation results on IFTTT. Results of retrieval, phrasal, sync, classifier and posclass are taken from (Quirk et al., 2015), LR and NN are from (Beltagy and Quirk, 2016), DOUBLYRNN is from (Alvarez-Melis and Jaakkola, 2017), SNM is from (Yin and Neubig, 2017). Methods in the last two blocks are neural models.

outperforms SEQ2SEQ, albeit with a small margin. The gains of F1 scores are non-significant, partly because the program trees of IFTTT have a fixed and simple structure. Compared to the previous datasets, the attention mechanism and our argument identification method yield less of an improvement. This may be due to the size of Quirk et al. (2015) and the way it was created – user curated descriptions are often of low quality, and thus align very loosely to their corresponding ASTs.

2.3.4 Error Analysis

Finally, we inspected the output of our model in order to identify the most common causes of errors which we summarize below.

Under-Mapping The attention model used in our experiments does not consider the alignment history. So, some question words may be ignored in the decoding process. For example, the input "*show me the flight to and from ap0*" in the development set of ATIS is under-mapped to `(lambda $0 e (and (flight $0) (from $0 ap0)))`, where `(or (from $0 ap0) (to $0 ap0))` should be predicted. This is a common problem for encoder-decoder models and can be addressed by explicitly modeling the decoding coverage of the source words (Tu et al., 2016; Cohn et al., 2016) or using length normalization to encourage longer sequences (Wu et al., 2016). Keeping track of the attention history would help adjust future attention and guide the decoder towards untranslated source words.

Argument Identification Some mentions are incorrectly identified as arguments. For example, the word "*may*" is sometimes identified as a month when it is simply a modal verb. Moreover, some argument mentions are ambiguous. For instance, "*6 o'clock*" can be used to express either "*6 am*" or "*6 pm*". We could disambiguate arguments based on contextual information. The execution results of logical forms could also help prune unreasonable arguments.

Well-Formedness Because no explicit constraint is used for the decoding process of sequence-to-sequence models, the well-formedness of predictions is not guaranteed. We use brackets to linearize tree-structured logical forms for sequence modeling. A typical prediction error is that the pairs of brackets are mismatched, i.e., the predicted result is not a well-formed tree. On the development set of ATIS, the predictions that are ill-formed trees account for 11.27% of the error cases. The sequence-to-tree model

partially solves the problem of well-formedness, since the decoder always produces valid trees. We can further integrate a grammar model into the decoder, which forces predictions to obey the predefined grammar of meaning representations (Xiao et al., 2016; Yin and Neubig, 2017; Rabinovich et al., 2017; Krishnamurthy et al., 2017).

Rare Words Because the data size of JOBS, GEO, and ATIS is relatively small, some question words are rare in the training set, which makes it hard to estimate reliable parameters for them. We propose confidence estimation and interpretation methods in Chapter 4 to provides feedbacks to users for uncertain predictions. Moreover, we leverage external resources to generate paraphrases for natural language inputs in order to improve model coverage.

2.4 Summary

In this chapter we presented an encoder-decoder neural network model for mapping natural language descriptions to their meaning representations. We encode natural language utterances into vectors and generate their corresponding logical forms as sequences or trees using recurrent neural networks with long short-term memory units. Experimental results show that enhancing the model with a hierarchical tree decoder and an attention mechanism improves performance across the board. Extensive comparisons with previous methods show that our approach performs competitively, without recourse to domain- or representation-specific features.

The proposed models are portable as they can be end-to-end trained by giving annotated data, namely, natural language utterances paired with their meaning representations. So we can easily adapt the models to different applications with minimal efforts. The structural gap between inputs and outputs is bridged by neural encoder-decoder networks augmented with attention mechanisms. Moreover, explicitly considering the hierarchical structure of formal languages constrains outputs in the space of well-formed trees, which enhances the model robustness. In the next chapter we will further explore how to decode structured representations and model meaning at different levels of granularity.

Chapter 3

Coarse-to-Fine Decoding

The fact that meaning representations are typically structured objects prompt efforts to develop neural architectures which explicitly account for their structure as presented in Chapter 2. Other examples include tree or graph decoders (Alvarez-Melis and Jaakkola, 2017; Chen et al., 2018), decoders constrained by a grammar model (Xiao et al., 2016; Yin and Neubig, 2017; Krishnamurthy et al., 2017), or modular decoders which use syntax to dynamically compose various submodels (Rabinovich et al., 2017). In this chapter, we propose to decompose the decoding process into two stages. The first decoder focuses on predicting a rough *sketch* of the meaning representation, which omits low-level details, such as arguments and variable names. Example sketches for various meaning representations are shown in Table 3.1. Then, a second decoder fills in missing details by conditioning on the natural language input and the sketch itself. Specifically, the sketch constrains the generation process and is encoded into vectors to guide decoding.

We argue that there are at least three advantages to the proposed approach. Firstly, the decomposition disentangles high-level from low-level semantic information, which enables the decoder to model meaning at different levels of granularity. As shown in Table 3.1, sketches are more compact and as a result easier to generate compared to decoding the entire meaning structure in one go. For examples in the dataset ATIS, the average length of meaning sketch is 9.2, while the original average length is 21.1. Secondly, the model can explicitly share knowledge of coarse structures for the examples that have the same sketch (i.e., basic meaning), even though their actual meaning representations are different (e.g., due to different variable names or argument values). Thirdly, after generating the sketch, the decoder knows what the basic meaning of the utterance looks like, and the model can use it as global context to improve the predic-

Dataset	Num.	Example
GEO	7.6	*x* : *which state has the most rivers running through it?*
	13.7	*y* : `(argmax $0 (state:t $0) (count $1 (and (river:t $1) (loc:t $1 $0))))`
	6.9	*a* : `(argmax#1 state:t@1 (count#1 (and river:t@1 loc:t@2 ) ) )`
ATIS	11.1	*x* : *all flights from dallas before 10am*
	21.1	*y* : `(lambda $0 e (and (flight $0) (from $0 dallas:ci) (< (departure_time $0) 1000:ti)))`
	9.2	*a* : `(lambda#2 (and flight@1 from@2 (< departure_time@1 ? ) ) )`
DJANGO	14.4	*x* : *if length of bits is lesser than integer 3 or second element of bits is not equal to string 'as' ,*
	8.7	*y* : `if len(bits) < 3 or bits[1] != 'as':`
	8.0	*a* : `if len ( NAME ) < NUMBER or NAME [ NUMBER ] != STRING :`
WIKISQL	17.9	Table: ‖*Pianist*‖*Conductor*‖*Record Company*‖*Year of Recording*‖*Format*‖
	13.3	*x* : *What record company did conductor Mikhail Snitko record for after 1996?*
	13.0	*y* : `SELECT Record Company WHERE (Year of Recording > 1996) AND (Conductor = Mikhail Snitko)`
	2.7	*a* : `WHERE > AND =`

Table 3.1: Examples of natural language expressions x, their meaning representations y, and meaning sketches a. The average number of tokens is shown in the second column.

tion of the final details. The sketches can also provide constraints for the generation of fine-grained meaning representations. For example, as shown in Table 3.1, sketch symbols indicate the number of missing tokens, and use type information to constrain the search space of decoding.

The proposed framework is flexible and not restricted to specific tasks or any particular model. We conduct experiments on four datasets representative of various semantic parsing tasks ranging from logical form parsing, to code generation, and SQL query generation. We adapt our architecture to these tasks and present several ways to obtain sketches from their respective meaning representations. Experimental results show that our framework achieves competitive performance compared with previous systems, despite employing relatively simple sequence decoders.

3.1 Related Work

Coarse-to-fine methods have been popular in the NLP literature, and are perhaps best known for syntactic parsing (Charniak et al., 2006; Petrov, 2011). The nonterminals of the grammar are clustered to represent syntactic structures at a coarse level. The training and inference processes have multiple stages. Each stage refines results of the previous stage until final outputs are predicted. Charniak et al. (2006) first identify the locations of constituents of the parse tree, and then distinguish only argument from modifier phrases. Next, prepositional phrases, nominal, verbal, and adjectival are predicted. In the last stage, all the categories are distinguished. Petrov (2011) develop a latent variable approach to automatically induce coarse-to-fine grammar rules from data. A similar idea is also used to build class-based (Brown et al., 1992; Niesler et al., 1998; Maltese et al., 2001) and discriminative (Morin and Bengio, 2005; Parvez et al., 2018) language models. The methods typically represent tokens by their type information or clusters. The clusters of words can be either manually derived (Maltese et al., 2001; Parvez et al., 2018) or automatically obtained by clustering algorithms (Brown et al., 1992; Maltese et al., 2001; Morin and Bengio, 2005). Language models first predict coarse categories of words, and then generate actual outputs.

For semantic parsing, Artzi and Zettlemoyer (2013) and Zhang et al. (2017) use coarse lexical entries or macro grammars to reduce the search space of semantic parsers. Compared with coarse-to-fine inference for lexical induction, sketches in our case are abstractions of the final meaning representation. Goldman et al. (2018) utilize a partially abstract representation to learn neural semantic parsers from weak supervision (i.e., denotations), where tokens are lifted to abstract forms according to lexical rules. The abstraction relieves the search and spuriousness challenges of weakly supervised semantic parsing. In our work, we explore different ways to define meaning sketches for various types of meaning representations, e.g., trees or other structured objects. Moreover, our model uses an additional encoder to encode the generated sketch into vectors, which provides global context for fine-grained meaning decoding.

The idea of using sketches as intermediate representations has also been explored in the field of program synthesis (Solar-Lezama, 2008; Zhang and Sun, 2013; Gaunt et al., 2016; Feng et al., 2017). Yaghmazadeh et al. (2017) use SEMPRE (Berant et al., 2013) to map a sentence into SQL sketches which are completed using program synthesis techniques and iteratively repaired if they are faulty. Bošnjak et al. (2017) present a differentiable Forth (Rather and Conklin, 2007) interpreter to complete slots for given

program sketches. The model is trained from pairs of input-output data with gradient descent algorithms. Murali et al. (2018) propose a method based on neural sketch learning for conditional program generation, which aims at generating source code in a Java-like programming language. The model learns to produce program sketches which abstract from names and operations. Then, the output code is concretized by using combinatorial techniques. In contrast, our model is trained using pairs of natural language descriptions and their meaning representations instead of input-output examples. Furthermore, we learn models for generation of both coarse- and fine-grained meaning representations, rather than using program synthesis techniques to produce final outputs.

3.2 Problem Formulation

Our goal is to learn semantic parsers from instances of natural language expressions paired with their structured meaning representations. Let $x = x_1 \cdots x_{|x|}$ denote a natural language expression, and $y = y_1 \cdots y_{|y|}$ its meaning representation. We wish to estimate $p(y|x)$, the conditional probability of meaning representation y given input x. We decompose $p(y|x)$ into a two-stage generation process:

$$p(y|x) = p(y|x,a)\, p(a|x) \tag{3.1}$$

where $a = a_1 \cdots a_{|a|}$ is an abstract sketch representing the meaning of y. We defer detailed description of how sketches are extracted to Section 3.3. Suffice it to say that the extraction amounts to stripping off arguments and variable names in logical forms, schema specific information in SQL queries, and substituting tokens with types in source code (see Table 3.1).

As shown in Figure 3.1, we first predict sketch a for input x, and then fill in missing details to generate the final meaning representation y by conditioning on both x and a. The sketch is encoded into vectors which in turn guide and constrain the decoding of y. We view the input expression x, the meaning representation y, and its sketch a as sequences. The generation probabilities are factorized as:

$$p(a|x) = \prod_{t=1}^{|a|} p(a_t|a_{<t}, x) \tag{3.2}$$

$$p(y|x,a) = \prod_{t=1}^{|y|} p(y_t|y_{<t}, x, a) \tag{3.3}$$

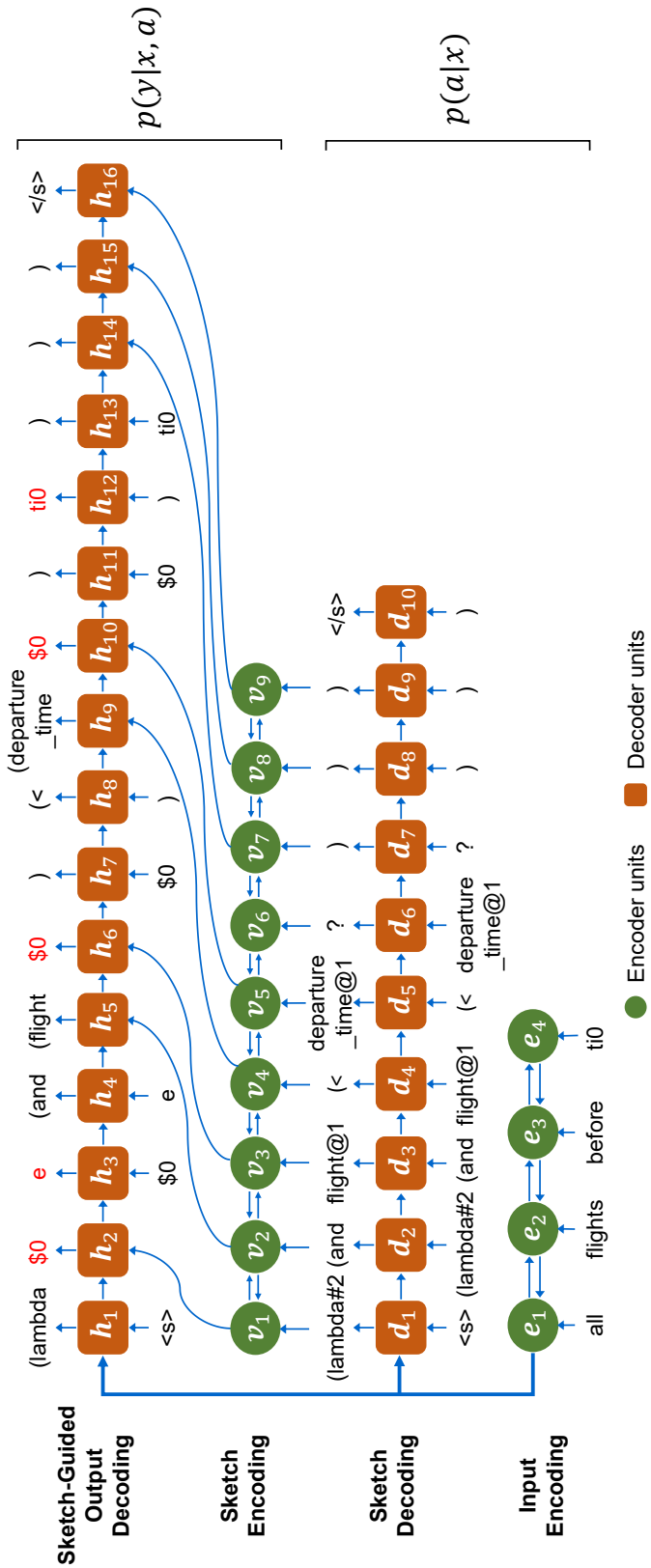


Figure 3.1: We first generate the meaning sketch a for natural language input x. Then, a fine meaning decoder fills in the missing details (shown in red) of meaning representation y. The coarse structure a is used to guide and constrain the output decoding.

where $a_{<t} = a_1 \cdots a_{t-1}$, and $y_{<t} = y_1 \cdots y_{t-1}$. In the following, we will explain how $p(a|x)$ and $p(y|x,a)$ are estimated.

3.2.1 Sketch Generation

An *encoder* is used to encode the natural language input x into vector representations. Then, a *decoder* learns to compute $p(a|x)$ and generate the sketch a conditioned on the encoding vectors.

Input Encoder Every input word is mapped to a vector via $\mathbf{x}_t = \mathbf{W}_x \mathbf{o}(x_t)$, where $\mathbf{W}_x \in \mathbb{R}^{n \times |\mathcal{V}_x|}$ is an embedding matrix, $|\mathcal{V}_x|$ is the vocabulary size, and $\mathbf{o}(x_t)$ a one-hot vector. We use a bi-directional recurrent neural network with long short-term memory units (LSTM, Hochreiter and Schmidhuber 1997) as the input encoder. The details of an LSTM unit are described in Section 2.2.1. The encoder recursively computes the hidden vectors at the t-th time step via:

$$\overrightarrow{\mathbf{e}}_t = \mathrm{f}_{\mathrm{LSTM}}\left(\overrightarrow{\mathbf{e}}_{t-1}, \mathbf{x}_t\right), t = 1, \cdots, |x| \tag{3.4}$$

$$\overleftarrow{\mathbf{e}}_t = \mathrm{f}_{\mathrm{LSTM}}\left(\overleftarrow{\mathbf{e}}_{t+1}, \mathbf{x}_t\right), t = |x|, \cdots, 1 \tag{3.5}$$

$$\mathbf{e}_t = [\overrightarrow{\mathbf{e}}_t, \overleftarrow{\mathbf{e}}_t] \tag{3.6}$$

where $[\cdot,\cdot]$ denotes vector concatenation, $\mathbf{e}_t \in \mathbb{R}^n$, and $\mathrm{f}_{\mathrm{LSTM}}$ is the LSTM function. Compared with SEQ2SEQ and SEQ2TREE that are described in Chapter 2, the input encoder used here is based on bi-directional LSTMs instead of uni-directional LSTMs.

Coarse Meaning Decoder The decoder's hidden vector at the t-th time step is computed by $\mathbf{d}_t = \mathrm{f}_{\mathrm{LSTM}}(\mathbf{d}_{t-1}, \mathbf{a}_{t-1})$, where $\mathbf{a}_{t-1} \in \mathbb{R}^n$ is the embedding of the previously predicted token. The hidden states of the first time step in the decoder are initialized by the concatenated encoding vectors $\mathbf{d}_0 = [\overrightarrow{\mathbf{e}}_{|x|}, \overleftarrow{\mathbf{e}}_1]$. Additionally, we use an attention mechanism (as described in Section 2.2.3) to learn soft alignments. We compute the attention score for the current time step t of the decoder, with the k-th hidden state in the encoder as:

$$s_{t,k} = \exp\{\mathbf{d}_t \cdot \mathbf{e}_k\} / Z_t \tag{3.7}$$

where $Z_t = \sum_{j=1}^{|x|} \exp\{\mathbf{d}_t \cdot \mathbf{e}_j\}$ is a normalization term. We then compute $p(a_t|a_{<t}, x)$ via:

$$\mathbf{e}_t^d = \sum_{k=1}^{|x|} s_{t,k}\mathbf{e}_k \tag{3.8}$$

$$\mathbf{d}_t^{att} = \tanh\left(\mathbf{W}_1\mathbf{d}_t + \mathbf{W}_2\mathbf{e}_t^d\right) \tag{3.9}$$

$$p(a_t|a_{<t}, x) = \text{softmax}_{a_t}\left(\mathbf{W}_o\mathbf{d}_t^{att} + \mathbf{b}_o\right) \tag{3.10}$$

where $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{n\times n}$, $\mathbf{W}_o \in \mathbb{R}^{|\mathcal{V}_a|\times n}$, and $\mathbf{b}_o \in \mathbb{R}^{|\mathcal{V}_a|}$ are parameters. Generation terminates once an end-of-sequence token "</s>" is emitted.

3.2.2 Meaning Representation Generation

Meaning representations are predicted by conditioning on the input x and the generated sketch a. The model uses the encoder-decoder architecture to compute $p(y|x, a)$, and decorates the sketch a with details to generate the final output.

Sketch Encoder As shown in Figure 3.1, a bi-directional LSTM encoder maps the sketch sequence a into vectors $\{\mathbf{v}_k\}_{k=1}^{|a|}$ as in Equation (3.6), where $\mathbf{v}_k$ denotes the vector of the k-th time step.

Fine Meaning Decoder The final decoder is based on recurrent neural networks with an attention mechanism, and shares the input encoder described in Section 3.2.1. The decoder's hidden states $\{\mathbf{h}_t\}_{t=1}^{|y|}$ are computed via:

$$\mathbf{i}_t = \begin{cases} \mathbf{v}_k & y_{t-1} \text{ is determined by } a_k \\ \mathbf{y}_{t-1} & \text{otherwise} \end{cases} \tag{3.11}$$

$$\mathbf{h}_t = \text{f}_{\text{LSTM}}(\mathbf{h}_{t-1}, \mathbf{i}_t) \tag{3.12}$$

where $\mathbf{h}_0 = [\overrightarrow{\mathbf{e}}_{|x|}, \overleftarrow{\mathbf{e}}_1]$, and $\mathbf{y}_{t-1}$ is the embedding of the previously predicted token. Apart from using the embeddings of previous tokens, the decoder is also fed with $\{\mathbf{v}_k\}_{k=1}^{|a|}$. If y_{t-1} is determined by a_k in the sketch (i.e., there is a one-to-one alignment between y_{t-1} and a_k), we use the corresponding token's vector $\mathbf{v}_k$ as input to the next time step.

The sketch constrains the decoding output. If the output token y_t is already in the sketch, we force y_t to conform to the sketch. The constrained decoding algorithm explicitly makes use of structural information of meaning representations. In some

cases, sketch tokens will indicate what information is missing (e.g., in Figure 3.1, token "*flight@1*" indicates that an argument is missing for the predicate "*flight*"). In other cases, sketch tokens will not reveal the number of missing tokens (e.g., "STRING" in DJANGO) but the decoder's output will indicate whether missing details have been generated (e.g., if the decoder emits a closing quote token for "STRING"). Moreover, type information in sketches can be used to constrain generation. In Table 3.1, sketch token "NUMBER" specifies that a numeric token should be emitted.

For the missing details, we use the hidden vector $\mathbf{h}_t$ to compute $p(y_t|y_{<t},x,a)$, analogously to Equations (3.7)–(3.10).

3.2.3 Training and Inference

The model's training objective is to maximize the log likelihood of the generated meaning representations given natural language expressions:

$$\max \sum_{(x,a,y)\in\mathcal{D}} \log p(y|x,a) + \log p(a|x) \tag{3.13}$$

where $\mathcal{D}$ represents natural language and meaning representation training pairs.

At test time, the prediction for input x is obtained via $\hat{a} = \arg\max_{a'} p(a'|x)$ and $\hat{y} = \arg\max_{y'} p(y'|x,\hat{a})$, where a' and y' represent coarse- and fine-grained meaning candidates. Because probabilities $p(a|x)$ and $p(y|x,a)$ are factorized as shown in Equations (3.2)–(3.3), we can obtain best results approximately by using greedy search to generate tokens one by one, rather than iterating over all candidates.

3.3 Task Description

In order to show that our framework applies across domains and meaning representations, we developed models for three tasks, namely parsing natural language to logical form, to Python source code, and to SQL query. These tasks represent typical use cases of natural language interfaces, i.e., querying database and completing desired actions using structured programs. We describe the datasets we used, and specify model details over and above the architecture presented in Section 3.2. For each of these tasks we also present the algorithms used to automatically extract sketches from output meaning representations.

Algorithm 2 Sketch for GEO and ATIS

Input: t: Tree-structured λ-calculus expression

$t.pred$: Predicate name, or operator name

Output: a: Meaning sketch

▷ *(count $0 (< (fare $0) 50:do)) → (count#1 (< fare@1 ?))*

function SKETCH(t)

  if t is leaf **then** ▷ *No nonterminal in arguments*

    return "%s@%d" % ($t.pred$, len($t.args$))

  if $t.pred$ is λ operator, or quantifier **then** ▷ *e.g., count*

    Omit variable information defined by $t.pred$

    $t.pred \leftarrow$ "%s#%d" % ($t.pred$, len($variable$))

  for $c \leftarrow$ argument in $t.args$ **do**

    if c is nonterminal **then**

      c ← SKETCH(c)

    else

      c ← "?" ▷ *Placeholder for terminal*

  return t

3.3.1 Natural Language to Logical Form

For our first task we used two benchmark datasets, namely GEO (880 language queries to a database of U.S. geography) and ATIS (5,410 queries to a flight booking system). These two datasets are also employed in Chapter 2. Examples are shown in Table 3.1 (see the first and second block). We used standard splits for both datasets: 600 training and 280 test instances for GEO (Zettlemoyer and Collins, 2005); 4,480 training, 480 development, and 450 test examples for ATIS. Meaning representations in these datasets are based on λ-calculus (Kwiatkowski et al., 2011). We use brackets to linearize the hierarchical structure. The first element between a pair of brackets is an operator or predicate name, and any remaining elements are its arguments. The JOBS dataset (as described in Chapter 2) is not used because its meaning representation (i.e., Prolog-style queries) is different with GEO and ATIS. The proposed model can also be applied to other semantic representations, while we leave it for future work.

