LEARNING TO COMPOSITIONALLY REASON OVER NATURAL LANGUAGE

Nitish Gupta

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2021

Supervisor of Dissertation

_____

Dan Roth, Professor of Computer and Information Science

Graduate Group Chairperson

_____

Mayur Naik, Professor of Computer and Information Science

Dissertation Committee

Mitch Marcus, Professor of Computer and Information Science

Lyle Ungar, Professor of Computer and Information Science

Chris Callison-Burch, Associate Professor of Computer and Information Science

Luke Zettlemoyer, Professor of Computer Science & Engineering, University of Washington

LEARNING TO COMPOSITIONALLY REASON OVER NATURAL LANGUAGE

© COPYRIGHT

2021

Nitish Gupta

This work is licensed under the

Creative Commons Attribution

NonCommercial-ShareAlike 3.0

License

To view a copy of this license, visit

http://creativecommons.org/licenses/by-nc-sa/3.0/

Dedicated to Mummy and Papa

ACKNOWLEDGMENT

It is hard to imagine a better advisor than Dan Roth. His exceptional enthusiasm for research and encouragement to always work on important and challenging problems are some of his many teachings that will help me throughout my career. Dan's guidance in technical aspects of research and his sage advice on various professional aspects has helped me towards becoming a well-rounded researcher. Dan has been extremely kind and considerate in helping me handle the various ups and downs I've encountered in the six years of PhD life, which has made working with him extremely joyous. I am immensely grateful to Dan for all that he has done for me.

I am forever indebted to Sameer Singh for the support and mentorship that he has provided me over all these years. Right from agreeing to mentor me, who was an unexperienced undergrad, to helping me with graduate schools applications, and eventually being a close collaborator throughout my PhD journey, Sameer's contributions to my growth are inexplicable in words. It is only fair to say that none of this would have been possible without you. I feel extremely fortunate to have gotten the opportunity to work with Matt Gardner towards the later half of my PhD. Matt's guidance has made a significant portion of this dissertation possible and I have learnt a great deal from him. Apart from being excellent researchers, Dan, Sameer and Matt are few of the nicest human beings I know. They have always treated me with utmost respect, and their kindness towards others, especially junior colleagues, is something I will always aim to emulate.

I would like to thank the members of my thesis committee – Mitch Marcus, Lyle Ungar, Chris Callison-Burch, and Luke Zettlemoyer. Their unique and valuable perspectives, as well as suggestions on improving the technical presentation have significantly helped me improve this dissertation.

I feel lucky to have been a part of vibrant NLP communities at UIUC and UPenn. Over the years, I have come across brilliant people from whom I have learned a lot and developed close friendships with. I would like to thank the former members of CogComp—Mark Sammons, Christos Christodoulopoulos, Subhro Roy, Shyam Upadhyay, Stephen Mayhew, Daniel Khashabi, Haoruo Peng, and Chen-Tse Tsai—who were extremely welcoming and helped me fit in the group. A huge

thanks to the past and present students at Penn: Dan Deutsch, Jordan Kodner, Anne Cocos, Joao Sedoc, Reno Kriz, Daphne Ippolito, Rebecca Iglesias-Flores, Oshin Agarwal, Sihao Chen, Hengfeng He, Ben Zhou, Xiaodong Yu, and Krunal Shah. I will dearly miss the stimulating and fun discussions we had over lunches and happy hours. I cannot thank Snigdha Chaturvedi and Shashank Srivastava enough for their invaluable friendship, mentorship, and sage advice regarding all aspects of life.

I am grateful to the following people for their tireless efforts behind the scenes to make sure things run smoothly. Dan Widyono and Marcus Lauer in CETS provided the systems support whenever and whatever I needed. Jennifer Sheffield in CogComp and Britton Carnevali in CIS have always efficiently handled all the administrative details, and I thank them for their patience and help.

During summer internships, I was fortunate to have fruitful collaborations with the industry. I worked with Mike Lewis (Facebook) and Tom Kwiatkowski (Google) closely and interacted with many other researchers on their teams. The lessons I learned during my internships have informed the thesis in very different ways and helped me realize the importance of broader impact and real-world application of academic research.

I am in debt of my friends at Urbana-Champaign and Philadelphia: Tanmay Gupta, Vishaal Mohan, Ajit Vikram, Arpit Agrawal, Manisha Muduli, Ashna Jaiswal, Ankit Saxena, Palak Agarwal, Prathmesh Patil, Meghana Denduluri, Dushyant Sahoo, and Aalok Thakkar. Without your friendship and craziness, my years in PhD would have been much colder and more lonesome. I hope you enjoyed the laughs as much as I did. Penn Sargam, thank you for giving me the opportunity to play music on stage, something I did not imagine would be possible during graduate school.

This thesis would not be possible without the encouragement, support and love of my family. To my (cousin) sister, Nupur didi, who has been a steady support, I am grateful for your unconditional love and care. My brother, Jeet bhaiya, who has been my role model since childhood, thank you for being a constant source of encouragement. Finally, Mummy and Papa, I cannot thank you enough for all the sacrifices you made to provide me with excellent education. I don't have words to express how grateful I am for the love and blessings you shower upon me. All that is the best in me is you.

ABSTRACT

LEARNING TO COMPOSITIONALLY REASON OVER NATURAL LANGUAGE

Nitish Gupta

Dan Roth

The human ability to understand the world in terms of reusable "building blocks" allows us to generalize in near-infinite ways. Developing language understanding systems that can compositionally reason in a similar manner is crucial to achieve human-like capabilities. Designing such systems presents key challenges in the architectural design of machine learning models and the learning paradigm used to train them. This dissertation addresses aspects of both of these challenges by exploring compositional structured models that can be trained using end-task supervision.

We believe that solving complex problems in a generalizable manner requires decomposition into sub-tasks, which in turn are solved using reasoning capabilities that can be reused in novel contexts. Motivated by this idea, we develop a neuro-symbolic model with a modular architecture for language understanding and focus on answering questions requiring multi-step reasoning against natural language text. We design an inventory of freely-composable, learnable neural modules for performing various atomic language understanding and symbolic reasoning tasks in a differentiable manner. The question guides how these modules are dynamically composed to yield an end-to-end differentiable model that performs compositional reasoning and can be trained using end-task supervision. However, we show that when trained using such supervision, having a compositional model structure is not sufficient to induce the intended problem decomposition in terms of the modules; Lack of supervision for the sub-tasks leads to modules that do not freely compose in novel ways, hurting generalization. To address this, we develop a new training paradigm that leverages *paired examples*—instances that share sub-tasks—to provide an additional training signal to that provided by individual examples. We show that this paradigm induces the intended compositional reasoning and leads to improved in- and out-of-distribution generalization.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

Chapter 1

Introduction

Language is a crucial part of a human being's life. We use language to perform some of our most important daily functions. Language facilitates our quest for gathering new knowledge about the world, expressing our feelings and emotions towards entities and their actions, communicating instructions to carry out various tasks, among many of our other undertakings. Being such an indispensable feature of the human experience, it is obvious that any artificially intelligent agent we develop needs to possess, along with the knowledge of the world, impeccable language understanding capabilities, to interact with humans and also understand the physical world that is often described using language. Over the past few decades, significant effort has been put into developing agents that perform various tasks that require understanding language. As a community, we have come a long way since the rule-based systems of the 1950s and 60s (e.g., ELIZA and SHRDLU) to the present day, large-scale, machine learning based models that aim to learn about our world and language from the 100s of billions of tokens of text found on the Internet. In these years, our systems' ability to perform language related tasks has improved many folds in terms of their accuracy, coverage, and robustness, though, the goal of achieving human-like language understanding capability still seems as far ahead.

The human capacity to comprehend language, and the world in general, extends far beyond direct experiences. We are able to reason about novel real-life scenarios and understand never-encountered

natural language sentences with utmost ease. This seemingly infinite capability in intelligence (cognition) is best attributed to the principle of compositionality (often attributed to Gottlob Frege), which states that the meaning of a complex expression is determined by the meanings of its constituents and the rules used to combine them. Human language is highly compositional—by knowing the meaning of a finite set of words, and having an understanding of the rules that govern how these words can be combined to form sentences, we are able to understand and produce an infinite number of sentences. The idea of compositionality extends far beyond language and more generally to our world—by having the knowledge about different entities and concepts in the world, and an understanding of the *rules* that govern how they combine, we are able to reason about novel scenarios. By decomposing the world in terms of reusable "building blocks", we are able to understand novel contexts in terms of known concepts, and thereby leverage our existing knowledge in near-infinite ways. We believe that any language understanding system we build will also need to perform compositional processing in order to achieve human-like capabilities. The work described in this dissertation was carried out with the idea of developing algorithmic approaches that are capable of representing and reasoning about the compositional nature of our world and language to solve complex language understanding problems. The focus of this dissertation is on the specific application of answering questions requiring multiple steps of reasoning. However, the techniques developed in this research are more widely applicable to other language understanding problems.

Which country had the most COVID-19 cases by October 2020?

The COVID-19 pandemic in **India** is part of the worldwide pandemic of coronavirus disease 2019 (COVID-19) caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2). India in October has the largest number of confirmed cases in Asia,[8] and has the **second-highest number of confirmed cases in the world,[9][10] with almost 8 million reported** cases of COVID-19 infection

...

The COVID-19 pandemic in the **United Kingdom** is part of the worldwide pandemic of coronavirus disease 2019 (COVID-19). The virus reached the country in late January 2020. **As of 6 October 2020 there have been 515,571 confirmed cases**[nb 3] and 42,369 deaths of confirmed cases.[nb 1]

The COVID-19 pandemic in the **United States** is part of the worldwide pandemic of coronavirus disease 2019 (COVID-19). **As of October 2020, there were more than 9,000,000 cases** and 230,000 COVID-19-related deaths in the U.S., representing 20% of the world's known COVID-19 deaths, and the most deaths of any country.[6]

Figure 1: Example of a question requiring compositional reasoning

At the core of solving complex problems in a compositional manner lies two important steps, problem decomposition into simpler sub-tasks and solving those sub-tasks by using reasoning capabilities that can be reused in novel contexts. Let us look at examples of complex problems in the context of language understanding, and the various representational and reasoning capabilities that an AI agent would require to successfully solve them. Consider the question, *Which country had the most COVID-19 cases by October 2020?* in Figure. 1.[1] The answer to the question is not stated *directly* in the accompanying context, but can be reached by performing a few steps of reasoning. To answer this, an agent would need to understand the semantics of the question and decompose it into multiple simpler but interrelated problems, find the answer to these sub-problems, and finally compose these answers to arrive at the answer to the question. In this particular example, an agent would need to solve the following sub-problems: locate the "countries" mentioned in the context, find the respective number of "COVID-19 cases by October 2020" for each one of them,[2] and compute the maximum value of cases among these. Solving such sub-problems requires a variety of language understanding, world knowledge, and symbolic reasoning capabilities. In this example, the system needs the ability to identify different mentions of "countries" in text (*India*, *United Kingdom*, and *United States*). It needs to tackle the various linguistic variations in which the number of "COVID-19 cases by October 2020" for each country is mentioned, in order to perform accurate information extraction. Lastly, it needs to know that the linguistic quantifier "most" requires performing the *argmax* operation and possess symbolic reasoning capability to carry out this operation. Similarly, consider the question, *What was the longest gap between two Radiohead albums?*[3] which, apart from challenges similar to the previous question of grounding the phrase "Radiohead albums" to its corresponding instantiations in text, poses new challenges. The system needs to infer that the linguistic construction "longest gap" in the context of two albums refers to the maximum time-span, measured in years, between the release dates of two consecutive Radiohead albums. Further, it needs to seek the release dates of the various albums to carry out this reasoning. Lastly, consider the utterance, *I liked the show, but not as much as Seinfeld*—to understand the sentiment expressed towards

[1] A lot of the work presented in this thesis was conducted during the COVID-19 pandemic. Text snippets are from Wikipedia.

[2] This query can be further broken down depending the provided context. We discuss this issue in §8.2.

[3] The answer is 5; between *The King of Limbs* (2011) and *A Moon Shaped Pool* (2016).

the "show", a system will need to compose the sentiment towards "Seinfeld", with the phrase "not" and "as much as", to reach the conclusion that this show rates lower on the "likeness" scale. In all the examples we saw, problem decomposition into simpler tasks and composition of their solutions is a common theme that needs to be addressed by language understanding systems. Also note that, solving these tasks and their composition requires symbolic operations in many cases.

In order for language understanding systems to truly perform compositional reasoning and systematically generalize to novel problem instances in a manner humans do, they also need to reuse their knowledge and reasoning capabilities in novel contexts. For example, an agent that is able to reason about the problem instances mentioned above, should also generalize to *Which Radiohead album sold the most records?* by reusing its capability to ground the concept "Radiohead albums" and understand the quantifier expression "most" in this previously unseen context of "albums" and "number of records sold". Therefore, for language understanding agents to achieve human-like capabilities, they need to be able to understand the world and language in terms of parts that can be processed (almost) independently and recombined in infinite ways.

Given that compositionality is at the core of intelligent behavior, it is surprising that majority of the NLP community's recent efforts have largely ignored this issue.[4] The two main components of building language understanding systems—the machine learning models developed for NLP tasks and the learning paradigm used for training these models—both are not designed in a manner that promotes compositional processing. Let us look at the most widely used instantiations of both these components and the issues underlying them.

In the last decade, the NLP community has narrowed down to using large-scale, black-box, neural network (NN) models for solving complex language understanding tasks. These models are trained to map raw input text, represented via dense word representations, directly to the output class. The computational structure of these models is *fixed*; irrespective of the problem, each input is processed in exactly the same manner by passing through all layers of the neural network to predict the out-

[4]There is indeed a long line of work incorporating formal compositional semantics into natural language processing. We discuss relevant approaches in Chapter 2.

put. These models do not perform any explicit problem decomposition and recombination, neither do they contain any architectural bias to perform such decomposition or identify building blocks of language understanding that can be reused. They are used on the assumption that this hypothesis class of universal function approximators will automatically learn to perform the required reasoning internally in its distributed representations. In this dissertation, we take a contrary view and propose models with modular architecture, thereby providing explicit inductive bias to process language in a compositional manner. The modules form the building blocks of the reasoning capability of the model, and can be dynamically combined depending on the input to perform the required compositional reasoning.

The learning paradigm used to train the models is another important aspect of developing a language understanding system. The dominant paradigm of learning in NLP has been labeling large quantities of input-output pairs for an end-task, followed by training an expressive function approximator to model the statistical regularities between some representation of text and output labels. The resulting models, evaluated on held-out test data obtained by drawing *i.i.d* samples from the data collection, have demonstrated impressive performance across a range of NLP benchmarks. However, it has been increasingly observed that this i.i.d training and evaluation paradigm is faulty. As a result, these models have been the subject of scrutiny in many studies testing generalization. Various studies have shown that our datasets often contain *spurious biases*—unintended correlations between input and output—and the expressive NNs exploit these spurious correlations, thus failing to solve the problems for the right reasons. When these models are evaluated on instances obtained by performing minor input perturbations (Khashabi et al., 2016; Jia and Liang, 2017; Gardner et al., 2020), or subject to *compositional generalization* tests—evaluation on instances obtained by composing observed sub-parts in novel ways—we find that these models show extremely poor generalization capabilities. We believe that the supervision provided by input-output pairs is not enough to teach the models about the compositional nature of problems we are trying to solve. We will never be able to gather as much data that contains a sufficient number of different combinations of sub-parts to induce the correct decomposition. Additionally, just output supervision does not provide any direct training signal for what the correct intermediate decisions should be. Given how extremely

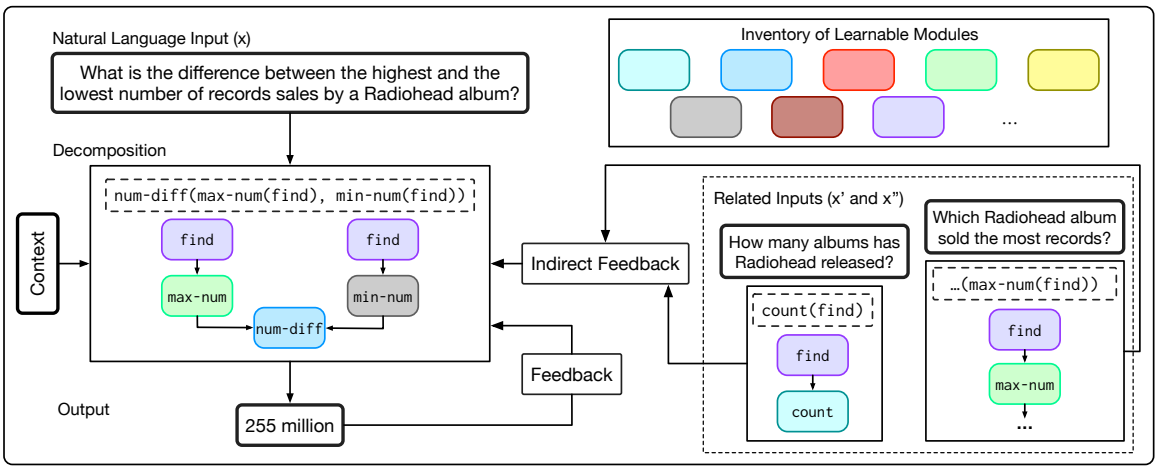


Figure 2: Framework for compositional reasoning over natural language. The system consists of an inventory of learnable neural modules that are designed to perform various atomic natural language and symbolic reasoning operations, and a model for decomposing the input task into sub-tasks in terms of these modules. For a given a natural language utterance (x) describing a task (e.g., question, instruction), the required modules are dynamically assembled into a structured model that is executed against the provided context (e.g., paragraph, knowledge-graph) to predict the output. The feedback from end-task supervision can be used to train the modules and the model for decomposition. We also explore ways to leverage related input utterances (x' and x"), that share some substructure with the input, to provide additional indirect feedback to improve learning.

high-dimensional natural language is, any realistic amount of i.i.d samples will contain spurious correlations, thus containing *shortcuts* that the models can exploit. We also believe that it is unrealistic to expect annotations containing stronger supervision signals for learning—in terms of the correct problem decomposition and the intermediate decisions that the model needs to make. Such annotations are expensive to collect and require experts in many cases. To address these limitations with the weak training signal provided by end-task supervision, we propose a new training paradigm that leverages related training examples to provide indirect supervision to the intermediate decisions, and thus provide stronger cues for learning the correct compositional processing.

1.1 Thesis Statement

The thesis of this research is that solving complex language understanding problems, in a generalizable manner similar to humans, requires decomposition in terms of reusable reasoning capabilities that can be combined in novel ways. We claim that machine learning models that are composed

of loosely decoupled modules can support such compositional understanding of natural language to solve complex tasks. Additionally, designing learnable modules that can perform various atomic reasoning tasks in a differentiable manner can yield models that facilitate diverse reasoning capabilities and can be learned from end-task supervision. However, since input-output examples only provide a weak indirect signal to the model's intermediate decisions—output of the modules in this case, such a learning paradigm can yield suboptimal modules that do not generalize to novel contexts. We further claim that appropriately selected groups of training examples can be simultaneously used to provide indirect supervision to the model's intermediate decisions. By applying consistency constraints between the model's intermediate decisions, this paradigm can provide an additional training signal beyond what is provided by individual examples.

1.2 Outline of this Dissertation

The rest of the document is organized as follows:

- Chapter 2 gives a brief history of different approaches for complex and compositional language understanding tasks. We focus on semantic parsing, approaches for question answering with a focus on reading comprehension, and neural module networks (Andreas et al., 2016).

- Chapter 3 introduces a modular, neuro-symbolic approach for answering complex, compositional questions against text. We design learnable modules for performing various atomic natural language and symbolic reasoning tasks (such as arithmetic, sorting, counting), which in turn can be composed to answer questions requiring multi-step reasoning.

- In Chapter 4 we show that the inductive bias for compositional processing via modular architecture is not sufficient to induce correct problem decomposition when such models are trained via input-output supervision.

- In Chapter 5 we propose a new training paradigm that leverages related instances to provide indirect supervision to the intermediate decisions that the model needs to make when trying to solve the problem in a compositional manner. We demonstrate the benefits of this paradigm in

the context of question answering against text, and show that it leads to significantly improved in- and out-of-distribution generalization.

- In Chapter 6 we apply our training paradigm based on related examples to improve semantic parsing in a weakly-supervised setting. We show that enforcing consistency between the interpretations of similar phrases in related examples leads to better generalization.

- Inspired by formal approaches to semantics, in Chapter 7 we present a fully-grounded question understanding approach that resembles Montague semantics (Montague, 1973). Each span in the question is represented by either a denotation in the context (in this case, a knowledge-graph) or some ungrounded aspect of meaning. Learned composition modules are used to recursively combine constituent spans, culminating in a grounding for the complete question. We show that the modules can learn a variety of challenging semantic operators, such as quantifiers, disjunctions and composed relations via end-to-end training.

Finally, Chapter 8 summarizes the contributions of this dissertation and provides an overview of some future directions. All code, data, and other resources used in this dissertation are available at: `https://nitishgupta.github.io/phd-thesis-resources/`

Chapter 2

Background

In this chapter, we overview previous approaches in three areas that relate to our work. We first discuss prior work in the area of semantic parsing. Next, we look approaches for question answering with a focus on reading comprehension and neural module networks.