Algorithm 2 shows the pseudocode used to extract sketches from λ-calculus-based meaning representations. We strip off arguments and variable names in logical forms, while keeping predicates, operators, and composition information. We use the symbol

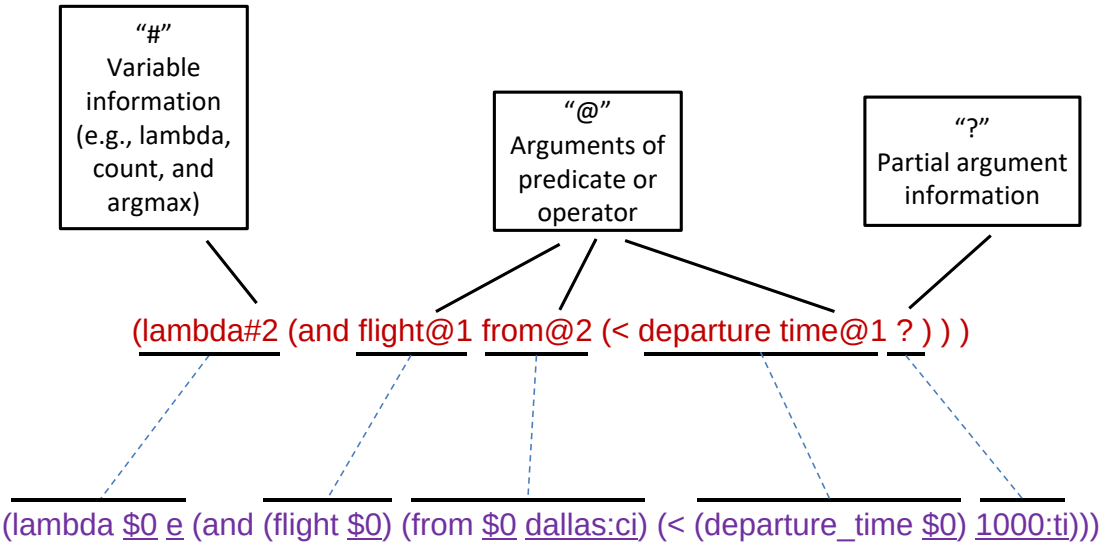


Figure 3.2: Example sketch for ATIS.

"@" to denote the number of missing arguments in a predicate. As shown in Figure 3.2, we extract "*from@2*" from the expression "*(from $0 dallas:ci)*" which indicates that the predicate "*from*" has two arguments. We use "*?*" as a placeholder in cases where only partial argument information can be omitted. We also omit variable information defined by the lambda operator and quantifiers (e.g., *exists*, *count*, and *argmax*). We use the symbol "#" to denote the number of omitted tokens. As shown in Figure 3.1, "*lambda $0 e*" is reduced to "*lambda#2*". For ATIS, about 85.9% of meaning sketches of the development set appear in the training set.

Parent Feeding The meaning representations of these two datasets are highly compositional, which motivates us to utilize the hierarchical structure of λ-calculus. Taking the meaning sketch "*(and flight@1 from@2)*" as an example, the parent of "*from@2*" is "*(and*". Let p_t denote the parent of the t-th time step in the decoder. Compared with Equation (3.10), we use the vector $\mathbf{d}_t^{att}$ and the hidden state of its parent $\mathbf{d}_{p_t}$ to compute the probability $p(a_t|a_{<t},x)$ via:

$$p(a_t|a_{<t},x) = \text{softmax}_{a_t}\left(\mathbf{W}_o[\mathbf{d}_t^{att},\mathbf{d}_{p_t}]+\mathbf{b}_o\right) \tag{3.14}$$

where $[\cdot,\cdot]$ denotes vector concatenation. Parent feeding is used for both generation stages in Equations (3.2)–(3.3) to take advantage of tree decoding. A similar idea is also explored in the tree decoder proposed in Section 2.2.2 and Yin and Neubig (2017) where parent hidden states are fed to the input gate of the LSTM units. On the contrary, parent hidden states serve as input to the softmax classifiers of both fine and

coarse meaning decoders.

3.3.2 Natural Language to Source Code

Our second semantic parsing task used DJANGO (Oda et al., 2015), a dataset built upon the Python code of the Django library. The dataset contains lines of code paired with natural language expressions (see the third block in Table 3.1) and exhibits a variety of use cases, such as iteration, exception handling, and string manipulation. The original split has 16,000 training, 1,000 development, and 1,805 test instances.

We used the built-in lexical scanner of Python[1] to tokenize the code and obtain token types. Sketches were extracted by substituting the original tokens with their token types, except delimiters (e.g., "[", and ":"), operators (e.g., "+", and "*"), and built-in keywords (e.g., "True", and "while"). For instance, the expression "if s[:4].lower() == 'http':" becomes "if NAME [: NUMBER] . NAME ( ) == STRING :", with details about names, values, and strings being omitted. For DJANGO, about 75.1% of meaning sketches of the development set appear in the training set.

Copying Mechanism DJANGO is a diverse dataset, spanning various real-world use cases and as a result models are often faced with out-of-vocabulary (OOV) tokens (e.g., variable names, and numbers) that are unseen during training. We handle OOV tokens with a copying mechanism (Gu et al., 2016; Gulcehre et al., 2016; Jia and Liang, 2016), which allows the fine meaning decoder (Section 3.2.2) to directly copy tokens from the natural language input.

Recall that we use a softmax classifier to predict the probability $p(y_t|y_{<t},x,a)$ over the pre-defined vocabulary. We also learn a copying gate $g_t \in [0,1]$ to decide whether y_t should be copied from the input or generated from the vocabulary. We compute the modified output distribution via:

$$g_t = \text{sigmoid}(\mathbf{w}_g \cdot \mathbf{h}_t + b_g) \tag{3.15}$$

$$\tilde{p}(y_t|y_{<t},x,a) = (1-g_t)p(y_t|y_{<t},x,a) + \mathbb{1}_{[y_t \notin \mathcal{V}_y]} g_t \sum_{k:x_k=y_t} s_{t,k} \tag{3.16}$$

where $\mathbf{w}_g \in \mathbb{R}^n$ and $b_g \in \mathbb{R}$ are parameters, the probability $p(y_t|y_{<t},x,a)$ is described in Section 3.2.2, and the indicator function $\mathbb{1}_{[y_t \notin \mathcal{V}_y]}$ is 1 only if y_t is not in the target vocabulary $\mathcal{V}_y$; the attention score $s_{t,k}$ (see Equation (3.7)) measures how likely it is to copy y_t from the input word x_k.

[1] https://docs.python.org/3/library/tokenize

3.3.3 Natural Language to SQL

The WIKISQL (Zhong et al., 2017) dataset contains 80,654 examples of questions and SQL queries distributed across 24,241 tables from Wikipedia. The goal is to generate the correct SQL query for a natural language question and table schema (i.e., table column names), without using the content values of tables (see the last block in Table 3.1 for an example). The dataset is partitioned into a training set (70%), a development set (10%), and a test set (20%). Each table is present in one split to ensure generalization to unseen tables.

WIKISQL queries follow the format "`SELECT agg_op agg_col WHERE (cond_col cond_op cond) AND ...`", which is a subset of the SQL syntax. `SELECT` identifies the column that is to be included in the results after applying the aggregation operator `agg_op`[2] to column `agg_col`. `WHERE` can have zero or multiple conditions, which means that column `cond_col` must satisfy the constraints expressed by the operator `cond_op`[3] and the condition value `cond`. Sketches for SQL queries are simply the (sorted) sequences of condition operators `cond_op` in `WHERE` clauses. For example, in Table 3.1, sketch "`WHERE > AND =`" has two condition operators, namely "$>$" and "=". All meaning sketches of the development set have been seen in the training set.

The generation of SQL queries differs from our previous semantic parsing tasks, in that the table schema serves as input in addition to natural language. We therefore modify our input encoder in order to render it table-aware, so to speak. Furthermore, due to the formulaic nature of the SQL query, we only use our decoder to generate the `WHERE` clause (with the help of sketches). The `SELECT` clause has a fixed number of slots (i.e., aggregation operator `agg_op` and column `agg_col`), which we straightforwardly predict with softmax classifiers (conditioned on the input). We briefly explain how these components are modeled below.

Table-Aware Input Encoder Given a table schema with M columns, we employ the special token "||" to concatenate its header names as "$\|\|c_{1,1}\cdots c_{1,|c_1|}\|\|\cdots\|\|c_{M,1}\cdots c_{M,|c_M|}\|\|$", where the k-th column ("$c_{k,1}\cdots c_{k,|c_k|}$") has $|c_k|$ words. As shown in Figure 3.3, the first column is "*college*", and the second one is "*number of presidents*". We use bi-directional LSTMs to encode the whole sequence of columns. Next, for column c_k, the LSTM hidden states at positions $c_{k,1}$ and $c_{k,|c_k|}$ are concatenated. Finally, the concatenated vectors are used as the encoding vectors $\{\mathbf{c}_k\}_{k=1}^{M}$ for table columns.

[2] `agg_op` $\in \{$*empty*, `COUNT`, `MIN`, `MAX`, `SUM`, `AVG`$\}$.
[3] `cond_op` $\in \{=, <, >\}$.

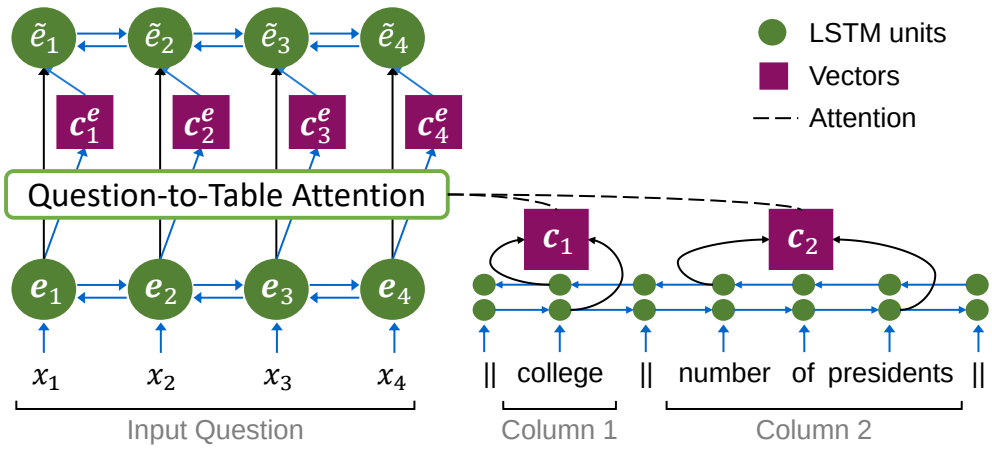


Figure 3.3: Table-aware input encoder (left) and table column encoder (right) used for WIKISQL. At first, we use the bidirectional LSTM encoders to encode the sequences of input and table schema. For each question word, we use an attention mechanism towards table to obtain its relevant column vectors. Finally, we use another bidirectional LSTM to aggregate the question encoding results and their attentional vectors. We then use these new vectors to represent the input question.

As mentioned earlier, the meaning representations of questions are dependent on the tables. As shown in Figure 3.3, we encode the input question x into $\{\mathbf{e}_t\}_{t=1}^{|x|}$ using LSTM units. At each time step t, we use an attention mechanism towards table column vectors $\{\mathbf{c}_k\}_{k=1}^{M}$ to obtain the most relevant columns for $\mathbf{e}_t$. The attention score from $\mathbf{e}_t$ to $\mathbf{c}_k$ is computed via $u_{t,k} \propto \exp\{\alpha(\mathbf{e}_t) \cdot \alpha(\mathbf{c}_k)\}$, where $\alpha(\cdot)$ is a one-layer neural network, and $\sum_{k=1}^{M} u_{t,k} = 1$. Then we compute the context vector $\mathbf{c}_t^e = \sum_{k=1}^{M} u_{t,k}\mathbf{c}_k$ to summarize the relevant columns for $\mathbf{e}_t$. We feed the concatenated vectors $\{[\mathbf{e}_t, \mathbf{c}_t^e]\}_{t=1}^{|x|}$ into a bi-directional LSTM encoder, and use the new encoding vectors $\{\tilde{\mathbf{e}}_t\}_{t=1}^{|x|}$ to replace $\{\mathbf{e}_t\}_{t=1}^{|x|}$ in other model components. We define the vector representation of input x as:

$$\tilde{\mathbf{e}} = [\overrightarrow{\tilde{\mathbf{e}}}_{|x|}, \overleftarrow{\tilde{\mathbf{e}}}_{1}] \tag{3.17}$$

analogously to Equations (3.4)–(3.6).

`SELECT` Clause We feed the question vector $\tilde{\mathbf{e}}$ into a softmax classifier to obtain the aggregation operator `agg_op`. If `agg_col` is the k-th table column, its probability is computed via:

$$\sigma(\mathbf{x}) = \mathbf{w}_3 \cdot \tanh(\mathbf{W}_4\mathbf{x} + \mathbf{b}_4) \tag{3.18}$$

$$p(\texttt{agg\_col} = k|x) \propto \exp\{\sigma([\tilde{\mathbf{e}}, \mathbf{c}_k])\} \tag{3.19}$$

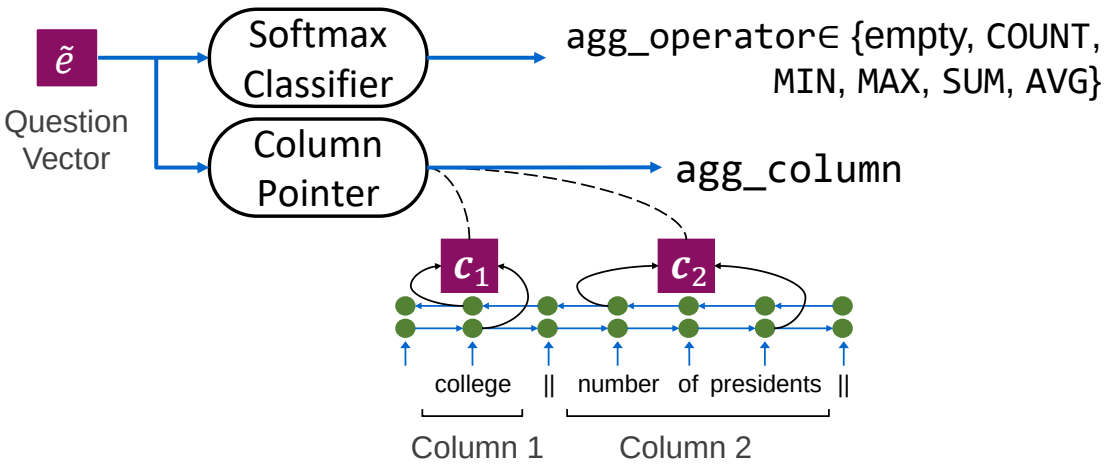


Figure 3.4: Decoder of the SELECT clause used for WIKISQL. We feed the question vector into a softmax classifier to obtain the aggregation operator. The column pointer network is used to compute the matching score between the question vector and table column vectors, which selects the column that is included in the results.

where $\sum_{j=1}^{M} p(\texttt{agg\_col} = j|x) = 1$, $\sigma(\cdot)$ is a scoring network, and $\mathbf{W}_4 \in \mathbb{R}^{2n\times m}, \mathbf{w}_3, \mathbf{b}_4 \in \mathbb{R}^m$ are parameters.

WHERE Clause We first generate sketches whose details are subsequently decorated by the fine meaning decoder described in Section 3.2.2. As the number of sketches in the training set is small (35 in total), we model sketch generation as a classification problem. We treat each sketch a as a category, and use a softmax classifier to compute $p(a|x)$:

$$p(a|x) = \text{softmax}_a(\mathbf{W}_a\tilde{\mathbf{e}} + \mathbf{b}_a) \tag{3.20}$$

where $\mathbf{W}_a \in \mathbb{R}^{|\mathcal{V}_a|\times n}, \mathbf{b}_a \in \mathbb{R}^{|\mathcal{V}_a|}$ are parameters, and $\tilde{\mathbf{e}}$ is the table-aware input representation defined in Equation (3.17).

Once the sketch is predicted, we know the condition operators and number of conditions in the WHERE clause which follows the format "WHERE (cond_op cond_col cond) AND ...". As shown in Figure 3.5, our generation task now amounts to populating the sketch with condition columns cond_col and their values cond.

Let $\{\mathbf{h}_t\}_{t=1}^{|y|}$ denote the LSTM hidden states of the fine meaning decoder, and $\{\mathbf{h}_t^{att}\}_{t=1}^{|y|}$ the vectors obtained by the attention mechanism as in Equation (3.9). The condition column $\texttt{cond\_col}_{y_t}$ is selected from the table's headers. For the k-th column in the table, we compute $p(\texttt{cond\_col}_{y_t} = k|y_{<t}, x, a)$ as in Equation (3.19), but use different parameters and compute the score via $\sigma([\mathbf{h}_t^{att}, \mathbf{c}_k])$. If the k-th table column is

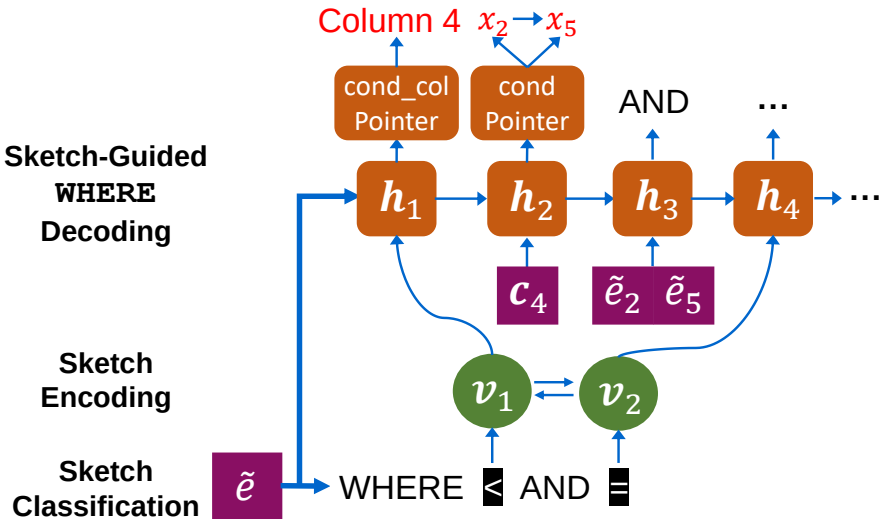


Figure 3.5: Fine meaning decoder of the WHERE clause used for WIKISQL.

selected, we use $\mathbf{c}_k$ for the input of the next LSTM unit in the decoder.

Condition values are typically mentioned in the input questions. These values are often phrases with multiple tokens (e.g., *Mikhail Snitko* in Table 3.1). We therefore propose to select a *text span* from input x for each condition value $\texttt{cond}_{y_t}$ rather than copying tokens one by one. Let $x_l \cdots x_r$ denote the text span from which $\texttt{cond}_{y_t}$ is copied. We factorize its probability as:

$$p\left(\texttt{cond}_{y_t} = x_l \cdots x_r | y_{<t}, x, a\right) = p\left([\![l]\!]^L_{y_t} | y_{<t}, x, a\right) p\left([\![r]\!]^R_{y_t} | y_{<t}, x, a, [\![l]\!]^L_{y_t}\right) \tag{3.21}$$

$$p\left([\![l]\!]^L_{y_t} | y_{<t}, x, a\right) \propto \exp\{\sigma([\mathbf{h}^{att}_t, \tilde{\mathbf{e}}_l])\} \tag{3.22}$$

$$p\left([\![r]\!]^R_{y_t} | y_{<t}, x, a, [\![l]\!]^L_{y_t}\right) \propto \exp\{\sigma([\mathbf{h}^{att}_t, \tilde{\mathbf{e}}_l, \tilde{\mathbf{e}}_r])\} \tag{3.23}$$

where $[\![l]\!]^L_{y_t} / [\![r]\!]^R_{y_t}$ represents the first/last copying index of $\texttt{cond}_{y_t}$ is l/r, the probabilities are normalized to 1, and $\sigma(\cdot)$ is the scoring network defined in Equation (3.18). Notice that we use different parameters for the scoring networks $\sigma(\cdot)$. The copied span is represented by the concatenated vector $[\tilde{\mathbf{e}}_l, \tilde{\mathbf{e}}_r]$, which is fed into a one-layer neural network and then used as the input to the next LSTM unit in the decoder.

3.4 Experiments

We present results on the three tasks discussed in Section 3.3. The code and pretrained models are available at https://github.com/donglixp/coarse2fine.

3.4.1 Experimental Setup

Preprocessing For GEO and ATIS, we used the preprocessing process as in Section 2.3, where natural language expressions are lowercased and stemmed with NLTK (Bird et al., 2009), and entity mentions are replaced by numbered markers. We combined predicates and left brackets that indicate hierarchical structures to make meaning representations compact. We employed the preprocessed DJANGO data provided by Yin and Neubig (2017), where input expressions are tokenized by NLTK, and quoted strings in the input are replaced with place holders. WIKISQL was preprocessed by the script provided by Zhong et al. (2017), where inputs were lowercased and tokenized by Stanford CoreNLP (Manning et al., 2014).

Configuration Model hyperparameters were cross-validated on the training set for GEO, and were validated on the development split for the other datasets. Dimensions of hidden vectors and word embeddings were selected from $\{250, 300\}$ and $\{150, 200, 250, 300\}$, respectively. The dropout rate was selected from $\{0.3, 0.5\}$. Label smoothing (Szegedy et al., 2016) was employed for GEO and ATIS. The smoothing parameter was set to 0.1. For WIKISQL, the hidden size of $\sigma(\cdot)$ and $\alpha(\cdot)$ in Equation (3.18) was set to 64. Word embeddings were initialized by GloVe (Pennington et al., 2014), and were shared by table encoder and input encoder in Section 3.3.3. We appended 10-dimensional part-of-speech tag vectors to embeddings of the question words in WIKISQL. The part-of-speech tags were obtained by the spaCy toolkit. We used the RMSProp optimizer (Tieleman and Hinton, 2012) to train the models. The learning rate was selected from $\{0.002, 0.005\}$. The batch size was 200 for WIKISQL, and was 64 for other datasets. Early stopping was used to determine the number of epochs.

Evaluation We use accuracy as the evaluation metric, i.e., the percentage of the examples that are correctly parsed to their gold standard meaning representations. For WIKISQL, we also execute generated SQL queries on their corresponding tables, and report the execution accuracy which is defined as the proportion of correct answers.

3.4.2 Results and Analysis

We compare our model (COARSE2FINE) against several previously published systems as well as various baselines. Specifically, we report results with a model which decodes

Method	GEO	ATIS
ZC07 (Zettlemoyer and Collins, 2007)	86.1	84.6
UBL (Kwiatkowksi et al., 2010)	87.9	71.4
FUBL (Kwiatkowski et al., 2011)	88.6	82.8
GUSP++ (Poon, 2013)	—	83.5
KCAZ13 (Kwiatkowski et al., 2013)	89.0	—
DCS+L (Liang et al., 2013)	87.9	—
TISP (Zhao and Huang, 2015)	88.9	84.2
SEQ2SEQ (Dong and Lapata, 2016)	84.6	84.2
SEQ2TREE (Dong and Lapata, 2016)	87.1	84.6
ASN (Rabinovich et al., 2017)	85.7	85.3
ASN+SUPATT (Rabinovich et al., 2017)	87.1	85.9
SEQ2ACT (Chen et al., 2018)	88.9	85.5
ONESTAGE	85.0	85.3
COARSE2FINE	88.2	87.7
− sketch encoder	87.1	86.9
+ oracle sketch	93.9	95.1

Table 3.2: Accuracies on GEO and ATIS.

meaning representations in one stage (ONESTAGE) without leveraging sketches. We also report the results of several ablation models, i.e., without a sketch encoder and without a table-aware input encoder.

Table 3.2 presents our results on GEO and ATIS. Overall, we observe that our method COARSE2FINE outperforms ONESTAGE. Improvements on both datasets are significant with $p < 0.05$ according to bootstrap hypothesis testing (Efron and Tibshirani, 1994). The results suggest that disentangling high-level from low-level information during decoding is beneficial. The results also show that removing the sketch encoder harms performance since the decoder loses access to additional contextual information. Compared with previous neural models that utilize syntax or grammatical information (SEQ2TREE, ASN, SEQ2ACT; the second block in Table 3.2), our method performs competitively despite the use of relatively simple decoders. As an upper bound, we report model accuracy when gold meaning sketches are given to the fine meaning decoder (+oracle sketch). As can be seen, predicting the sketch correctly

Method	Accuracy
Retrieval System	14.7
Phrasal SMT	31.5
Hierarchical SMT	9.5
SEQ2SEQ+UNK replacement	45.1
SEQ2TREE+UNK replacement	39.4
LPN+COPY (Ling et al., 2016)	62.3
SNM+COPY (Yin and Neubig, 2017)	71.6
ONESTAGE	69.5
COARSE2FINE	74.1
− sketch encoder	72.1
+ oracle sketch	83.0

Table 3.3: DJANGO results. Accuracies in the first and second block are taken from Ling et al. (2016) and Yin and Neubig (2017).

boosts performance. The oracle results also indicate the accuracy of the fine meaning decoder.

Table 3.3 reports results on the dataset DJANGO where we observe similar tendencies. COARSE2FINE outperforms ONESTAGE by a wide margin, which is significant at $p < 0.05$. It is also superior to the best reported result in the literature (SNM+COPY; see the second block in the table). Again we observe that the sketch encoder is beneficial and that there is an 8.9 point difference in accuracy between COARSE2FINE and the oracle.

Results on WIKISQL are shown in Table 3.4. Our model is superior to ONESTAGE. Improvements over the baseline model are significant at $p < 0.05$. COARSE2FINE's accuracies on aggregation `agg_op` and `agg_col` are 90.2% and 92.0%, respectively, which is comparable to SQLNET (Xu et al., 2017). So most gain is obtained by the improved decoder of the `WHERE` clause. We also find that a table-aware input encoder is critical for doing well on this task, since the same question might lead to different SQL queries depending on the table schemas. Consider the question "*how many presidents are graduated from A*". The SQL query over table "||*President*||*College*||" is "`SELECT COUNT`(*President*) `WHERE` (*College* = *A*)", but the query over table "||*College*||*Number of Presidents*||" would be "`SELECT` *Number of Presidents* `WHERE` (*College* = *A*)".

Method	Accuracy	Execution Accuracy
SEQ2SEQ	23.4	35.9
Aug Ptr Network	43.3	53.3
SEQ2SQL (Zhong et al., 2017)	48.3	59.4
SQLNET (Xu et al., 2017)	61.3	68.0
METALEARN (Huang et al., 2018)	62.8	68.0
TYPESQL (Yu et al., 2018)	66.7	73.5
MQAN (McCann et al., 2018)	75.4	81.4
ONESTAGE	68.8	75.9
COARSE2FINE	71.7	78.5
− sketch encoder	70.8	77.7
− table-aware input encoder	68.6	75.6
+ oracle sketch	73.0	79.6

Table 3.4: Evaluation results on WIKISQL. Accuracies of SEQ2SEQ and Aug Ptr Network are taken from Zhong et al. (2017).

Method	GEO	ATIS	DJANGO	WIKISQL
ONESTAGE	85.4	85.9	73.2	95.4
COARSE2FINE	89.3	88.0	77.4	95.9

Table 3.5: Sketch accuracy. For ONESTAGE, sketches are extracted from the meaning representations it generates.

We also examine the predicted sketches themselves in Table 3.5. We compare sketches generated by COARSE2FINE against ONESTAGE. The latter model generates meaning representations without an intermediate sketch generation stage. Nevertheless, we can extract sketches from the output of ONESTAGE following the procedures described in Section 3.3. Sketches produced by COARSE2FINE are more accurate across the board. This is not surprising because our model is trained explicitly to generate compact meaning sketches. Taken together (Tables 3.2–3.4), our results show that better sketches bring accuracy gains on GEO, ATIS, and DJANGO. On WIKISQL, the sketches predicted by COARSE2FINE are marginally better com-

pared with ONESTAGE. Performance improvements on this task are mainly due to the fine meaning decoder. We conjecture that by decomposing decoding into two stages, COARSE2FINE can better match table columns and extract condition values without interference from the prediction of condition operators. Moreover, the sketch provides a canonical order of condition operators, which is beneficial for the decoding process (Vinyals et al., 2016; Xu et al., 2017).

3.5 Summary

In this chapter we presented a coarse-to-fine decoding algorithm for neural semantic parsing. We first generate meaning sketches which abstract away from low-level information such as arguments and variable names and then predict missing details in order to obtain full meaning representations. The proposed framework can be easily adapted to different domains and meaning representations. Experimental results show that coarse-to-fine decoding improves performance across tasks.

The structure-aware decoders model meanings at different levels of granularity. The decomposition of decoding improves performance of both meaning sketches and final predictions. The sketch constrains the fine-grained meaning decoding process, which forces the prediction to conform the sketch. The decoding constraints prune invalid candidates, so that the output well-formedness can be improved.

In addition to enhancing the robustness of decoders, we would like to make the models robust to uncertain examples. In other words, the models should be able to alert users when they are unsure about the predictions. However, neural models tend to always predict some answers even if they could not handle the input example. To make the proposed models less black-box, we present confidence modeling algorithms for neural semantic parsing in the next chapter.

Chapter 4

Confidence Modeling

In previous chapters, we present neural semantic parsing models that map natural language text to a formal meaning representation (e.g., logical forms or SQL queries). However, despite achieving promising results, the neural semantic parsers remain difficult to interpret, acting in most cases as a black box, not providing any information about what made them arrive at a particular decision. In this chapter, we explore ways to estimate and interpret the model's confidence in its predictions, which we argue can provide users with immediate and meaningful feedback regarding uncertain outputs.

An explicit framework for confidence modeling would benefit the development cycle of neural semantic parsers which, contrary to more traditional methods, do not make use of lexicons or templates and as a result the sources of errors and inconsistencies are difficult to trace. Moreover, from the perspective of application, semantic parsing is often used to build natural language interfaces, such as dialogue systems. In this case it is important to know whether the system understands the input queries with high confidence in order to make decisions more reliably. For example, knowing that some of the predictions are uncertain would allow the system to generate clarification questions, prompting users to verify the results before triggering unwanted actions. In addition, the training data used for semantic parsing can be small and noisy, and as a result, models do indeed produce uncertain outputs, which we would like our framework to identify.

A widely-used confidence scoring method is based on posterior probabilities $p(y|x)$ where x is the input and y the model's prediction. For a linear model, this method makes sense: as more positive evidence is gathered, the score becomes larger. Neural models, in contrast, learn a complicated function that often overfits the training data. Posterior probability is effective when making decisions about model output, but is

no longer a good indicator of confidence due in part to the nonlinearity of neural networks (Johansen and Socher, 2017; Guo et al., 2017). This observation motivates us to develop a confidence modeling framework for sequence-to-sequence models. We categorize the causes of uncertainty into three types, namely *model uncertainty*, *data uncertainty*, and *input uncertainty* and design different metrics to characterize them.

We compute these confidence metrics for a given prediction and use them as features in a regression model which is trained on held-out data to fit prediction F1 scores. At test time, the regression model's outputs are used as confidence scores. Our approach does not interfere with the training of the model, and can be thus applied to various architectures, without sacrificing test accuracy. Furthermore, we propose a method based on backpropagation which allows to interpret model behavior by identifying which parts of the input contribute to uncertain predictions.

Experimental results on two semantic parsing datasets (IFTTT, Quirk et al. 2015; and DJANGO, Oda et al. 2015) show that our model is superior to a method based on posterior probability. We also demonstrate that thresholding confidence scores achieves a good trade-off between coverage and accuracy. Moreover, the proposed uncertainty backpropagation method yields results which are qualitatively more interpretable compared to those based on attention scores.

4.1 Related Work

Confidence Estimation Confidence estimation has been studied in the context of a few NLP tasks, such as statistical machine translation (Blatz et al., 2004; Ueffing and Ney, 2005; Soricut and Echihabi, 2010), and question answering (Gondek et al., 2012). To the best of our knowledge, confidence modeling for semantic parsing remains largely unexplored.

A common scheme for modeling uncertainty in neural networks is to place distributions over the network's weights (Denker and Lecun, 1991; MacKay, 1992; Neal, 1996; Blundell et al., 2015; Gan et al., 2017). But the resulting models often contain more parameters, and the training pipeline has to be accordingly changed, which makes these approaches difficult to work with.

Another strand of related work needs minor modifications to the standard training process of neural networks. Gal and Ghahramani (2016) develop a theoretical framework which shows that the use of dropout in neural networks can be interpreted as a Bayesian approximation of Gaussian processes. We adapt their framework so as to rep-

resent uncertainty in encoder-decoder architectures, and extend it by leveraging other metrics to estimate confidence. Li and Gal (2017) propose an approximate inference technique to avoid the uncertainty underestimation of Dropout variational inference. Lakshminarayanan et al. (2017) use ensembles and adversarial training to modify the training pipeline, and investigate how to use them to compute predictive uncertainty estimates.

Recently, Ott et al. (2018) analyze the effects of uncertainty in neural machine translation model fitting and search strategy, which is closely related to our work. The source of uncertainty is categorized to two types in their work, i.e., intrinsic uncertainty and extrinsic uncertainty. Intrinsic uncertainty is due to the existence of several semantically equivalent translations and under-specification, caused by the one-to-many nature of the machine translation task. Extrinsic uncertainty is caused by noise in the training data. The goal is to analyze how uncertainty affects model fitting and candidate searching. In contrast, we aim at quantifying the confidence scores for given examples, which can be used to indicate how likely the predictions are correct. We also develop an uncertainty interpretation model based on estimated scores to provide fine-grained analyses for predictions. We further identify model uncertainty and design various metrics.

Previous work also investigates how to detect out-of-distribution examples for neural models, which can be a cause of uncertainty. Hendrycks and Gimpel (2017) utilize a pre-trained classifier to measure a maximum value of the predictive distribution, and compare this value to a threshold that determines whether the example is out-of-distribution or not. Liang et al. (2018b) enhance the threshold-based method with temperature scaling (Guo et al., 2017) and adding controlled perturbations to the input. DeVries and Taylor (2018) add a confidence estimation branch in the network architecture to detect out-of-distribution examples based on the encoding vectors of the input. Lee et al. (2018) jointly train both generative and classification networks for out-of-distribution detection. The generator produces samples that are most effective to train the classifier, while the classifier is encouraged to assign uniform class probabilities to out-of-distribution examples. In our work, we use the probability of the input computed by a language model and the number of unknown tokens to indicate whether it is out-of-distribution.

Interpretability Another strand of related work concerns the interpretation of neural models. Network gradients can be used to compute how much a unit contributes

to the final score (Baehrens et al., 2010; Li et al., 2016), e.g., in a classification task. This method assumes that the scores of interest are differentiable, whereas uncertainty scores do not satisfy this requirement. Attention scores between the encoder and decoder are often interpreted as alignments and used to analyze model output (Xu et al., 2015; Bahdanau et al., 2015). However, attention scores offer little information when the input and output are loosely aligned as is the case with IFTTT in semantic parsing (Dong and Lapata, 2016), and any interpretation method based on them will be equally ineffective. Besides, the attention component is learned as a part of the model to obtain good predictions, rather than being specifically designed for the purpose of interpretation. Lei et al. (2016) jointly train a generator and an encoder to extract rationales for given input texts. The generator identifies text fragments as candidate rationales which are passed through the encoder to predict the results. Bach et al. (2015), Zhang et al. (2016), Montavon et al. (2017), Ding et al. (2017), and Kindermans et al. (2018) propose to use rules to backpropagate salience scores to input layers. The intuition is to decompose the neuron activation in terms of contributions from its input values. We expand on this idea, defining new propagation rules for encoder-decoder networks that clarify the relation between input tokens and uncertainty scores of predictions. In addition, our goal is to identity the input words that contribute to prediction uncertainty, rather than explaining how the model obtains results.

4.2 Neural Semantic Parsing Model

In the following section we describe the neural semantic parsing model we assume throughout this chapter. The model is built upon the sequence-to-sequence architecture and is illustrated in Figure 4.2. An *encoder* is used to encode natural language input $q = q_1 \cdots q_{|q|}$ into a vector representation, and a *decoder* learns to generate a logical form representation of its meaning $a = a_1 \cdots a_{|a|}$ conditioned on the encoding vectors. The encoder and decoder are two different recurrent neural networks with long short-term memory units (LSTMs; Hochreiter and Schmidhuber 1997) which process tokens sequentially. The model is presented in Section 2.2.1, while we use one-layer LSTMs in this chapter. The proposed confidence modeling methods can also be applied to multi-layer models. Specifically, the probability of generating the whole sequence $p(a|q)$ is factorized as:

$$p(a|q) = \prod_{t=1}^{|a|} p(a_t|a_{<t}, q) \tag{4.1}$$

where $a_{<t} = a_1 \cdots a_{t-1}$.

Let $\mathbf{e}_t \in \mathbb{R}^n$ denote the hidden vector of the encoder at time step t. It is computed via $\mathbf{e}_t = \mathrm{f}_{\mathrm{LSTM}}(\mathbf{e}_{t-1}, \mathbf{q}_t)$, where $\mathrm{f}_{\mathrm{LSTM}}$ refers to the LSTM unit, and $\mathbf{q}_t \in \mathbb{R}^n$ is the word embedding of q_t. Once the tokens of the input sequence are encoded into vectors, $\mathbf{e}_{|q|}$ is used to initialize the hidden states of the first time step in the decoder.

Similarly, the hidden vector of the decoder at time step t is computed by $\mathbf{d}_t = \mathrm{f}_{\mathrm{LSTM}}(\mathbf{d}_{t-1}, \mathbf{a}_{t-1})$, where $\mathbf{a}_{t-1} \in \mathbb{R}^n$ is the word vector of the previously predicted token. Additionally, we use an attention mechanism (Luong et al., 2015a) to utilize relevant encoder-side context. For the current time step t of the decoder, we compute its attention score with the k-th hidden state in the encoder as:

$$r_{t,k} \propto \exp\{\mathbf{d}_t \cdot \mathbf{e}_k\} \tag{4.2}$$

where $\sum_{j=1}^{|q|} r_{t,j} = 1$. The probability of generating a_t is computed via:

$$\mathbf{c}_t = \sum_{k=1}^{|q|} r_{t,k} \mathbf{e}_k \tag{4.3}$$

$$\mathbf{d}_t^{att} = \tanh(\mathbf{W}_1 \mathbf{d}_t + \mathbf{W}_2 \mathbf{c}_t) \tag{4.4}$$

$$p(a_t | a_{<t}, q) = \mathrm{softmax}_{a_t}\left(\mathbf{W}_o \mathbf{d}_t^{att}\right) \tag{4.5}$$

where $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{n \times n}$ and $\mathbf{W}_o \in \mathbb{R}^{|V_a| \times n}$ are three parameter matrices.

The training objective is to maximize the likelihood of the generated meaning representation a given input q, i.e.,

$$\text{maximize} \sum_{(q,a) \in \mathcal{D}} \log p(a|q)$$

where $\mathcal{D}$ represents training pairs. At test time, the model's prediction for input q is obtained via:

$$\hat{a} = \arg\max_{a'} p\left(a' | q\right)$$

where a' represents candidate outputs. Because $p(a|q)$ is factorized as shown in Equation (4.1), we can use beam search to generate tokens one by one rather than iterating over all possible results.

4.3 Confidence Estimation

As shown in Figure 4.1, given input q and its meaning representation a predicted by the semantic parser, the confidence estimation method computes score $s(q,a) \in (0,1)$.

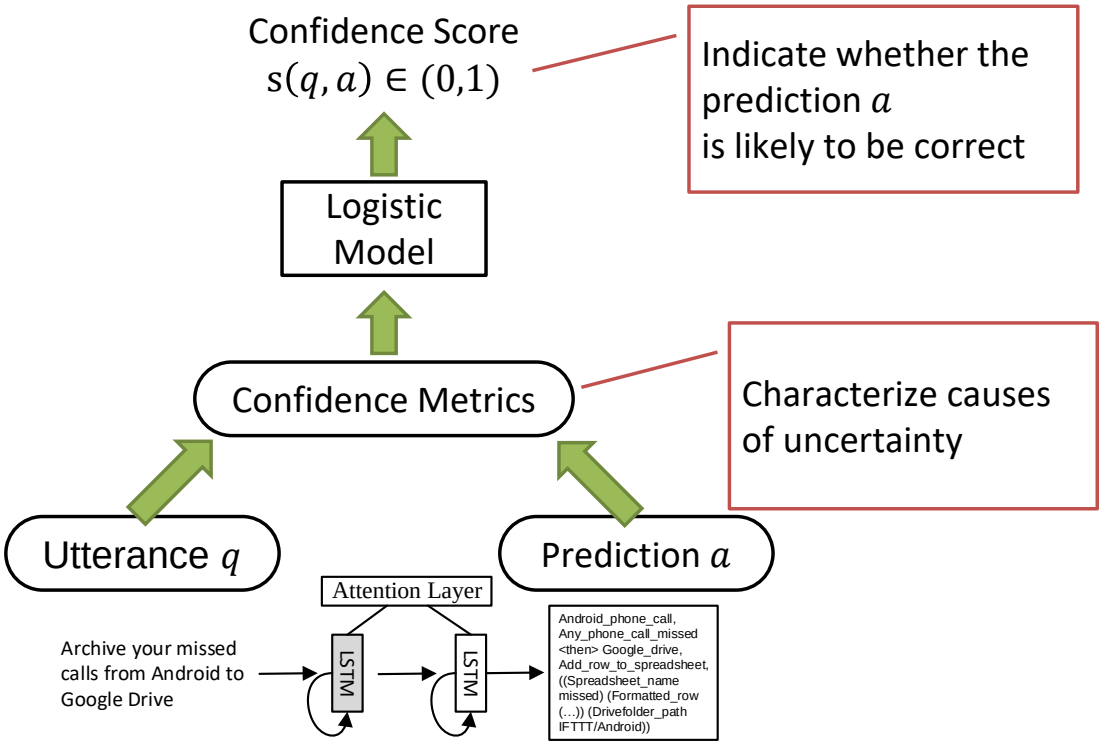


Figure 4.1: Overview of confidence estimation.

A large score indicates the semantic parsing model is confident that its prediction is correct. In order to gauge confidence, we need to estimate "what we do not know". To this end, we identify three causes of uncertainty, and design various metrics characterizing each one of them. We then feed these metrics into a regression model (see upper part in Figure 4.1) in order to predict $s(q,a)$.

4.3.1 Model Uncertainty

The model's parameters or structures contain uncertainty, which makes the model less confident about the values of $p(a|q)$. For example, noise in the training data and the stochastic learning algorithm itself can result in model uncertainty. We describe metrics for capturing uncertainty below:

Dropout Perturbation Our first metric uses dropout (Srivastava et al., 2014) as approximate Bayesian inference to estimate model uncertainty (Gal and Ghahramani, 2016). Dropout is a widely used regularization technique during training, which relieves overfitting by randomly masking some input neurons to zero according to a Bernoulli distribution. In our work, we use dropout at *test time*, instead.

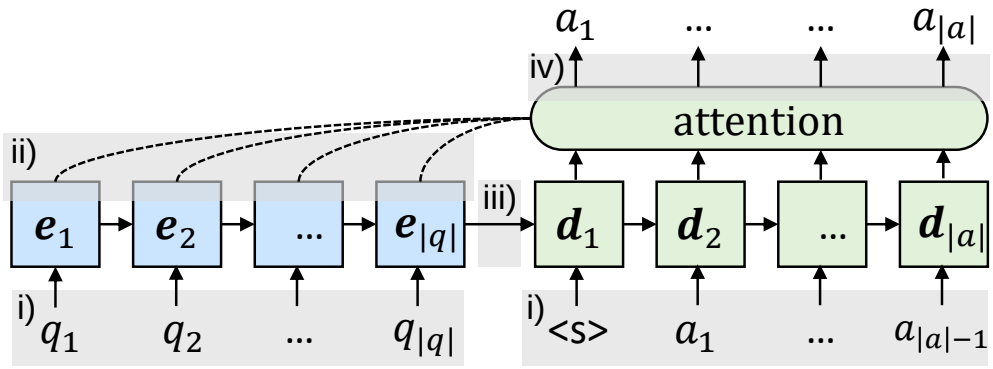


Figure 4.2: We use dropout as approximate Bayesian inference to obtain model uncertainty. The dropout layers are applied to i) token input vectors; ii) the encoder's output vectors; iii) bridge vectors between encoder and decoder; and iv) decoding vectors.

Algorithm 3 Dropout Perturbation

Input: q, a: Input and its prediction

$\mathcal{M}$: Model parameters

▷ *Perform F forward passes through the network*

for $i \leftarrow 1, \cdots, F$ **do**

  ▷ *Get perturbed networks*

  $\hat{\mathcal{M}}^i \leftarrow$ Apply dropout layers to $\mathcal{M}$ as in Figure 4.2

  Run forward pass and compute $\hat{p}(a|q;\hat{\mathcal{M}}^i)$

▷ *Compute the metric as in Equation (4.6)*

return $\text{var}\{\hat{p}(a|q;\hat{\mathcal{M}}^i)\}_{i=1}^F$

As shown in Algorithm 3, we perform F forward passes through the network, and collect the results $\{\hat{p}(a|q;\hat{\mathcal{M}}^i)\}_{i=1}^F$ where $\hat{\mathcal{M}}^i$ represents the perturbed networks. Then, the uncertainty metric is computed by the variance of results. We define the metric on the sequence level as:

$$\text{var}\{\hat{p}(a|q;\hat{\mathcal{M}}^i)\}_{i=1}^F. \tag{4.6}$$

In addition, we compute uncertainty score u_{a_t} for each token a_t via:

$$u_{a_t} = \text{var}\{\hat{p}(a_t|a_{<t},q;\hat{\mathcal{M}}^i)\}_{i=1}^F \tag{4.7}$$

where $\hat{p}(a_t|a_{<t},q;\hat{\mathcal{M}}^i)$ is the probability of generating token a_t (Equation (4.5)) using perturbed model $\hat{\mathcal{M}}^i$. We operationalize token-level uncertainty in two ways, as the average score $\text{avg}\{u_{a_t}\}_{t=1}^{|a|}$ and the maximum score $\max\{u_{a_t}\}_{t=1}^{|a|}$ (since the uncertainty of a sequence is often determined by the most uncertain token).

As shown in Figure 4.2, we add dropout layers in i) the word vectors of the encoder and decoder $\mathbf{q}_t, \mathbf{a}_t$; ii) the output vectors of the encoder $\mathbf{e}_t$; iii) bridge vectors $\mathbf{e}_{|q|}$ used to initialize the hidden states of the first time step in the decoder; and iv) decoding vectors $\mathbf{d}_t^{att}$ (Equation (4.4)).