2.1 Semantic Parsing

Building systems that understand natural language utterances and produce an appropriate response has been a long-standing goal of artificial intelligence. Consider the utterances in Figure 3; a system will require deep understanding of natural language and reasoning capability in order to produce the correct action. Systems with such capability have useful applications as natural language interfaces into knowledge graphs, databases, software systems, robots, etc. The area of *semantic parsing* within NLP aims to develop language understanding systems with such capability.

A semantic parser maps a natural language utterance into a semantic meaning representation called *logical forms*. These logical forms can be seen as *programs* that are *executed* against the appropriate context to produce the desired response. The response can be an answer, action, a set of entities in a knowledge-base, a truth-value, etc. depending on the context. More generally, we will refer to this response as the *denotation* of the logical form.

Context:	knowledge of mathematics
Utterance:	*What is the largest prime less than 10?*
Action:	7
Context:	knowledge of geography
Utterance:	*What is the tallest mountain in Europe?*
Action:	Mt. Elbrus
Context:	user's calendar
Utterance:	*Cancel all my meetings after 4pm tomorrow.*
Action:	(removes meetings from calendar)

Figure 3: Example of natural language utterances in different domains with their corresponding actions

For example, consider the second utterance in Fig. 3 which would map to the following logical form:

What is the tallest mountain in Europe?

$$\texttt{tallest}(\lambda \texttt{x.mountain(x)} \wedge \lambda \texttt{y.europe(y)} \wedge \texttt{isLocated(x, y)})$$

In this example, the logical form can be executed against a database containing geographical facts to yield the answer.

Semantic parsing is rooted in formal semantics and strongly inherits two insights from it. First, the idea of *compositionality*—the meaning of an expression can be recursively determined from the meaning of its subexpressions (Montague, 1973)—to interpret complex natural language utterances. Second, the idea of *model theory*, which states that utterances are mere symbols which only obtain their meaning or denotation when executed against a context (e.g., database). Semantic parsing goes beyond other forms of parsing that identify shallow agent-patient roles, such as in semantic role labeling. Since the parse, the logical form, is executed against a context to yield the denotation, this form of semantic parsing is sometimes referred to as *executable semantic parsing*.

Early semantic parsing systems were built using hand-crafted rules to handle complex linguistic phenomena and integrate syntax, semantics, and reasoning. Early examples include the LUNAR system - a natural language interface into the moon rocks database (Woods, 1972), and the famous SHRDLU - a system that could answer questions and perform actions in a toy blocks world environment (Winograd, 1972). While these systems were capable of handling complex utterances, due

their rule-based nature, they suffered from generalizing beyond the narrow domain they were developed for and were brittle in handling variations and intricacies of natural language. With the rise of machine learning in NLP, the paradigm of collecting input-output examples and fitting a statistical model also influenced the area of semantic parsing. Henceforth, we will focus on statistical method for semantic parsing.

Statistical semantic parsing has been the dominant approach of building a semantic parser since the 1990s (Zelle and Mooney, 1996; Miller et al., 1996). Though semantic parsing systems have undergone several drastic improvements, they roughly still consist of a few key components. One beneficial property of semantic parsing systems has been their modular nature, where different choices of these components can be used together to yield different semantic parsers. A semantic parser consists of - (1) the semantic representation or the logical language used to represent the meaning and a grammar to produce candidate derivations, (2) a parsing model that scores different logical forms corresponding to an utterance based on its learnable parameters, and a parsing algorithm to search for high scoring programs, and (3) a learning algorithm to estimate the parameters of the model (and possible rules in the grammar) given training examples. We will go over these components in detail in the following sections.

2.1.1 Semantic Meaning Representation

Over the years many logical language formalisms have been used in semantic parsing models. The decision to which language to use is dependent on two main factors – the ability to execute the logical form in the required context and the expressivity of the logical language to handle the semantic phenomena required in the domain.

First-order logic - GeoQuery (Zelle and Mooney, 1996) is a first-order logic based language augmented with higher-order predicates. It is the first formalism to be used in a statistical semantic parser.

Lambda Calculus - First-order logic augmented with *lambda calculus* (Carpenter, 1997) is the most commonly used logical language. Lambda calculus has been used for querying databases (Zettle-

moyer and Collins, 2005), in implementing conversational agents (Artzi and Zettlemoyer, 2011), as well as for providing instructions to robots (Artzi and Zettlemoyer, 2013).

Lambda Dependency-based Compositional Semantics (λ-DCS) is a logic-based formalism introduced by Liang et al. (2011); Liang (2013). It is designed as a more compact representation of lambda calculus. In this formalism, existential quantification is made implicit, allowing the elimination of variables. For example, an utterance such as *People above the age of 20.* would be

Lambda calculus – λx.$\exists$y.AgeGreaterThan(x, y)$\wedge$Age(y, 20)

λ-DCS – AgeGreaterThan.Age.20

This formalism has been used in applications such as developing semantic parsers for question-answering over a knowledge-graph, such as Freebase (Berant et al., 2013), and over semi-structured tables (Pasupat and Liang, 2015).

Structured Query Language (SQL) is a domain-specific query language used for managing data stored in a relational database management system (RDBMS). Due to the wide-spread usage of databases, Text-to-SQL is an ever growing subarea in semantic parsing. Many datasets in a variety of domains and semantic parsing models have been developed to map natural language statements into SQL queries.

Programming Languages Similar to SQL, many approaches have looked at converting natural language into code written in a high-level general purpose programming language, such as Python (Ling et al., 2016; Yin and Neubig, 2017) and Java (Iyer et al., 2018).

Functional Query Language Due to the difficulty in handling variables in a logical language while parsing, variable-free functional query languages have been developed for some domains. FunQL (Kate et al., 2005) was developed as a variable-free variant of the Geoquery language, Dasigi et al. (2019) develop a functional language for visual reasoning on the NLVR dataset (Suhr et al., 2017), Wang et al. (2016) develop a language for a blocks world similar to SHRDLU, etc. As we will see later in Section 2.3, this formalism is also used in neural module networks.

Grammar The grammar in a semantic parsing model is used to define a set of candidate derivations for a given natural language utterance. In the simplest form, a grammar contains rules that map the utterance tokens or phrases to the different logical form units it can be mapped to. For example, *10* $\implies$ `10` – indicates that the natural language token "10" can map to the constant `10` in the logical language. Another rule, *largest NP[z]* $\implies$ `max(z)` indicates that a natural language phrase like "largest state" can map to a logical form sub-part `maximum(state)` where `maximum` is an operation in the logical language and `state` is a variable.

Combinatory Categorial Grammar (CCG) has an integrated treatment of syntax and semantics that makes use of a compositional semantics based on the lambda calculus (Steedman, 2004). CCG consists of two parts – a CCG lexicon in which words (phrases) are mapped to *syntactic types*, and a set of *combinatory rules* which describe how adjacent syntactic categories in a string can be recursively combined. Along with a syntactic type, CCG also contains an a *semantic type* for each lexical entry when used for semantic parsing. Similarly, combinatory rules are extended to functional application rules that define how the semantic types can be composed. For example, the following are two entries in a CCG lexicon for Geoquery:

$$\text{Utah} = NP \ : \ \texttt{utah} \tag{2.1}$$

$$\text{borders} = (S\backslash NP) / NP \ : \ \lambda\texttt{x}.\lambda\texttt{y}.\texttt{borders(y,x)} \tag{2.2}$$

Zettlemoyer and Collins (2005) generalize CCGs to Probabilistic CCGs (PCCGs) which is used to score different derivations of the same utterance in a statistical semantic parser. CCG as a formalism is used in many semantic parsing models for various applications (Zettlemoyer and Collins, 2007; Artzi and Zettlemoyer, 2011; Kwiatkowski et al., 2010, 2011, 2013).

In CCG, the lexical rules can become quite complicated. λ-DCS mitigate this issue by describing a much crude grammar with very few rules. Secondly, the requirement to have all predicates in the logical form be anchored to lexical items via the grammar leads to a brittle parser when the grammar is incomplete in terms of lexical or linguistic variations. To overcome this, floating rules—rules that can be applied in the absence of any lexical trigger—have been proposed (Berant and Liang, 2014;

Pasupat and Liang, 2015). These relaxations definitely lead to a blowup in the space of possible programs, especially non-sensical ones, but the statistical parser is expected to learn to score such incorrect derivations poorly.

With the rise of deep learning approaches in semantic parsing the role of a lexicalized grammar has diminished quite significantly. Semantic parsers no longer require a grammar mapping lexical items to semantic units to enumerate candidate derivations. Though, recent work has shown that the grammar of the target language can be useful in decoding only well-formed logical forms which leads to improved performance. Work on general purpose code generation (such as SQL, Python etc) leverage the well-defined grammar associated with the programming language by directly generating the Abstract Syntax Tree corresponding to the query/code. For well-typed domain-specific languages designed for particular tasks (for example, the language used in Krishnamurthy et al. (2017)), the type-signature of predicates induces a type-constrained grammar which is used to perform grammar-constrained decoding to output well-formed logical forms. We will see this style of grammar-constrained decoding in detail later in this section.

2.1.2 Parsing models and learning

In this section we will see the two most prominent statistical parsing models used in our community; feature-based log-linear models and neural network based Seq2Seq models. The idea is, given a training dataset containing natural language utterances and their corresponding logical forms, to learn a parsing model to predict the correct logical form for a new test utterance.

Log-linear model Zettlemoyer and Collins (2005) were the first to use a log-linear model to learn parameters of a Probabilistic CCG (PCCG). It takes as input pairs of natural language utterances and their meaning representations as lambda-calculus expressions. For example, in the GeoQuery domain, *What states border Texas?* would be mapped to λ`x.state(x)`$\wedge$`borders(x, texas)`. Given a CCG lexixon L, there would be many different parses that conform to the utterance the corresponding logical form. To deal with this ambiguity, the PCCG of Zettlemoyer and Collins (2005) learns to rank possible parses for an utterance based on the estimated probability of the parse. More

concretely, the PCCG is modeled using a log linear model that uses a feature vector f defined over an (utterance, parse, logical-form) tuple and a parameter vector θ which is used to used a particular parse. For a given utterance x, parse s and logical expression z, the log linear model is defined as :

$$P(s, z|x; \theta, L) = \frac{e^{\theta^T f(x,s,z)}}{\sum_{(s',z')} e^{\theta^T f(x,s',z')}} \tag{2.3}$$

For a given utterance, finding the most likely logical meaning representation z^* involves performing the following marginalization over all the latent parses s:

$$z^* = \underset{z}{\text{argmax}}\, P(z|x; \theta, L) = \underset{z}{\text{argmax}} \sum_{s} P(s, z|x; \theta, L) \tag{2.4}$$

Usually, the feature function f decomposes locally over the inputs giving way to perform dynamic programming using the CKY parsing algorithm yielding an efficient inference methodology. Zettlemoyer and Collins (2005) induce a CCG lexicon L as a part of the learning procedure. Several improvements to this basic framework have been proposed. Please see Zettlemoyer and Collins (2007); Kwiatkowski et al. (2010, 2011, 2013) for further reading.

Sequence-to-Sequence (Seq2Seq) models The introduction of encoder-decoder architectures based on recurrent neural networks (Kalchbrenner and Blunsom, 2013; Cho et al., 2014; Sutskever et al., 2014) for machine translation has revolutionized the way many NLP tasks are modeled. This neural architecture, commonly known as sequence-to-sequence (Seq2Seq), takes as input a natural language string in one language and learns to output its translation in the desired language in an end-to-end manner. This framework has been used to model various NLP tasks as a generic string transduction problem; for example, syntactic parsing (Vinyals et al., 2015a), image captioning (Vinyals et al., 2015b), text-summarization (Rush et al., 2015), etc. The basic Seq2Seq architecture has been augmented with neural attention (Bahdanau et al., 2015) to model the latent alignment between the input and output tokens. This *soft* attention is computed using a deterministic network which estimates the expected value of the latent alignment variable (Deng et al., 2018), and has proved to be extremely beneficial in improving performance of Seq2Seq models on various problems.

While Wong and Mooney (2006) had proposed treating semantic parsing as a machine translation (MT) problem, their approach made use of pipelined statistical models for MT to perform semantic parsing. Dong and Lapata (2016) introduced an end-to-end neural semantic parsing model based on the Seq2Seq with attention architecture. In their basic model, a neural LSTM encoder encodes the input utterance as a dense vector representation. This utterance representation is used to condition the decoder that generates the corresponding logical form in a token-by-token manner. Given an utterance $x = [x_1, \ldots, x_N]$, the model learns to output a tokenized, linearized representation of the corresponding logical form $z = [z_1, \ldots, z_M]$. For example, for *What states border Texas?* with the logical form `λx.state(x)∧borders(x, texas)`, the model will learn to produce the following sequence of logical form tokens: [‘`λ`’, ‘`x`’, . . ., ‘`x`’, ‘`,`’ ‘`texas`’, ‘`)`’]. The decoder performs decoding in a left-to-right manner and models the probability of the next token based on the representation of the last decoded token, hidden representation of the LSTM state, and the encoded representation of the input utterance. The probability for the predicted logical form decomposes as the product of probabilities of the individual tokens in it, just like any language model:

$$p(z|x) = \prod_{t=1}^{M} p(z_t|z_{<t}, x) \tag{2.5}$$

Here $p(z_t|z_{<t}, x)$ is modeled in a standard manner as any Seq2Seq model. Dong and Lapata (2016) also present a Sequence-to-Tree model to take the hierarchical tree structure of the logical form into account; the decoder here generates the logical form in a top-down manner. Such architectural inductive bias provides stronger syntactic cues to the model which are absent when trained to perform decoding in a linear manner. Note that this end-to-end framework for parsing provides a significant departure from previous approaches that needed one to define a feature function over the (utterance, logical form) combination and mitigates the need to define precise grammars, such as the lexicon used in CCG. On the other hand, also note that neither of these models ensures that the produced logical form is syntactically well-formed and *executable*.

Grammar-constrained decoding The neural semantic parser of Krishnamurthy et al. (2017) uses an encoder-decoder model where the decoder generates the logical from a grammar instead of out-

putting tokens from the logical language without any constraints. Such grammar-constrained decoding guarantees that the parser generates only those logical forms that are well-typed and executatble. We use this grammar-constrained decoder based semantic parser model in this dissertation (chapters 3, 4, 5, and 6). Though Krishnamurthy et al. (2017) use a λ-calculus language for expressing logical forms in the WikiTables domain (Pasupat and Liang, 2015), here we describe their grammar-constrained decoding using a toy example for clarity.

Imagine designing the logical language for the toy task of representing arithmetic expressions, such as `add(2, 3)`, `square(5)`, etc. This language contains constants, such as numbers `2`, `3`, `5`, etc. and functions such as `add`, `square` etc. Krishnamurthy et al. (2017) prescribe designing the logical language in a manner such that all constants and functions (or operators, predicates, etc.) are assigned a type; this allows the decoder to enforce type constraints on the generated logical form. In our toy logical language, the constant numbers will be assigned the type `int`, and the functions will have a higher-order functional type. For example, `add` will have the functional type $\langle$`int`,`int`:`int`$\rangle$, indicating that `add` is a function that takes as input two arguments of `int` type and produces an output of the `int` type. Similarly, `square` will have the functional type $\langle$`int`:`int`$\rangle$ representing a function that takes a single `int` type argument as input, and produces an `int` type output. The parser then applies standard programming language type inference algorithms to automatically assign types to larger expressions. Given the types of constants and predicates, the parser enumerates all possible production rules to generate a non-terminal literal of a particular type, which is later used to perform constrained decoding. For example, for this toy logical language, the following production rules are

valid:

$$\texttt{int} \rightarrow 2$$

$$\texttt{int} \rightarrow 3$$

$$\texttt{int} \rightarrow 5$$

$$\ldots$$

$$\langle \texttt{int}, \texttt{int} : \texttt{int} \rangle \rightarrow \texttt{add}$$

$$\langle \texttt{int} : \texttt{int} \rangle \rightarrow \texttt{square}$$

$$\texttt{int} \rightarrow [\langle \texttt{int} : \texttt{int} \rangle, \texttt{int}]$$

$$\texttt{int} \rightarrow [\langle \texttt{int}, \texttt{int} : \texttt{int} \rangle, \texttt{int}, \texttt{int}]$$

Here, the last production rule defines that an `int` type literal can be produced using a function with the type $\langle$`int`,`int`:`int`$\rangle$, and two `int` type inputs.

Given the logical language where all constants and predicates are typed, the parser defines a grammar consisting of production rules (actions) as shown above. The decoder performs top-down grammar-constrained decoding, generating the linearized abstract syntax tree (in an in-order traversal) of the logical form. At any given time-step of decoding, the decoder maintains the logical form with non-terminals that are yet to be decoded, where the current state defines the actions that are valid. This ensures that the generated logical forms are well-typed. For example, to produce the program

`add(2,square(3))`, the decoder will generate the following actions:

$$z_1 : \texttt{start} \rightarrow \texttt{int}$$

$$z_2 : \texttt{int} \rightarrow [\langle \texttt{int}, \texttt{int} : \texttt{int} \rangle, \texttt{int}, \texttt{int}]$$

$$z_3 : \langle \texttt{int}, \texttt{int} : \texttt{int} \rangle \rightarrow \texttt{add}$$

$$z_4 : \texttt{int} \rightarrow 2$$

$$z_5 : \texttt{int} \rightarrow [\langle \texttt{int} : \texttt{int} \rangle, \texttt{int}]$$

$$z_6 : \langle \texttt{int} : \texttt{int} \rangle \rightarrow \texttt{square}$$

$$z_7 : \texttt{int} \rightarrow 3$$

Krishnamurthy et al. (2017) show that their parser with grammar-constrained decoding significantly outperforms Seq2Seq and Seq2Tree decoders.

2.1.3 Learning Semantic Parsers from Denotation

The semantic parsing approaches we saw until now rely on annotated training data mapping natural language utterances to their corresponding logical forms as supervision for learning. Acquiring such supervision is time consuming and requires expert annotators who understand the formal meaning representation being used. Also, such annotation restricts the meaning representation that can be used by the semantic parser. Since in all most all cases these semantic parsers are used as natural language interfaces, we only care about the output and any restriction on the meaning representation formalism that can be used limits the capability of the parser.

Over the last few years exciting innovations in learning semantic parsers directly from denotations[1] instead of supervised logical forms (Clarke et al., 2010) has lead to increased interest in semantic parsing. When learning in such a setting, the semantic parse for an utterance is treated as a structured latent variable which needs to be inferred from the answer supervision. This results in a significantly

[1] We refer to the response of the logical form after execution as its denotation; see §2.1

challenging learning problem since the space of all possible logical forms is exponentially large and *spurious* logical forms—logical forms that evaluate to the correct answer but are not the correct meaning representation of the utterance—provide a noisy training signal to the learner. Typically this learning framework is called weakly-supervised semantic parsing. In contrast to fully-supervised semantic parsing we saw in the previous section, here we are provided with a dataset $\mathcal{D}$ containing pairs of natural language utterances and their corresponding denotation $\{(x^i, y^i)\}_{i=1}^{i=M}$. Using this weak supervision, the aim is to learn a semantic parser to map the utterance x to the logical form z such that z executes to the correct denotation, i.e. $[\![z]\!] = y$.

Iterative Structured Learning In the first work that looked at this problem, Clarke et al. (2010) propose an iterative structured learning framework to learn a semantic parser from weak supervision. Their approach converts the provided weak supervision in terms of (x, y) pairs into strong supervision in terms of (x, z) pairs, and by that convert the problem into a structured prediction problem. Their iterative approach consists of two steps: (1) mining a dataset of (x, z) pairs based on the current estimate of the parameters, and (2) using this dataset to update the estimate of the parameters by employing a standard structured prediction learning algorithm. The iterative training is stopped when no new (x, z) pairs are generated in step 1. For the semantic parsing model, they use a linear model (Zettlemoyer and Collins, 2005) to score a (x, z) pair via $w^T\Phi(x, z)$, where Φ is a feature function over the input. For a given (x, y) pair, they find the top-scoring $\hat{z} = \text{argmax}\, w^T\Phi(x, z)$ by running inference, and add $(x, \hat{z})$ to the supervised dataset if its execution leads to the correct output, i.e., $[\![\hat{z}]\!] = y$. This framework has also been used to learn semantic representations of natural language instructions that are aimed at teaching machines about concepts from world feedback instead of using labeled training examples (Goldwasser and Roth, 2011).

Maximum Marginal Likelihood (MML) Following this, the work of Liang et al. (2011), which introduced the λ-DCS, also propose learning paradigm for weakly-supervised semantic parsing by maximizing the marginal log-likelihood (MML) of predicting the correct denotation. The marginal

likelihood of the correct denotation for a single instance is

$$p(y|x) = \sum_{z \in \mathcal{Z}(x) s.t. [\![z]\!]=y} p(z|x;\theta) \tag{2.6}$$

which sums over all logical forms z plausible for x (denoted by the set $\mathcal{Z}(x)$) that evaluate to the correct denotation y. Since enumerating all possible logical forms in $\mathcal{Z}(x)$ is computationally infeasible, their work resorts to running beam search to gather a set of possible logical forms, and maximizes the approximate marginal likelihood based on the logical forms in this set. The final training objective for the approximate MML is

$$J = \sum_{(x,y)} \log \sum_{z \in \mathcal{Z}_\theta(x) s.t. [\![z]\!]=y} p(z|x;\theta) \tag{2.7}$$

where $\mathcal{Z}_\theta(x)$ denotes the set of all logical forms for x produced by running beam search. Liang et al. (2011) places the log-linear distribution over the space of logical forms similar to Zettlemoyer and Collins (2005). Their approximate MML learning framework has been successfully adopted by other works in weakly-supervised semantic parsing using floating parsers (Pasupat and Liang, 2015) and Seq2Seq models (Krishnamurthy et al., 2017).