Gaussian Noise Perturbation Standard dropout can be viewed as applying noise sampled from a Bernoulli distribution to the network parameters. We instead use Gaussian noise, and apply the metrics in the same way discussed above. Let $\mathbf{v}$ denote a vector. The perturbed vectors $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2$ are obtained via:

$$\mathbf{g} \sim \mathcal{N}(0, \sigma^2)$$
$$\hat{\mathbf{v}}_1 = \mathbf{v} + \mathbf{g}$$
$$\hat{\mathbf{v}}_2 = \mathbf{v} + \mathbf{v} \odot \mathbf{g}$$

where $\mathbf{g}$ is a noise vector sampled from the Gaussian distribution $\mathcal{N}(0, \sigma^2)$, and σ is the standard deviation. Intuitively, if the model is more confident in an example, it should be more robust to perturbations.

Posterior Probability Our last class of metrics is based on posterior probability. We use the log probability $\log p(a|q)$ as a sequence-level metric. The token-level metric $\min\{p(a_t|a_{<t}, q)\}_{t=1}^{|a|}$ can identify the most uncertain predicted token. The perplexity per token $-\frac{1}{|a|}\sum_{t=1}^{|a|} \log p\left(a_t|a_{<t}, q\right)$ is also employed.

4.3.2 Data Uncertainty

The coverage of training data also affects the uncertainty of predictions. If the input q does not match the training distribution or contains unknown words, it is difficult to predict $p\left(a|q\right)$ reliably. We define two metrics as follows.

Probability of Input We train a language model on the training data, and use it to estimate the probability of input $p(q|\mathcal{D})$ where $\mathcal{D}$ represents the training data. The probability measures whether the input utterance is an out-of-domain/distribution instance.

Number of Unknown Tokens Tokens that do not appear in the training data harm robustness, and lead to uncertainty. So, we use the number of unknown tokens in the input q as a metric.

4.3.3 Input Uncertainty

Even if the model can estimate $p(a|q)$ reliably, the input itself may be ambiguous. For instance, the input *the flight is at 9 o'clock* can be interpreted as either `flight_time(9am)` or `flight_time(9pm)`. Selecting between these predictions is difficult, especially if they are both highly likely. We use the following metrics to measure uncertainty caused by ambiguous inputs.

Variance of Top Candidates We use the variance of the probabilities of the top candidates to indicate whether these are similar. The sequence-level metric is computed by:

$$\text{var}\{p(a^i|q)\}_{i=1}^{K}$$

where $a^1 \dots a^K$ are the K-best predictions obtained by the beam search during inference (Section 4.2).

Entropy of Decoding The sequence-level entropy of the decoding process is computed via:

$$H[a|q] = -\sum_{a'} p(a'|q) \log p(a'|q)$$

which we approximate by Monte Carlo sampling rather than iterating over all candidate predictions. The token-level metrics of decoding entropy are computed by:

$$\text{avg}\{H[a_t|a_{<t}, q]\}_{t=1}^{|a|}$$

and

$$\max\{H[a_t|a_{<t}, q]\}_{t=1}^{|a|} .$$

4.3.4 Confidence Scoring

The sentence- and token-level confidence metrics defined in Section 4.3 are fed into a gradient tree boosting model (Chen and Guestrin, 2016) in order to predict the overall confidence score $s(q,a)$. The gradient tree boosting model is an ensemble of a set of classification and regression trees which classify instances into different leaves and assign these examples the corresponding scores. The model is wrapped with a logistic function so that confidence scores are in the range of $(0,1)$.

Because the confidence score indicates whether the prediction is likely to be correct, we can use the prediction's F1 (see Section 4.5.2) as target value. The training

loss is defined as:

$$\sum_{(q,a)\in\mathcal{D}} \ln(1+e^{-\hat{s}(q,a)})^{y_{q,a}} + \ln(1+e^{\hat{s}(q,a)})^{(1-y_{q,a})}$$

where $\mathcal{D}$ represents the data, $y_{q,a}$ is the target F1 score, and $\hat{s}(q,a)$ the predicted confidence score. We refer readers to Chen and Guestrin (2016) for mathematical details of how the gradient tree boosting model is trained. In short, second-order Taylor expansion is used to approximate the loss function. A regularization term is also added to control the model complexity (such as the number of leaves of boosted trees). Because it is intractable to learn all the decision trees at once, only one new tree is added at a time while the other trees are kept fixed. The parameters and decision tree structures are trained to optimize the objective function. Notice that we learn the confidence scoring model on the held-out set (rather than on the training data of the semantic parser) to avoid overfitting.

Once we have a scoring model, we use it to estimate confidence scores for new examples. Given the input and its prediction, we first extract the confidence metrics as described in Section 4.3.1–4.3.3. Then these metrics are used as features and fed into the scoring model, which produces the estimated confidence score.

4.4 Uncertainty Interpretation

Confidence scores are useful in so far they can be traced back to the inputs causing the uncertainty in the first place. For semantic parsing, identifying which input words contribute to uncertainty would be of value, e.g., these could be treated explicitly as special cases or refined if they represent noise.

In this section, we introduce an algorithm that backpropagates token-level uncertainty scores (see Equation (4.7)) from predictions to input tokens, following the ideas of Bach et al. (2015) and Zhang et al. (2016). Let u_m denote neuron m's uncertainty score, which indicates the degree to which it contributes to uncertainty. As shown in Figure 4.3, u_m is computed by the summation of the scores backpropagated from its child neurons:

$$u_m = \sum_{c\in\text{Child}(m)} v_m^c u_c$$

where $\text{Child}(m)$ is the set of m's child neurons, and the non-negative contribution ratio v_m^c indicates how much we backpropagate u_c to neuron m. Intuitively, if neuron m contributes more to c's value, ratio v_m^c should be larger.

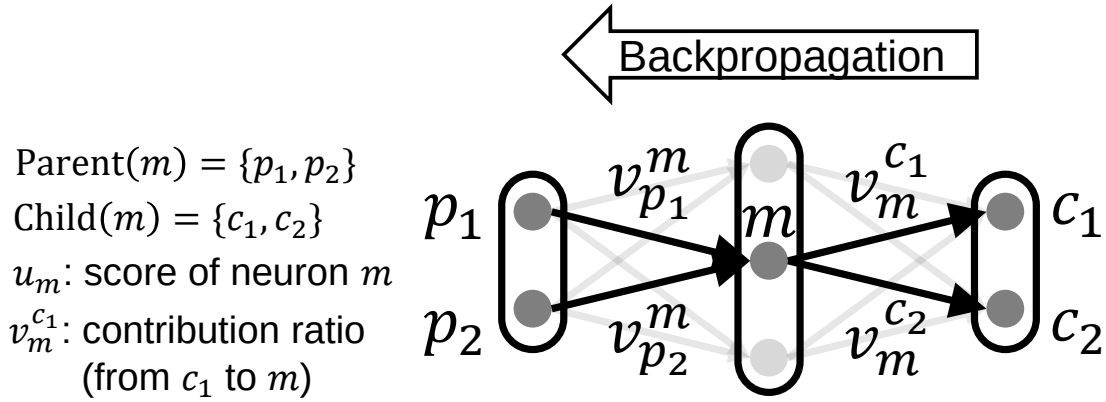


Figure 4.3: Uncertainty backpropagation at the neuron level. Neuron m's score u_m is collected from child neurons c_1 and c_2 by $u_m = v_m^{c_1} u_{c_1} + v_m^{c_2} u_{c_2}$. The score u_m is then redistributed to its parent neurons p_1 and p_2, which satisfies $v_{p_1}^m + v_{p_2}^m = 1$.

After obtaining score u_m, we redistribute it to its parent neurons in the same way. Contribution ratios from m to its parent neurons are normalized to 1:

$$\sum_{p \in \text{Parent}(m)} v_p^m = 1$$

where $\text{Parent}(m)$ is the set of m's parent neurons.

Given the above constraints, we now define different backpropagation rules for the operators used in neural networks. We first describe the rules used for fully-connected layers. Let $\mathbf{x}$ denote the input. The output is computed by $\mathbf{z} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$, where σ is a nonlinear function, $\mathbf{W} \in \mathbb{R}^{|\mathbf{z}| * |\mathbf{x}|}$ is the weight matrix, $\mathbf{b} \in \mathbb{R}^{|\mathbf{z}|}$ is the bias, and neuron $\mathbf{z}_i$ is computed via $\mathbf{z}_i = \sigma(\sum_{j=1}^{|\mathbf{x}|} \mathbf{W}_{i,j}\mathbf{x}_j + \mathbf{b}_i)$. Neuron $\mathbf{x}_k$'s uncertainty score u_{x_k} is gathered from the next layer:

$$\begin{aligned} u_{x_k} &= \sum_{i=1}^{|\mathbf{z}|} v_{x_k}^{z_i} u_{z_i} \\ &= \sum_{i=1}^{|\mathbf{z}|} \frac{|\mathbf{W}_{i,k}\mathbf{x}_k|}{\sum_{j=1}^{|\mathbf{x}|} |\mathbf{W}_{i,j}\mathbf{x}_j|} u_{z_i} \end{aligned}$$

ignoring the nonlinear function σ and the bias $\mathbf{b}$. The ratio $v_{x_k}^{z_i}$ is proportional to the contribution of $\mathbf{x}_k$ to the value of $\mathbf{z}_i$.

We next define backpropagation rules for element-wise vector operators. For $\mathbf{z} = \mathbf{x} \pm \mathbf{y}$, these are:

$$u_{x_k} = \frac{|\mathbf{x}_k|}{|\mathbf{x}_k| + |\mathbf{y}_k|} u_{z_k}$$
$$u_{y_k} = \frac{|\mathbf{y}_k|}{|\mathbf{x}_k| + |\mathbf{y}_k|} u_{z_k}$$

where the contribution ratios $v_{x_k}^{z_k}$ and $v_{y_k}^{z_k}$ are determined by $|\mathbf{x}_k|$ and $|\mathbf{y}_k|$. For multiplication, the contribution of two elements in $\frac{1}{3} * 3$ should be the same. So, the propagation rules for $\mathbf{z} = \mathbf{x} \odot \mathbf{y}$ are:

$$u_{x_k} = \frac{|\log |\mathbf{x}_k||}{|\log |\mathbf{x}_k|| + |\log |\mathbf{y}_k||} u_{z_k}$$
$$u_{y_k} = \frac{|\log |\mathbf{y}_k||}{|\log |\mathbf{x}_k|| + |\log |\mathbf{y}_k||} u_{z_k}$$

where the contribution ratios are determined by $|\log |\mathbf{x}_k||$ and $|\log |\mathbf{y}_k||$. The propagation rule of element-wise multiplication is important to the gating mechanism of LSTM units as shown in Equation (2.3).

For scalar multiplication, $\mathbf{z} = \lambda \mathbf{x}$ where λ denotes a constant. We directly assign $\mathbf{z}$'s uncertainty scores to $\mathbf{x}$ and the backpropagation rule is $u_{x_k} = u_{z_k}$.

As shown in Algorithm 4, we first initialize uncertainty backpropagation in the decoder (lines 1–5). For each predicted token a_t, we compute its uncertainty score u_{a_t} as in Equation (4.7). Next, we find the dimension of a_t in the decoder's softmax classifier (Equation (4.5)), and initialize the neuron with the uncertainty score u_{a_t}. We then backpropagate these uncertainty scores through the network (lines 6–9), and finally into the neurons of the input words. We summarize them and compute the token-level scores for interpreting the results (line 10–13). For input word vector $\mathbf{q}_t$, we use the summation of its neuron-level scores as the token-level score:

$$\hat{u}_{q_t} \propto \sum_{c \in \mathbf{q}_t} u_c$$

where $c \in \mathbf{q}_t$ represents the neurons of word vector $\mathbf{q}_t$, and $\sum_{t=1}^{|q|} \hat{u}_{q_t} = 1$. We use the normalized score $\hat{u}_{q_t}$ to indicate token q_t's contribution to prediction uncertainty.

4.5 Experiments

In this section we describe the datasets used in our experiments and various details concerning our models. We also present our experimental results and analysis of model behavior. The code and pretrained models are available at https://github.com/donglixp/confidence.

Algorithm 4 Uncertainty Interpretation

Input: q, a: Input and its prediction

Output: $\{\hat{u}_{q_t}\}_{t=1}^{|q|}$: Interpretation scores for input tokens

Function: TokenUnc: Get token-level uncertainty

▷ *Get token-level uncertainty for predicted tokens*

$\{u_{a_t}\}_{t=1}^{|a|} \leftarrow \mathsf{TokenUnc}(q, a)$

▷ *Initialize uncertainty scores for backpropagation*

for $t \leftarrow 1, \cdots, |a|$ **do**

    Decoder classifier's output neuron $\leftarrow u_{a_t}$

▷ *Run backpropagation*

for $m \leftarrow$ neuron in backward topological order **do**

    ▷ *Gather scores from child neurons*

    $u_m \leftarrow \sum_{c \in \text{Child}(m)} v_m^c u_c$

▷ *Summarize scores for input words*

for $t \leftarrow 1, \cdots, |q|$ **do**

    $u_{q_t} \leftarrow \sum_{c \in \mathbf{q}_t} u_c$

$\{\hat{u}_{q_t}\}_{t=1}^{|q|} \leftarrow$ normalize $\{u_{q_t}\}_{t=1}^{|q|}$

Dataset	Example

IFTTT	*turn android phone to full volume at 7am monday to friday*
	`date_time−every_day_of_the_week_at−((time_of_day (07)(:)(00))`
	`(days_of_the_week (1)(2)(3)(4)(5))) THEN android_device−`
	`set_ringtone_volume−(volume ({volume_level':1.0,'name':'100%'}))`

DJANGO	*for every key in sorted list of user_settings* `for key in sorted(user_settings):`

Table 4.1: Natural language descriptions and their meaning representations from IFTTT and DJANGO.

4.5.1 Datasets

We trained the neural semantic parser introduced in Section 4.2 on two datasets covering different domains and meaning representations. Examples are shown in Table 4.1. IFTTT and DJANGO are also used in Section 2.3 and Section 3.3.2, respectively. The datasets contain uncertainty because of the method of data collection and the ambiguity

of examples. The natural language descriptions and meaning representations of IFTTT are written by end users from the IFTTT website[1], which renders the dataset noisy. The other dataset DJANGO contains pairs of pseudocodes and Python statements, which is annotated by software engineers. Because of the flexibility of Python language, the same natural language expression can have various implementations. For example, the pseudocode "*summation of variables a and b*" can be mapped into both `a+b` and `sum((a,b))` in Python.

IFTTT This dataset (Quirk et al., 2015) contains a large number of if-this-then-that programs. The programs are paired with natural language descriptions and are written for various applications, such as home security (e.g., "*email me if the window opens*"), and task automation (e.g., "*save instagram photos to dropbox*"). Whenever a program's trigger is satisfied, an action is performed. Triggers and actions represent functions with arguments; they are selected from different channels (160 in total) representing various services (e.g., Android). There are 552 trigger functions and 229 action functions. The original split contains 77,495 training, 5,171 development, and 4,294 test instances. The subset that removes non-English descriptions was used in our experiments.

DJANGO This dataset (Oda et al., 2015) is built upon the code of the Django web framework. Each line of Python code has a manually annotated natural language description. Our goal is to map the English pseudocode to Python statements. This dataset contains diverse use cases, such as iteration, exception handling, and string manipulation. The original split has 16,000 training, 1,000 development, and 1,805 test examples.

4.5.2 Settings

We followed the data preprocessing used in previous work (Dong and Lapata, 2016; Yin and Neubig, 2017). Input sentences were tokenized using NLTK (Bird et al., 2009) and lowercased. We filtered words that appeared less than four times in the training set. Numbers and URLs in IFTTT and quoted strings in DJANGO were replaced with place holders. Hyperparameters of the semantic parsers were validated on the development set. The learning rate and the smoothing constant of RMSProp (Tieleman and Hinton,

[1] http://ifttt.com

2012) were 0.002 and 0.95, respectively. The dropout rate was 0.25. A two-layer LSTM was used for IFTTT, while a one-layer LSTM was employed for DJANGO. Dimensions for the word embedding and hidden vector were selected from $\{150, 250\}$. The beam size during decoding was 5.

For IFTTT, we view the predicted trees as a set of productions, and use balanced F1 as evaluation metric (Quirk et al., 2015). We do not measure accuracy because the dataset is very noisy and there rarely is an exact match between the predicted output and the gold standard. The F1 score of our neural semantic parser is 50.1%, which is comparable to Dong and Lapata (2016). For DJANGO, we measure the fraction of exact matches, where F1 score is equal to accuracy. Because there are unseen variable names at test time, we use attention scores as alignments to replace unknown tokens in the prediction with the input words they align to (Luong et al., 2015b). The accuracy of our parser is 53.7%, which is better than the result (45.1%) of the sequence-to-sequence model reported in Yin and Neubig (2017).

To estimate model uncertainty, we set dropout rate to 0.1, and performed 30 inference passes. The standard deviation of Gaussian noise was 0.05. The language model was estimated using KenLM (Heafield et al., 2013). For input uncertainty, we computed variance for the 10-best candidates. The confidence metrics were implemented in batch mode, to take full advantage of GPUs. Hyperparameters of the confidence scoring model were cross-validated. The number of boosted trees was selected from $\{20, 50\}$. The maximum tree depth was selected from $\{3, 4, 5\}$. We set the subsample ratio to 0.8. All other hyperparameters in XGBoost (Chen and Guestrin, 2016) were left with their default values.

4.5.3 Results

Confidence Estimation We compared our approach (CONFIDENCE) against confidence scores based on posterior probability $p(a|q)$ (POSTERIOR). We also report the results of three ablation variants ($-$MODEL, $-$DATA, $-$INPUT) by removing each group of confidence metrics described in Section 4.3. Specifically, $-$MODEL removes the metrics computed by dropout perturbation, Gaussian noise perturbation, and posterior probability. The second variant $-$DATA removes the confidence metrics based on probability of input, and number of unknown tokens. The last variant $-$INPUT does not use metrics computed by variance of top candidates, and entropy of decoding. We measure the relationship between confidence scores and F1 using Spearman's ρ cor-

Method	IFTTT	DJANGO
POSTERIOR	0.477	0.694
CONFIDENCE	**0.625**	**0.793**
– MODEL	0.595	0.759
– DATA	0.610	0.787
– INPUT	0.608	0.785

Table 4.2: Spearman ρ correlation between confidence scores and F1. Best results are shown in **bold**. All correlations are significant at $p < 0.01$.

	F1	Dropout	Noise	Posterior	Perplexity	LM	#UNK	Variance
Dropout	**0.59**							
Noise	**0.59**	0.90						
Posterior	**0.52**	0.84	0.82					
Perplexity	0.48	0.78	0.78	0.89				
LM	0.30	0.26	0.32	0.27	0.25			
#UNK	0.27	0.31	0.33	0.29	0.25	0.32		
Variance	0.49	0.83	0.78	0.88	0.79	0.25	0.27	
Entropy	0.53	0.78	0.78	0.80	0.75	0.27	0.30	0.76

Table 4.3: Correlation matrix for F1 and individual confidence metrics on the IFTTT dataset. All correlations are significant at $p < 0.01$. Best predictors are shown in **bold**. Posterior is short for posterior probability, LM for probability based on a language model, #UNK for number of unknown tokens, and Variance for variance of top candidates.

relation coefficient which varies between -1 and 1 (0 implies there is no correlation). High ρ indicates that the confidence scores are high for correct predictions and low otherwise.

As shown in Table 4.2, our method CONFIDENCE outperforms POSTERIOR by a large margin. The ablation results indicate that model uncertainty plays the most important role among the confidence metrics. In contrast, removing the metrics of data uncertainty affects performance less, because most examples in the datasets are in-domain. Improvements for each group of metrics are significant with $p < 0.05$ according to bootstrap hypothesis testing (Efron and Tibshirani, 1994).

	F1	Dropout	Noise	Posterior	Perplexity	LM	#UNK	Variance
Dropout	**0.76**							
Noise	**0.78**	0.94						
Posterior	**0.73**	0.89	0.90					
Perplexity	0.64	0.80	0.81	0.84				
LM	0.32	0.41	0.40	0.38	0.30			
#UNK	0.27	0.28	0.28	0.26	0.19	0.35		
Variance	0.70	0.87	0.87	0.89	0.87	0.37	0.23	
Entropy	0.72	0.89	0.90	0.92	0.86	0.38	0.26	0.90

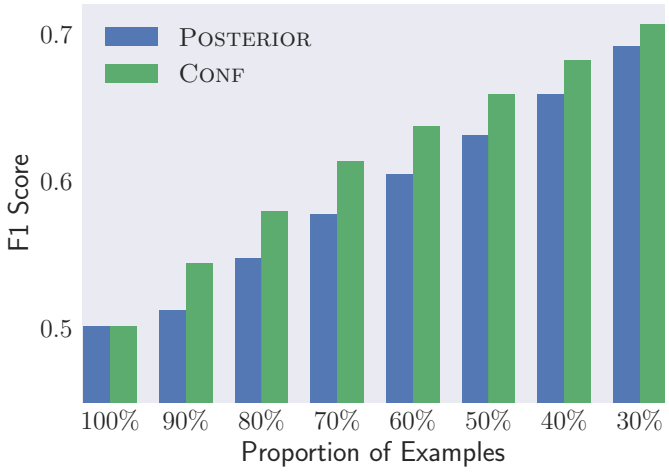
Table 4.4: Correlation matrix for F1 and individual confidence metrics on the DJANGO dataset. All correlations are significant at $p < 0.01$. Best predictors are shown in **bold**. Same shorthands apply as in Table 4.3.

Metric	Dropout	Noise	Posterior	Perplexity	LM	#UNK	Variance	Entropy
IFTTT	0.39	**1.00**	**0.89**	0.27	0.26	0.46	0.43	0.34
DJANGO	**1.00**	**0.59**	0.22	**0.58**	0.49	0.14	0.24	0.25

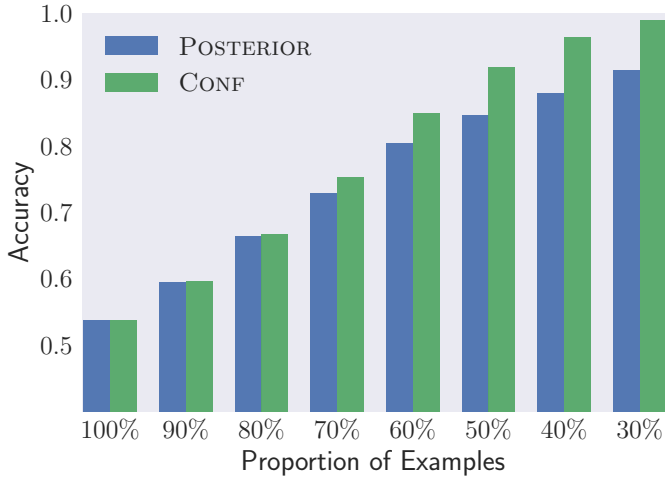
Table 4.5: Importance scores of confidence metrics (normalized by maximum value on each dataset). Best results are shown in **bold**. Same shorthands apply as in Table 4.3.

Tables 4.3–4.4 show the correlation matrix for F1 and individual confidence metrics on the IFTTT and DJANGO datasets, respectively. As can be seen, metrics representing model uncertainty and input uncertainty are more correlated to each other compared with metrics capturing data uncertainty. Perhaps unsurprisingly metrics of the same group are highly inter-correlated since they model the same type of uncertainty. Table 4.5 shows the relative importance of individual metrics in the regression model. As importance score we use the average gain (i.e., loss reduction) brought by the confidence metric once added as feature to the branch of the decision tree (Chen and Guestrin, 2016). The results indicate that model uncertainty (Noise/Dropout/Posterior/Perplexity) plays the most important role. On IFTTT, the number of unknown tokens (#UNK) and the variance of top candidates (var(K-best)) are also very helpful because this dataset is relatively noisy and contains many ambiguous inputs.

Finally, in real-world applications, confidence scores are often used as a threshold



(a) IFTTT



(b) DJANGO

Figure 4.4: Confidence scores are used as threshold to filter out uncertain test examples. As the threshold increases, performance improves. The horizontal axis shows the proportion of examples beyond the threshold.

to trade-off precision for coverage. Figure 4.4 shows how F1 score varies as we increase the confidence threshold, i.e., reduce the proportion of examples that we return answers for. F1 score improves monotonically for POSTERIOR and our method, which, however, achieves better performance when coverage is the same.

Uncertainty Interpretation We next evaluate how our backpropagation method (see Section 4.4) allows us to identify input tokens contributing to uncertainty.

We compare against a method that interprets uncertainty based on the attention mechanism (ATTENTION). As shown in Equation (4.2), attention scores $r_{t,k}$ can be

used as soft alignments between the time step t of the decoder and the k-th input token. We compute the normalized uncertainty score $\hat{u}_{q_t}$ for a token q_t via:

$$\hat{u}_{q_t} \propto \sum_{t=1}^{|a|} r_{t,k} u_{a_t} \tag{4.8}$$

where u_{a_t} is the uncertainty score of the predicted token a_t (Equation (4.7)), and $\sum_{t=1}^{|q|} \hat{u}_{q_t} = 1$.

Unfortunately, the evaluation of uncertainty interpretation methods is problematic. For our semantic parsing task, we do not a priori know which tokens in the natural language input contribute to uncertainty and these may vary depending on the architecture used, model parameters, and so on. We work around this problem by creating a proxy gold standard. We inject noise to the vectors representing tokens in the encoder (see Section 4.3.1) and then estimate the uncertainty caused by each token q_t (Equation (4.6)) under the assumption that addition of noise should only affect genuinely uncertain tokens. Notice that here we inject noise to one token at a time[2] instead of all parameters (see Figure 4.2). Tokens identified as uncertain by the above procedure are considered gold standard and compared to those identified by our method. We use Gaussian noise to perturb vectors in our experiments (dropout obtained similar results).

We define an evaluation metric based on the overlap ($overlap@K$) among tokens identified as uncertain by the model and the gold standard. Given an example, we first compute the interpretation scores of the input tokens according to our method, and obtain a list τ_1 of K tokens with highest scores. We also obtain a list τ_2 of K tokens with highest ground-truth scores and measure the degree of overlap between these two lists:

$$\text{overlap@}K = \frac{|\tau_1 \cap \tau_2|}{K}$$

where $K \in \{2,4\}$ in our experiments. For example, the overlap@4 metric of the lists $\tau_1 = [q_7, q_8, q_2, q_3]$ and $\tau_2 = [q_7, q_8, q_3, q_4]$ is $3/4$, because there are three overlapping tokens.

Table 4.6 reports results with overlap@2 and overlap@4. Overall, BACKPROP achieves better interpretation quality than the attention mechanism. On both datasets, about 80% of the top-4 tokens identified as uncertain agree with the ground truth. The evaluation results demonstrate that the backpropagation algorithm is relatively good at identifying the most important words that cause prediction uncertainty.

[2]Noise injection is used for evaluation purposes only since we need to perform forward passes multiple times (see Section 4.3.1) for each token, and the running time increases linearly with the input length.

Method	IFTTT		DJANGO	
	@2	@4	@2	@4
ATTENTION	0.525	0.737	0.637	0.684
BACKPROP	**0.608**	**0.791**	**0.770**	**0.788**

Table 4.6: Uncertainty interpretation against inferred ground truth; we compute the overlap between tokens identified as contributing to uncertainty by our method and those found in the gold standard. Overlap is shown for top 2 and 4 tokens. Best results are in **bold**.

```
google_calendar—any_event_starts THEN facebook—create_a_status_message
    —(status_message({description}))
```
ATT post calendar event to facebook

BP post calendar event to facebook

```
feed—new_feed_item—(feed_url(_url_sports.espn.go.com)) THEN ...
```
ATT espn mlb headline to readability

BP espn mlb headline to readability

```
weather—tomorrow's_low_drops_below—((temperature(0)) (degrees_in(c)))
    THEN ...
```
ATT warn me when it's going to be freezing tomorrow

BP warn me when it's going to be freezing tomorrow

```
if str_number[0] == '_STR_':
```
ATT if first element of str_number equals a string _STR_ .

BP if first element of str_number equals a string _STR_ .

```
start = 0
```
ATT start is an integer 0 .

BP start is an integer 0 .

```
if name.startswith('_STR_'):
```
ATT if name starts with an string _STR_ ,

BP if name starts with an string _STR_ ,

Table 4.7: Uncertainty interpretation for ATTENTION (ATT) and BACKPROP (BP) . The first line in each group is the model prediction. Predicted tokens and input words with large scores are shown in red and blue, respectively.

Table 4.7 shows examples where our method has identified input tokens contributing to the uncertainty of the output. We highlight token a_t if its uncertainty score u_{a_t} is greater than $0.5 * \text{avg}\{u_{a_{t'}}\}_{t'=1}^{|a|}$. The same criterion is used to highlight input words that have larger scores. The results illustrate that the parser tends to be uncertain about tokens which are function arguments (e.g., URLs, and message content), and ambiguous inputs. The examples show that BACKPROP is qualitatively better compared to ATTENTION; attention scores often produce inaccurate alignments while BACKPROP can utilize information flowing through the LSTMs rather than only relying on the attention mechanism. In the first example, the model identifies the trigger function `any_event_starts` as a uncertain token in the prediction. Because both trigger functions `any_new_event_added` and `any_event_starts` are correct for the input utterance. Another uncertain span in the first prediction is `{description}`, as the message content is absent in the input. In the third instance, the model is unsure about the trigger function `tomorrow's_low_drops_below` and the function argument `0`. BACKPROP highlights the input words "*freezing tomorrow*" for uncertainty interpretation, which helps us to quickly verify or post-edit the results.

4.6 Summary

In this chapter we presented a confidence estimation model and an uncertainty interpretation method for neural semantic parsing. We identified three types of uncertainty, and designed various metrics for them. A regression model uses the metrics as features to estimate confidence scores for model predictions. We also proposed a method that allows to interpret uncertainty: by backpropagating and aggregating the uncertainly scores through the neural network, identifying which tokens contribute to uncertain predictions. Experimental results show that our method achieves better performance than competitive baselines on two datasets. Directions for future work are many and varied. The proposed framework could be applied to a variety of tasks (such as machine translation) employing encoder-decoder architectures. We could also utilize the confidence estimation model within an active learning framework for neural semantic parsing (Hwa, 2000; Duong et al., 2018).

The confidence modeling algorithms help the neural semantic parsers to make robust decisions for uncertain predictions, rather than always guessing some outputs. The estimated confidence scores can be used as threshold to avoid unwanted actions. Furthermore, fine-grained uncertainty interpretations provide valuable clues about what is

not learned by neural semantic parsers, which makes the proposed models more interpretable. As indicated by the experimental results, model coverage is one of the important factors influencing prediction uncertainty. The findings motivate us to utilize external resources to handle the many different ways natural language expresses the same information need. In the next chapter, we learn paraphrase models to improve the model robustness to variations in semantically equivalent utterances.

Chapter 5

Query Paraphrasing

As described in Section 1.2, one of the challenges to build a robust natural language interface is model coverage. Due to the limited size of training data, it is challenging to handle the many different ways natural language expresses the same information need. As a result, small variations in semantically equivalent inputs may yield different results. For example, a hypothetical natural language interface must recognize that the questions "*who created microsoft*" and "*who started microsoft*" have the same meaning and that they both convey the `founder` relation in order to obtain the correct answer from a knowledge base. Moreover, in Chapter 4, one of the main causes of uncertainty is defined as data uncertainty, namely, uncertainty of predictions affected by the coverage of training data. If the pattern of input is unseen by the model on training data, it is difficult to predict reliable outputs.

In this chapter, we leverage external resources to rewrite the natural language input during both training and test, so that model coverage can be increased by augmenting the original expression with its variations. We focus on natural language interfaces that are used to automatically answer questions posed in human language on any domain or topic, as open-domain question answering (QA) tasks contain more language variations.

Given the great variety of surface forms for semantically equivalent expressions, it should come as no surprise that previous work has investigated the use of paraphrases in relation to natural language interfaces. There have been three main strands of research. The first one applies paraphrasing to match natural language and logical forms in the context of semantic parsing. Berant and Liang (2014) use a template-based method to heuristically generate canonical text descriptions for candidate logical forms, and then compute paraphrase scores between the generated texts and input ques-

tions in order to rank the logical forms. Another strand of work uses paraphrases in the context of neural question answering models (Bordes et al., 2014a,b; Dong et al., 2015b). These models are typically trained on question-answer pairs, and employ question paraphrases in a multi-task learning framework in an attempt to encourage the neural networks to output similar vector representations for the paraphrases.

The third strand of research uses paraphrases more directly. The idea is to paraphrase the question and then submit the rewritten version to a QA module. Various resources have been used to produce question paraphrases, such as rule-based machine translation (Duboue and Chu-Carroll, 2006), lexical and phrasal rules from the Paraphrase Database (Narayan et al., 2016), as well as rules mined from Wiktionary (Chen et al., 2016) and large-scale paraphrase corpora (Fader et al., 2013). A common problem with the generated paraphrases is that they often contain inappropriate candidates. Hence, treating all paraphrases as equally felicitous and using them to answer the question could degrade performance. To remedy this, a scoring model is often employed, however independently of the QA system used to find the answer (Duboue and Chu-Carroll, 2006; Narayan et al., 2016). Problematically, the separate paraphrase models used in previous work do not fully utilize the supervision signal from the training data, and as such cannot be properly tuned to the question answering tasks at hand. Based on the large variety of possible transformations that can generate paraphrases, it seems likely that the kinds of paraphrases that are useful would depend on the QA application of interest (Bhagat and Hovy, 2013). Fader et al. (2014) use features that are defined over the original question and its rewrites to score paraphrases. Examples include the pointwise mutual information of the rewrite rule, the paraphrase's score according to a language model, and POS tag features. In the context of semantic parsing, Chen et al. (2016) also use the ID of the rewrite rule as a feature. However, most of these features are not informative enough to model the quality of question paraphrases, or cannot easily generalize to unseen rewrite rules.

We present a general framework for learning paraphrases for question answering tasks. Given a natural language question as input, our model estimates a probability distribution over candidate answers. We first generate paraphrases for the question, which can be obtained by one or several paraphrasing systems. A neural scoring model predicts the quality of the generated paraphrases, while learning to assign higher weights to those which are more likely to yield correct answers. The paraphrases and the original question are fed into a QA model that predicts a distribution over answers given the question. The entire system is trained end-to-end using question-answer pairs

as a supervision signal. The framework is flexible, it does not rely on specific paraphrase or QA models. In fact, this plug-and-play functionality allows to learn specific paraphrases for different QA tasks and to explore the merits of different paraphrasing models for different applications.

We evaluate our approach on question answering over Freebase and text-based answer sentence selection. We employ a range of paraphrase models based on the Paraphrase Database (PPDB; Pavlick et al. 2015), neural machine translation (Mallinson et al., 2016), and rules mined from the WikiAnswers corpus (Fader et al., 2014). Results on three datasets show that our framework consistently improves performance; it achieves state-of-the-art results on GraphQuestions and competitive performance on two additional benchmark datasets using simple QA models.

5.1 Related Work

The task of automatically generating and acquiring semantic equivalences for natural language expressions has been widely studied in previous work. The key to the problem is to learn surface forms that express the same meaning. In our work, we focus on generating paraphrases for questions.

Apart from relying on dictionaries, manually defined rules, and formal grammars, various methods (Madnani and Dorr, 2010) have been used to generate paraphrases at different levels (e.g., lexical, phrasal, and sentential). For example, statistical machine translation techniques are widely used in the context of "translating" the input to its paraphrases (Quirk et al., 2004; Bannard and Callison-Burch, 2005; Wubben et al., 2010; Zhao et al., 2010).

Recently, neural networks have also been used for the task of paraphrase generation. Sequence-to-sequence models are trained from pairs of paraphrases to map sentences in the learned vector space (Prakash et al., 2016; Cao et al., 2017; Gupta et al., 2018). Moreover, the techniques of neural machine translation and the idea of bilingual pivoting are integrated, which utilizes large-scale bilingual parallel corpora to produce multiple paraphrases for the input sentence (Mallinson et al., 2016; Wieting and Gimpel, 2018). The proposed method can be adapted for wide domains, because it is easier to collect bilingual parallel data compared with pairs of paraphrases. Iyyer et al. (2018) further use a target syntactic form as extra input to generate a paraphrase of the sentence with the desired syntax. They also show that the model robustness to syntactic variation is improved when adversarial paraphrase examples are used for

training data augmentation. In our work, we employ several generators (as described in Section 5.2.1) to rewrite input queries and regard them as candidates.

5.2 Problem Formulation

Let q denote a natural language question, and a its answer. Our aim is to estimate $p(a|q)$, the conditional probability of candidate answers given the question. We decompose $p(a|q)$ as:

$$p(a|q) = \sum_{q' \in H_q \cup \{q\}} \underbrace{p_\psi(a|q')}_{\text{QA Model}} \underbrace{p_\theta(q'|q)}_{\text{Paraphrase Model}} \tag{5.1}$$

where H_q is the set of paraphrases for question q, ψ are the parameters of a QA model, and θ are the parameters of a paraphrase scoring model.

As shown in Figure 5.1, we first generate candidate paraphrases H_q for question q. Then, a neural scoring model predicts the quality of the generated paraphrases, and assigns higher weights to the paraphrases which are more likely to obtain the correct answers. These paraphrases and the original question simultaneously serve as input to a QA model that predicts a distribution over answers for a given question. Finally, the results of these two models are fused to predict the answer. In the following we will explain how $p(q'|q)$ and $p(a|q')$ are estimated.

5.2.1 Paraphrase Generation

As shown in Equation (5.1), the term $p(a|q)$ is the sum over q and its paraphrases H_q. Ideally, we would generate all the paraphrases of q. However, since this set could quickly become intractable, we restrict the number of candidate paraphrases to a given size. In order to increase the coverage and diversity of paraphrases, we employ three methods based on: (1) lexical and phrasal rules from the Paraphrase Database (Pavlick et al., 2015); (2) neural machine translation models (Sutskever et al., 2014; Bahdanau et al., 2015); and (3) paraphrase rules mined from clusters of related questions (Fader et al., 2014). We briefly describe these models below, however, there is nothing inherent in our framework that is specific to these, any other paraphrase generator could be used instead.

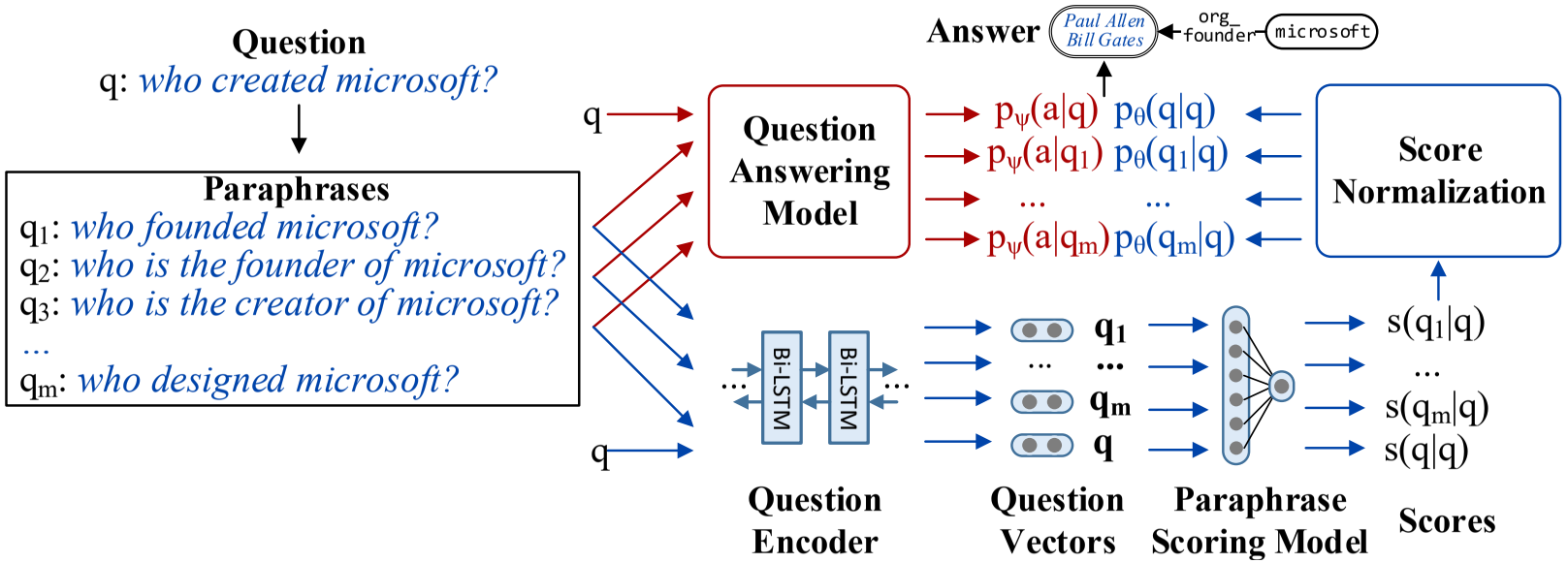


Figure 5.1: We use three different methods to generate candidate paraphrases for input q. The question and its paraphrases are fed into a neural model which scores how suitable they are. The scores are normalized and used to weight the results of the question answering model. The entire system is trained end-to-end using question-answer pairs as a supervision signal.

Input: what be the zip code of the largest car manufacturer	
what be the zip code of the largest vehicle manufacturer	PPDB
what be the zip code of the largest car producer	PPDB
what be the postal code of the biggest automobile manufacturer	NMT
what be the postcode of the biggest car manufacturer	NMT
what be the largest car manufacturer 's postal code	Rule
zip code of the largest car manufacturer	Rule
Input: which country have the largest hi-tech company in europe	
which country have the largest high-technology company in europe	PPDB
which country have the largest high-tech company in europe	PPDB
which country own europe 's biggest high-tech company	NMT
which country have europe 's largest high-tech company	NMT
what european country have the largest hi-tech company	Rule
which country have the biggest hi-tech company in europe	Rule

Table 5.1: Paraphrases obtained for an input question from different models (PPDB, NMT, Rule). Words are lowercased and stemmed.

5.2.1.1 PPDB-based Generation

Bilingual pivoting (Bannard and Callison-Burch, 2005) is one of the most well-known approaches to paraphrasing; it uses bilingual parallel corpora to learn paraphrases based on techniques from phrase-based statistical machine translation (SMT, Koehn et al. 2003). The intuition is that two English strings that translate to the same foreign string can be assumed to have the same meaning. The method first extracts a bilingual phrase table and then obtains English paraphrases by pivoting through foreign language phrases.

Drawing inspiration from syntax-based SMT, Callison-Burch (2008) and Ganitkevitch et al. (2011) extended this idea to syntactic paraphrases, leading to the creation of PPDB (Ganitkevitch et al., 2013), a large-scale paraphrase database containing over a billion of paraphrase pairs in 24 different languages. Pavlick et al. (2015) further used a supervised model to automatically label paraphrase pairs with entailment relationships based on natural logic (MacCartney, 2009). In our work, we employ bidirectionally entailing rules from PPDB. Specifically, we focus on lexical (single word) and

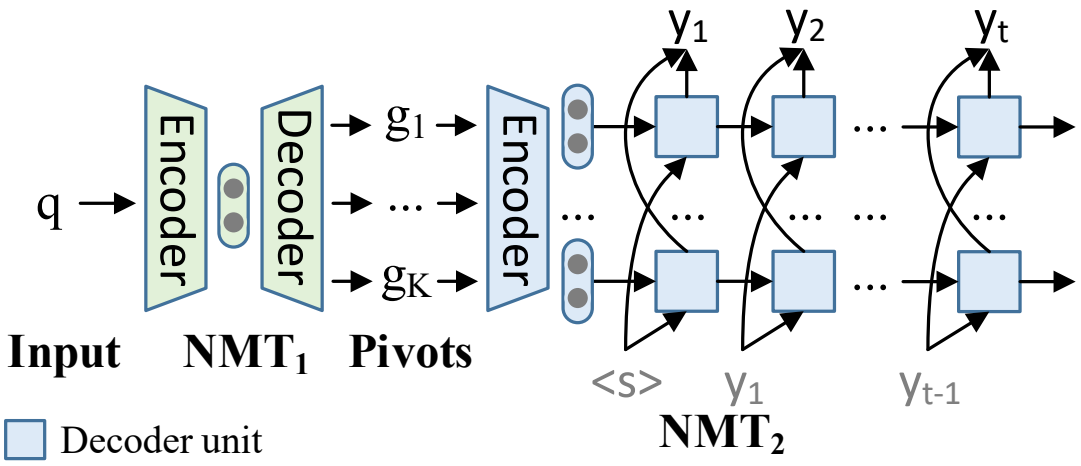


Figure 5.2: Overview of NMT-based paraphrase generation. NMT_1 (green) translates question q into pivots $g_1 \ldots g_K$ which are then back-translated by NMT_2 (blue) where K decoders jointly predict tokens at each time step, rather than only conditioning on one pivot and independently predicting outputs.

phrasal (multiword) rules which we use to paraphrase questions by replacing words and phrases in them. An example is shown in Table 5.1 where we substitute *car* with *vehicle* and *manufacturer* with *producer*.

5.2.1.2 NMT-based Generation

Mallinson et al. (2016) revisit bilingual pivoting in the context of neural machine translation (NMT, Sutskever et al. 2014; Bahdanau et al. 2015) and present a paraphrasing model based on neural networks. At its core, NMT is trained end-to-end to maximize the conditional probability of a correct translation given a source sentence, using a bilingual corpus. Paraphrases can be obtained by translating an English string into a foreign language and then back-translating it into English. NMT-based pivoting models offer advantages over conventional methods such as the ability to learn continuous representations and to consider wider context while paraphrasing.

In our work, we select German as our pivot following Mallinson et al. (2016) who show that it outperforms other languages in a wide range of paraphrasing experiments, and pretrain two NMT systems, English-to-German (EN-DE) and German-to-English (DE-EN). A naive implementation would translate a question to a German string and then back-translate it to English. However, using only one pivot can lead to inaccuracies as it places too much faith on a single translation which may be wrong. Instead, we translate from multiple pivot sentences (Mallinson et al., 2016). As shown in Fig-

Source	Target
what be the zip code of __	what be __ 's postal code
the average size of __	what be __ average size
what be the money in __	what currency do __ use
__ be locate on which continent	what continent be __ a part of
what be some famous place in __	what be some place of interest in __
language speak in __	what be the official language of __
in which state be __ in	what state be __ at
what can be use instead of __	what can you use to substitute __

Table 5.2: Examples of rules used in the rule-based paraphrase generator.

ure 5.2, question q is translated to K-best German pivots, $\mathcal{G}_q = \{g_1, \ldots, g_K\}$. The probability of generating paraphrase $q' = y_1 \ldots y_{|q'|}$ is decomposed as:

$$\begin{aligned} p\left(q'|\mathcal{G}_q\right) &= \prod_{t=1}^{|q'|} p\left(y_t|y_{<t}, \mathcal{G}_q\right) \\ &= \prod_{t=1}^{|q'|} \sum_{k=1}^{K} p\left(g_k|q\right) p\left(y_t|y_{<t}, g_k\right) \end{aligned} \tag{5.2}$$

where $y_{<t} = y_1, \ldots, y_{t-1}$, and $|q'|$ is the length of q'. Probabilities $p\left(g_k|q\right)$ and $p\left(y_t|y_{<t}, g_k\right)$ are computed by the EN-DE and DE-EN models, respectively. We use beam search to decode tokens by conditioning on multiple pivoting sentences. The results with the best decoding scores are considered candidate paraphrases. Examples of NMT paraphrases are shown in Table 5.1.