Reinforcement Learning Weakly supervised semantic parsing has also been looked at from the lens of reinforcement learning (Liang et al., 2018). Given a training instance (x, y), the agent uses the policy to take a sequence of actions $z = [z_1, \ldots, z_M]$ and then receives a reward $R(z)$, which is 1 if z evaluates to the correct denotation ($[\![z]\!] = y$) and 0 otherwise. In policy gradient methods, a stochastic policy function, in this case the semantic parser, is trained to maximize the expected reward. For a single instance, the expected reward is $\sum_{z \in \mathcal{Z}(x)} R(z)p(z|x;\theta)$. Given a dataset of (x, y) pairs, the final objective is

$$\begin{aligned} J &= \sum_{(x,y)} \sum_{z \in \mathcal{Z}(x)} R(z)p(z|x;\theta) \\ &= \sum_{(x,y)} \sum_{z \in \mathcal{Z}(x) s.t. [\![z]\!]=y} p(z|x;\theta) \end{aligned} \tag{2.8}$$

It can be easily observed that the training objectives for MML (Eq. 2.7) and RL (Eq. 2.8) look very similar. Guu et al. (2017) perform a very interesting and useful analysis comparing MML and RL, and conclude that the MML training objective is better in theory and practice.

2.2 Question Answering

In this section we focus our attention on approaches that develop "question answering" (QA) models, i.e., systems that are capable of answering questions posed in natural language. Question answering is a very natural manner in which humans acquire knowledge and fill their information needs. Hence, it is no wonder that the NLP community has taken keen interest in developing question answering systems. Apart from its use an end-goal application, QA also provides a natural framework for probing an agent's understanding of language and the world in general. As a result, QA has been also been viewed as a way to measure progress in NLP, and AI in general, by many. At this point, it is important to note that question answering is just a format, instead of a task, in which the input and output are natural language strings (Gardner et al., 2019). This flexibility has also contributed to its wide-spread adoption by the community. In some extreme cases, people have pushed for treating *every* NLP task as question answering, even classification and translation (Kumar et al., 2016; McCann et al., 2018). Though, QA only becomes useful as a format for tasks in which *question understanding* is a crucial component of the task itself.

It is natural to assume that a system capable of "question answering" will need to deal with a broad range of language and reasoning phenomena; ideally encompassing all possible phenomena. Such an ask is unrealistic from our current machine learning models. In its original use in NLP, QA was used to only refer to the narrow scope of answering factoid questions against a small database (Woods, 1972; Zelle and Mooney, 1996) or answering short, open-domain, factoid questions that a human might pose to a retrieval system (Voorhees, 1999; Kwok et al., 2001). However, as the NLP systems have improved in recent times, a wide variety of tasks are being cast as question answering. Though the scope of question answering has widened from just answering factoid questions, the terminology has remained with us. Question answering approaches have been developed for a wide variety of use cases – answering questions against the web as a resource (Voorhees, 1999; Talmor and Berant,

2018), a short story (Richardson et al., 2013), knowledge-graphs (Cai and Yates, 2013; Berant et al., 2013), images (Agrawal et al., 2015), semi-structured tables (Pasupat and Liang, 2015), solving math word problems (Hosseini et al., 2014; Kushman et al., 2014; Roy and Roth, 2015), answering scientific questions against text (Khot et al., 2015) or diagrams (Krishnamurthy et al., 2016; Sachan and Xing, 2017), among many others. Since the main theme in this dissertation is of QA systems that answer questions against a paragraph of text, in the rest of this section we will only discuss relevant approaches and datasets.

Reading comprehension (RC) is a *task* in which an agent is provided with a natural language paragraph as context, and it is required to answer questions about the paragraph. Such a format allows for testing the short text comprehension capability of an agent, and is widely used as a benchmark for testing comprehension capability of students in schools. Richardson et al. (2013) introduced the **M**achine **C**omprehension Test (MCTest) dataset as a way to test the language comprehension capability of machines, and since then this format has been widely adopted in NLP. Such a format allows for testing the machine's capability to understand diverse linguistic phenomena, such as predicate-argument structure, coreference, discourse, pragmatics, etc., in a unified but relatively controlled environment. As discussed above, QA is a format is which the reasoning phenomena that a model might have to deal with has an unbounded scope. As a result, the range of linguistic, semantic, and world knowledge concepts that the model needs deal with is a function of the training and evaluation data. For example, the MCTest dataset was carefully designed with short fictional stories restricted to words and concepts that a 7 year old could understand. Thus, while being open-domain in nature, a system developed for this data only needs to understand simple world concepts described in simple language, and does not need to worry about full complexity of the real world. On the other hand, SQuAD (Rajpurkar et al., 2016), which is the first large-scale RC dataset containing over 100,000 questions, contains paragraphs obtained from Wikipedia on a diverse range of topics. Due to the semi-automatic method of collecting these paragraphs, there is no guarantee on the complexity of language and concepts that a model needs to deal with in order to perform well on this dataset. However, over the years it has been realized that a model only needs to find a relevant sentence in the passage and perform simple predicate-argument extraction to answer questions in SQuAD. With the

rise of data-intensive neural methods, many large-scale reading comprehension datasets have been developed. They span a wide variety of domains and use-cases; for example, diagnostic datasets to measure progress in NLP (bAbI; Weston et al., 2016), datasets to answer trivia questions (TriviaQA; Joshi et al., 2017), questions from middle- and high-school examinations (RACE; Lai et al., 2017), questions asked by humans as web-search queries (NaturalQuestions; Kwiatkowski et al., 2019), or datasets containing questions that require complex compositional and symbolic reasoning (DROP; Dua et al., 2019). In this dissertation, we will apply our techniques and approaches mainly to DROP.

2.2.1 Approaches to Reading Comprehension

Here we will briefly discuss different approaches to reading comprehension that have been developed over the years. The factors that influenced the complexity (or expressiveness) of the model depended on two main factors; (a) the state of the art in NLP at the time, and (b) the complexity of the question, context text, and output in the dataset for which the approach was developed. Consequently, as for the rest of NLP, reading comprehension models also saw a change from statistical techniques, to neural models, to large-scale pre-trained language models that are currently state of the art.

Latent alignment models The initially successful RC models were based on modeling QA inference procedure via latent alignment between the question, context and candidate answer. One of the most promising approaches for the MCTest dataset (Richardson et al., 2013) was the answer-entailing structures proposed by Sachan et al. (2015). It used rhetorical structure theory, and event and entity coreference to form a semi-structured representation of the context (story), and learned to form a latent alignment between this representation and a hypothesis statement to score the later. Different hypothesis statements were constructed by converting the question and different candidate answer pairs into a declarative sentence which were scored using the latent alignment. Similar framework of inference via latent alignment between semi-structured representations of text have been further explored to answer scientific questions against semi-structured tables (Khashabi et al., 2016) and paragraphs of natural language text (Khashabi et al., 2018).

Neural approaches With the rise of neural methods in NLP, several end-to-end neural-network based RC models were proposed. These models are called end-to-end since they do not require intermediate linguistic structures (e.g. entity coreference) during learning and inference. On a high-level, all these models use RNNs (LSTMs or GRUs) to represent the context and the question, and differ in the manner they architecture intermediate layers to model complex interaction between the contextual representations of the context and question words. At the end, an output layer is used to predict a distribution over the possible answers. For example, distribution over entity mentions in the context in CNN & DailyMail dataset (Hermann et al., 2015) or distribution over spans in SQuAD (Rajpurkar et al., 2016). The intermediate layers mainly use the attention mechanism (Bahdanau et al., 2015) between the question and context in different ways to compute intermediate representations. Some of the more popular approaches in this realm are BiDAF (Seo et al., 2017), Attention-over-attention (Cui et al., 2017), and Gated-attention reader (Dhingra et al., 2017).

Large pre-trained Language Models The complete space of NLP has been changed in the last 2-3 years with the development of large-scale pre-trained language models to get contextualized word representations. ELMo (Peters et al., 2018) showed that using contextualized word representations from a pre-trained language model with BiDAF (or other RC architectures) significantly improves performance as compared to using static pre-trained word embeddings (such as GloVe (Pennington et al., 2014)). Since then, even larger pre-trained models for contextualized text representations, such as BERT (Devlin et al., 2019), BART (Lewis et al., 2020), T5 (Raffel et al., 2020), etc., have shown further significant improvements in many language understanding tasks, RC included. These are different from ELMo in that the same model architecture can be used for many different tasks; the only architectural change is the thin output layer that takes as input the contextualized word representations and produces a distribution over the output support (e.g., answer candidates in RC, topics is document classification, etc.). For example, when using BERT as a RC model for SQuAD, the output layer has a single feed-forward layer to convert the contextualized word representations into two distributions over the tokens, each representing the probability of the token being the start or end point of the span answer.

2.3 Neural Module Networks

In most RC datasets, the questions that have been tackled are simple, in the sense, usually involve selecting the retrieving a relevant sentence from the passage, and extracting its predicate-argument structure to find the answer. Now, consider the image in Figure 4[2] and the question *What is the color of the sphere?* - this is analogous to the RC question in the sense that it requires finding a relevant sub-part in the context (sphere) and determining one of its argument (color). Consider another question, *What color is the thing with the same size as the blue cylinder?* - this question is more complex than the previous one and requires multiple steps of *reasoning*: finding the blue cylinder, determining its size, finding the other object with the same size, and finally determining the color of this object. The exact computation required to answer such compositional questions is dependent on the question itself, something which the standard fixed neural network architecture approach does not account for. On the other hand, semantic parsing models are designed to handle exactly this kind of compositionality in computation, but the output logical forms can be executed on structured knowledge sources.

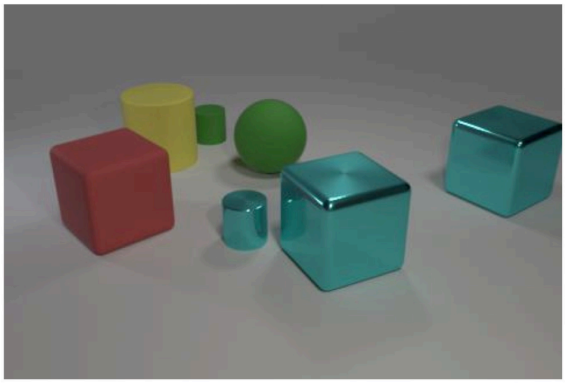


Figure 4: Example image from the CLEVR dataset containing multiple objects with different attributes.

Neural module networks (NMNs; Andreas et al., 2016) is a class of models that combines the compositional nature of logical meaning representations and the expressive power of neural networks to answer complex, compositional questions against unstructured contexts, such as images and natural

[2]Image from the CLEVR dataset (Johnson et al., 2017)

language. Just like semantic parsing, the idea in NMNs is to parse a given question into a logical meaning representation that defines the computation required to answer the question. The difference here is that, the logical form (or program) needs to be executed against an unstructured context to reach the denotation. The logical language used in NMNs, i.e., the set of predicates/functions used to produce programs are usually defined by the user in a variable-free language depending on the domain for which the NMN is being developed. Typically, these predicates are the basic operations required in the domain and can be combined to perform complex reasoning. For example, *What color is the thing with the same size as the blue cylinder?* will be parsed into the program `color(same-size(find[`*blue cylinder*`]))` where the program is composed of three basic operations - `find` which locates the *blue cylinder*, `same-size` that locates the object of the same size as the blue cylinder, and `color` which determines the color of the object located in the previous step.

Modules In contrast to standard semantic parsing, where the program is executed against a structured knowledge base, in NMNs, the program is executed against an unstructured knowledge source. Therefore, the execution of functions in a NMN cannot be hard-coded a priori in the same manner as standard semantic parsing. For example, the execution of the predicate `isState(s)` to check if the atom `s` in a knowledge-graph is a *state* or not can be hard-coded, but the execution of a predicate `find` to find the *blue cylinder* in an image cannot be pre-coded. NMNs propose to implement the execution of these predicates as independent neural networks, called modules, to leverage the representation power of neural networks. Therefore, the program that the question gets parsed into, defines the structure in which the relevant modules are arranged to solve the question.

Consider the two question in Figure 5[3] - their respective programs arrange the relevant modules in the required manner and effectively form different networks to solve the questions. Since the `compare-size` module is shared between the programs, those parameters would be shared in the final network.

Learning A neural module network is built with two components, both containing learnable parameters - (1) a semantic parser that models $p(z|x;\theta_z)$, the distribution over programs for a given

[3]Figure from `https://bair.berkeley.edu/blog/2017/06/20/learning-to-reason-with-neural-module-networks/`

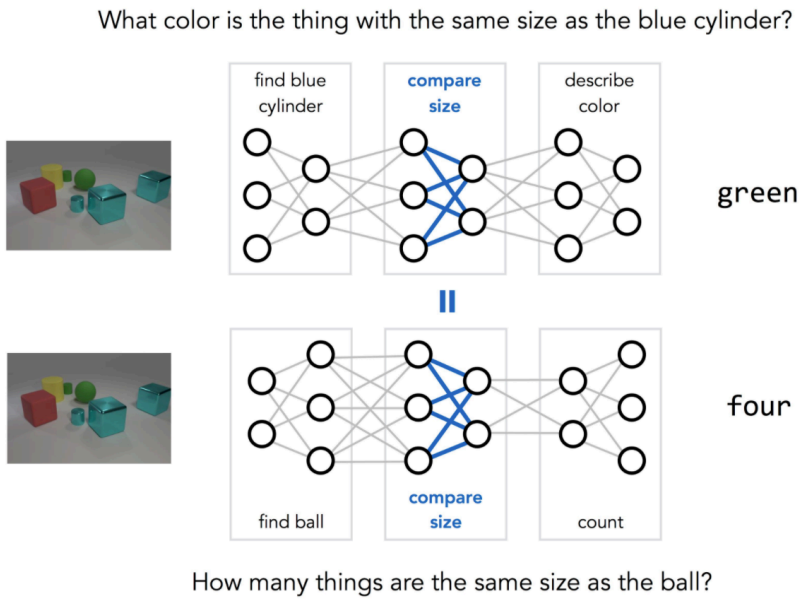


Figure 5: Example of NMN computation graphs

question x, and (2) an execution model that models $p(y|x, z; \theta_m)$, the distribution over answer candidates for the given program z. The parameters of the execution model θ_m are the parameters of all the modules combined, though, only the parameters of the modules occurring in z take part in modeling the answer distribution for a given question.

Given weak supervision in terms of (question, answer) pairs, the idea in NMNs is to jointly learn the parameters of the parser and the modules. Via marginalizing over all possible programs, we can write

$$p(y|x; \theta_z, \theta_m) = \sum_z p(z|x; \theta_z) * p(y|x, z; \theta_m) \tag{2.9}$$

We need to optimize for two sets of parameters, θ_z and θ_m. Optimizing for the parameters of the parser θ_z is similar to weakly-supervised semantic parsing only that the reward $R(z)$ now is $p(y|x, z; \theta_m)$. The original NMN work (Andreas et al., 2016) and subsequent followups (Hu et al., 2017) use policy-gradients, specifically the REINFORCE algorithm (Williams, 1992), to carry out this optimization. Optimizing for the module parameters θ_m can be done via backpropagation since the neural modules compose to result in a differentiable network, just like any standard neural network. See Andreas et al. (2016) for details.

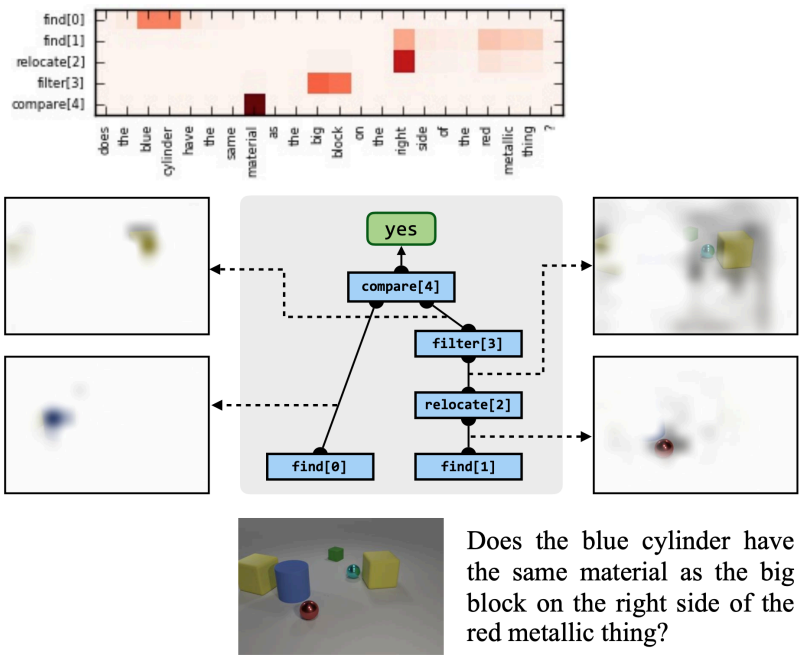


Figure 6: Example prediction of a neural module network trained on the CLEVR dataset

Benefits of NMNs A NMN has an explicit structural bias towards solving a problem in a compositional manner, and hence, is well suited for problems which can be decomposed into simpler problems whose solution can be combined to reach the final output. Such compositional structural bias is also expected to generalize better to novel problem instances that are new compositions of basic concepts observed in training. The separation of model parameters into modules, the use of only relevant modules when solving a problem, and explicit sharing of parameters when solving similar sub-problems (via reusing modules), all should lead to better sample complexity while learning. Also, the explicit parsing of the question into a logical program and the intermediate outputs of the modules during execution, both offer a great deal of interpretability in the model's decision making process. In Figure 6 (from Hu et al. (2017)) we see an example of the level of interpretability provided by a NMN.

Chapter 3

Neural Module Networks for Reasoning over Text

This chapter presents a modular approach to answer complex, compositional questions against a paragraph of text. Developing such a model poses significant challenges of understanding the question semantics, extracting various information from the provided passage, and performing symbolic reasoning to compose the extracted information in the required manner. Inspired by semantic parsing and neural module networks (NMNs; Andreas et al., 2016), our proposed model is modular in nature which provides the required architectural bias to perform question decomposition and reasoning in a compositional manner. We design learnable neural modules to perform atomic language understanding and symbolic reasoning tasks, that can be combined to perform multi-step reasoning. These modules are designed in a differentiable manner to maintain end-to-end differentiability which allows us to train this model via backpropagation using just question-answer supervision. This chapter is based on work originally described in Gupta et al. (2020a).

3.1 Introduction

Consider the question *Who kicked the longest field goal in the second quarter?* in Figure 7. Multiple reasoning steps are needed to answer such a question: find all instances of "field goal" in the paragraph, select the ones "in the second quarter", find their lengths, compute the "longest" of them, and then find "who kicked" it. We would like to develop machine reading models that are capable of understanding the context and the compositional semantics of such complex questions in order to provide the correct answer, ideally while also explaining the reasoning that led to that answer.

Semantic parsing techniques, which map natural language utterances to executable programs, have been used for compositional question understanding for a long time (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Liang et al., 2011), but have been limited to answering questions against structured and semi-structured knowledge sources. Neural module networks (NMNs; Andreas et al., 2016) extend semantic parsers by making the program executor a *learned function* composed of neural network modules. NMNs perform well on synthetic visual question answering (VQA) domains such as CLEVR (Johnson et al., 2017) and it is appealing to apply them to answer questions over text due to their interpretable, modular, and inherently compositional nature. However, it is non-trivial to extend NMNs for answering non-synthetic questions against open-domain text, where a model needs to deal with the ambiguity and variability of real-world text while performing a diverse range of reasoning.

In this chapter we extend NMNs to answer compositional questions against a paragraph of text as context. We introduce neural modules to perform atomic language understanding operations over text using distributed representations, and perform symbolic reasoning, such as arithmetic, sorting, comparisons, and counting. The modules we define are probabilistic and differentiable, which lets us maintain uncertainty about intermediate decisions and train the entire model via end-to-end differentiability.