Compared to PPDB, NMT-based paraphrases are syntax-agnostic, operating on the surface level without knowledge of any underlying grammar. Furthermore, paraphrase rules are captured implicitly and cannot be easily extracted, e.g., from a phrase table. As mentioned earlier, the NMT-based approach has the potential of performing major rewrites as paraphrases are generated while considering wider contextual information, whereas PPDB paraphrases are more local, and mainly handle lexical variation.

5.2.1.3 Rule-Based Generation

Our third paraphrase generation approach uses rules mined from the WikiAnswers corpus (Fader et al., 2014) which contains more than 30 million question clusters labeled

as paraphrases by WikiAnswers[1] users. This corpus is a large resource (the average cluster size is 25), but is relatively noisy due to its collaborative nature – 45% of question pairs are merely related rather than genuine paraphrases. We therefore followed the method proposed in (Fader et al., 2013) to harvest paraphrase rules from the corpus. We first extracted question templates (i.e., questions with at most one wild-card) that appear in at least ten clusters. Any two templates co-occurring (more than five times) in the same cluster and with the same arguments were deemed paraphrases. Table 5.2 shows examples of rules extracted from the corpus.

During paraphrase generation, we consider substrings of the input question as arguments, and match them with the mined template pairs. For example, the stemmed input question in Table 5.1 can be paraphrased using the rules ("*what be the zip code of __*", "*what be __ 's postal code*") and ("*what be the zip code of __*", "*zip code of __*"). If no exact match is found, we perform fuzzy matching by ignoring stop words in the question and templates.

5.2.2 Paraphrase Scoring

Recall from Equation (5.1) that $p_\theta(q'|q)$ scores the generated paraphrases $q' \in H_q \cup \{q\}$. We estimate $p_\theta(q'|q)$ using neural networks given their successful application to paraphrase identification tasks (Socher et al., 2011; Yin and Schütze, 2015; He et al., 2015). As shown in Figure 5.3, the input question and its paraphrases are encoded as vectors. Then, we employ a neural network to obtain the score $s(q'|q)$ which after normalization becomes the probability $p_\theta(q'|q)$.

Encoding Let $q = q_1 \dots q_{|q|}$ denote an input question. Every word is initially mapped to a d-dimensional vector. In other words, vector $\mathbf{q}_t$ is computed via $\mathbf{q}_t = \mathbf{W}_q \mathbf{e}(q_t)$, where $\mathbf{W}_q \in \mathbb{R}^{d \times |\mathcal{V}|}$ is a word embedding matrix, $|\mathcal{V}|$ is the vocabulary size, and $\mathbf{e}(q_t)$ is a one-hot vector. Next, we use a bi-directional recurrent neural network with long short-term memory units (LSTM, Hochreiter and Schmidhuber 1997) as the question encoder, which is shared by the input questions and their paraphrases. The encoder recursively processes tokens one by one, and uses the encoded vectors to represent questions. We compute the hidden vectors at the t-th time step via:

$$\begin{aligned} \overrightarrow{\mathbf{h}}_t &= \mathrm{f}_{\mathrm{LSTM}}\left(\overrightarrow{\mathbf{h}}_{t-1}, \mathbf{q}_t\right), t = 1, \dots, |q| \\ \overleftarrow{\mathbf{h}}_t &= \mathrm{f}_{\mathrm{LSTM}}\left(\overleftarrow{\mathbf{h}}_{t+1}, \mathbf{q}_t\right), t = |q|, \dots, 1 \end{aligned} \tag{5.3}$$

[1] http://wiki.answers.com

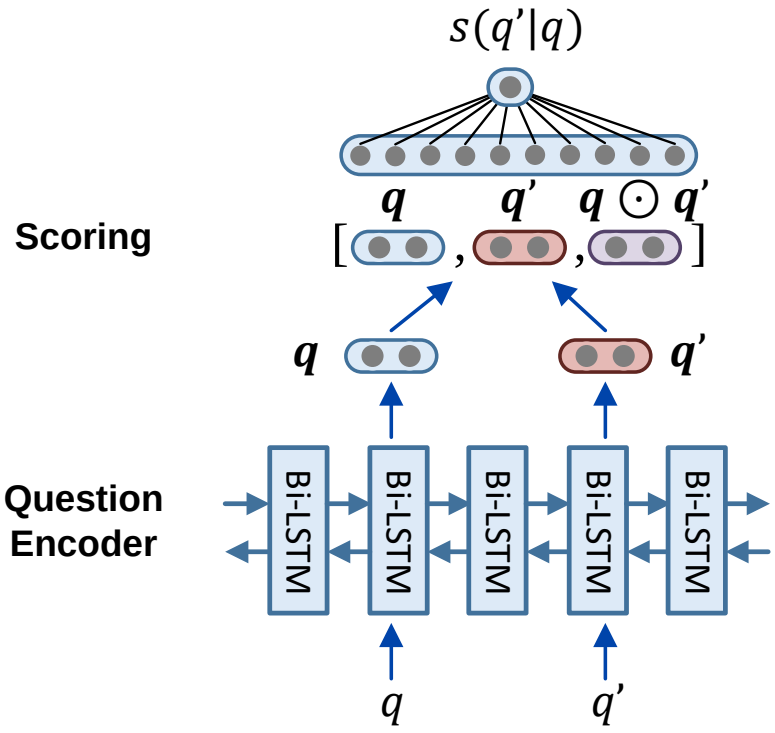


Figure 5.3: Overview of paraphrase scoring model. A bidirectional LSTM is used to encode the question q and the generated paraphrase $q' \in H_q \cup \{q\}$. We then employ a neural network to obtain the score $s(q'|q)$ for paraphrase identification.

where $\overrightarrow{\mathbf{h}}_t, \overleftarrow{\mathbf{h}}_t \in \mathbb{R}^n$. We use the $\mathrm{f}_{\mathrm{LSTM}}$ function as described in Equation (2.3). The representation of q is obtained by:

$$\mathbf{q} = \left[\overrightarrow{\mathbf{h}}_{|q|}, \overleftarrow{\mathbf{h}}_1\right] \tag{5.4}$$

where $[\cdot,\cdot]$ denotes concatenation, and $\mathbf{q} \in \mathbb{R}^{2n}$.

Scoring After obtaining vector representations for q and q', we compute the score $s(q'|q)$ via:

$$s\left(q'|q\right) = \mathbf{w}_s \cdot \left[\mathbf{q}, \mathbf{q}', \mathbf{q} \odot \mathbf{q}'\right] + b_s \tag{5.5}$$

where $\mathbf{w}_s \in \mathbb{R}^{6n}$ is a parameter vector, $[\cdot,\cdot,\cdot]$ denotes concatenation, $\odot$ is element-wise multiplication, and b_s is the bias. Alternative ways to compute $s(q'|q)$ such as dot product or with a bilinear term were not empirically better than Equation (5.5) and we omit them from further discussion.

Normalization For paraphrases $q' \in H_q \cup \{q\}$, the probability $p_\theta(q'|q)$ is computed via:

$$p_\theta(q'|q) = \frac{\exp\{s(q'|q)\}}{\sum_{r \in H_q \cup \{q\}} \exp\{s(r|q)\}} \tag{5.6}$$

where the paraphrase scores are normalized over the set $H_q \cup \{q\}$.

5.2.3 QA Models

The framework defined in Equation (5.1) is relatively flexible with respect to the QA model being employed as long as it can predict $p_\psi(a|q')$. We illustrate this by performing experiments across different tasks and describe below the QA models used for these tasks.

Knowledge Base QA In our first task we use the Freebase knowledge base to answer questions. Query graphs for the questions typically contain more than one predicate. For example, to answer the question "*who is the ceo of microsoft in 2008*", we need to use one relation to query "*ceo of microsoft*" and another relation for the constraint "*in 2008*". For this task, we employ the SIMPLEGRAPH model described in Reddy et al. (2016, 2017), and follow their training protocol and feature design. In brief, their method uses rules to convert questions to ungrounded logical forms, which are subsequently matched against Freebase subgraphs. SIMPLEGRAPH is simple yet effective, and achieves competitive performance compared to state-of-the-art methods (Reddy et al., 2017). The QA model learns from question-answer pairs: it extracts features for pairs of questions and Freebase subgraphs, and uses a logistic regression classifier to predict the probability that a candidate answer is correct. We perform entity linking using the Freebasee/KG API on the original question (Reddy et al., 2016, 2017), and generate candidate Freebase subgraphs. The QA model estimates how likely it is for a subgraph to yield the correct answer.

Answer Sentence Selection Given a question and a collection of relevant sentences, the goal of this task is to select sentences which contain an answer to the question. The assumption is that correct answer sentences have high semantic similarity to the questions (Yu et al., 2014; Yang et al., 2015; Miao et al., 2016). We employ a QA model that is similar to the paraphrase scoring model described in Section 5.2.2. We use two bi-directional recurrent neural networks (BILSTM) to separately encode questions and answer sentences to vectors (Equation (5.4)). Similarity scores are computed as shown

in Equation (5.5), and then squashed to $(0,1)$ by a sigmoid function in order to predict $p_\psi(a|q')$.

5.2.4 Training and Inference

We use a log-likelihood objective for training, which maximizes the likelihood of the correct answer given a question:

$$\max \sum_{(q,a)\in\mathcal{D}} \log p(a|q) \tag{5.7}$$

where $\mathcal{D}$ is the set of all question-answer training pairs. According to Equation (5.1), the objective becomes:

$$\max_{\psi,\theta} \sum_{\substack{(q,a)\in\mathcal{D} \\ q'\in H_q\cup\{q\}}} \log p_\psi\left(a|q'\right) + \log p_\theta\left(q'|q\right) \tag{5.8}$$

where the QA model and the paraphrase scoring model are jointly learned by using question-answer pairs as a supervision signal.

For the knowledge base QA task, we predict how likely it is that a subgraph obtains the correct answer, and the answers of some candidate subgraphs are partially correct. So, we use the binary cross entropy between the candidate subgraph's F1 score and the prediction as the objective function. The RMSProp algorithm (Tieleman and Hinton, 2012) is employed to solve this non-convex optimization problem. Moreover, dropout is used for regularizing the recurrent neural networks (Pham et al., 2014).

At test time, we generate paraphrases for the question q, and then predict the answer by:

$$\hat{a} = \arg\max_{a'\in\mathcal{C}_q} p\left(a'|q\right) \tag{5.9}$$

where $\mathcal{C}_q$ is the set of candidate answers (e.g., knowledge base subgraphs, and answer sentences), and $p(a'|q)$ is computed as shown in Equation (5.1).

5.3 Experiments

We compared our model which we call PARA4QA (as shorthand for learning to paraphrase for question answering) against multiple previous systems on three datasets. In the following we introduce these datasets, provide implementation details for our model, describe the systems used for comparison, and present our results.

5.3.1 Datasets

Our model was trained on three datasets, representative of different types of QA tasks. The first two datasets focus on question answering over a structured knowledge base, whereas the third one is specific to answer sentence selection.

WEBQUESTIONS This dataset (Berant et al., 2013) contains 3,778 training instances and 2,032 test instances. Questions were collected by querying the Google Suggest API. A breadth-first search beginning with *wh-* was conducted and the answers were crowd-sourced using Freebase as the backend knowledge base.

GRAPHQUESTIONS The dataset (Su et al., 2016) contains 5,166 question-answer pairs (evenly split into a training and a test set). It was created by asking crowd workers to paraphrase 500 Freebase graph queries in natural language.

WIKIQA This dataset (Yang et al., 2015) has 3,047 questions sampled from Bing query logs. The questions are associated with 29,258 candidate answer sentences, 1,473 of which contain the correct answers to the questions.

5.3.2 Implementation Details

Paraphrase Generation Candidate paraphrases were stemmed (Minnen et al., 2001) and lowercased. We discarded duplicate or trivial paraphrases which only rewrite stop words or punctuation.

For the NMT model, we followed the implementation[2] and settings described in Mallinson et al. (2016), and used English↔German as the language pair. The system was trained on data released as part of the WMT15 shared translation task (4.2 million sentence pairs). We also had access to back-translated monolingual training data (Sennrich et al., 2016a). Rare words were split into subword units (Sennrich et al., 2016b) to handle out-of-vocabulary words in questions. We used the top 15 decoding results as candidate paraphrases.

In the PPDB-based generator, we used the S size package of PPDB 2.0 (Pavlick et al., 2015) for high precision. For each question, we utilized at most two lexical rules. At most ten candidate paraphrases were considered.

[2] http://github.com/sebastien-j/LV_groundhog

We mined paraphrase rules from WikiAnswers (Fader et al., 2014) as described in Section 5.2.1.3. The extracted rules were ranked using the pointwise mutual information between template pairs in the WikiAnswers corpus. The top ten candidate paraphrases were used.

Training For the paraphrase scoring model, we used GloVe (Pennington et al., 2014) vectors[3] pretrained on Wikipedia 2014 and Gigaword 5 to initialize the word embedding matrix. We kept this matrix fixed across datasets. Out-of-vocabulary words were replaced with a special unknown symbol. We also augmented questions with start-of- and end-of-sequence symbols. Word vectors for these special symbols were updated during training. Model hyperparameters were validated on the development set. The dimensions of hidden vectors and word embeddings were selected from $\{50, 100, 200\}$ and $\{100, 200\}$, respectively. The dropout rate was selected from $\{0.2, 0.3, 0.4\}$. The BILSTM for the answer sentence selection QA model used the same hyperparameters. Parameters were randomly initialized from a uniform distribution $\mathcal{U}(-0.08, 0.08)$. The learning rate and decay rate of RMSProp were 0.01 and 0.95, respectively. The batch size was set to 150. To alleviate the exploding gradient problem (Pascanu et al., 2013), the gradient norm was clipped to 5. Early stopping was used to determine the number of epochs.

5.3.3 Paraphrase Statistics

Table 5.3 presents descriptive statistics on the paraphrases generated by the various systems across datasets (training set). As can be seen, the average paraphrase length is similar to the average length of the original questions. The NMT method generates more paraphrases and has wider coverage, while the average number and coverage of the other two methods vary per dataset. As a way of quantifying the extent to which rewriting takes place, we report BLEU (Papineni et al., 2002) and TER (Snover et al., 2006) scores between the original questions and their paraphrases. The NMT method and the rules extracted from WikiAnswers tend to paraphrase more (i.e., have lower BLEU and higher TER scores) compared to PPDB.

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

Metric	GRAPHQ			WEBQ			WIKIQA		
	NMT	PPDB	Rule	NMT	PPDB	Rule	NMT	PPDB	Rule
avg($\|q\|$)		10.87			7.71			6.47	
avg($\|q'\|$)	10.87	12.40	10.51	8.13	8.55	7.54	6.60	7.85	7.15
avg($\#q'$)	13.85	3.02	2.50	13.76	0.71	7.74	13.95	0.62	5.64
Coverage (%)	99.67	73.52	31.16	99.87	35.15	83.61	99.89	31.04	63.12
BLEU (%)	42.33	67.92	54.23	35.14	56.62	42.37	32.40	54.24	40.62
TER (%)	39.18	14.87	38.59	45.38	19.94	43.44	46.10	17.20	48.59

Table 5.3: Statistics of generated paraphrases across datasets (training set). avg($|q|$): average question length; avg($|q'|$): average paraphrase length; avg($\#q'$): average number of paraphrases; coverage: the proportion of questions that have at least one candidate paraphrase.

5.3.4 Comparison Systems

We compared our framework to previous work and several ablation models which either do not use paraphrases or paraphrase scoring, or are not jointly trained.

The first baseline only uses the base QA models (SIMPLEGRAPH and BILSTM) described in Section 5.2.3. The second baseline (AVGPARA) does not take advantage of paraphrase scoring. The paraphrases for a given question are used while the QA model's results are directly averaged to predict the answers. The third baseline (DATAAUGMENT) employs paraphrases for data augmentation during training. Specifically, we use the question, its paraphrases, and the correct answer to automatically generate new training samples.

In the fourth baseline (SEPPARA), the paraphrase scoring model is separately trained on paraphrase classification data, without taking question-answer pairs into account. In the experiments, we used the Quora question paraphrase dataset[4] which contains question pairs and labels indicating whether they constitute paraphrases or not. We removed questions with more than 25 tokens and sub-sampled to balance the dataset. We used 90% of the resulting 275K examples for training, and the remaining for development. The paraphrase score $s(q'|q)$ (Equation (5.5)) was wrapped by a sigmoid function to predict the probability of a question pair being a paraphrase. A binary cross-entropy loss was used as the objective. The classification accuracy on the dev set

[4] http://goo.gl/kMP46n

was 80.6%.

Finally, in order to assess the individual contribution of different paraphrasing resources, we compared the PARA4QA model against versions of itself with one paraphrase generator removed (−NMT/−PPDB/−RULE).

5.3.5 Results

We first discuss the performance of PARA4QA on GRAPHQUESTIONS and WEBQUESTIONS. The first block in Table 5.4 shows a variety of systems previously described in the literature using average F1 as the evaluation metric (Berant et al., 2013). Among these, PARASEMP, SUBGRAPH, MCCNN, and BILAYERED utilize paraphrasing resources. The second block compares PARA4QA against various related baselines (see Section 5.3.4). SIMPLEGRAPH results on WEBQUESTIONS and GRAPHQUESTIONS are taken from Reddy et al. (2016) and Reddy et al. (2017), respectively.

Overall, we observe that PARA4QA outperforms baselines which either do not employ paraphrases (SIMPLEGRAPH) or paraphrase scoring (AVGPARA, DATAAUGMENT), or are not jointly trained (SEPPARA). Improvements over SIMPLEGRAPH on both datasets are significant with $p < 0.05$ according to bootstrap hypothesis testing (Efron and Tibshirani, 1994). On GRAPHQUESTIONS, our model PARA4QA outperforms the previous state of the art by a wide margin. Ablation experiments with one of the paraphrase generators removed show that performance drops most when the NMT paraphrases are not used on GRAPHQUESTIONS, whereas on WEBQUESTIONS removal of the rule-based generator hurts performance most. One reason is that the rule-based method has higher coverage on WEBQUESTIONS than on GRAPHQUESTIONS (see Table 5.3).

Results on WIKIQA are shown in Table 5.5. We report MAP and MMR which evaluate the relative ranks of correct answers among the candidate sentences for a question. Again, we observe that PARA4QA outperforms related baselines (see BILSTM, DATAAUGMENT, AVGPARA, and SEPPARA). Improvements over BILSTM are significant at $p < 0.1$. Ablation experiments show that performance drops most when NMT paraphrases are removed. When word matching features are used (see +CNT in the third block), PARA4QA reaches state of the art performance.

Examples of paraphrases and their probabilities $p_\theta(q'|q)$ (see Equation (5.6)) learned by PARA4QA are shown in Table 5.6. The two examples are taken from the development set of GRAPHQUESTIONS and WEBQUESTIONS, respectively. We also show the

Method	Average F1 (%)	
	GRAPHQ	WEBQ
SEMPRE (Berant et al., 2013)	10.8	35.7
JACANA (Yao and Van Durme, 2014)	5.1	33.0
PARASEMP (Berant and Liang, 2014)	12.8	39.9
SUBGRAPH (Bordes et al., 2014a)	-	40.4
MCCNN (Dong et al., 2015b)	-	40.8
YAO15 (Yao, 2015)	-	44.3
AGENDAIL (Berant and Liang, 2015)	-	49.7
STAGG (Yih et al., 2015)	-	48.4 (52.5)
MCNN (Xu et al., 2016)	-	47.0 (53.3)
TYPERERANK (Yavuz et al., 2016)	-	**51.6** (52.6)
BILAYERED (Narayan et al., 2016)	-	47.2
UDEPLAMBDA (Reddy et al., 2017)	**17.6**	49.5
SIMPLEGRAPH (baseline)	15.9	48.5
AVGPARA	16.1	48.8
SEPPARA	18.4	49.6
DATAAUGMENT	16.3	48.7
PARA4QA	**20.4**	**50.7**
−NMT	18.5	49.5
−PPDB	19.5	50.4
−RULE	19.4	49.1

Table 5.4: Model performance on GRAPHQUESTIONS and WEBQUESTIONS. Results with additional task-specific resources are shown in parentheses. The base QA model is SIMPLEGRAPH. The ablation models AVGPARA and DATAAUGMENT directly use paraphrases without paraphrase scoring. SEPPARA separately trains a paraphrase scoring model on paraphrase classification data. Best results in each group are shown in **bold**.

Freebase relations used to query the correct answers. In the first example, the original question cannot yield the correct answer because of the mismatch between the question and the knowledge base. The paraphrase contains "*role*" in place of "*sort of part*", increasing the chance of overlap between the question and the predicate words. The second question contains an informal expression "*play 4*", which confuses the QA model.

Method	MAP	MRR
BIGRAMCNN (Yu et al., 2014)	0.6190	0.6281
BIGRAMCNN+CNT (Yu et al., 2014)	0.6520	0.6652
PARAVEC (Le and Mikolov, 2014)	0.5110	0.5160
PARAVEC+CNT (Le and Mikolov, 2014)	0.5976	0.6058
LSTM (Miao et al., 2016)	0.6552	0.6747
LSTM+CNT (Miao et al., 2016)	0.6820	0.6988
NASM (Miao et al., 2016)	0.6705	0.6914
NASM+CNT (Miao et al., 2016)	0.6886	0.7069
KVMEMNET+CNT (Miller et al., 2016)	0.7069	0.7265
COMPAGG (Wang and Jiang, 2017)	**0.7433**	**0.7545**
BILSTM (baseline)	0.6456	0.6608
AVGPARA	0.6587	0.6753
SEPPARA	0.6613	0.6765
DATAAUGMENT	0.6578	0.6736
PARA4QA	**0.6759**	**0.6918**
−NMT	0.6528	0.6680
−PPDB	0.6613	0.6767
−RULE	0.6553	0.6756
BILSTM+CNT (baseline)	0.6722	0.6877
PARA4QA+CNT	**0.6978**	**0.7131**

Table 5.5: Model performance on WIKIQA. +CNT: word matching features introduced in Yang et al. (2015). The base QA model is BILSTM. Same ablation models apply as in Table 5.4. Best results in each group are shown in **bold**.

The paraphrase model generates "*play for*" and predicts a high paraphrase score for it. More generally, we observe that the model tends to give higher probabilities $p_\theta(q'|q)$ to paraphrases biased towards delivering appropriate answers.

We also analyzed which structures were mostly paraphrased within a question. We manually inspected 50 (randomly sampled) questions from the development portion of each dataset, and their three top-scoring paraphrases (Equation (5.5)). We grouped the most commonly paraphrased structures into the following categories: a) question words, i.e., wh-words and "*how*"; b) question focus structures, i.e., cue words or cue phrases for an answer with a specific entity type (Yao and Van Durme, 2014); c) verbs

Examples	$p_\theta(q'\|q)$
(music.concert_performance.performance_role)	
what sort of part do queen play in concert	0.0659
what role do queen play in concert	0.0847
what be the role play by the queen in concert	0.0687
what role do queen play during concert	0.0670
what part do queen play in concert	0.0664
which role do queen play in concert concert	0.0652
(sports.sports_team_roster.team)	
what team do shaq play 4	0.2687
what team do shaq play for	0.2783
which team do shaq play with	0.0671
which team do shaq play out	0.0655
which team have you play shaq	0.0650
what team have we play shaq	0.0497

Table 5.6: Questions and their top-five paraphrases with probabilities learned by the model. The Freebase relations used to query the correct answers are shown in brackets. The original question is underlined. Questions with incorrect predictions are in *red*.

or noun phrases indicating the relation between the question topic entity and the answer; and d) structures requiring aggregation or imposing additional constraints the answer must satisfy (Yih et al., 2015). In the example "*which year did Avatar release in UK*", the question word is "*which*", the question focus is "*year*", the verb is "*release*", and "*in UK*" constrains the answer to a specific location.

Figure 5.4 shows the degree to which different types of structures are paraphrased. As can be seen, most rewrites affect Relation Verb, especially on WEBQUESTIONS. Question Focus, Relation NP, and Constraint & Aggregation are more often rewritten in GRAPHQUESTIONS compared to the other datasets.