The model needs to correctly make multiple intermediate decisions to arrive at the answer which is challenging to learn only using end-task QA supervision. We show that these challenges can be alle-

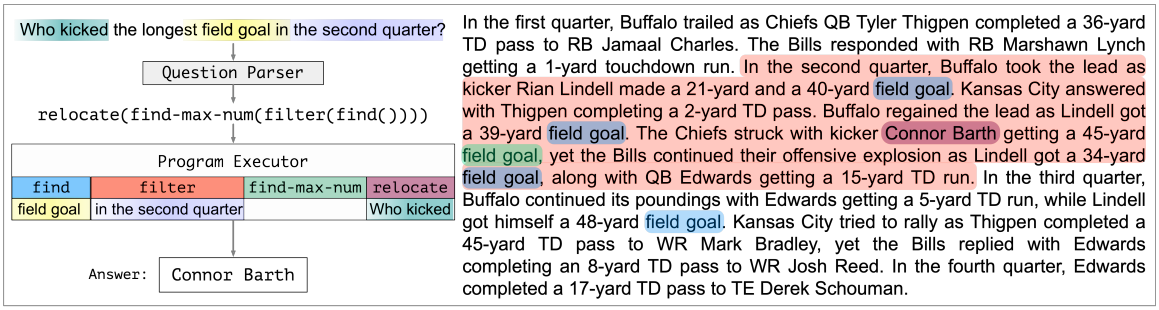


Figure 7: Model Overview: Given a question, our model parses it into a program composed of neural modules. This program is executed against the context to compute the final answer. The modules operate over soft attention values (on the question, passage, numbers, and dates). For example, `filter` takes as input attention over the question (*in the second quarter*) and filters the output of the `find` module by producing an attention mask over tokens that belong to the *second quarter*.

viated with auxiliary objectives over the intermediate latent decisions of the model. Specifically, we introduce an unsupervised objective that provides an inductive bias to perform accurate information extraction from the context. Additionally, we show that providing heuristically-obtained supervision for question programs and outputs for intermediate modules in a program for a small subset of the training data (5–10%) is sufficient for accurate learning. Experiment on the DROP dataset (Dua et al., 2019) demonstrates the feasibility of our approach.

3.2 Overview of Approach

Our goal is to develop a question-answering model that given a question q against a paragraph p predicts the correct answer y. The answer in our case can be a span of text from the passage or a number. We extend NMNs to perform question answering against text. A NMN is composed of two main components; a semantic parser to map the question q into a logical meaning representation or program z, and an executor made up of modules to execute this program z against some representation of the context p to arrive at the answer y.

Consider the question *"Who kicked the longest field goal in the second quarter?"* in Figure 7. Our NMN would parse this question into an executable program, such as,

```
extract-argument(find-max-num(filter(find())))        (3.1)
```

whose execution against the given paragraph yields the answer. These programs capture the abstract compositional reasoning structure required to answer the question correctly and are composed of learnable modules designed to solve sufficiently independent reasoning tasks. For example, the `find` module should ground the question span "field goal" to its various occurrences in the paragraph; the module `find-max-num` should output the span amongst its input that is associated with the largest length; and finally, the `extract-argument` module should find "who kicked" the *field goal* corresponding to its input span.

3.2.1 Components of a NMN for Text

Modules To perform natural language and symbolic reasoning operations over different types of information, such as text, numbers, and dates, we define a diverse set of differentiable modules to operate over these different data types. We describe these modules and the data types in §3.3.

Contextual Token Representations Our model represents the question q as $\mathbf{Q} \in \mathbb{R}^{n \times d}$ and the context paragraph p as $\mathbf{P} \in \mathbb{R}^{m \times d}$ using contextualized token embeddings. These are outputs of either the same bidirectional-GRU or a pre-trained BERT (Devlin et al., 2019) model. Here n and m are the number of tokens in the question and the paragraph, respectively, and d is the dimensionality of the embeddings.

Question Parser We use an encoder-decoder model with attention to map the question into an executable program. Similar to end-to-end NMN (N2NMN; Hu et al., 2017), at each timestep of decoding, the attention that the parser puts on the question is available as a side argument to the module produced at that timestep during execution. This lets the modules have access to question information without making hard decisions about which question words to put into the program.

In our model, the data types of the inputs and output of modules automatically induce a type-constrained grammar which lends itself to top-down grammar-constrained decoding as performed by Krishnamurthy et al. (2017). This ensures that the decoder always produces well-typed programs. For example, if a module f_1 inputs a *number*, and f_2 outputs a *date*, then $f_1(f_2)$ is invalid and

would not be explored while decoding. The output of the decoder is a linearized abstract syntax tree (in an in-order traversal). Please refer to the background chapter (§2.1.2) for details about the grammar-constrained decoder.

Learning We define our model probabilistically, i.e., for any given program z, we can compute the likelihood of the gold-answer $p(y^*|z)$. Combined with the likelihood of the program under the question-parser model $p(z|q)$, we can maximize the marginal likelihood of the answer by enumerating all possible programs.

$$J = \sum_{z} p(y^*|z)p(z|q) \tag{3.2}$$

Since the space of all programs is intractable, we run beam search to enumerate top-K programs and maximize the approximate marginal-likelihood. Owing to type-constrained decoding, given the gold answer, we only explore programs that would yield an answer of the gold type. For example, if the answer is a number, we do *not* search for programs that would yield a string answer, i.e., we only search for programs in which the outermost function returns a *number*-type output.

3.2.2 Learning Challenges in NMN for Text

As mentioned above, the question parser and the program executor both contain learnable parameters. Each of them is challenging to learn in its own right and joint training further exacerbates the situation.

Question Parser Our model needs to parse free-form real-world questions into the correct program structure and identify its arguments (e.g., "who kicked", "field goal", etc.). This is challenging since the questions are not generated from a small fixed grammar (unlike CLEVR), involve lexical variability, and have no program supervision. Additionally, many incorrect programs can yield the same correct answer thus training the question parser to highly score incorrect interpretations. Such an incorrect training signal is adversarial in nature for the question parser.

Program Executor The output of each intermediate module in the program is a latent decision by the model since the only feedback available is for the final output of the program. The absence

of any direct feedback to the intermediate modules complicates learning since the errors of one module would be passed on to the next. Differentiable modules that propagate uncertainties in intermediate decisions help here, such as attention on pixels in CLEVR, but do not fully solve the learning challenges.

Joint Learning Jointly training the parser and executor increases the latent choices available to the model by many folds while the only supervision available is the gold answer. Specifically, for each program in the program search space, there are *multiple* intermediate module outputs that are possible that will yield the correct answer. Among this large space of latent configurations, only *one* (ideally) is correct but discovering the correct program and intermediate outputs is a hard learning challenge. Additionally, joint learning is challenging as prediction errors from one component will provide incorrect training signal to the other component, in turn leading to incorrect feedback from that component to the first. This forms a cycle of incorrect feedback that hurts learning. E.g., if the parser predicts the program `extract-argument(find())` for the question in Fig. 7, then the associated modules would be incorrectly trained to predict the gold answer. On the next iteration, incorrect program execution would provide the wrong feedback to the question parser and lead to its incorrect training, and learning fails.

3.3 Modules for Reasoning over Text

Modules are designed to perform basic independent reasoning tasks and form the basis of the compositional reasoning that the model is capable of. We identify a set of tasks that need to be performed to support diverse enough reasoning capabilities over text, numbers, and dates, and define modules accordingly. Since the module parameters will be learned jointly with the rest of the model, we would like the modules to maintain uncertainties about their decisions and propagate them through the decision making layers via end-to-end differentiability. One of our main contributions is introducing differentiable modules that perform reasoning over text and symbols in a probabilistic manner.

We first discuss how we make our design choices for the modules to use and the data types they operate over. Then, we present the specifics of the data types and their respective representations on

which our modules operate. Finally, we describe the various modules and how they are implemented. Table 1 gives an overview of the representative modules in our model.

3.3.1 Design Choices for Modules and Data Types

We perform manual analysis of around 200 questions from the dataset we work with (DROP; Dua et al., 2019) and identify their decomposition in terms of basic reasoning tasks that need to be performed to answer these questions. These basic tasks guide our choice of modules to define, which are typed-functions that operate over text, numbers, and dates. The idea is that the composition of these modules should support diverse reasoning capabilities required to answer the questions. The format of input and output required by the modules guides our choice of the data-types to define.

Using few questions as motivating examples, let us look at some of the basic tasks we identify and a high-level sketch of the corresponding modules. Consider the questions, Q_1: *What happened last, commission being granted to Robert or death of his cousin?*, Q_2: *Who scored the longest field goal?*, Q_3: *How many touchdowns were scored in the third quarter?*, and Q_4: *Were there more of cultivators or main agricultural labourers in Sweden?*.

1. The most common task associated to *all* questions is grounding a question span (e.g., concept, entity, event) to relevant span(s) in the associated paragraph. For example, finding the mention(s) of events like *touchdowns*, *field goals*, *commission being granted to Robert*, etc. and grounding to the mention(s) of entities like *cultivators*, *main agricultural labourers*, etc. in the paragraph. To perform this task we need to define a module, say `find`, with a signature like `find(question-span)` $\rightarrow$ `Set[passage-span]`, to output the set of passage spans found relevant to the input question span.

2. Consider question Q_3 where after finding the set of *touchdown* mentions, this set needs to be pruned to the ones *scored in the third quarter*. For a filtering task like such, where a set of mentions needs to be pruned based on a condition specified in the question, we define a module with a signature `filter(Set[passage-span], question-span)` $\rightarrow$ `Set[passage-span]`.

3. Another common basic task that is required for almost all questions is extracting an argument (string, number, or date) associated to the entity/event(s) being queried about in the question. For example the following arguments need to be extracted in the questions above—Q_1: the date when the two events in the question took place, Q_2: the length (number) associated with each *field goal* and the *who scored* argument associated to the *longest field goal*, and Q_4: the number of *cultivators* and *main agricultural labourers* mentioned in the paragraph. For such operations, we define three modules:
   `extract-argument(Set[passage-span], question-span)` $\rightarrow$ `Set[passage-span]`,
   `find-num(Set[passage-span])` $\rightarrow$ `Set[number]`, and
   `find-date(Set[passage-span])` $\rightarrow$ `Set[date]`.

4. Other common basic tasks we identify require discrete symbolic operations; such as, computing the *maximum*/*minimum* number among a set of numbers (e.g., for finding the *longest field goal* in Q_2), comparing two numbers/dates and checking which one is smaller/greater (e.g., comparing dates and numbers in Q_1 and Q_4, respectively), and counting the set of relevant passage spans that are identified (e.g., in Q_3).

3.3.2 Discrete vs. Continuous Representations

In the previous section we saw that the modules need to operate over inputs and outputs of various types—e.g., `question-span`, `Set[passage-span]`, `Set[number]` and `Set[date]`. Also, note that the modules are functions with learnable parameters initialized from scratch; they are expected to learn their intended task behavior from end-task supervision. Since all modules are trained jointly, we would like to maintain uncertainties in a module's predictions and propagate them through the decision making process to aid module parameter learning from end-task supervision. Therefore, we need to represent the values that the different data types can take in a probabilistic manner. We also need to be able to perform discrete operations, such as *min/max*, *count* and *comparisons*, in a probabilistic manner. To facilitate these requirements in a computationally efficient manner, there are certain design decisions that need to be taken in terms of how to represent values of the data types and how to design modules. We will discuss these issues in this section.

Let us first consider the option of a discrete probabilistic representation for values of the data types. Consider the type `Set[passage-span]` – a paragraph with T tokens will contain a total number of $\frac{T(T+1)}{2}$ or $\mathcal{O}(T^2)$ spans. Even if we consider spans with a maximum length L, the total number of spans are $L \cdot T$. For a passage of length $T = 500$ and with $L = 20$, the total number of spans is still quite a large number, $L \cdot T = 10000$. Representing a discrete distribution over the power-set of these spans as the underlying sample space is computationally expensive; the size of this space $\mathcal{O}(2^{L \cdot T})$ is exponential in nature.

Further, such a discrete representation also introduces combinatorial search issues while training when used in conjunction with discrete operations, especially in the absence of any supervision for the output of the modules. For example, no supervision for the set of passage-spans that should be output by a `find` module operations. Consider a program, `max(find-num)`, where the `find-num` outputs a discrete distribution over the power set of all numbers present in the passage, and `max` outputs the maximum value in any particular set of numbers input to it. Given the gold output of this program, let's say a single number x that should be output after the `max` operation, based on this feedback we will like to improve the model's estimate of the underlying distribution that is output by the `find-num` operation. Now, since the `max` module operates on a discrete distribution over sets, we will need to enumerate all sets for which the `max` module can yield the output x, i.e., all possible sets that contain the number x, and update the distribution accordingly. A similar issue will occur in a program like `count(find)` where `find` outputs a discrete distribution over the power set of all possible passage spans and the only feedback available is the size of *gold* set that should be output by `find`. Therefore, due to the issues of exponential space and search complexity, we cannot represent data type values as discrete probability distributions.

One way to mitigate this issue is to represent data type values as compact continuous-value distributions. Further, by performing discrete operations in a continuous and differentiable manner, we will be able to directly pass gradients to the underlying input distribution without incurring combinatorial search costs. To handle the issue of exponential size of the power set of a set of instances, we instead represent sets as a probability distribution over the underlying instances. By doing so, we reduce the

space complexity of representing distributions over power sets from $\mathcal{O}(2^{|S|})$ to $\mathcal{O}(|S|)$ where $|S|$ is the number of underlying instances. All discrete and continuous operations are then conducted over this surrogate representation of sets. For example, to represent `Set[passage-span]`, we model a distribution over passage tokens where the idea is that a contiguous span of high probability values is indicative of a potential span. Using such a representation, we are able to probabilistically represent values of the type `Set[passage-span]` with a vector of size $\mathcal{O}(T)$ instead of $\mathcal{O}(2^{L \cdot T})$.

One drawback of using continuous probabilistic representations for inherently discrete data types is the need to define ways to perform discrete operations in a continuous and differentiable manner. In this work, we are able to define analytical solutions to perform operations like *minimum*/*maximum* and *comparisons* in a differentiable manner. On the other hand, we needed to define modules with learnable parameters to perform operations like counting the number of underlying spans and figuring out discrete spans from a continuous distribution over tokens. We give details about the data types, their representations, and the modules in the subsequent sections.

Module	In	Out	Task
`find`	`Q`	`P`	For question spans in the input, find similar spans in the passage
`filter`	`Q, P`	`P`	Based on the question, select a subset of spans from the input
`extract-argument`	`Q, P`	`P`	Find the argument asked for in the question for input paragraph spans
`find-num`	`P`	`N`	} Find the number(s) / date(s) associated to the input paragraph spans
`find-date`	`P`	`D`	
`count`	`P`	`C`	Count the number of input passage spans
`compare-num-lt`	`P, P`	`P`	Output the span associated with the smaller number.
`time-diff`	`P, P`	`TD`	Difference between the dates associated with the paragraph spans
`find-max-num`	`P`	`P`	Select the span that is associated with the largest number
`span`	`P`	`S`	Identify a contiguous span from the attended tokens

Table 1: Description of the modules we define and their expected behaviour. All inputs and outputs are represented as distributions over tokens, numbers, and dates as described in §3.3.3.

3.3.3 Data Types

The modules operate over the following data types. Each data type represents its underlying value as a normalized distribution over the relevant support.

- **Question (`Q`) and Paragraph (`P`) attentions:** soft subsets of relevant tokens in the text.
- **Number (`N`) and Date (`D`):** soft subset of unique numbers and dates from the passage.

We pre-process the paragraphs to extract the numbers and dates in them. For numbers, we use a simple strategy where all tokens in the paragraph that can be parsed as a number are extracted. For example, 200 in "200 women". The total number of number-tokens in the paragraph is denoted by N_{tokens}. We do not normalize numbers based on their units and leave it for future work.

To extract dates from the paragraph, we run the spaCy-NER[1] and collect all DATE mentions. To normalize the date mentions we use an off-the-shelf date-parser[2]. For example, a date mention "19th November, 1961" would be normalized to $(19, 11, 1961)$ (day, month, year). The total number of date-tokens is denoted by D_{tokens}.

- **Count Number (`C`):** count value as a distribution over the supported count values $(0 - 9)$.
- **Time Delta (`TD`):** a value amongst all possible unique differences between dates in the paragraph. In this work, we consider differences in terms of years.
- **Span (`S`):** span-type answers as two probability values (start/end) for each paragraph token.

3.3.4 Neural Modules for Question Answering

The question and paragraph contextualized embeddings (**Q** and **P**) are available as global variables to all modules in the program. The question attention computed by the decoder during the timestep the module was produced is also available to the module as a side argument, as described in §3.2.1.

`find(Q)` $\rightarrow$ `P` This module is used to ground attended question tokens to similar tokens in the paragraph (e.g., "field goal" in Figure 7). We use a question-to-paragraph attention matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ whose i-th row is the distribution of similarity over the paragraph tokens for the i-th question token. The output is an *expected* paragraph attention; a weighted-sum of the rows of **A**, weighed by the input question attention.

$$P = \sum_i Q_i \cdot \mathbf{A}_{i:} \in \mathbb{R}^m$$

A is computed by normalizing (using softmax) the rows of a question-to-paragraph similarity matrix $\mathbf{S} \in \mathbb{R}^{n \times m}$. Here $\mathbf{S}_{ij}$ is the similarity between the contextual embeddings of the i-th question token

[1] https://spacy.io/
[2] https://github.com/scrapinghub/dateparser

and the j-th paragraph token computed as,

$$\mathbf{S}_{ij} = \mathbf{w_f}^T[\mathbf{Q}_{i:}; \mathbf{P}_{j:}; \mathbf{Q}_{i:} \circ \mathbf{P}_{j:}]$$

where $\mathbf{w_f} \in \mathbb{R}^{3d}$ is a learnable parameter vector of this module, $[;]$ denotes the concatenation operation, and $\circ$ is element-wise multiplication.

`filter(Q, P)` $\rightarrow$ `P` This module masks the input paragraph attention conditioned on the question, selecting a subset of the attended paragraph (e.g., selecting fields goals "in the second quarter" in Fig. 7). We compute a *locally-normalized* paragraph-token mask $M \in \mathbb{R}^m$ where M_j is the masking score for the j-th paragraph token computed as

$$M_j = \sigma(\mathbf{w_{filter}}^T[\mathbf{q}; \mathbf{P}_{j:}; \mathbf{q} \circ \mathbf{P}_{j:}])$$

Here $\mathbf{q} = \sum_i Q_i \cdot \mathbf{Q}_{i:} \in \mathbb{R}^d$, is a weighted sum of question-token embeddings, $\mathbf{w_{filter}}^T \in \mathbb{R}^{3d}$ is a learnable parameter vector, and σ is the *sigmoid* non-linearity function. The output is a normalized masked input paragraph attention, $P_{\text{filtered}} = \text{normalize}(M \circ P)$.

`extract-argument(Q, P)` $\rightarrow$ `P` This module re-attends to the paragraph based on the question and is used to find the arguments for paragraph spans (e.g., shifting the attention from "field goals" to "who kicked" them). We first compute a paragraph-to-paragraph attention matrix $\mathbf{R} \in \mathbb{R}^{m \times m}$ based on the question, as

$$\mathbf{R}_{ij} = \mathbf{w_{relocate}}^T[(\mathbf{q} + \mathbf{P}_{i:}); \mathbf{P}_{j:}; (\mathbf{q} + \mathbf{P}_{i:}) \circ \mathbf{P}_{j:}]$$

where $\mathbf{q} = \sum_i Q_i \cdot \mathbf{Q}_{i:} \in \mathbb{R}^d$, and $\mathbf{w_{relocate}} \in \mathbb{R}^{3d}$ is a learnable parameter vector. Each row of $\mathbf{R}$ is also normalized using the softmax operation. The output attention is a weighted sum of the rows $\mathbf{R}$ weighted by the input paragraph attention, $P_{\text{relocated}} = \sum_i P_i \cdot \mathbf{R}_{i:}$

`find-num(P)` $\rightarrow$ `N` This module finds a number distribution associated with the input paragraph attention. We use a paragraph token-to-number-token attention map $\mathbf{A}^{\text{num}} \in \mathbb{R}^{m \times N_{\text{tokens}}}$ whose i-th row is probability distribution over number-containing tokens for the i-th paragraph token. We first compute a token-to-number similarity matrix $\mathbf{S}^{\text{num}} \in \mathbb{R}^{m \times N_{\text{tokens}}}$ as,

$$\mathbf{S}^{\text{number}}{}_{i,j} = \mathbf{P}_{i:}^{T} \mathbf{W}_{\text{number}} \mathbf{P}_{n_j:}$$

where n_j is the index of the j-th number token and $\mathbf{W}_{\text{num}} \in \mathbb{R}^{d \times d}$ is a learnable parameter. Normalizing the rows of $\mathbf{S}^{\text{number}}$ using *softmax* yields $\mathbf{A}^{\text{number}}$, $\mathbf{A}^{\text{num}}{}_{i:} = \text{softmax}(\mathbf{S}^{\text{num}}{}_{i:})$. We compute an *expected* distribution over the number tokens $T = \sum_i P_i \cdot \mathbf{A}^{\text{num}}{}_{i:}$ and aggregate the probabilities for number-tokens with the same value to compute the output distribution N. For example, if the values of the number-tokens are [*2*, *2*, *3*, *4*] and $T = [0.1, 0.4, 0.3, 0.2]$, the output will be a distribution over $\{2, 3, 4\}$ with $N = [0.5, 0.3, 0.2]$.

`find-date(P)` $\rightarrow$ `D` follows the same process as above to compute a distribution over dates for the input paragraph attention. The corresponding learnable parameter matrix is $\mathbf{W}_{\text{date}} \in \mathbb{R}^{d \times d}$.

`count(P)` $\rightarrow$ `C` This module is used to count the number of attended paragraph spans. The idea is to learn a module that detects contiguous spans of attention values and counts each as one. For example, if an attention vector is $[0, 0, 0.3, 0.3, 0, 0.4]$, the count module should produce an output of 2. The module first scales the attention using the values $[1, 2, 5, 10]$ to convert it into a matrix $P_{\text{scaled}} \in \mathbb{R}^{m \times 4}$. A bidirectional-GRU then represents each token attention as a hidden vector h_t. A single-layer feed-forward network maps this representation to a soft 0/1 score to indicate the presence of a span surrounding it. These scores are summed to compute a count value, $c_v = \sum \sigma\left(FF(\text{countGRU}(P_{\text{scaled}}))\right) \in \mathbb{R}$. We hypothesize that the output count value is normally distributed with c_v as mean, and a constant variance $v = 0.5$, and compute a categorical distribution over the supported count values, as $p(c) \propto \exp(^{-(c-c_v)^2}/_{2v^2}) \;\; \forall c \in [0, 9]$. We find that training the `count` module is challenging and pre-training the parameters of the `count` module helps.

Pretraining `count` module: We generate synthetic data to pre-train this module; each instance is a

normalized-attention vector $x = \mathbb{R}^m$ and a count value $y \in [0, 9]$. This is generated by sampling m uniformly between $200 - 600$, then sampling a count value y uniformly in $[0, 9]$. We then sample y span-lengths between $5 - 15$ and also sample y non-overlapping span-positions in the attention vector x. For all these y spans in x, we put a value of 1.0 and zeros everywhere else. We then add 0-mean, 0.01-variance gaussian noise to all elements in x and normalize to make the normalized attention vector that can be input to the count module. We train the parameters of the count module using these generated instances using L_2-loss between the true count value and the predicted c_v.

`compare-num-lt(P1, P2)` $\rightarrow$ `P` This module performs a soft less-than operation between two passage distributions. For example, to find *the city with fewer people, cityA or cityB*, the module would output a linear combination of the two input attentions weighted by which city was associated with a lower number. This module internally calls the `find-num` module to get a number distribution for each of the input paragraph attentions, N_1 and N_2. It then computes two soft boolean values, $p(N_1 < N_2)$ and $p(N_2 < N_1)$, and outputs a weighted sum of the input paragraph attentions. The boolean values are computed by marginalizing the relevant joint probabilities:

$$p(N_1 < N_2) = \sum_i \sum_j \mathbb{1}_{N_1^i < N_2^j} N_1^i N_2^j \qquad p(N_2 < N_1) = \sum_i \sum_j \mathbb{1}_{N_2^i < N_1^j} N_2^i N_1^j$$

The final output is:

$$\mathsf{P}_{out} = p(N_1 < N_2) * \mathsf{P}_1 + p(N_2 < N_1) * \mathsf{P}_2$$

When the the predicted number distributions are peaky, $p(N_1 < N_2)$ or $p(N_2 < N_1)$ is close to 1, and the output is either P_1 or P_2.

We similarly include the comparison modules `compare-num-gt`, `compare-date-lt`, and `compare-date-gt`, defined in an essentially identical manner, but for greater-than and for dates.

`time-diff(P1, P2)` $\rightarrow$ `TD` The module outputs the difference between the dates associated with the two paragraph attentions as a distribution over all possible difference values. The module internally calls the `find-date` module to get a date distribution for the two paragraph attentions, D_1

and D_2. The probability of the difference being t_d is computed by marginalizing over the joint probability for the dates that yield this value, as

$$p(t_d) = \sum_{i,j} \mathbb{1}_{(d_i - d_j = t_d)} D_1^i D_2^j$$

In this work, date differences are computed as a difference between their year values.

`find-max-num(P)` $\rightarrow$ `P`, `find-min-num(P)` $\rightarrow$ `P` Given a passage attention attending to multiple spans, this module outputs an attention for the span associated with the largest (or smallest) number. We first compute an expected number token distribution T using `find-num`, then use this to compute the expected probability that each number token is the one with the maximum value, $T^{\text{max}} \in \mathbb{R}^{N_{\text{tokens}}}$ (explained below). We then re-distribute this distribution back to the original passage tokens associated with those numbers. The contribution from the i-th paragraph token to the j-th number token, T_j, was $P_i \cdot \mathbf{A}^{\text{num}}{}_{ij}$. To compute the new attention value for token i, we re-weight this contribution based on the ratio T_j^{max}/T_j and marginalize across the number tokens to get the new token attention value:

$$\bar{P}_i = \sum_j T_j^{\text{max}}/T_j \cdot P_i \cdot \mathbf{A}^{\text{num}}{}_{ij}$$

Computing T^{max}: Consider a distribution over numbers N, sorted in an increasing order. Say we sample a set S (size n) of numbers from this distribution. The probability that N_j is the largest number in this set is $p(x \leq N_j)^n - p(x \leq N_{j-1})^n$ i.e. all numbers in S are less than or equal to N_j, and at least one number is N_j. By picking the set size $n = 3$ as a hyperparameter, we can analytically (and differentiably) convert the expected distribution over number tokens, T, into a distribution over the maximum value T^{max}.

`span(P)` $\rightarrow$ `S` This module is used to convert a paragraph attention into a contiguous answer span and only appears as the outermost module in a program. The module outputs two probability distributions, P_s and $P_e \in \mathbb{R}^m$, denoting the probability of a token being the start and end of a span, respectively. This module is implemented similar to the `count` module. The input paragraph

attention is first scaled using $[1, 2, 5, 10]$, then a bidirectional-GRU represents each attention as a hidden vector, and a single-layer feed-forward network maps this to 2 scores, for span start and end. A softmax operation on these scores gives the output probabilities.

3.4 Auxiliary Supervision

As mentioned in §3.2.2, jointly learning the parameters of the parser and the modules using only end-task QA supervision is extremely challenging. To overcome issues in learning, (a) we introduce an unsupervised auxiliary loss to provide an inductive bias to the execution of `find-num`, `find-date`, and `relocate` modules; and (b) provide heuristically-obtained supervision for question program and intermediate module output for a subset of questions (5–10%).

3.4.1 Unsupervised Auxiliary Loss for IE

The `find-num`, `find-date`, and `extract-argument` modules perform information extraction by finding relevant arguments for entities and events mentioned in the context. In our initial experiments we found that these modules would often spuriously predict a high attention score for output tokens that appear far away from their corresponding inputs. We introduce an auxiliary objective to induce the idea that the arguments of a mention should appear near it. For any token, the objective increases the sum of the attention probabilities for output tokens that appear within a window $W = 10$, letting the model distribute the mass within that window however it likes. The objective for the `find-num` is

$$H^{\text{n}}_{\text{loss}} = -\sum_{i=1}^{m} \log\Big(\sum_{j=0}^{N_{\text{tokens}}} \mathbb{1}_{n_j \in [i \pm W]} \mathbf{A}^{\text{num}}{}_{ij}\Big)$$

We compute a similar loss for the date-attention map $\mathbf{A}^{\text{date}}$ ($H^{\text{d}}_{\text{loss}}$) and the argument-map $\mathbf{R}$ ($H^{\text{r}}_{\text{loss}}$). The final auxiliary loss is

$$H_{\text{loss}} = H^{\text{n}}_{\text{loss}} + H^{\text{d}}_{\text{loss}} + H^{\text{r}}_{\text{loss}} \tag{3.3}$$

3.4.2 Question Parse and Intermediate Module Output Supervision

Question Parse Supervision Learning to parse questions in a noisy feedback environment is very challenging. For example, even though the questions in CLEVR are programmatically generated, Hu et al. (2017) needed to pre-train their parser using external supervision for all questions. For the dataset we work with, DROP (Dua et al., 2019), we have no such external supervision. In order to bootstrap the parser, we analyze some questions manually and come up with a few heuristic patterns to get program and corresponding question attention supervision (for modules that require it) for a subset of the training data (10% of the questions; see Appendix A.1). For example, for program `find-num(find-max-num(find()))`, we provide supervision for question tokens to attend to when predicting the `find` module.

Intermediate Module Output Supervision. Consider the question, "how many yards was the shortest goal?". The model only gets feedback for how long the *shortest goal* is, but not for other *goals*. Such feedback biases the model in predicting incorrect values for intermediate modules (only the shortest goal instead of all in `find-num`) which in turn hurts model generalization.

We provide heuristically-obtained noisy supervision for the output of the `find-num` and `find-date` modules for a subset of the questions (5%) for which we also provide question program supervision. For questions like "how many yards was the longest/shortest touchdown?", we identify all instances of the token "touchdown" in the paragraph and assume the closest number to it should be an output of the `find-num` module. We supervise this as a multi-hot vector N^* and use an auxiliary loss, similar to question-attention loss, against the output distribution N of `find-num`. We follow the same procedure for a few other question types involving dates and numbers; see Appendix A.1 for details.

3.5 Experimental Setup

This section describes the DROP dataset used for training and evaluation, the details of our model, and the previous approaches from the literature used in the experiments for comparison.

3.5.1 Dataset

We perform experiments on a portion of the DROP dataset (Dua et al., 2019), which to the best of our knowledge is the only dataset that requires the kind of compositional and symbolic reasoning that our model aims to solve. Our model possesses diverse but limited reasoning capability; hence, we try to automatically extract questions in the scope of our model based on their first n-gram. These n-grams were selected by performing manual analysis on a small set of questions. The dataset we construct contains $20,000$ questions for training/validation, and 1800 questions for testing (25% of DROP). Since the DROP test set is hidden, this test set is extracted from the validation data. Though this is a subset of the full DROP dataset it is still a significantly-sized dataset that allows drawing meaningful conclusions.

Based on the manual analysis we classify these questions into different categories, which are:
Date-Compare e.g. *What happened last, commission being granted to Robert or death of his cousin?*
Date-Difference e.g. *How many years after his attempted assassination was James II coronated?*
Number-Compare e.g. *Were there more of cultivators or main agricultural labourers in Sweden?*
Extract-Number e.g. *How many yards was Kasay's shortest field goal during the second half?*
Count e.g. *How many touchdowns did the Vikings score in the first half?*
Extract-Argument e.g. *Who threw the longest touchdown pass in the first quarter?*

Auxiliary Supervision Out of the $20,000$ training questions, we provide question program supervision for 10% (2000), and intermediate module output supervision for 5% (1000) of training questions. We use curriculum learning (Bengio et al., 2009) where the model is trained only on heuristically-supervised non-count questions for the first 5 epochs.

3.5.2 Model Details

We implement our model using AllenNLP (Gardner et al., 2018).

Question and Paragraph Representation Our model represents the question q as $\mathbf{Q} \in \mathbb{R}^{n \times d}$ and paragraph p as $\mathbf{P} \in \mathbb{R}^{m \times d}$ using contextualized token embeddings. These embeddings are either produced using a multi-layer GRU network that is trained from scratch, or a pre-trained BERT model (Devlin et al., 2019) that is fine-tuned during training. GRU: We use a 2-layer, 64-dimensional (d = 128, effectively), bi-directional GRU. The same GRU is used for both, the question and the paragraph. The token embeddings input to the contextual encoder are a concatenation of 100-d pre-trained GloVe embeddings, and 200-d embeddings output from a CNN over the token's characters. The CNN uses filters of size=5 and character embeddings of 64-d. The pre-trained glove embeddings are fixed, but the character embeddings and the parameters for the CNN are jointly learned with the rest of the model. BERT: The input to the BERT model is the concatenation of the question and paragraph in the following format: `[CLS] Question [SEP] Passage [SEP]`. The question and context tokens input to the BERT model are sub-words extracted by using BERT's tokenizer. We separate the question and context representation from the output of BERT as **Q** and **P**, respectively. We use `bert-base-uncased` model for all our experiments.

Question Parser The decoder for question parsing is a single-layer, 100-dimensional, LSTM. For each module, we use a 100-dimensional embedding to present it as an action in the decoder's input/output vocabulary. The attention is computed as a dot-product between the decoder hidden-state and the encoders hidden states which is normalized using the softmax operation. As the memory-state for the zero-eth time-step in the decoder, we use the last hidden-state of the question encoder GRU, or the `[CLS]` embedding for the BERT-based model. We use a beam-size of 4 for the approximate maximum marginal likelihood objective. Optimization is performed using the Adam algorithm with a learning rate of 0.001 or using BERT's optimizer with a learning rate of $1e-5$. The countGRU in the `count` module (spanGRU – `span` module) is a 2-layer, bi-directional GRU with input-dim = 4 and output-dim = 20. The final feed-forward comprises of a single-layer to map the output of the countGRU into a scalar score.

Model	F1	EM
NAQANet	62.1	57.9
TAG-NABERT+	74.2	70.6
NABERT+	75.4	72.0
MTMSN	76.5	73.1
Our Model (w/ GRU)	73.1	69.6
Our Model (w/ BERT)	**77.4**	**74.0**

(a) Performance on DROP (pruned)

Question Type	MTMSN	Our Model (w/ BERT)
Date-Compare (18.6%)	**85.2**	82.6
Date-Difference (17.9%)	72.5	**75.4**
Number-Compare (19.3%)	85.1	**92.7**
Extract-Number (13.5%)	80.7	**86.1**
Count (17.6%)	**61.6**	55.7
Extract-Argument (12.8%)	66.6	**69.7**

(b) Performance by Question Type (F1)

Table 2: Performance of different models on the dataset and across different question types

3.5.3 Comparative approaches

We compare to publicly available best performing models: NAQANet (Dua et al., 2019), NABERT+ (Kinley and Lin, 2019), TAG-NABERT+ (Efrat and Shoham, 2019), and MTMSN (Hu et al., 2019), all trained on the same data as our model. All these models are black-box neural networks models either based on GRUs trained from scratch, or fine-tuned BERT-based models. The arithmetic and counting operations in these models are performed by having two simple-multi-class classifier heads: count is performed by predicting a count value in the range 0-9, and arithmetic is performed by selecting two passage numbers and a +/- sign between them. None of these models perform any explicit decomposition of the question and hence their decision making process is non interpretable.

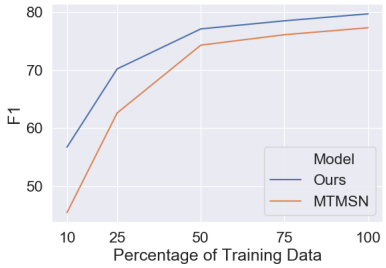
3.6 Results

3.6.1 Overall

Table 2a compares our model's performance to state-of-the-art models on our full test set. Our model achieves an F1 score of 73.1 (w/ GRU) and significantly outperforms NAQANet (62.1 F1). Using BERT representations, our model's performance increases to 77.4 F1 and outperforms models that use BERT representations, such as MTMSN (76.5 F1). This shows the efficacy of our proposed model in understanding complex compositional questions and performing multi-step reasoning over natural language text. Additionally, this shows that structured models still benefit when used over representations from large pretrained-LMs, such as BERT.

Supervision Type		w/ BERT	w/ GRU
H_{loss}	Mod-sup		
✓	✓	**77.4**	**73.1**
✓		76.3	71.8
	✓	–*	57.3

(a) **Effect of Auxiliary Supervision:** The auxiliary loss contributes significantly to the performance, whereas module output supervision has little effect. **Training diverges without H_{loss} for the BERT-based model.*



(b) **Performance with less training data**: Our model performs significantly better than the baseline with less training data, showing the efficacy of explicitly modeling compositionality.

Figure 8: Effect of auxiliary losses and the size of training data on model performance.

3.6.2 Performance by Question Type.

Table 2b shows the performance for different question types as identified by our heuristic labeling. Our model outperforms MTMSN on majority of question types but struggles with counting questions. Even after pre-training the count module using synthetic data, training it is particularly unstable. We believe this is because feedback from count questions is weak, i.e., the model only gets feedback about the count value and not what the underlying set is; and because it was challenging to define a categorical count distribution given a passage attention distribution— finding a better way to parameterize this function is an interesting problem for future work.

3.6.3 Effect of Additional Supervision

Figure 8a shows that the unsupervised auxiliary objective significantly improves model performance (from 57.3 to 73.1 F1). The model using BERT diverges while training without the auxiliary objective. Additionally, the intermediate module output supervision has slight positive effect on the model performance.

3.6.4 Effect of Training Data Size

Figure 8b shows that our model significantly outperforms MTMSN when training using less data, especially using 10-25% of the available supervision. This shows that by explicitly modeling compositionality, our model is able to use additional auxiliary supervision effectively and achieves improved

Question: Which group was smaller, the 25 to 44 year olds or the 45 to 64 year olds?

Program: `span(compare-num-lt(find, find))`

Answer: 45 to 64

compare-num-lt passage-attention: In the city, the population was spread out with 12.0% under the age of 18, 55.2% from 18 to 24, 15.3% from 25 to 44, 10.3% from 45 to 64, and 7.1% who were 65 years of age or older. The median age was 22 years. For every 100 females, there were 160.7 males. For every 100 females age 18 and over, there were 173.2 males.

number-distribution: 7.1 10.3 12 15.3 18 22 24 25 44 45 55.2 64 65 100 160.7 173.2

question-attention: Which group was smaller, the 25 to 44 year olds or the 45 to 64 year olds?

passage-attention: In the city, the population was spread out with 12.0% under the age of 18, 55.2% from 18 to 24, 15.3% from 25 to 44, 10.3% from 45 to 64, and 7.1% who were 65 years of age or older. The median age was 22 years. For every 100 females, there were 160.7 males. For every 100 females age 18 and over, there were 173.2 males.

Figure 9: Example usage of `num-compare-lt`: For the given question, our model predicts the program: `span(compare-num-lt(find, find))`. We show the question attentions and the predicted passage attentions of the two `find` operations using color-coded highlights on the same question and paragraph (to save space) at the bottom. The number grounding for the two paragraph attentions predicted in the `compare-num-lt` module are shown using the same colors in *number-distribution*. Since the number associated to the passage span "45 to 64" is lower (10.3 vs. 15.3), the output of the `compare-num-lt` module is "45 to 64" as shown in the passage above.

model generalization.

3.6.5 Qualitative Analysis

Example Prediction In Figure 9 we show an example prediction by our trained model.

Incorrect Program Predictions Mistakes by our model can be classified into two types; incorrect program prediction and incorrect execution. Here we show few mistakes of the first type that highlight the need to parse the question in a context conditional manner:

1. How many touchdown passes did Tom Brady throw in the season? - `count(find)` is incorrect since the correct answer requires a simple lookup from the paragraph.

2. Which happened last, failed assassination attempt on Lenin, or the Red Terror? - `date-compare-gt(find, find))` is incorrect since the correct answer requires natural language inference about the order of events and not symbolic comparison between dates.

3. Who caught the most touchdown passes? - `relocate(find-max-num(find)))`. Such questions,

that require nested counting, are out of scope of our defined modules because the model would first need to to count the passes caught by each player.

3.7 Future Directions

Since we only work with a subset of the DROP dataset, we try adding a simple arithmetic module to gauge the performance of our on the full dataset. This aim of this analysis is to realize directions for future work in this class of models.

We try a trivial extension to our model by adding a module that allows for addition & subtraction between two paragraph numbers. The resulting model achieves a score of 65.4 F1 on the complete validation data of DROP, as compared to MTMSN that achieves 72.8 F1. Manual analysis of predictions reveals that a significant majority of mistakes are due to insufficient reasoning capability in our model and would require designing additional modules. For example, questions such as (a) "How many languages each had less than $115,000$ speakers in the population?" and "Which racial groups are smaller than 2%?" would require pruning passage spans based on the numerical comparison mentioned in the question; (b) "Which quarterback threw the most touchdown passes?" and "In which quarter did the teams both score the same number of points?" would require designing modules that considers some key-value representation of the paragraph; (c) "How many points did the packers fall behind during the game?" would require IE for implicit argument (points scored by the other team). It is not always clear how to design interpretable modules for certain operations; for example, for the last two cases above.

It is worth emphasizing here what happens when we try to train our model on these questions for which our modules *can't* express the correct reasoning. The modules in the predicted program get updated to try to perform the reasoning anyway, which harms their ability to execute their intended operations (cf. §3.2.2). This is why we focus on only a subset of the data when training our model.

In part due to this training problem, some other mistakes of our model relative to MTMSN on the full dataset are due to incorrect execution of the intermediate modules. For example, incorrect grounding by the find module, or incorrect argument extraction by the find-num module. For mistakes such as

these, our NMN based approach allows for identifying the cause of mistakes and supervising these modules using additional auxiliary supervision that is not possible in black-box models. In Chapter 5 we propose a novel training paradigm to provide indirect supervision for such intermediate decisions. This additionally opens up avenues for transfer learning where modules can be independently trained using indirect or distant supervision from different tasks. Direct transfer of reasoning capability in black-box models is not so straight-forward.

To solve both of these classes of errors, one could use black-box models, which gain performance on some questions at the expense of limited interpretability. It is not trivial to combine the two approaches, however. Allowing black-box operations inside of a neural module network significantly harms the interpretability—e.g., an operation that directly answers a question after an encoder encourages the encoder to perform complex reasoning in a non-interpretable way. This also harms the ability of the model to use the interpretable modules even when they would be sufficient to answer the question. Additionally, due to our lack of supervised programs, training the network to use the interpretable modules instead of a black-box shortcut module is challenging, further compounding the issue. Combining these black-box operations with the interpretable modules that we have presented is an interesting and important challenge for future work.