Finally, we examined how our method fares on simple versus complex questions. We performed this analysis on GRAPHQUESTIONS as it contains a larger proportion of complex questions. We consider questions that contain a single relation as simple. Complex questions have multiple relations or require aggregation. Table 5.7 shows how our model performs in each group. We observe improvements for both types of

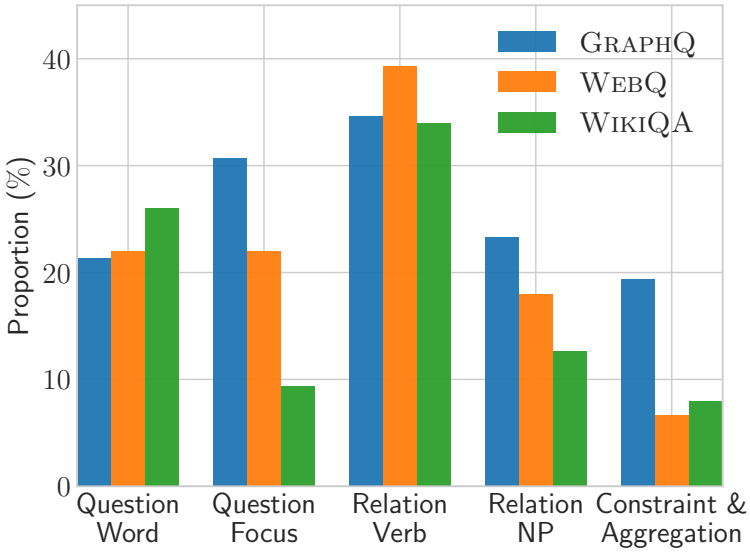


Figure 5.4: Proportion of linguistic phenomena subject to paraphrasing within a question. The results of three datasets are shown in different colors.

Method	**Average F1** (%)	
	Simple	**Complex**
SIMPLEGRAPH	20.9	12.2
PARA4QA	**27.4** (+6.5)	**16.0** (+3.8)

Table 5.7: We group GRAPHQUESTIONS into simple and complex questions and report model performance in each split. Best results in each group are shown in **bold**. The values in brackets are absolute improvements of average F1 scores.

questions, with the impact on simple questions being more pronounced. This is not entirely surprising as it is easier to generate paraphrases and predict the paraphrase scores for simpler questions.

5.4 Summary

In this chapter we proposed a general framework for learning paraphrases for question answering. We employ various paraphrase generators based on the Paraphrase Database, neural machine translation, and rules mined from a QA website. Paraphrase scoring and QA models are trained end-to-end on question-answer pairs, which results in learning paraphrases with a purpose. Experimental results on three datasets show

that our method improves performance across tasks. There are several directions for future work. The framework can be used for the semantic parsing datasets in Chapter 2 and Chapter 3, as Ray et al. (2018) show that paraphrases improve performance of semantic parsers on out-of-vocabulary words and phrases. We would like to explore more advanced paraphrase scoring models (Parikh et al., 2016; Cheng et al., 2016; Wang and Jiang, 2016) as well as additional paraphrase generators since improvements in the diversity and the quality of paraphrases could also enhance performance.

As an important application of natural language interfaces, question answering tasks usually need to handle many language variations, which makes model coverage important for the robustness of QA models. The proposed framework is portable, and not tied to a specific paraphrase generator or QA system. In fact it allows to incorporate several paraphrasing modules, and can serve as a testbed for exploring their coverage and rewriting capabilities. Moreover, we can plug the paraphrase model into the existing natural language interfaces, which enables the systems to handle semantically equivalent expressions.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we aim at developing portable and robust neural models for natural language interfaces that allow users to interact with computers in human language. In particular, the proposed neural semantic parsers are portable across tasks and meaning representations. Our main motivation is that neural networks can be end-to-end trained without manually-designed domain- or representation-specific features, and their strong modeling capability is suitable for learning structural mismatches between natural language and formal language. The modeling paradigm simplifies the development process of natural language interfaces. Furthermore, we enhance the model robustness in the following ways. Firstly, constraints can be added to neural models in order to prune invalid outputs by taking the well-formedness of formal languages into consideration. Secondly, confidence modeling helps models to make robust decisions by estimating and interpreting uncertainties in the predictions. Thirdly, paraphrasing resources can be leveraged to improve the model coverage and its robustness to language variations, so that natural language interfaces can handle different input utterances that express the same intention.

One of the core techniques in natural language interfaces is semantic parsing that maps human language expressions onto machine-interpretable meaning representations. We developed neural semantic parsing models built upon the encoder-decoder architecture in Chapter 2. Given an input utterance, the encoder first encodes it into vector representations. Then the decoder generates the output meaning representation by conditioning on the encoding vectors. The framework is, in principle, simple and flexible, allowsing us to design various architectures for both the encoder and the

decoder. The structural mismatches between natural language and formal language are bridged by neural networks, which enable end-to-end training without employing hand-crafted domain- or representation-specific features. So the models can be easily ported across tasks. Experimental results showed that compared to previous systems our models achieve competitive performance across datasets and meaning representations.

Apart from modeling formal language as sequences, the fact that meaning representations are typically structured objects motivates us to explicitly model the structures of meaning representations. Specifically, we proposed a tree decoder (Chapter 2) and a coarse-to-fine decoding algorithm (Chapter 3) for neural semantic parsing. The proposed tree decoder recursively generates hierarchical structures in a top-down, and left-to-right manner, which ensures the well-formedness of predicated trees. Moreover, we have proposed a method which models meaning at different levels of granularity, and decodes semantic representations from coarse to fine. We first generate a rough meaning sketch that omits low-level details. Then, a second decoder is used to fill in missing details. The sketch constrains the fine meaning generation process and is encoded into vectors to guide decoding. We found that structure-aware neural decoders improve the robustness and performance of natural language interfaces on various tasks.

Although neural models can predict outputs with good accuracy, they are mosty black-box models and remain difficult to interpret as they do not provide any uncertainty information. We further studied confidence modeling for the proposed neural semantic parsing framework in Chapter 4. We explored ways to estimate and interpret the model's confidence in its predictions. We designed various metrics to characterize the causes of uncertainty. The estimated confidence indicates how likely the prediction is correct. The proposed algorithm does not interfere with model training, so that we can apply it to various models without sacrificing performance. We conducted experiments for the neural model proposed in Chapter 2. Evaluation results suggested that our confidence estimator outperforms the approach based on posterior probabilities. We further proposed an uncertainty interpretation algorithm, which interprets model behavior by identifying which parts of the input contribute to uncertain predictions. We backpropagate uncertainty scores from the prediction to the input words at the neuron level. The scores indicate their contributions to the prediction uncertainty. We found that the backpropagation algorithm obtains more accurate uncertainty interpretations compared to using attention scores.

There are usually many different ways natural language expresses the same infor-

mation need. Due to the limited size of training data, model coverage would cause uncertainty in natural language interfaces. In Chapter 5, we present a model for learning paraphrases for natural language interfaces, which help the system handle variations in semantically equivalent questions. Question answering and neural paraphrase scoring models are jointly trained, thereby learning paraphrases with a purpose. As the framework is not restricted to a specific paraphrase generator, we explore three ways (i.e., Paraphrase Database, neural back-translation, and mined rewriting rules) to produce diverse candidate paraphrases. On several question answering datasets, we observed that the proposed paraphrasing model boosts the performance of natural language interfaces.

6.2 Future Work

The models presented in this thesis were developed and evaluated on a single domain, and were trained on English utterances paired with their meaning representations. It is worth studying and exploring how to improve the proposed models' scalability in terms of supporting many different domains, languages, and supervision signals. Avenues for future research about the scalability of natural language interfaces are many and varied. We discuss some promising directions as follows.

Cross-Domain Sharing For real-world applications (such as voice assistants), a system usually needs to handle many different domains. It is helpful to transfer and share knowledge across domains and meaning representations, especially when the data size of each domain is not large enough. We can share the common operators, predicates, and composition structures for similar examples. The idea of using meaning sketches described in Chapter 3 provides a promising approach to share high-level semantics across relevant domains by defining unified meaning representations. So the model can explicitly share knowledge of coarse structures for the examples that have the same sketch (i.e., basic meaning), even though their actual semantic representations are different (e.g., due to different details). Specifically, we can share the same coarse meaning decoder that generates rough meaning sketches, and learns multiple fine meaning decoders to fill in the missing details for various domains.

Zero/Few-Shot Learning The training data size for a new domain is often small or even non-existent. It is valuable to conduct few-shot or zero-shot learning for a

cold start (Bapna et al., 2017; Herzig and Berant, 2018), so that the model can be quickly adapted to a new similar domain. The problem is related to cross-domain sharing. We can also share the composition structures and common operators across domains, which would help boost performance on a new domain. The difference is that there are significantly fewer training examples in this setting. We assume that high-level structures of semantic representations are similar in related domains, while the predicates are different according to the tasks. So our main goal is to enable the model to handle new predicates. Based on the coarse-to-fine decoding framework, the coarse decoder could be used to learn high-level meaning sketches across domains, while a fine decoder would predict predicates and other details. For each predicate, we could use natural language explanations, trigger words, and typed grammars to describe its usage. Then, the fine meaning decoder would learn to match predicates according to their descriptions and the natural language input.

Data Collection Model performance can usually be improved if more annotated data is fed to the model. However, sometimes it is difficult to directly annotate meaning representations for ordinary users. A new paradigm of training data collection is critical for the acceleration of model deployment. A practical solution is to utilize active learning to reduce the amount of required annotations (Hwa, 2000; Duong et al., 2018). With the help of the confidence estimation approach proposed in Chapter 4, we can adaptively select the examples that the current model is least confident about, and annotate these first. The method potentially avoids spending time on instances which the model can handle well. Furthermore, under the coarse-to-fine decoding framework (as described in Chapter 3), we can only annotate the coarse-grained meaning sketches if predicting them is the performance bottleneck, which would accelerate the annotation process. Moreover, we could choose examples that contain as many meaning sketches as possible. So the training dataset would cover broader semantic phenomena. The paraphrasing techniques presented in Chapter 5 could also be used for data collection. Once we have a training dataset, we can then leverage the paraphrase generators to produce paraphrases for these input queries. The data augmentation method can generate multiple surface forms for a desired meaning.

Weakly Supervised Learning The models presented in this thesis were trained on natural language utterances paired with their meaning representations. Given the data paucity of such parallel corpora, we can learn models from weak supervision signals

(e.g., question-answer pairs), which would reduce the annotation burden. Weakly supervised learning also provides a way to utilize online user feedback (Iyer et al., 2017). A typical solution is to search for latent intermediate representations that lead to a correct outcome. Then, the obtained results are used to train the model. One of the challenges is that spurious meaning representations (Pasupat and Liang, 2016; Guu et al., 2017) can deliver the desired answers but the meaning representations might be incorrect. For language understanding tasks, only a few semantic representations are correct among candidates entertained by the model. The parameter learning process is greatly distracted by spurious programs. In order to solve this problem, we can add grammar constraints and inductive biases (e.g., using a structured decoder) in the method, so the learned model prunes spurious meaning representations during the search process. Another method would be to involve a human in the loop, which would enable users to identify the correct candidates when the model is unsure about the results.

Multilingual Semantic Parsing Natural language interfaces should accept multiple languages, so users from different world regions can freely use their native language. A typical problem is that for some languages there is less data compared to others, which inevitably results in inferior semantic parsing performance. Although input utterances are different, the meaning representations obey the same grammar. We can learn cross-lingual word embeddings or share the decoder across languages (Duong et al., 2017; Susanto and Lu, 2017; Zou and Lu, 2018; Richardson et al., 2018). So, systems in different languages can benefit from each other.

Multi-Turn Interactions Sometimes users express their intentions in multiple utterances or update the requests according to the system responses. Users often tend to omit information which has been expressed in the conversation history. So the prediction should be conditioned on the current utterance as well as the interaction history (Long et al., 2016; Iyyer et al., 2017; Suhr et al., 2018). As there is no explicit annotation about dependencies between the current utterance and the history, the model has to automatically learn how to understand the context. The problem is challenging because the combination of multi-turn interactions grows exponentially, which makes model learning data-hungry. Moreover, the model needs to identify co-reference relations within the same dialogue. In addition, error recovery is an interesting topic in the setting. The model should be able to correct previous predictions according to the user's clarification. An interesting idea would be to generate questions and ask users

to clarify their intentions, in cases when the model is uncertain about the predictions according to the results of confidence modeling presented in Chapter 4.

Bibliography

Aho, A. V., Sethi, R., and Ullman, J. D. (2007). *Compilers: principles, techniques, and tools*, volume 2. Addison-wesley Reading.

Alshawi, H. and van Eijck, J. (1989). Logical forms in the core language engine. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 25–32, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Alvarez-Melis, D. and Jaakkola, T. (2017). Tree-structured decoding with doubly-recurrent neural networks. In *Proceedings of the 5th International Conference on Learning Representations*, Toulon, France.

Andreas, J., Vlachos, A., and Clark, S. (2013). Semantic parsing as machine translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 47–52, Sofia, Bulgaria.

Artzi, Y. and Zettlemoyer, L. (2011). Bootstrapping semantic parsers from conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 421–432, Edinburgh, Scotland.

Artzi, Y. and Zettlemoyer, L. (2013). Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association of Computational Linguistics*, 1:49–62.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):1–46.

Baehrens, D., Schroeter, T., Harmeling, S., Kawanabe, M., Hansen, K., and Müller, K.-R. (2010). How to explain individual classification decisions. *Journal of Machine Learning Research*, 11:1803–1831.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations*, San Diego, California.

Ballard, B. W. (1984). The syntax and semantics of user-defined modifiers in transportable natural language processor. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd Annual Meeting of the Association for Computational Linguistics*, pages 52–56, Stanford, California, USA. Association for Computational Linguistics.

Ballard, B. W. and Stumberger, D. E. (1986). Semantic acquisition in TELI: A transportable, user-customized natural language processor. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 20–29, New York, New York, USA. Association for Computational Linguistics.

Bannard, C. and Callison-Burch, C. (2005). Paraphrasing with bilingual parallel corpora. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 597–604, Ann Arbor, Michigan. Association for Computational Linguistics.

Bapna, A., Tür, G., Hakkani-Tür, D., and Heck, L. (2017). Towards zero-shot frame semantic parsing for domain scaling. In *Proceedings of 18th Annual Conference of the International Speech Communication Association*, pages 2476–2480, Stockholm, Sweden.

Beck, D., Haffari, G., and Cohn, T. (2018). Graph-to-sequence learning using gated graph neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 273–283. Association for Computational Linguistics.

Beltagy, I. and Quirk, C. (2016). Improved semantic parsers for if-then statements. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 726–736, Berlin, Germany. Association for Computational Linguistics.

Berant, J., Chou, A., Frostig, R., and Liang, P. (2013). Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington. Association for Computational Linguistics.

Berant, J. and Liang, P. (2014). Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 1415–1425, Baltimore, Maryland. Association for Computational Linguistics.

Berant, J. and Liang, P. (2015). Imitation learning of agenda-based semantic parsers. *Transactions of the Association for Computational Linguistics*, 3:545–558.

Bhagat, R. and Hovy, E. (2013). What is a paraphrase? *Computational Linguistics*, 39(3):463–472.

Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.

Blatz, J., Fitzgerald, E., Foster, G., Gandrabur, S., Goutte, C., Kulesza, A., Sanchis, A., and Ueffing, N. (2004). Confidence estimation for machine translation. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 315–321, Geneva, Switzerland.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning*, pages 1613–1622, Lille, France.

Bobrow, D. G. (1964). *Natural Language Input for a Computer Problem Solving System*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Bobrow, R. J., Resnik, P., and Weischedel, R. M. (1990). Multiple underlying systems: Translating user requests into programs to produce answers. In *Proceedings of the 28th Annual Meeting on Association for Computational Linguistics*, ACL '90, pages 227–234, Stroudsburg, PA, USA. Association for Computational Linguistics.

Bordes, A., Chopra, S., and Weston, J. (2014a). Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 615–620, Doha, Qatar. Association for Computational Linguistics.

Bordes, A., Weston, J., and Usunier, N. (2014b). Open question answering with weakly supervised embedding models. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 8724*, ECML PKDD 2014, pages 165–180, New York, NY, USA.

Bošnjak, M., Rocktäschel, T., Naradowsky, J., and Riedel, S. (2017). Programming with a differentiable forth interpreter. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 547–556, Sydney, Australia.

Brown, P. F., deSouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

Cai, Q. and Yates, A. (2013). Semantic parsing freebase: Towards open-domain semantic parsing. In *2nd Joint Conference on Lexical and Computational Semantics*, pages 328–338, Atlanta, Georgia.

Callison-Burch, C. (2008). Syntactic constraints on paraphrases extracted from parallel corpora. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 196–205, Honolulu, Hawaii. Association for Computational Linguistics.

Cao, Z., Luo, C., Li, W., and Li, S. (2017). Joint copying and restricted generation for paraphrase. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 3152–3158, San Francisco, California.

Charniak, E., Johnson, M., Elsner, M., Austerweil, J., Ellis, D., Haxton, I., Hill, C., Shrivaths, R., Moore, J., Pozar, M., and Vu, T. (2006). Multilevel coarse-to-fine PCFG parsing. In *Proceedings of the Human Language Technology Conference of the NAACL*, pages 168–175, New York, NY.

Chen, B., Sun, L., and Han, X. (2018). Sequence-to-action: End-to-end semantic graph generation for semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 766–777. Association for Computational Linguistics.

Chen, B., Sun, L., Han, X., and An, B. (2016). Sentence rewriting for semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 766–777, Berlin, Germany. Association for Computational Linguistics.

Chen, D. L. and Mooney, R. J. (2011). Learning to interpret natural language navigation instructions from observations. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, pages 859–865, San Francisco, California.

Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, San Francisco, California.

Cheng, J., Dong, L., and Lapata, M. (2016). Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561, Austin, Texas. Association for Computational Linguistics.

Cheng, J., Reddy, S., Saraswat, V., and Lapata, M. (2017). Learning structured natural language representations for semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 44–55. Association for Computational Linguistics.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734, Doha, Qatar.

Chopra, S., Auli, M., and Rush, A. M. (2016). Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98, San Diego, California. Association for Computational Linguistics.

Clarke, J., Goldwasser, D., Chang, M.-W., and Roth, D. (2010). Driving semantic parsing from the world's response. In *Proceedings of the 14th Conference on Computational Natural Language Learning*, pages 18–27, Uppsala, Sweden.

Cohn, T., Hoang, C. D. V., Vymolova, E., Yao, K., Dyer, C., and Haffari, G. (2016). Incorporating structural alignment biases into an attentional neural translation model. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 876–885, San Diego, California.

Damerau, F. J. (1981). Operating statistics for the transformational question answering system. *Computational Linguistics*, 7(1):30–42.

Denker, J. S. and Lecun, Y. (1991). Transforming neural-net output levels to probability distributions. In *Advances in neural information processing systems*, pages 853–859.

DeVries, T. and Taylor, G. W. (2018). Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*.

Ding, Y., Liu, Y., Luan, H., and Sun, M. (2017). Visualizing and understanding neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1150–1159. Association for Computational Linguistics.

Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2015). Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634.

Dong, L., Huang, S., Wei, F., Lapata, M., Zhou, M., and Xu, K. (2017a). Learning to generate product reviews from attributes. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 623–632, Valencia, Spain. Association for Computational Linguistics.

Dong, L. and Lapata, M. (2016). Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 33–43, Berlin, Germany.

Dong, L. and Lapata, M. (2018). Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 731–742. Association for Computational Linguistics.

Dong, L., Mallinson, J., Reddy, S., and Lapata, M. (2017b). Learning to paraphrase for question answering. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 875–886, Copenhagen, Denmark. Association for Computational Linguistics.

Dong, L., Quirk, C., and Lapata, M. (2018). Confidence modeling for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 743–753. Association for Computational Linguistics.

Dong, L., Wei, F., Sun, H., Zhou, M., and Xu, K. (2015a). A hybrid neural model for type classification of entity mentions. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 1243–1249.

Dong, L., Wei, F., Zhou, M., and Xu, K. (2015b). Question answering over freebase with multi-column convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 260–269, Beijing, China. Association for Computational Linguistics.

Duboue, P. and Chu-Carroll, J. (2006). Answering the question you wish they had asked: The impact of paraphrasing for question answering. In *Proceedings of the Human Language Technology Conference of the NAACL*, pages 33–36, New York City, USA. Association for Computational Linguistics.

Duong, L., Afshar, H., Estival, D., Pink, G., Cohen, P., and Johnson, M. (2017). Multilingual semantic parsing and code-switching. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 379–389, Vancouver, Canada. Association for Computational Linguistics.

Duong, L., Afshar, H., Estival, D., Pink, G., Cohen, P., and Johnson, M. (2018). Active learning for deep semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 43–48, Melbourne, Australia. Association for Computational Linguistics.

Efron, B. and Tibshirani, R. J. (1994). *An introduction to the bootstrap*. CRC press.

Fader, A., Zettlemoyer, L., and Etzioni, O. (2013). Paraphrase-driven learning for open question answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 1608–1618, Sofia, Bulgaria. Association for Computational Linguistics.

Fader, A., Zettlemoyer, L., and Etzioni, O. (2014). Open question answering over curated and extracted knowledge bases. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1156–1165, New York, NY, USA.

Fan, X., Monti, E., Mathias, L., and Dreyer, M. (2017). Transfer learning for neural semantic parsing. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 48–56, Vancouver, Canada.

Feng, Y., Martins, R., Wang, Y., Dillig, I., and Reps, T. (2017). Component-based synthesis for complex apis. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 599–612, New York, NY.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA.

Gan, Z., Li, C., Chen, C., Pu, Y., Su, Q., and Carin, L. (2017). Scalable bayesian learning of recurrent neural networks for language modeling. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 321–331, Vancouver, Canada.

Ganitkevitch, J., Callison-Burch, C., Napoles, C., and Van Durme, B. (2011). Learning sentential paraphrases from bilingual parallel corpora for text-to-text generation. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language*

Processing, pages 1168–1179, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Ganitkevitch, J., Van Durme, B., and Callison-Burch, C. (2013). PPDB: The paraphrase database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 758–764, Atlanta, Georgia. Association for Computational Linguistics.

Gaunt, A. L., Brockschmidt, M., Singh, R., Kushman, N., Kohli, P., Taylor, J., and Tarlow, D. (2016). TerpreT: A probabilistic programming language for program induction. In *Proceedings of the 1st Workshop on Neural Abstract Machines & Program Induction*, Barcelona, Spain.

Ge, R. and Mooney, R. J. (2005). A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 9–16, Ann Arbor, Michigan.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1243–1252, International Convention Centre, Sydney, Australia.

Giordani, A. and Moschitti, A. (2010). Semantic mapping between natural language questions and SQL queries via syntactic pairing. In *Proceedings of the 14th International Conference on Applications of Natural Language to Information Systems*, NLDB'09, pages 207–221, Berlin, Heidelberg.

Goldberg, Y. (2017). Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309.

Goldman, O., Latcinnik, V., Nave, E., Globerson, A., and Berant, J. (2018). Weakly supervised semantic parsing with abstract examples. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 1809–1819, Melbourne, Australia. Association for Computational Linguistics.

Goldwasser, D. and Roth, D. (2014). Learning from natural instructions. *Machine Learning*, 94(2):205–232.

Gondek, D. C., Lally, A., Kalyanpur, A., Murdock, J. W., Duboue, P. A., Zhang, L., Pan, Y., Qiu, Z. M., and Welty, C. (2012). A framework for merging and ranking of answers in DeepQA. *IBM Journal of Research and Development*, 56(3.4):14:1–14:12.

Green, Jr., B. F., Wolf, A. K., Chomsky, C., and Laughery, K. (1961). Baseball: An automatic question-answerer. In *Western Joint IRE-AIEE-ACM Computer Conference*, pages 219–224, New York, NY, USA. ACM.

Green, L. E. S., Berkeley, E. C., and Gotlieb, C. (1959). Conversation with a computer. *Computers and Automation*, 8(10):9–11.