3.8 Summary

In this chapter, we show how to use neural module networks to answer compositional questions requiring symbolic reasoning against natural language text. We define probabilistic modules that propagate uncertainty about symbolic reasoning operations in a way that is end-to-end differentiable. Additionally, we show that injecting inductive bias using unsupervised auxiliary losses significantly helps learning. While we have demonstrated marked success in broadening the scope of neural modules and applying them to open-domain text, it remains a significant challenge to extend these models to the full range of reasoning required even just for the DROP dataset. NMNs provide interpretability, compositionality, and improved generalizability, but at the cost of restricted expressivity as compared to more black box models. Future research is necessary to continue bridging these reasoning gaps.

Chapter 4

Module Faithfulness in Compositional Neural Networks

In the previous chapter, we described a modular architecture for performing compositional reasoning over text. Similarly, modular architecture based models have been widely used for compositional visual question answering against images (Andreas et al., 2016; Hu et al., 2017). One key assumption in such models is that the designed modules will learn to perform their intended reasoning operation only using the end-task supervision that is used for training. In this chapter, we study this assumption on both textual and visual neural module networks and find that it is possible that the modules will *not* learn to perform their intended operation from such training signal. This happens even when the model is trained using gold programs which provides strong supervision about the correct problem decomposition. Since the modules are parameterized as expressive neural networks and their composition via the program structure leads to a larger neural network, we find that end-goal supervision is not sufficient to induce correct intermediate outputs by the modules. In this chapter, we propose a systematic evaluation methodology for quantifying the correctness of module outputs and propose methods to improve it. We also show that incorrect module behavior affects the model's generalization. This chapter is based on work originally described in Subramanian* et al. (2020).

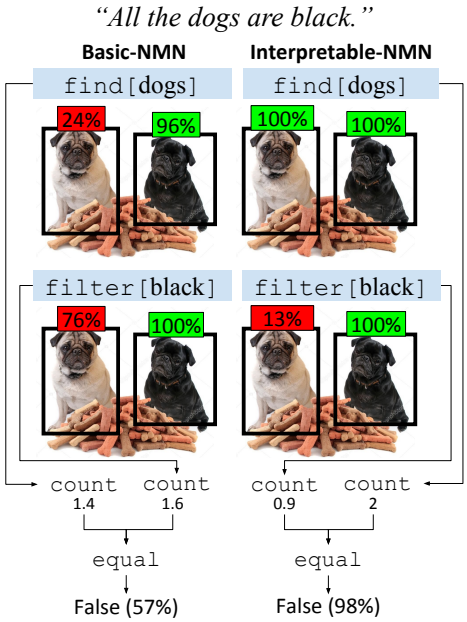


Figure 10: An example for a visual reasoning problem where both the Basic and Faithful NMNs produce the correct answer. The Basic NMN, however, fails to give meaningful intermediate outputs for the `find` and `filter` modules, whereas our improved Faithful-NMN assigns correct probabilities in all cases. Boxes are green if probabilities are as expected, red otherwise.

4.1 Introduction

Consider the example in Figure 10 from NLVR2 (Suhr et al., 2019): a model must understand the compositional utterance in order to then ground *dogs* in the input, count those that are *black* and verify that the count of all dogs in the image is equal to the number of black dogs.

A neural module network designed for this task will parse the input utterance into an executable program composed of learnable modules. These modules would be designed to perform atomic reasoning tasks in the visual domain and can be composed to perform complex reasoning. The decision making process of a NMN is interpretable: it provides a logical meaning representation of the utterance and also the outputs of the intermediate steps (outputs of the modules) to reach the final answer.

However, because module parameters are typically learned from end-task supervision only, it is possible that the program will not be a *faithful* explanation of the behaviour of the model (Ross

et al., 2017; Wiegreffe and Pinter, 2019), i.e., the model will solve the task by executing modules according to the program structure, but the modules will not perform the reasoning steps *as intended*. For example, in Figure 10, a *basic NMN* predicts the correct answer `False`, but incorrectly predicts the output of the `find`[*dogs*] operation. It does not correctly locate one of the *dogs* in the image because two of the reasoning steps (`find` and `filter`) have collapsed into one module (`find`). This behavior of the `find` module is not faithful to its intended reasoning operation; a human reading the program would expect `find`[*dogs*] to *locate* all dogs. Such unfaithful module behaviour yields an unfaithful explanation of the model behaviour.

Unfaithful behaviour of modules, such as multiple reasoning steps collapsing into one, are undesirable for two reasons: (a) it hurts the interpretability of a model–if a model fails to answer some question correctly, it is hard to tell which modules are the sources of error, and (b) such behaviour indicates that the model is not performing problem decomposition in a manner indicated by the program structure, and should hurt generalization to instances requiring novel compositional processing.

In this chapter, we provide three primary contributions regarding faithfulness in NMNs. First, we propose the concept of module-wise faithfulness – a systematic evaluation of individual module performance in NMNs that judges whether they have learned their intended operations, and define metrics to quantify this for both visual and textual reasoning. Empirically, we show on both NLVR2 (Suhr et al., 2019) and DROP (Dua et al., 2019) that training a NMN using end-task supervision, even using *gold programs*, does *not* yield module-wise faithfulness, i.e., the modules do not perform their intended reasoning task. Second, we provide strategies for improving module-wise faithfulness in NMNs. Specifically, (a) we demonstrate how module architecture affects faithfulness, (b) propose supervising module outputs with either a proxy task or heuristically generated data, and (c) show that providing modules with uncontexualized token representations improves faithfulness. Figure 10 shows an example where our approach (*Faithful-NMN*) results in expected module outputs as compared to the *Basic-NMN*.

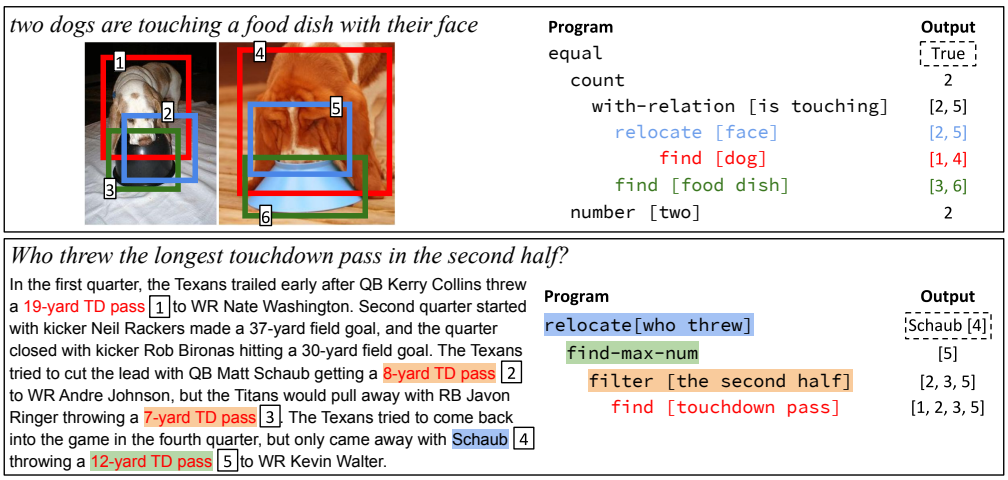


Figure 11: An example for a mapping of an utterance to a gold program and a perfect execution in a reasoning problem from NLVR2 (top) and DROP (bottom).

4.2 Background

In this chapter we use two different neural module networks—Text-NMN: described in the previous chapter to answer questions against natural language paragraphs; and Visual-NMN: to perform reasoning over images in NLVR2 (Suhr et al., 2019). Both NMNs, map a natural language utterance x into an executable program z that is executed against the given context (e.g., images, text) to output the denotation y (e.g., truth value in NLVR2, or a text answer in DROP). The program is computed of learnable modules that perform atomic reasoning tasks. Please refer to section 2.3 in the background chapter for a detailed explanation of NMNs, and chapter 3 for details about the modules used in Text-NMN. Below we describe details about our implementation of the Visual-NMN.

4.2.1 Visual-NMN

In NLVR2, given two images and a sentence that describes the images, the model should output `True` iff the sentence correctly describes the images. In Visual-NMN, we use the same seq2seq with grammar-constrained decoding parser used in Text-NMN. We base the implementation of modules in the Visual-NMN, on LXMERT (Tan and Bansal, 2019), which takes as input the sentence x and raw pixels, uses Faster R-CNN (Ren et al., 2015) to propose a set of bounding boxes, $\mathcal{B}$, that cover the objects in the image, and passes the tokens of x and the bounding boxes through a Trans-

former (Vaswani et al., 2017), encoding the interaction between both modalities. This produces a contextualized representation $\boldsymbol{t} \in \mathbb{R}^{|x| \times h}$ for each one of the tokens, and a representation $\boldsymbol{v} \in \mathbb{R}^{|\mathcal{B}| \times h}$ for each one of the bounding boxes, for a given hidden dimension h.

We provide a full list of modules and their implementation in Appendix A.3. Broadly, modules take as input representations of utterance tokens through an *utterance attention* mechanism (Hu et al., 2017), i.e., whenever the parser outputs a module, it also predicts a distribution over the utterance tokens $(p_1, \ldots, p_{|x|})$, and the module takes as input $\sum_{i=1}^{|x|} p_i \boldsymbol{t}_i$, where $\boldsymbol{t}_i$ is the hidden representation of token i. In addition, modules produce as output (and take as input) vectors $\boldsymbol{p} \in [0, 1]^{|\mathcal{B}|}$, indicating for each bounding box the probability that it should be output by the module (Mao et al., 2019). For example, in the program `filter`[*black*](`find`[*dog*]), the `find` module takes the word 'dog' (using *utterance attention*, which puts all probability mass on the word 'dog'), and outputs a probability vector $\boldsymbol{p} \in [0, 1]^{|\mathcal{B}|}$, where ideally all bounding boxes corresponding to dogs have high probability. Then, the `filter` module takes $\boldsymbol{p}$ as input as well as the word 'black', and is meant to output high probabilities for bounding boxes with 'black dogs'.

4.3 Module-wise Faithfulness

Neural module networks (NMNs) are designed in a modular manner to facilitate compositional reasoning. Its compositional structure encourages explicit problem decomposition in terms of independent and composable modules. The idea is that these modules will learn specialized operations, and can be composed in new ways to generalize to novel problem instances. NMNs also facilitate interpretability of their predictions via the reasoning steps in the structured program and providing the outputs of those intermediate steps during execution. For example, in Figure 11, all reasoning steps taken by both the Visual-NMN and Text-NMN can be discerned from the program and the intermediate module outputs. However, because module parameters are learned from an end-task, there is no guarantee that the modules will learn to perform their intended reasoning operation. In such a scenario, when modules do not perform their intended reasoning, the program is no longer a faithful explanation of the model behavior since it is not possible to reliably predict the outputs of the intermediate reasoning steps given the program.

We introduce the concept of *module-wise faithfulness* aimed at evaluating whether each module has correctly learned its intended operation by judging the correctness of its outputs in a trained NMN. For example, in Figure 11 (top), a model would be judged module-wise faithful if the outputs of all the modules, `find`, `relocate`, and `with_relation`, are correct – i.e. similar to the outputs that a human would expect. We provide gold programs when evaluating faithfulness, to not conflate faithfulness with parser accuracy.

4.3.1 Measuring Faithfulness in Visual-NMN

Modules in Visual-NMN provide for each bounding box a probability for whether it should be a module output. To evaluate intermediate outputs, we sampled examples from the development set, and annotated gold bounding boxes for each instance of `find`, `filter`, `with-relation` and `relocate`. The annotator draws the correct bounding-boxes for each module in the gold program, similar to the output in Figure 11 (top).

A module of a faithful model should assign high probability to bounding-boxes that are aligned with the annotated bounding boxes and low probabilities to other boxes. Since the annotated bounding boxes do not align perfectly with the model's bounding boxes, our evaluation must first induce an alignment. We consider two bounding boxes as "aligned" if the intersection-over-union (IOU) between them exceeds a pre-defined threshold $T = 0.5$. Note that it is possible for an annotated bounding box to be aligned with several proposed bounding boxes and vice versa. Next, we consider an *annotated* bounding box B_A as "matched" w.r.t a module output if B_A is aligned with a proposed bounding box B_P, and B_P is assigned by the module a probability > 0.5. Similarly, we consider a *proposed* bounding box B_P as "matched" if B_P is assigned by the module a probability > 0.5 and is aligned with some annotated bounding box B_A.

We compute precision and recall for each module type (e.g. `find`) in a particular example by considering all instances of the module in that example. We define *precision* as the ratio between the number of matched proposed bounding boxes and the number of proposed bounding boxes assigned a probability of more than 0.5. We define *recall* as the ratio between the number of matched an-

notated bounding boxes and the total number of annotated bounding boxes.[1] F_1 is the harmonic mean of precision and recall. Similarly, we compute an "overall" precision, recall, and F_1 score for an example by considering all instances of all module types in that example. The final score is an average over all examples.

4.3.2 Measuring Faithfulness in Text-NMN

Each module in Text-NMN produces a distribution over passage tokens (§3.3.4) which is a soft distributed representation for the selected spans. To measure module-wise faithfulness in Text-NMN, we obtain annotations for the set of spans that should be output by each module in the gold program (as seen in Figure 11 (bottom)) Ideally, all modules (`find`, `filter`, etc.) should predict high probability for tokens that appear in the gold spans and *zero* probability for other tokens.

To measure a module output's correctness, we use a metric akin to cross-entropy loss to measure the deviation of the predicted module output p_{att} from the gold spans $S = [s_1, \ldots, s_N]$. Here each span $s_i = (t^i_{\text{s}}, t^i_{\text{e}})$ is annotated as the start and end tokens. Faithfulness of a module is measured by:

$$I = -\sum_{i=1}^{N} \left(\log \sum_{j=t^i_{\text{s}}}^{t^i_{\text{e}}} p^j_{\text{att}} \right). \tag{4.1}$$

Lower cross-entropy corresponds to better faithfulness of a module.

4.4 Improving Faithfulness in NMNs

Module-wise faithfulness is affected by various factors: the choice of modules and their implementation, use of auxiliary supervision, and the use of contextual utterance embeddings. We discuss ways of improving faithfulness of NMNs across these dimensions.

[1] The numerators of the precision and the recall are different. Please see Appendix A.2.1 for an explanation.

4.4.1 Choice of Modules

Visual reasoning The `count` module always appears in NLVR2 as one of the top-level modules (see Figures 10 and 11).[2] We now discuss how its architecture affects faithfulness. Consider the program, `count(filter[black](find[dogs]))`. Its gold denotation (correct count value) would provide minimal feedback using which the *descendant* modules in the program tree, such as `filter` and `find`, need to learn their intended behavior. However, if `count` is implemented as an expressive neural network, it might learn to perform tasks designated for `find` and `filter`, hurting faithfulness. Thus, an architecture that allows counting, but also encourages *descendant* modules to learn their intended behaviour through backpropagation, is desirable. We discuss three possible `count` architectures, which take as input the bounding box probability vector $\boldsymbol{p} \in [0, 1]^{|\mathcal{B}|}$ and the visual features $\boldsymbol{v} \in \mathbb{R}^{|\mathcal{B}| \times h}$.

Layer-count module is motivated by the count architecture of Hu et al. (2017), which uses a linear projection from image attention, followed by a softmax. This architecture explicitly uses the visual features, $\boldsymbol{v}$, giving it greater expressivity compared to simpler methods. First we compute $\boldsymbol{p} \cdot \boldsymbol{v}$, the weighted sum of the visual representations, based on their probabilities, and then output a scalar count using: $\text{FF}_1(\text{LayerNorm}(\text{FF}_2(\boldsymbol{p} \cdot \boldsymbol{v}))$, where FF_1 and FF_2 are feed-forward networks, and the activation function of FF_1 is ReLU in order to output positive numbers only.

As discussed, since this implementation has access to the visual features of the bounding boxes, it can learn to perform certain tasks itself, without providing proper feedback to descendant modules.

Sum-count module on the other extreme, ignores $\boldsymbol{v}$, and simply computes the sum $\sum_{i=1}^{|\mathcal{B}|} \boldsymbol{p}_i$. Being parameter-less, this architecture provides direct feedback to descendant modules on how to change their output to produce better probabilities. However, such a simple functional-form ignores the fact that bounding boxes are overlapping, which might lead to over-counting objects. In addition, we would want `count` to ignore boxes with low probability. For example, if `filter` predicts a 5%

[2] Top-level modules are Boolean quantifiers, such as number comparisons like `equal` (which require `count`) or `exist`. We implement `exist` using a call to `count` and `greater-equal` (see Appendix A.3), so `count` always occurs in the program.

probability for 20 different bounding boxes, we would not want the output of `count` to be 1.0.

Graph-count module (Zhang et al., 2018) is a middle ground between both approaches - the naïve *Sum-Count* and the flexible *Layer-Count*. Like *Sum-Count*, it does not use visual features, but learns to ignore overlapping and low-confidence bounding boxes while introducing only a minimal number of parameters (less than 300). It does so by treating each bounding box as a node in a graph, and then learning to prune edges and cluster nodes based on the amount of overlap between their bounding boxes (see paper for further details). Because this is a light-weight implementation that does not access visual features, proper feedback from the module can propagate to its descendants, encouraging them to produce better predictions.

Textual reasoning In the context of Text-NMN (on DROP), we study the effect of several modules on interpretability.

First, we introduce an `extract-answer` module. This module bypasses all compositional reasoning and directly predicts an answer from the input contextualized representations. This has potential to improve performance, in cases where a question describes reasoning that cannot be captured by pre-defined modules, in which case the program can be the `extract-answer` module only. However, introducing `extract-answer` adversely affects interpretability and learning of other modules, specifically in the absence of gold programs. First, `extract-answer` does not provide any interpretability. Second, whenever the parser predicts the `extract-answer` module, the parameters of the more interpretable modules are not trained. Moreover, the parameters of the encoder are trained to perform reasoning *internally* in a non-interpretable manner. We study the interpretability vs. performance trade-off by training Text-NMN with and without `extract-answer`.

Second, consider the program `find-max-num(find[touchdown])` that aims to find the *longest touchdown*. `find-max-num` should sort spans by their value and return the maximal one; if we remove `find-max-num`, the program would reduce to `find[touchdown]`, and the `find` module would have to select the longest touchdown rather than all touchdowns, following the true denotation. More generally, omitting atomic reasoning modules pushes other modules to compensate and perform com-

plex tasks that were not intended for them, hurting faithfulness. To study this, we train Text-NMN by removing sorting and comparison modules (e.g., `find-max-num` and `num-compare`), and evaluate how this affects module-wise interpretability.

4.4.2 Supervising Module Output

As explained, given end-task supervision only, modules may not act as intended, since their parameters are only trained for minimizing the end-task loss. Thus, a straightforward way to improve interpretability is to train modules with additional atomic-task supervision.

Visual reasoning For Visual-NMN, we pre-train `find` and `filter` modules with explicit intermediate supervision, obtained from the GQA balanced dataset (Hudson and Manning, 2019). Note that this supervision is used only during pre-training – we do not assume we have full-supervision for the actual task at hand. GQA questions are annotated by gold programs; we focus on "exist" questions that use `find` and `filter` modules only, such as *"Are there any red cars?"*.

Given gold annotations from Visual Genome (Krishna et al., 2017), we can compute a label for each of the bounding boxes proposed by Faster-RCNN. We label a proposed bounding box as 'positive' if its IOU with a gold bounding box is > 0.75, and 'negative' if it is < 0.25. We then train on GQA examples, minimizing both the usual denotation loss, as well as an auxiliary loss for each instance of `find` and `filter`, which is binary cross entropy for the labeled boxes. This loss rewards high probabilities for 'positive' bounding boxes and low probabilities for 'negative' ones.

Textual reasoning In Chapter 3 we proposed heuristic methods to extract supervision for the `find-num` and `find-date` modules in DROP. On top of the end-to-end objective, we use an auxiliary objective that encourages these modules to output the "gold" numbers and dates according to the heuristic supervision. We evaluate the effect of such supervision on the faithfulness of both the supervised modules, as well as other modules that are trained jointly.

4.4.3 Decontextualized Word Representations

The goal of decomposing reasoning into multiples steps, each focusing on different parts of the utterance, is at odds with the widespread use of contextualized representations such as BERT or LXMERT. While the *utterance attention* is meant to capture information only from tokens relevant for the module's reasoning, contextualized token representations carry global information. For example, consider the program `filter[red](find[`*car*`])` for the phrase *red car*. Even if `find` attends only to the token *car*, its representation might also express the attribute *red*, so `find` might learn to find just *red cars*, rather than all *cars*, rendering the `filter` module useless, and harming faithfulness. To avoid such contextualization in Visual-NMN, we zero out the representations of tokens that are unattended, thus the input to the module is computed (with LXMERT) from the remaining tokens only.

4.5 Experimental Setup

This section describes the data used for measuring faithfulness and the details of the experiments.

Visual reasoning We use the published pre-trained weights and the same training configuration of LXMERT (Tan and Bansal, 2019), with 36 bounding boxes proposed per image. Due to memory constraints, we restrict training data to examples having a gold program with at most 13 modules.