Grosz, B. J., Appelt, D. E., Martin, P. A., and Pereira, F. C. N. (1987). TEAM: An experiment in the design of transportable natural-language interfaces. *Artificial Intelligence*, 32(2):173–243.

Gu, J., Lu, Z., Li, H., and Li, V. O. (2016). Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1631–1640, Berlin, Germany.

Gulcehre, C., Ahn, S., Nallapati, R., Zhou, B., and Bengio, Y. (2016). Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 140–149, Berlin, Germany. Association for Computational Linguistics.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330, International Convention Centre, Sydney, Australia.

Gupta, A., Agarwal, A., Singh, P., and Rai, P. (2018). A deep generative framework for paraphrase generation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 5149–5156, New Orleans, Louisiana.

Guu, K., Pasupat, P., Liu, E., and Liang, P. (2017). From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1051–1062. Association for Computational Linguistics.

Hafner, C. D. (1984). Interaction of knowledge sources in a portable natural language interface. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd Annual Meeting of the Association for Computational Linguistics*, pages 57–60, Stanford, California, USA. Association for Computational Linguistics.

He, H., Gimpel, K., and Lin, J. (2015). Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1576–1586, Lisbon, Portugal. Association for Computational Linguistics.

He, Y. and Young, S. (2006). Semantic processing using the hidden vector state model. *Speech Communication*, 48(3-4):262–275.

Heafield, K., Pouzyrevsky, I., Clark, J. H., and Koehn, P. (2013). Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 690–696, Sofia, Bulgaria.

Hendrix, G. G., Sacerdoti, E. D., Sagalowicz, D., and Slocum, J. (1978). Developing a natural language interface to complex data. *ACM Transactions on Database Systems*, 3(2):105–147.

Hendrycks, D. and Gimpel, K. (2017). A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *Proceedings of International Conference on Learning Representations*.

Hermann, K. M., Kočiský, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1693–1701, Cambridge, MA, USA. MIT Press.

Herzig, J. and Berant, J. (2017). Neural semantic parsing over multiple knowledge-bases. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 623–628, Vancouver, Canada.

Herzig, J. and Berant, J. (2018). Decoupling structure and lexicon for zero-shot semantic parsing. *arXiv preprint arXiv:1804.07918*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9:1735–1780.

Huang, P.-S., Wang, C., Singh, R., Yih, W.-t., and He, X. (2018). Natural language to structured query generation via meta-learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 732–738, New Orleans, Louisiana. Association for Computational Linguistics.

Hwa, R. (2000). Sample selection for statistical grammar induction. In *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 45–52, Hong Kong, China. Association for Computational Linguistics.

Iyer, S., Konstas, I., Cheung, A., Krishnamurthy, J., and Zettlemoyer, L. (2017). Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 963–973, Vancouver, Canada.

Iyyer, M., Wieting, J., Gimpel, K., and Zettlemoyer, L. (2018). Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1875–1885, New Orleans, Louisiana. Association for Computational Linguistics.

Iyyer, M., Yih, W.-t., and Chang, M.-W. (2017). Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1821–1831, Vancouver, Canada. Association for Computational Linguistics.

Jean, S., Cho, K., Memisevic, R., and Bengio, Y. (2015). On using very large target vocabulary for neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1–10, Beijing, China.

Jia, R. and Liang, P. (2016). Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 12–22, Berlin, Germany.

Johansen, A. and Socher, R. (2017). Learning when to skim and when to read. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 257–264, Vancouver, Canada.

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, Seattle, Washington.

Karpathy, A. and Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, Boston, Massachusetts.

Kate, R. J. and Mooney, R. J. (2006). Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 913–920, Sydney, Australia.

Kate, R. J., Wong, Y. W., and Mooney, R. J. (2005). Learning to transform natural to formal languages. In *Proceedings of the 20th National Conference on Artificial Intelligence*, pages 1062–1068, Pittsburgh, Pennsylvania.

Kindermans, P.-J., Schütt, K. T., Alber, M., Müller, K.-R., Erhan, D., Kim, B., and Dähne, S. (2018). Learning how to explain neural networks: Patternnet and patternattribution. In *International Conference on Learning Representations*.

Kirsch, R. A. (1964). Computer interpretation of english text and picture patterns. *IEEE Transactions on Electronic Computers*, EC-13(4):363–376.

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54, Sapporo, Japan.

Kočiský, T., Melis, G., Grefenstette, E., Dyer, C., Ling, W., Blunsom, P., and Hermann, K. M. (2016). Semantic parsing with semi-supervised sequential autoencoders. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1078–1087, Austin, Texas.

Krishnamurthy, J., Dasigi, P., and Gardner, M. (2017). Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1517–1527, Copenhagen, Denmark.

Krishnamurthy, J. and Mitchell, T. M. (2012). Weakly supervised training of semantic parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 754–765, Jeju Island, Korea.

Kwiatkowksi, T., Zettlemoyer, L., Goldwater, S., and Steedman, M. (2010). Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1223–1233, Cambridge, MA.

Kwiatkowski, T., Choi, E., Artzi, Y., and Zettlemoyer, L. (2013). Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1545–1556, Seattle, Washington.

Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., and Steedman, M. (2011). Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1512–1523, Edinburgh, Scotland.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413.

Lawrence, C. and Riezler, S. (2018). Improving a neural semantic parser by counterfactual learning from human bandit feedback. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1820–1830, Melbourne, Australia. Association for Computational Linguistics.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *Proceedings of The 31st International Conference on Machine Learning*, pages 1188–1196.

Lee, K., Lee, H., Lee, K., and Shin, J. (2018). Training confidence-calibrated classifiers for detecting out-of-distribution samples. In *International Conference on Learning Representations*.

Lei, T., Barzilay, R., and Jaakkola, T. (2016). Rationalizing neural predictions. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 107–117. Association for Computational Linguistics.

Li, J., Chen, X., Hovy, E., and Jurafsky, D. (2016). Visualizing and understanding neural models in NLP. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 681–691, San Diego, California.

Li, Y. and Gal, Y. (2017). Dropout inference in Bayesian neural networks with alpha-divergences. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine*

Learning Research, pages 2052–2061, International Convention Centre, Sydney, Australia.

Liang, C., Berant, J., Le, Q., Forbus, K. D., and Lao, N. (2017). Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 23–33. Association for Computational Linguistics.

Liang, C., Norouzi, M., Berant, J., Le, Q. V., and Lao, N. (2018a). Memory augmented policy optimization for program synthesis and semantic parsing. In *Advances in Neural Information Processing Systems*, pages 10014–10026.

Liang, P., Jordan, M. I., and Klein, D. (2013). Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2).

Liang, P. and Potts, C. (2015). Bringing machine learning and compositional semantics together. *Annual Reviews of Linguistics*, 1(1):355–376.

Liang, S., Li, Y., and Srikant, R. (2018b). Enhancing the reliability of out-of-distribution image detection in neural networks. In *International Conference on Learning Representations*.

Ling, W., Blunsom, P., Grefenstette, E., Hermann, K. M., Kočiský, T., Wang, F., and Senior, A. (2016). Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 599–609, Berlin, Germany.

Ling, X., Singh, S., and Weld, D. S. (2015). Design challenges for entity linking. *Transactions of the Association for Computational Linguistics*, 3:315–328.

Ling, X. and Weld, D. (2012). Fine-grained entity recognition. In *Proceedings of the 26th Conference on Artificial Intelligence*.

Long, R., Pasupat, P., and Liang, P. (2016). Simpler context-dependent logical forms via model projections. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1456–1465. Association for Computational Linguistics.

Lu, W., Ng, H. T., Lee, W. S., and Zettlemoyer, L. (2008). A generative model for parsing natural language to meaning representations. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 783–792, Honolulu, Hawaii.

Luong, T., Pham, H., and Manning, C. (2015a). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal.

Luong, T., Sutskever, I., Le, Q., Vinyals, O., and Zaremba, W. (2015b). Addressing the rare word problem in neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 11–19, Beijing, China.

MacCartney, B. (2009). *Natural Language Inference*. PhD thesis, Stanford University.

MacKay, D. J. C. (1992). A practical bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472.

Madnani, N. and Dorr, B. J. (2010). Generating phrasal and sentential paraphrases: A survey of data-driven methods. *Computational Linguistics*, 36(3):341–387.

Mallinson, J., Sennrich, R., and Lapata, M. (2016). Paraphrasing revisited with neural machine translation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, Valencia, Spain.

Maltese, G., Bravetti, P., Crépy, H., J. Grainger, B., Herzog, M., and Palou, F. (2001). Combining word- and class-based language models: A comparative study in several languages using automatic and manual word-clustering techniques. In *Seventh European Conference on Speech Communication and Technology*, pages 21–24.

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics System Demonstrations*, pages 55–60, Baltimore, Maryland.

McCann, B., Keskar, N. S., Xiong, C., and Socher, R. (2018). The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*.

Miao, Y., Yu, L., and Blunsom, P. (2016). Neural variational inference for text processing. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1727–1736.

Miller, A., Fisch, A., Dodge, J., Karimi, A.-H., Bordes, A., and Weston, J. (2016). Key-value memory networks for directly reading documents. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1400–1409, Austin, Texas. Association for Computational Linguistics.

Miller, S., Stallard, D., Bobrow, R., and Schwartz, R. (1996). A fully statistical approach to natural language interfaces. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 55–61, Santa Cruz, California.

Minnen, G., Carroll, J., and Pearce, D. (2001). Applied morphological processing of english. *Natural Language Engineering*, 7(3):207–223.

Montavon, G., Lapuschkin, S., Binder, A., Samek, W., and Müller, K.-R. (2017). Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65(C):211–222.

Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pages 246–252. Society for Artificial Intelligence and Statistics.

Murali, V., Qi, L., Chaudhuri, S., and Jermaine, C. (2018). Neural sketch learning for conditional program generation. In *International Conference on Learning Representations*.

Narayan, S., Reddy, S., and Cohen, S. B. (2016). Paraphrase generation from Latent-Variable PCFGs for semantic parsing. In *Proceedings of the 9th International Natural Language Generation Conference*, pages 153–162, Edinburgh, Scotland.

Neal, R. (1996). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

Neelakantan, A., Le, Q. V., and Sutskever, I. (2016). Neural programmer: Inducing latent programs with gradient descent. In *International Conference on Learning Representations*.

Niesler, T. R., Whittaker, E. W. D., and Woodland, P. C. (1998). Comparison of part-of-speech and automatically derived category-based language models for speech recognition. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 177–180 vol.1.

Oda, Y., Fudaba, H., Neubig, G., Hata, H., Sakti, S., Toda, T., and Nakamura, S. (2015). Learning to generate pseudo-code from source code using statistical machine translation. In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering*, pages 574–584, Washington, DC.

Ott, M., Auli, M., Grangier, D., and Ranzato, M. (2018). Analyzing uncertainty in neural machine translation. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3953–3962, Stockholmsmässan, Stockholm Sweden.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Parikh, A., Täckström, O., Das, D., and Uszkoreit, J. (2016). A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255, Austin, Texas. Association for Computational Linguistics.

Parvez, M. R., Chakraborty, S., Ray, B., and Chang, K.-W. (2018). Building language models for text with named entities. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 2373–2383, Melbourne, Australia. Association for Computational Linguistics.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 1310–1318, Atlanta, Georgia.

Pasupat, P. and Liang, P. (2015). Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.

Pasupat, P. and Liang, P. (2016). Inferring logical forms from denotations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 23–32. Association for Computational Linguistics.

Pavlick, E., Rastogi, P., Ganitkevitch, J., Van Durme, B., and Callison-Burch, C. (2015). PPDB 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 425–430, Beijing, China. Association for Computational Linguistics.

Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, Doha, Qatar.

Petrov, S. (2011). *Coarse-to-fine natural language processing*. Springer Science & Business Media.

Pham, V., Bluche, T., Kermorvant, C., and Louradour, J. (2014). Dropout improves recurrent neural networks for handwriting recognition. In *2014 14th International Conference on Frontiers in Handwriting Recognition*, pages 285–290.

Poon, H. (2013). Grounded unsupervised semantic parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 933–943, Sofia, Bulgaria.

Popescu, A.-M., Etzioni, O., and Kautz, H. (2003). Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, pages 149–157, Miami, Florida.

Prakash, A., Hasan, S. A., Lee, K., Datla, V., Qadir, A., Liu, J., and Farri, O. (2016). Neural paraphrase generation with stacked residual lstm networks. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2923–2934, Osaka, Japan.

Quirk, C., Brockett, C., and Dolan, W. (2004). Monolingual machine translation for paraphrase generation. In Lin, D. and Wu, D., editors, *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 142–149, Barcelona, Spain. Association for Computational Linguistics.

Quirk, C., Mooney, R., and Galley, M. (2015). Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 878–888, Beijing, China.

Rabinovich, M., Stern, M., and Klein, D. (2017). Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1139–1149, Vancouver, Canada.

Rather, E. D. and Conklin, E. K. (2007). *Forth Programmer's Handbook*. BookSurge Publishing, 3rd edition.

Ravichander, A., Manzini, T., Grabmair, M., Neubig, G., Francis, J., and Nyberg, E. (2017). How would you say it? eliciting lexically diverse dialogue for supervised semantic parsing. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 374–383, Saarbrücken, Germany. Association for Computational Linguistics.

Ray, A., Shen, Y., and Jin, H. (2018). Learning out-of-vocabulary words in intelligent personal agents. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 4309–4315. International Joint Conferences on Artificial Intelligence Organization.

Reddy, S., Lapata, M., and Steedman, M. (2014). Large-scale semantic parsing without question-answer pairs. *Transactions of the Association of Computational Linguistics*, 2:377–392.

Reddy, S., Täckström, O., Collins, M., Kwiatkowski, T., Das, D., Steedman, M., and Lapata, M. (2016). Transforming dependency structures to logical forms for semantic parsing. *Transactions of the Association for Computational Linguistics*, 4:127–140.

Reddy, S., Täckström, O., Petrov, S., Steedman, M., and Lapata, M. (2017). Universal semantic parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 89–101, Copenhagen, Denmark. Association for Computational Linguistics.

Richardson, K., Berant, J., and Kuhn, J. (2018). Polyglot semantic parsing in apis. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 720–730, New Orleans, Louisiana. Association for Computational Linguistics.

Rush, A. M., Chopra, S., and Weston, J. (2015). A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal.

See, A., Liu, P., and Manning, C. (2017). Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1073–1083. Association for Computational Linguistics.

Sennrich, R., Haddow, B., and Birch, A. (2016a). Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Sennrich, R., Haddow, B., and Birch, A. (2016b). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Serban, I. V., García-Durán, A., Gulcehre, C., Ahn, S., Chandar, S., Courville, A., and Bengio, Y. (2016). Generating factoid questions with recurrent neural networks: The 30m factoid question-answer corpus. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 588–598, Berlin, Germany. Association for Computational Linguistics.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *In Proceedings of Association for Machine Translation in the Americas*, pages 223–231.

Socher, R., Huang, E., Pennin, J., Manning, C., and Ng, A. (2011). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*, pages 801–809.

Solar-Lezama, A. (2008). *Program Synthesis by Sketching*. PhD thesis, University of California at Berkeley, Berkeley, CA.

Soricut, R. and Echihabi, A. (2010). TrustRank: Inducing trust in automatic translations via ranking. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 612–621, Uppsala, Sweden.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Steedman, M. (2000). *The Syntactic Process*. MIT Press, Cambridge, MA, USA.

Su, Y., Sun, H., Sadler, B., Srivatsa, M., Gur, I., Yan, Z., and Yan, X. (2016). On generating characteristic-rich question sets for qa evaluation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 562–572, Austin, Texas. Association for Computational Linguistics.

Suhr, A., Iyer, S., and Artzi, Y. (2018). Learning to map context-dependent sentences to executable formal queries. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2238–2249, New Orleans, Louisiana. Association for Computational Linguistics.

Sun, Y., Tang, D., Duan, N., Ji, J., Cao, G., Feng, X., Qin, B., Liu, T., and Zhou, M. (2018). Semantic parsing with syntax- and table-aware SQL generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 361–372. Association for Computational Linguistics.

Susanto, R. H. and Lu, W. (2017). Neural architectures for multilingual semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 38–44, Vancouver, Canada.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA. MIT Press.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826.

Tang, L. R. and Mooney, R. J. (2000). Automated construction of database interfaces: Intergrating statistical and relational learning for semantic parsing. In *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 133–141, Hong Kong, China.

Tang, L. R. and Mooney, R. J. (2001). Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the 12th European Conference on Machine Learning*, pages 466–477, London, UK.

Templeton, M. and Burger, J. (1983). Problems in natural-language interface to dsms with examples from eufid. In *Proceedings of the First Conference on Applied Natural Language Processing*, pages 3–16, Santa Monica, California, USA. Association for Computational Linguistics.

Thompson, B. H. and Thompson, F. B. (1983). Introducing ask, a simple knowledgeable system. In *Proceedings of the First Conference on Applied Natural Language Processing*, pages 17–24, Santa Monica, California, USA. Association for Computational Linguistics.

Thomspon, C. A. and Mooney, R. J. (2003). Acquiring word-meaning mappings for natural language interfaces. *Journal of Artifical Intelligence Research*, 18:1–44.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude. Technical report.

Tu, Z., Lu, Z., Liu, Y., Liu, X., and Li, H. (2016). Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 76–85, Berlin, Germany.

Ueffing, N. and Ney, H. (2005). Word-level confidence estimation for machine translation using phrase-based translation models. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 763–770, Vancouver, Canada.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R. J., Darrell, T., and Saenko, K. (2015a). Sequence to sequence - video to text. In *IEEE International Conference on Computer Vision*.

Venugopalan, S., Xu, H., Donahue, J., Rohrbach, M., Mooney, R., and Saenko, K. (2015b). Translating videos to natural language using deep recurrent neural networks. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1494–1504, Denver, Colorado.

Vinyals, O., Bengio, S., and Kudlur, M. (2016). Order matters: Sequence to sequence for sets. In *Proceedings of the 4th International Conference on Learning Representations*, San Juan, Puerto Rico.

Vinyals, O., Kaiser, L., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. (2015a). Grammar as a foreign language. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, pages 2773–2781, Montreal, Canada.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015b). Show and tell: A neural image caption generator. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, Boston, Massachusetts.

Wang, A., Kwiatkowski, T., and Zettlemoyer, L. (2014). Morpho-syntactic lexical generalization for ccg semantic parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1284–1295, Doha, Qatar. Association for Computational Linguistics.

Wang, S. and Jiang, J. (2016). Learning natural language inference with lstm. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1442–1451, San Diego, California. Association for Computational Linguistics.

Wang, S. and Jiang, J. (2017). A compare-aggregate model for matching text sequences. In *Proceedings of International Conference on Learning Representations*.

Wang, Y., Berant, J., and Liang, P. (2015). Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1332–1342, Beijing, China. Association for Computational Linguistics.

Warren, D. H. D. and Pereira, F. C. N. (1982). An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*, 8(3-4):110–122.

Wieting, J. and Gimpel, K. (2018). ParaNMT-50M: Pushing the limits of paraphrastic sentence embeddings with millions of machine translations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 451–462. Association for Computational Linguistics.

Winograd, T. (1972). Understanding natural language. *Cognitive Psychology*, 3(1):1–191.

Wiseman, S., Shieber, S., and Rush, A. (2017). Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263, Copenhagen, Denmark. Association for Computational Linguistics.

Wong, Y. W. and Mooney, R. (2006). Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL*, pages 439–446, New York City, USA.

Wong, Y. W. and Mooney, R. (2007). Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967, Prague, Czech Republic.

Woods, W. A., Kaplan, R., and Webber, N. B. (1972). The LUNAR sciences natural language information system: Final report. Technical Report BBN Report No. 2378, Bolt Beranek and Newman, Cambridge, Massachusetts.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Łukasz Kaiser, Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Wubben, S., van den Bosch, A., and Krahmer, E. (2010). Paraphrase generation as monolingual translation: Data and evaluation. In *Proceedings of the 6th International Natural Language Generation Conference*, INLG '10, pages 203–207, Stroudsburg, PA, USA. Association for Computational Linguistics.

Xiao, C., Dymetman, M., and Gardent, C. (2016). Sequence-based structured prediction for semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1341–1350, Berlin, Germany.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 2048–2057, Lille, France.

Xu, K., Reddy, S., Feng, Y., Huang, S., and Zhao, D. (2016). Question answering on freebase via relation extraction and textual evidence. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 2326–2336, Berlin, Germany. Association for Computational Linguistics.

Xu, X., Liu, C., and Song, D. (2017). SQLNet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Yaghmazadeh, N., Wang, Y., Dillig, I., and Dillig, T. (2017). SQLizer: Query synthesis from natural language. *Proceedings of the ACM on Programming Languages*, 1:63:1–63:26.

Yang, Y., Yih, W.-t., and Meek, C. (2015). Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2013–2018, Lisbon, Portugal. Association for Computational Linguistics.

Yao, X. (2015). Lean question answering over freebase from scratch. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 66–70, Denver, Colorado. Association for Computational Linguistics.

Yao, X. and Van Durme, B. (2014). Information extraction over structured data: Question answering with freebase. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 956–966, Baltimore, Maryland. Association for Computational Linguistics.

Yavuz, S., Gur, I., Su, Y., Srivatsa, M., and Yan, X. (2016). Improving semantic parsing via answer type inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 149–159, Austin, Texas. Association for Computational Linguistics.

Yih, W.-t., Chang, M.-W., He, X., and Gao, J. (2015). Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1321–1331, Beijing, China. Association for Computational Linguistics.

Yin, P., Lu, Z., Li, H., and Kao, B. (2016). Neural enquirer: Learning to query tables in natural language. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 2308–2314. AAAI Press.

Yin, P. and Neubig, G. (2017). A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 440–450, Vancouver, Canada.

Yin, P., Zhou, C., He, J., and Neubig, G. (2018). StructVAE: Tree-structured latent variable models for semi-supervised semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 754–765. Association for Computational Linguistics.

Yin, W. and Schütze, H. (2015). Convolutional neural network for paraphrase identification. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 901–911, Denver, Colorado. Association for Computational Linguistics.

Yu, L., Hermann, K. M., Blunsom, P., and Pulman, S. (2014). Deep Learning for Answer Sentence Selection. In *NIPS Deep Learning Workshop*.

Yu, T., Li, Z., Zhang, Z., Zhang, R., and Radev, D. (2018). TypeSQL: Knowledge-based type-aware neural text-to-SQL generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 588–594, New Orleans, Louisiana. Association for Computational Linguistics.

Zaremba, W., Sutskever, I., and Vinyals, O. (2015). Recurrent neural network regularization. In *Proceedings of the 3rd International Conference on Learning Representations*, San Diego, California.

Zelle, J. M. and Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 1050–1055, Portland, Oregon.

Zettlemoyer, L. and Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, pages 658–666, Edinburgh, Scotland.

Zettlemoyer, L. and Collins, M. (2007). Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 678–687, Prague, Czech Republic.

Zhang, J., Lin, Z., Brandt, J., Shen, X., and Sclaroff, S. (2016). Top-down neural attention by excitation backprop. In *European Conference on Computer Vision*, pages 543–559, Amsterdam, Netherlands.

Zhang, S. and Sun, Y. (2013). Automatically synthesizing SQL queries from input-output examples. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, pages 224–234, Piscataway, NJ.

Zhang, Y., Pasupat, P., and Liang, P. (2017). Macro grammars and holistic triggering for efficient semantic parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1214–1223. Association for Computational Linguistics.

Zhao, K. and Huang, L. (2015). Type-driven incremental semantic parsing with polymorphism. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1416–1421, Denver, Colorado.

Zhao, S., Wang, H., Lan, X., and Liu, T. (2010). Leveraging multiple MT engines for paraphrase generation. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 1326–1334, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zhong, V., Xiong, C., and Socher, R. (2017). Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Zou, Y. and Lu, W. (2018). Learning cross-lingual distributed logical representations for semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 673–679. Association for Computational Linguistics.