We automatically generate gold program annotations for $26,311$ training set examples and for $5,772$ development set examples from NLVR2. The input to this generation process is the set of crowdsourced question decompositions from the Break dataset (Wolfson et al., 2020). We generated program annotations for NLVR2 by automatically canonicalizing its question decompositions in the Break dataset. Decompositions were originally annotated by Amazon Mechanical Turk workers. For each utterance, the workers were asked to produce the correct decomposition and an *utterance attention* for each operator (module), whenever relevant.

Limitations of Program Annotations: Though our annotations for gold programs in NLVR2 are largely correct, we find that there are some examples for which the programs are unnecessarily

Figure 12: An example of a gold program for NLVR2 that is unnecessarily complicated.

complicated. For instance, for the sentence "the right image contains a brown dog with its tongue extended." the gold program is shown in Figure 12. This program could be simplified by replacing the `with-relation` with the second argument of `with-relation`. Programs like this make learning more difficult for the NMNs since they use modules (in this case, `with-relation`) in degenerate ways. There are also several sentences that are beyond the scope of our language, e.g. comparisons such as "the right image shows exactly two virtually identical trifle desserts."

For module-wise faithfulness evaluation, 536 examples from the development set were annotated with the gold output for each module by experts.

Textual reasoning We train Text-NMN on DROP, which is augmented with program supervision for $4,000$ training questions collected heuristically as described in Chapter 3. The model is evaluated on the complete development set of DROP which does not contain any program supervision. Module-wise faithfulness is measured on 215 manually-labeled questions from the development set, which are annotated with gold programs and module outputs (passage spans).

4.6 Results

4.6.1 Faithfulness Evaluation in Visual Reasoning

Results are seen in Table 3. Accuracy for LXMERT, when trained and evaluated on the same subset of data, is 71.7%; slightly higher than NMNs, but without providing any evidence for the compositional structure of the problem.

Model	Performance (Accuracy)	Overall Faithful. (↑)			Module-wise Faithfulness F_1(↑)			
		Prec.	Rec.	F_1	find	filter	with-relation	relocate
LXMERT	**71.7**							
Upper Bound		1	0.84	0.89	0.89	0.92	0.95	0.75
NMN w/ Layer-count	71.2	0.39	0.39	0.11	0.12	0.20	0.37	**0.27**
NMN w/ Sum-count	68.4	**0.49**	0.31	0.28	0.31	0.32	0.44	0.26
NMN w/ Graph-count	69.6	0.37	0.39	0.28	0.31	0.29	0.37	0.19
NMN w/ Graph-count + decont.	67.3	0.29	0.51	0.33	0.38	0.30	0.36	0.13
NMN w/ Graph-count + pretraining	69.6	0.44	0.49	0.36	0.39	0.34	0.42	0.21
NMN w/ Graph-count + decont. + pretraining	68.7	0.42	**0.66**	**0.47**	**0.52**	**0.41**	**0.47**	0.21

Table 3: Faithfulness and accuracy on NLVR2. "decont." refers to decontextualized word representations. Precision, recall, and F_1 are averages across examples, and thus F_1 is **not** the harmonic mean of the corresponding precision and recall.

For faithfulness, we measure an upper-bound on the faithfulness score. Recall that this score measures the similarity between module outputs and annotated outputs. Since module outputs are constrained by the bounding boxes proposed by Faster-RCNN, while annotated boxes are not, perfect faithfulness could only be achieved by a model if there are suitable bounding boxes. *Upper Bound* shows the maximal faithfulness score conditioned on the proposed bounding boxes.

We now compare the performance and faithfulness scores of the different components. When training our NMN with the most flexible count module, (*NMN w/ Layer-count*), an accuracy of 71.2% is achieved, a slight drop compared to LXMERT but with low faithfulness scores. Using *Sum-count* drops about 3% of performance, but increases faithfulness. Using *Graph-count* increases accuracy while faithfulness remains similar.

Next, we analyze the effect of decontextualized word representations (abbreviated "decont.") and pre-training. First, we observe that *NMN w/ Graph-count + decont.* increases faithfulness score to 0.33 F_1 at the expense of accuracy, which drops to 67.3%. Pre-training (*NMN w/ Graph-count + pretraining*) achieves higher faithfulness scores with a higher accuracy of 69.6%. Combining the two achieves the best faithfulness (0.47 F_1) with a minimal accuracy drop. We perform a paired permutation test to compare *NMN w/ Graph-count + decont. + pretraining* with *NMN w/ Layer-count* and find that the difference in F_1 is statistically significant ($p < 0.001$). Please see Appendix A.4 for further details.

Model	Performance (F_1 Score)	Overall Faithful. (cross-entropy* ↓)	Module-wise Faithfulness* (↓)				
			find	filter	relocate	min-max†	find-arg†
Text-NMN w/o prog-sup							
w/ `extract-answer`	63.5	9.5	13.3	9.5	3.5	2.6	9.9
w/o `extract-answer`	60.8	**6.9**	8.1	7.3	1.3	1.7	8.5
Text-NMN w/ prog-sup							
no auxiliary sup	65.3	11.2	13.7	16.9	1.5	2.2	13.0
w/o sorting & comparison	63.8	8.4	9.6	11.1	1.6	1.3	10.6
w/ module-output-sup	**65.7**	**6.5**	**7.6**	**10.7**	**1.3**	**1.2**	**7.6**

Table 4: Faithfulness and performance scores for various NMNs on DROP. *lower is better. †min-max is average faithfulness of `find-min-num` and `find-max-num`; find-arg of `find-num` and `find-date`.

4.6.2 Faithfulness Evaluation in Textual Reasoning

As seen in Table 4, when trained on DROP using question-program supervision, the model achieves 65.3 F_1 performance and a faithfulness score of 11.2 (lower is better). When adding supervision for intermediate modules, we find that the module-wise faithfulness score improves to 6.5. Similar to Visual-NMN, this shows that supervising intermediate modules in a program leads to better faithfulness.

To analyze how choice of modules affects faithfulness, we train without sorting and comparison modules (`find-max-num`, `num-compare`, etc.). We find that while performance drops slightly, faithfulness deteriorates significantly to 8.4, showing that modules that perform atomic reasoning are crucial for faithfulness. When trained without program supervision, removing `extract-answer` improves faithfulness ($9.5 \rightarrow 6.9$) but at the cost of performance ($63.5 \rightarrow 60.8$ F_1). This shows that such a black-box module encourages reasoning in an opaque manner, but can improve performance by overcoming the limitations of pre-defined modules. All improvements in faithfulness are significant as measured using paired permutation tests ($p < 0.001$).

4.6.3 Measuring Generalization

A natural question is whether models that are more faithful also generalize better. We conducted a few experiments to see whether this is true for our models. For NLVR2, we performed (1) an exper-

iment in which programs in training have length at most 7, and programs at test time have length greater than 7, (2) an experiment in which programs in training have at most 1 `filter` module and programs at test time have at least 2 `filter` modules, and (3) an experiment in which programs in training do not have both `filter` and `with-relation` modules in the same program, while each program in test has both modules. We compared three of our models – *NMN w/ Layer-count*, *NMN w/ Sum-count*, and *NMN w/ Graph-count + decont. + pretraining*. We did not observe that faithful models generalize better (in fact, the most unfaithful model tended to achieve the best generalization).

To measure if faithful model behavior leads to better generalization in Text-NMN we conducted the following experiment. We selected the subset of data for which we have gold programs and split the data such that questions that require maximum and greater-than operations are present in the training data while questions that require computing minimum and less-than are in the test data. We train and test our model by providing gold-programs under two conditions, in the presence and absence of additional module supervision. We find that providing auxiliary module supervision (that leads to better module faithfulness; see above) also greatly helps in model generalization (performance increases from 32.3 $F_1 \rightarrow 78.3$ F_1). This result demonstrates that when the modules correctly learn to perform their intended reasoning, they generalize better to novel program contexts.

4.6.4 Qualitative Analysis

We analyze outputs of different modules in Figure 13. Figures 13a, 13b show the output of `find`[*llamas*] when trained with contextualized and decontextualized word representations. With contextualized representations (13a), the `find` fails to select any of the *llamas*, presumably because it can observe the word *eating*, thus effectively searching for *eating llamas*, which are not in the image. Conversely, the decontextualized model correctly selects the boxes. Figure 13c shows that `find` outputs meaningless probabilities for most of the bounding boxes when trained with *Layer-count*, yet the count module produces the correct value (three). Figure 13d shows that `find` fails to predict all relevant spans when trained without sorting modules in Text-NMN.

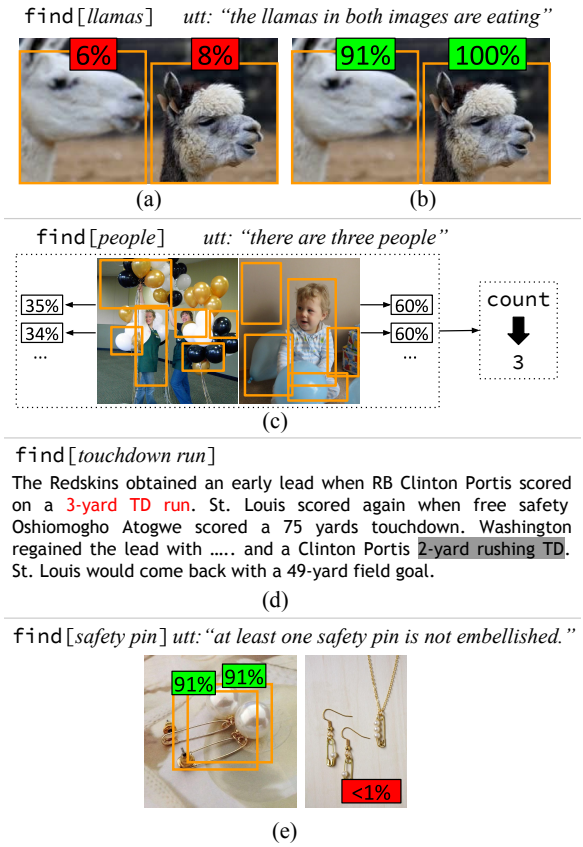


Figure 13: Comparison of module outputs between NMN versions: (a) Visual-NMN with contextualized representations, (b) Visual-NMN with decontextualized representations, (c) model using a parameter-rich count layer (Layer-Count), (d) Text-NMN trained without sorting module produces an incorrect `find` output (misses *2-yard rushing TD*), and (e) Visual-NMN failure case with a rare object (of *w/ Graph-count + decont. + pretraining*)

4.7 Summary

In this chapter, we introduce the concept of *module-wise faithfulness*, a systematic evaluation of correctness of module execution in neural module networks (NMNs) for visual and textual reasoning. We show that training NMNs using end-task supervision alone does not produce faithful modules, even when trained using gold programs. This shows that compositional model structure alone is not sufficient to encourage compositional processing; since the modules are implemented using neural networks, there is no guarantee that modules will learn to perform independent tasks that were intended for them. Additionally, we propose several techniques to improve module-wise faithfulness

in NMNs and show that our approach leads to much higher module-wise faithfulness at a marginal cost to performance. We encourage future work to judge the correctness of individual modules in models with such compositional structure using the proposed evaluation and publicly published annotations. We also hope that the community will explore techniques that encourage modules to learn their intended behaviour, without resorting to expensive annotations.

Chapter 5

Paired Examples as Indirect Supervision in Latent Decision Models

Developing models that are capable of reasoning about complex real-world problems is challenging. It involves decomposing the problem into sub-tasks, making intermediate decisions, and combining them to make the final prediction. In the previous chapters, we propose a neural module network based question answering model that parses the question into an explicit program that describes the sub-tasks and the structure in which they need to be composed. These sub-tasks are performed by learnable modules that are designed to learn to perform specific atomic reasoning tasks, with the idea that they can be composed in novel ways to perform complex reasoning. In Chapter 4, we saw that end-task supervision provides a very weak training signal for what the correct outputs of the intermediate modules should be. As a result, the modules do not learn to perform their intended reasoning operation which hurts generalization when these modules are used in novel contexts. Therefore, we can say that individual training examples do not provide sufficient training signal for learning intermediate decisions, and hence induce compositional processing. In this chapter, we introduce a way to leverage *paired examples* that provide stronger cues for learning the latent decisions (outputs of the modules) that the model should predict. When two related training examples share internal substructure, we add an additional training objective to encourage consistency between their latent

decisions. This indirect supervision constraints the model from exploiting any shortcuts and encourages problem decomposition in the manner proposed by the program structure . We empirically demonstrate that our proposed approach improves both in- and out-of-distribution generalization of our question answering model, and leads to correct latent decision predictions. This chapter is based on work originally described in Gupta et al. (2021b).

5.1 Introduction

Let us first quickly recall how a compositional structured model (e.g., our NMN based QA model in §3) would answer a compositional question. To answer *How many field goals were scored in the first half?* against a passage containing a football-game summary, the model would first ground the set of *field goals* mentioned in the passage, then filter this set to the ones scored *in the first half*, and then return the size of the resulting set as the answer.

In Chapter 4 we saw that learning such models using just the end-task supervision is difficult, since the decision boundary that the model is trying to learn is complex, and the lack of any supervision for the latent decisions provides only a weak training signal. Moreover, the presence of dataset artifacts (Lai and Hockenmaier, 2014; Gururangan et al., 2018; Min et al., 2019, *among others*), and degeneracy in the model, where incorrect latent decisions can still lead to the correct output, further complicates learning. As a result, models often fail to predict meaningful intermediate outputs and instead end up fitting to dataset quirks, thus hurting generalization. In our application of NMNs, modules do not learn to correctly perform the tasks independently, thus hurting compositional reasoning if they are combined in novel ways.

We propose a method to leverage related training examples to provide an indirect supervision to these intermediate decisions. Our method is based on the intuition that related examples involve similar sub-tasks; hence, we can use an objective on the outputs of these sub-tasks to provide an additional training signal. Concretely, we use *paired examples*—instances that share internal substructure—and apply an additional training objective relating the outputs from the shared substructures resulting from partial model execution. Using this objective does not require supervision for the output of the

shared substructure, or even the end-task of the paired example. This additional training objective imposes weak constraints on the intermediate outputs using related examples and provides the model with a richer training signal than what is provided by a single example. For example, *What was the shortest field goal?* shares the substructure of finding all *field goals* with *How many field goals were scored?*. For this *paired example*, our proposed objective would enforce that the output of this latent decision for the two questions is the same.

We demonstrate the benefits of our paired training objective using our textual-NMN proposed in Chapter 3 on the DROP dataset. While there can be many ways of acquiring paired examples, we explore three directions. First, we show how naturally occurring paired questions can be automatically found from within the dataset. Further, since our method does not require end-task supervision for the paired example, one can also use data augmentation techniques to acquire paired questions without requiring additional annotation. We show how paired questions can be constructed using simple templates, and how a pre-trained question generation model can be used to generate paired questions.

We empirically demonstrate that our paired training objective leads to overall performance improvement of the NMN model. Based on the evaluation methodology described in Chapter 4, we quantitatively show that using this paired objective results in significant improvement in predicting the correct latent decisions, and thus demonstrate that the model's performance is improving *for the right reasons*. Finally, we show that the proposed approach leads to better *compositional generalization* to out-of-distribution examples. Our results show that we achieve the stated promise of compositional structured models: an interpretable model that naturally encodes compositional reasoning and uses its modular architecture for better generalization.

5.2 Paired Examples as Indirect Supervision for Latent Decisions

We first describe our approach generally as it can be applied to a host of compositional structured models that perform an explicit problem decomposition and predict interpretable latent decisions that are composed to predict the final output. We will then describe how we apply this idea specifically

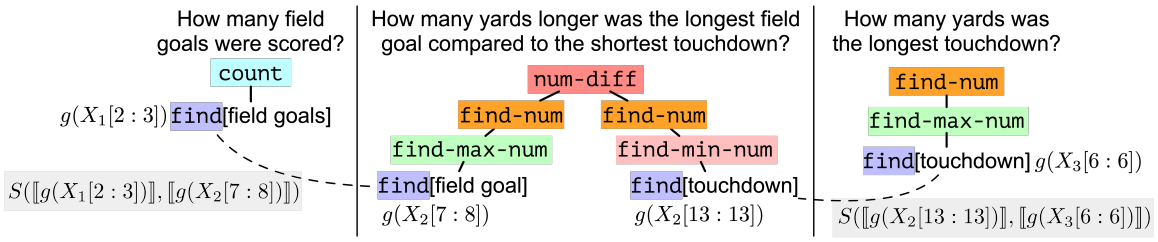


Figure 14: Proposed paired objective: For training examples that share substructure, we propose an additional training objective relating their latent decisions; S in the shaded gray area. In this figure, $g(X_i[m:n]) = g(\text{BERT}(x_i, p)[m:n])$, where $\text{BERT}(x_i, p)$ is the contextualized representation of x_i-th question/passage, and $[m:n]$ is its slice for the m through n token. g = `find` in all cases. Here, since the outputs of the shared substructures should be the same, S would encourage *equality* between them.

to our NMN.

Let us say, for a given input x, a model performs the computation $f(g(x), h(x))$ to predict the output y. This computation tree $f(g(x), h(x)) = z$ could represent a program in our NMN, for example. Following our notation, we will use the notation z to denote a computation tree, and $[\![z]\!]$ to denote the output of its execution. Hence we can write,

$$y = [\![f(g(x), h(x))]\!] \tag{5.1}$$

where f, g, and h perform the three sub-tasks required for x and the computations $g(x)$ and $h(x)$ are the intermediate decisions. The actual computation tree would be dependent on the input and the structure of the model. For example, to answer *How many field goals were scored?*, our NMN would perform $f(g(x))$ where $g(x)$ would be the `find` module to output the set of *field goals* and f would be the `count` module to return the size of this set. While we focus on NMNs, other models that have similar structures where our techniques would be applicable include language models with latent variables for coreference (Ji et al., 2017), syntax trees (Dyer et al., 2016), or knowledge graphs (Logan et al., 2019); checklist-style models that manage coverage over parts of the input (Kiddon et al., 2016); or any neural model that has some interpretable intermediate decision, including standard attention mechanisms (Bahdanau et al., 2015). In the next chapter we will see how we apply this idea to improve weakly-supervised semantic parsing.

Typically, the only supervision provided to the model are gold (x, y^*) pairs, without the outputs of the intermediate decisions ($[\![g(x)]\!]$ and $[\![h(x)]\!]$ above), from which it is expected to jointly learn the parameters of all of its components. In our NMN for QA, we are only provided with question-answer supervision and expected to jointly learn the parameters for all modules without any intermediate module output supervision. In Chaper 4, we saw that such weak supervision is not enough for accurate learning, and the fact that incorrect latent decisions can lead to the correct prediction further complicates learning. Consequently, models fail to learn to perform these latent tasks correctly and usually end up modeling irrelevant correlations in the data.

We propose a method to leverage *paired examples*—examples whose one or more latent decisions are related to each other—to provide an indirect supervision to these latent decisions. Consider paired training examples x_i and x_j with the following computation trees:

$$z_i = f(g(x_i), h(x_i)) \tag{5.2}$$

$$z_j = f(k(g(x_j))) \tag{5.3}$$

These trees share the internal substructure $g(x)$. In such a scenario, we propose an additional training objective $S([\![g(x_i)]\!], [\![g(x_j)]\!])$ to enforce consistency of partial model execution for the shared substructure:

$$\mathcal{L}_{\text{paired}} = S([\![g(x_i)]\!], [\![g(x_j)]\!]) \tag{5.4}$$

For example, the two questions on the LHS of Figure 14 share the intermediate decision of finding the field goals. i.e., their computation trees share the substructure $g(x)$ = `find`[*field goal*]. In such a case, where the outputs of the intermediate decision should be the same for the paired examples, using a similarity measure for S would enforce equality of the latent outputs $[\![g(x)]\!]$. We will go into the specifics of this example later in this section. By adding this consistency objective, we are able to provide an additional training signal to the latent decision using related examples, and hence indirectly share supervision among multiple training examples. As a result, we are able to more densely characterize the decision boundary around an instance (x_i), by using related instances (x_j), than what was possible by using the original instance alone.

To use this consistency objective for x_i, we do not require supervision for the latent output $[\![g(x_i)]\!]$, nor the gold end-task output y_j^* for the paired example x_j; we only enforce that the intermediate decisions are consistent. Additionally, we are not limited to enforcing consistency for a single intermediate decision from a single paired example; if x_i shares an additional substructure $h(x)$ with a paired example x_k, we can add an additional term $S'([\![h(x_i)]\!], [\![h(x_k)]\!])$ to Eq. 5.4.

5.2.1 Training via Paired Examples in Neural Module Networks

We apply our approach to improve question answering using our neural module network based QA model described in Chapter 3.

Recall that, to answer $q =$ *How many field goals were scored in the first half?*, our NMN would parse it into a program $z =$ `count(filter[`*in the first half*`](find[`*field goals*`]))`; essentially performing, $y = [\![f(g(q, h(q)))]\!]$, where $f =$ `count`, $g =$ `filter`, and $h =$ `find`. Similarly, to answer $q =$ *How many field goals were scored?*, our NMN would parse it into a program $z =$ `count(find[`*field goals*`])`; essentially performing $y = [\![f(g(q))]\!]$, where $f =$ `count` and $g =$ `find`.

In this chapter we assume that for every question q in the training data, we are also given the gold program z^*, along with the correct answer a^*. Therefore, the model only needs to learn the parameters for the modules (e.g., `find`, `count`, `filter`, etc.). In Chapter 4 we saw that correctly learning the module parameters only using the answer supervision is challenging. Learning is further complicated by the fact that the space of possible intermediate outputs is quite large and incorrect module output prediction can still lead to the correct answer. For example, the `find` module in the question above needs to learn to select the spans describing *field goals* among all possible spans in the passage using just the *count value* as answer supervision. With no direct supervision for the module outputs, the modules can learn incorrect behavior but still predict the correct answer, effectively memorizing the training data. Such a model would presumably fail to generalize.

In this chapter, we use our BERT-based model since that lead to better performance. Let us first recall how a program execution actually works. Given a question q and passage p, BERT is used to compute joint contextualized representations for the (question, passage) combination, $\text{BERT}(q, p) \in$

$\mathbb{R}^{(|q|+|p|)\times d}$. During execution, the modules that take a question span argument as input (e.g. `find`) operate on the corresponding slice of this contextualized representation. For example, in q = *How many field goals were scored?* with program z = `count(find[`*field goals*`])`, to execute `find[`*field goals*`]`, the model actually executes `find`(BERT$(q, p)[2:3]$).[1] Here the slice $[2:3]$ corresponds to the contextualized representations for the 2nd through the 3rd token (*field goals*) of the question.

Paired training in NMNs We consider a pair of questions whose program trees z share a subtree as paired examples. A shared subtree implies that a part of the reasoning required to answer the questions is the same. Since some modules take as input a string argument, we define two subtrees to be equivalent *iff* their structure matches and the string arguments to the modules that require them are *semantically equivalent*. For example, subtrees `find-num(find[`*passing touchdowns*`])` and `find-num(find[`*touchdown passes*`])` are equivalent, while they are not the same as `find-num(find[`*touchdown runs*`])`. We describe how we detect semantic equivalence in the next section.

Consider a question q_i that shares the substructure $g(q)$ with a paired question q_j. Since shared substructures are common program subtrees in our case, we encourage the latent outputs, the outputs $[\![g(q)]\!]$ of the subtree, to be equal to enforce consistency. As the outputs of modules are probability distributions, enforcing consistency amounts to minimizing the KL-divergence between the two outputs. We therefore maximize the following paired objective from Eq. 5.4,

$$\mathcal{L}_{\text{paired}} = -\big(\text{KL}\big[[\![g(q_i)]\!] \parallel [\![g(q_j)]\!]\big] + \text{KL}\big[[\![g(q_j)]\!] \parallel [\![g(q_i)]\!]\big]\big) \tag{5.5}$$

where $S(p_1, p_2) = -(\text{KL}[p_1 \| p_2] + \text{KL}[p_2 \| p_1])$ is the negative symmetric KL-divergence.

To understand why such an objective is helpful even though the paired examples share exact subtrees, consider the paired examples on the LHS of Figure 14. The substructure $g(q)$ = `find[`*field goal*`]` is shared between them. Even though the input string argument to `find` is the same, what gets executed is $g(q_i)$ = `find`(BERT$(x_1, p)[2:3]$) and $g(q_j)$ = `find`(BERT$(x_2, p)[7:8]$), i.e. `find`

[1] All modules also take the (BERT-encoded) passage p as an implicit argument, as well as additional state extracted from the passage such as which tokens are numbers, which we omit for notational simplicity.

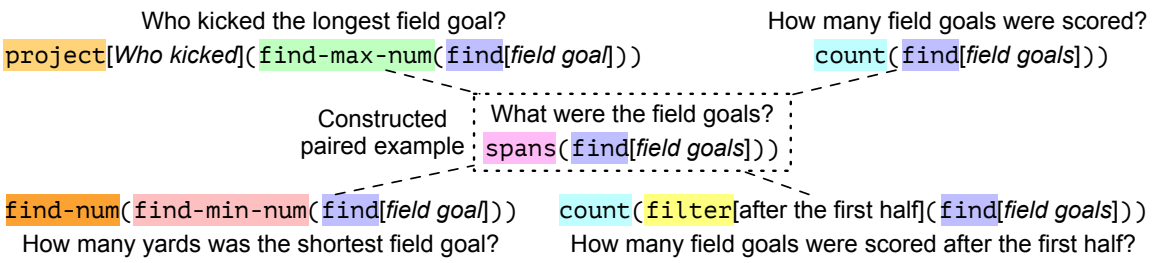


Figure 15: Templated Construction of Paired Examples: Constructed paired examples can help in indirectly enforcing consistency between different training examples (§5.3.2).

gets as input different contextualized representations of *field goal* from the two questions. Due to different inputs, the output of `find` could be different, which would lead to inconsistent behavior and inefficient learning. Our paired objective (Eq. 5.5) would encourage that these two outputs are consistent, thereby allowing sharing of supervision across examples.

Complete Example We describe the benefits of training with paired data using an example. Consider the four questions in the periphery of Figure 15; all of them share the substructure of finding the field goal scoring events. However, we find that for the questions requiring the `find-{max/min}-num` operation, our vanilla NMN directly grounds to the longest/shortest field goal as the `find` execution. Due to the use of powerful NNs (i.e., BERT) for contextualized question/passage representations and no constraints on the modules to perform as intended, the model performs the symbolic *min*/*max* operation internally in its parameters. Such `find` execution results in non-interpretable behavior, and substantially hurts generalization to the `count` questions. By enforcing consistency between all the `find` executions, the model can no longer shortcut the compositional reasoning defined by the programs; this results in correct `find` outputs and better generalization, as we will later see in the results section. Note that in this example we do not know the correct answer a^* for the constructed question *What were the field goals?*, nor do we know the intermediate output $[\![\texttt{find}[\textit{field goals}]]\!]$. The *only* additional supervision given to the model is that there is a pairing between substructures in all of these examples, and so the model should be consistent.

5.3 Many Ways of Getting Paired Data

We explore three ways of acquiring paired questions. We show how questions that share substructures can be automatically found from within the dataset, and how new paired questions can be constructed using templates, or generated using a question-generation model (§5.3.2).

5.3.1 Finding Naturally Occurring Paired Data

Any dataset that contains multiple questions against the same context could have questions that query different aspects of the same underlying event or entity. These examples can potentially be paired by finding the elements in common between them. As the DROP data that we are using has annotated programs, this process is simplified somewhat in that we can simply find pairs of programs in the training data that share a subtree. While the subtrees could be of arbitrary size, we limit ourselves to programs that share a leaf `find` module. Recall that `find` requires a question string argument, so the challenge of finding paired questions reduces to discovering pairs of `find` modules in different questions about the same paragraph whose question string arguments are semantically equivalent. To this end, we use BERTScore (Zhang* et al., 2020) to measure string similarity.

We consider two string arguments to be semantically equivalent if their BERTScore-F1 exceeds a threshold (0.6), and if the same entities are mentioned in the arguments. This additional constraint allows us to judge that *Jay Feely's field goal* and *Janikowski's field goal* are semantically different, even though they receive a high BERTScore. This approach would find paired examples like,
What term is used to describe the Yorkist defeat at Ludford Bridge in 1459?
What happened first: Yorkist defeat at Ludford Bridge or widespread pillaging by Queen Margaret?

5.3.2 Paired Data via Augmentation

One benefit of our consistency objective (Eq. 5.4) is that it only requires that the paired example shares substructure. This allows us to augment training data with new paired questions without knowing their gold answer. We explore two ways for such augmentation; (a) constructing paired questions using templates, and (b) generating paired questions using a question-generation model.

Templated Construction of Paired Examples

Grounding `find` event(s) Using the question argument from the `find` module of certain frequently occurring programs, we construct a paired question that aims to ground the mentions of the event queried in the `find` module. For example, *Who scored the longest touchdown?* would be paired with *What were the touchdowns?*. This templated paired question construction is carried out for,

(1) `count(find[])`

(2) `count(filter(find[]))`

(3) `find-num(find-max-num(find[]))`

(4) `find-num(find-max-num(filter[](find[])))`

(5) `project[](find-max-num(find[]))`

(6) `project[](find-max-num(filter[](find[])))`

(7) `date-compare-gt(find[], find[])`

(8) `time-diff(find[], find[])`, and their versions with `find-min-num` or `date-compare-lt`.

For questions with a program in (1) - (6), we append *What were the* to the program's `find` argument to construct a paired question. We annotate this paired question with the program `spans(find[])`, and enforce consistency among the `find` modules. Such a construction allows us to indirectly enforce consistency among multiple related questions via the constructed question; see Figure 15.

For questions with a program in (7) - (8), we append *When did the* to the two `find` modules' arguments and construct two paired questions, one for each `find` operation. We label the constructions with `find-date(find[])` and enforce consistency among the `find` modules. For example, *How many years after the Battle of Rullion Green was the Battle of Drumclog?* would result in the construction of *When did the Battle of Rullion Green?* and *When did the Battle of Drumclog?*. While this method can lead to ungrammatical questions, it should help in decomposing the two `find` executions.

Inverting Superlatives For questions with a program in (3) - (6) or its `find-min-num` equivalent, we construct a paired question by replacing the superlative in the question with its antonym (e.g. *largest* $\rightarrow$ *smallest*) and inverting the min/max module. We enforce consistency among the `find`

modules of the original and the paired question.

Model-generated Paired Examples

We show how question generation (QG) models (Du et al., 2017; Krishna and Iyyer, 2019) can be used to generate paired questions. QG models are seq2seq models that generate a question corresponding to an answer span marked in a passage as input. We follow Wang et al. (2020) and fine-tune a BART model (Lewis et al., 2020) on SQuAD (Rajpurkar et al., 2016) to use as a QG model.

We generate paired questions for non-football passages in DROP by randomly choosing 10 numbers and dates as answer spans, and generating questions for them. We assume that the generated questions are SQuAD-like—they query an argument about an event/entity mentioned in text—and label them with the program `find-num(find)` or `find-date(find)`. We use the whole question apart from the Wh-word as the string argument to `find`. We then follow the same procedure as §5.3.1—for each of the `find` module in a DROP question's program, we see if a generated question with a *semantically similar* `find` module exists. If such an augmented question is found, it is used as a paired example for the DROP question to enforce consistency between the `find` modules. For example, *How many percentage points did the population of non-Hispanic Whites drop from 1990 to 2010?* is paired with the generated question *What percentage of the population was non-Hispanic Whites in 2010?*.

5.4 Experimental Setup

This section describes the dataset used for training and evaluation, details about the number of paired examples we use from different techniques, the details of our model, and the baseline approaches used in the experiments for comparison.

5.4.1 Dataset

We perform experiments on a portion of the DROP dataset (Dua et al., 2019) that is composed of two subsets: (1) the subset of DROP used in Chapter 3—this contains 3881 passages and 19204 ques-

tions; and (2) question-decomposition meaning representation (QDMR) annotations from Break (Wolfson et al., 2020)—this contains 2756 passages with a total of 4762 questions. After removing duplicate questions we are left with 23215 questions in total. All questions in our dataset contain program annotations (heuristically annotated questions from Chapter 3 and crowd-sourced question decompositions in Break). We convert the program annotations in QDMR to programs that conform to the grammar induced by the modules in Text-NMN using simple transformations. For an i.i.d. split, since the DROP test set is hidden, we split the training set into train/validation and use the provided validation set as the test set. Our train / validation / test sets contain 18299 / 2460 / 2456 questions, respectively.

Paired Training Data In the training data, we found 7018 naturally-occurring pairings for 6039 questions (§5.3.1); construct template-based paired examples for 10882 questions (§5.3.2); and generate 2632 questions paired with 2079 DROP questions (§5.3.2).

5.4.2 Training Objective

We use these paired examples to compute $\mathcal{L}_{\text{paired}}$, which is added as an additional training objective on top of the standard training regime for Text-NMN We simply add the paired objective $\mathcal{L}_{\text{paired}}$ to the training objective of Text-NMN (Gupta et al., 2020a) which includes a maximum likelihood objective to predict the gold program, a maximum likelihood objective for gold answer prediction from the program execution, and an unsupervised auxiliary loss to aid information extraction. We do not use any heuristically-obtained intermediate module output supervision used in Chapter 3. Note that we do not add any additional (question, answer) pairs to the data, only new (unlabeled) questions.

5.4.3 Model Details

All models use the bert-base-uncased model to compute the question and passage contextualized representations. For all experiments (including all baselines), we train two versions of the model with different seed values, and choose the one that results in higher validation performance. All models are trained for a maximum number of 40 epochs, with early stopping if validation F1 does

Model	dev		test	
	F1	**EM**	**F1**	**EM**
MTMSN	66.2	62.4	72.8	70.3
NMN Baseline	62.6	58.0	70.3	67.0
NMN + $\mathcal{L}_{\text{paired, found}}$	66.0	61.5	71.0	67.8
NMN + $\mathcal{L}_{\text{paired, temp}}$	66.2	61.4	72.3	69.2
NMN + $\mathcal{L}_{\text{paired, qgen}}$	63.7	58.9	71.2	68.4
NMN + $\mathcal{L}_{\text{paired, all}}$	**66.3**	**61.6**	**73.5**	**70.5**

Table 5: Performance on DROP (pruned): Using our paired objective with all different kinds of paired-data leads to improvements in NMN. Model achieves the best performance when all kinds of paired-data are used together.

not improve for 10 consecutive epochs. We use a batch size of 2 (constrained by a 12GB GPU) and a learning rate of `1e-5`. The question parser in the Text-NMN uses a 100-dimensional, single-layer LSTM decoder. Our code is written using the AllenNLP library (Gardner et al., 2018).

5.4.4 Baseline Approaches

As we are studying the impact of our new paired learning objective, our main point of comparison is our Text-NMN trained without that objective. Though the focus of our work is improving learning in structured interpretable models, we also show results from a strong, reasonably comparable black-box model for DROP, MTMSN (Hu et al., 2019), to better situate the relative performance of this class of models.

Model	**Performance** (F_1 Score)	**Overall Faithfulness** (cross-entropy* ↓)	Module-wise Faithfulness* (↓)				
			`find`	`filter`	`num-date`†	`project`	`min-max`†
NMN	70.3	46.3	14.3	21.0	30.6	0.9	1.4
NMN + $\mathcal{L}_{\text{paired, all}}$	73.5	13.0	4.4	5.7	8.3	1.4	1.2

Table 6: Faithfulness scores: Using the paired objective significantly improves intermediate output predictions. †denotes the average of `find-num` & `find-date` and `find-min-num` & `find-max-num`.

5.5 Results

5.5.1 In-distribution Performance

We first evaluate the impact of our proposed paired objective on in-distribution generalization. Table 5 shows the performance of the NMNs, trained with and without the paired objective, using different types of paired examples. We see that the paired objective always leads to improved performance; test F1 improves from 70.3 F1 for the vanilla NMN to (a) 71 F1 using naturally-occurring paired examples ($\mathcal{L}_{\text{paired, found}}$), (b) 72.3 F1 using template-based paired examples ($\mathcal{L}_{\text{paired, temp}}$), and (c) 71.2 F1 using model-generated paired examples ($\mathcal{L}_{\text{paired, qgen}}$). Further, the model achieves the best performance when all kinds of paired examples are combined, improving the performance to 73.5 F1 ($\mathcal{L}_{\text{paired, all}}$). The improvement over the baseline is statistically significant ($p = 0.01$) based on the Student's t-test. Test numbers are much higher than dev since the test set contains 5 answer annotations for each question. Our final model also outperforms the black-box MTMSN model.

5.5.2 Measuring Faithfulness of NMN execution

As observed in Chapter 4, training a NMN only using the end-task supervision can lead to learned modules whose behaviour is *unfaithful* to their intended reasoning operation, even when trained and evaluated with gold programs. That is, even though the NMN might produce the correct final output, the outputs of the modules are not as expected according to the program (e.g., outputting only the longest field goal for the `find`[*field goal*] execution), and this leads to markedly worse generalization on DROP. We evaluate whether the use of our paired objective to indirectly supervise latent decisions (module outputs) in a NMN indeed leads to more faithful execution. We use the module output annotations, i.e., the correct spans that should be output by each module in a program, from Chapter 4, and report the proposed cross-entropy-based metric to quantify the divergence between the output distribution over passage tokens and the annotated spans. A lower value of this metric denotes better faithfulness of the produced outputs.

In Table 6 we see that the NMN trained with the proposed paired objective greatly improves the

Model	Complex Arithmetic			Filter-ArgMax		
	dev	test w/o G.P.	test w/ G.P.	dev	test w/o G.P.	test w/ G.P.
MTMSN	67.3	44.1		67.5	59.3	
NMN	64.3	29.5	42.1	65.0	55.6	59.7
NMN + $\mathcal{L}_{\text{paired, all}}$	67.2	47.2	**54.7**	65.5	62.3	**71.5**

Table 7: Measuring compositional-generalization: NMN performs substantially better when trained with the paired objective and performs even better when gold-programs are used for evaluation (w/ G.P).

overall faithfulness ($46.3 \rightarrow 13.0$) and also leads to huge improvements in most modules. This faithfulness evaluation shows that enforcing consistency between shared substructures provides the model with a dense enough training signal to learn correct module execution. That is, not only does the model performance improve by using the paired objective, this faithfulness evaluation shows that the model's performance is improving *for the right reasons*. In §5.5.4 we explore how this faithfulness is actually achieved.

5.5.3 Evaluating Compositional Generalization

Ou primary objective of developing a compositional structured models is that the explicit structure should help the model learn *reusable* operations that generalize to novel contexts. We test this capability using the *compositional generalization* setup of Finegan-Dollak et al. (2018), where the model is tested on questions whose program templates are unseen during training. In our case, this tests whether module executions generalize to new contexts in a program.

We create two test sets to measure our model's capability to generalize to such out-of-distribution examples. In both settings, we identify certain program templates to keep in a held-out test set, and use the remaining questions for training and validation purposes.

Complex Arithmetic This test set contains questions that require addition and subtraction operations in complex contexts: questions whose program contains `num-add`/`num-diff` as the root node, but the program is *not* the simple addition or subtraction template `num-add`/`num-diff`(`find-num`(`find`), `find-num`(`find`)). For example, *How many more mg/L is the highest amount of arsenic in drinking*

Model	Test F1			Faithful.-score (↓)
	Overall	Min-Max	Count	
NMN	57.4	82.1	36.2	110.4
+ $\mathcal{L}_{\text{max+min}}$	60.9	**85.5**	39.7	56.5
+ $\mathcal{L}_{\text{max+count}}$	60.8	81.4	43.0	99.2
+ $\mathcal{L}_{\text{max+min+count}}$	**71.1**	85.4	**58.8**	**25.9**

Table 8: Using constructed paired examples for all three types of questions—min, max, and count—leads to dramatically better count performance. Without all three, the model finds shortcuts to satisfy the consistency constraint and does not learn correct module execution.

water linked to skin cancer risk than the lowest mg/L amount?, with program `num-diff(find-num(find-max-num(find)), find-num(find-min-num(find)))`.

Filter-Argmax This test set contains questions that require an argmax operation after filter: programs that contain the subtree `find-max-num`/`find-min-num(filter(·))`. For example, *Who scored the shortest touchdown in the first half?*, with program `project(find-max-num(filter(find)))`.

Performance In Table 7 we see that a NMN using our paired objective outperforms both the vanilla NMN and the black-box MTMSN on both test sets.[2] This shows that enforcing consistent module behavior also improves their performance in novel contexts and as a result allows the model to generalize to out-of-distribution examples. We see a further dramatic improvement in performance when the model is evaluated using gold programs. This is not surprising since it is known that semantic parsers (including the one in our model) often fail to generalize compositionally (Finegan-Dollak et al., 2018; Lake and Baroni, 2018; Bahdanau et al., 2019). Recent advancements in semantic parsing models that aim at compositional generalization should help improve overall model performance (Lake, 2019; Korrel et al., 2019; Herzig and Berant, 2020).

5.5.4 Analysis

We perform an analysis to understand how augmented paired examples—ones that do not contain end-task supervision—help in improving latent decision predictions. We conduct an experiment on

[2]The test set size is quite small, so while the w/ G.P. results are significantly better than MTMSN ($p = 0.05$), we can't completely rule out noise as the cause for w/o G.P. outperforming MTMSN ($p = 0.5$), based on the Student's t-test.

a subset of the data containing only min, max and count type questions; programs in (1)-(6) from §5.3.2. We see a dramatic improvement over the baseline in count-type performance when paired examples for all three types of questions are used; answer-F1 improves from $36.2 \rightarrow 58.8$, and faithfulness from $110.4 \rightarrow 25.9$. This verifies that without additional supervision the model does indeed perform the min/max operation internal to its parameters and ground to the output event instead of performing the correct `find` operation (§5.2.1). As a result, the `find` computation that *should* be shared with the count questions is not actually shared, hurting performance. By indirectly constraining the `find` execution to produce consistent outputs for all three types of questions via the constructed question (Fig. 15), the model learns to correctly execute `find`, resulting in much better count performance. Using paired examples only for max and count questions ($\mathcal{L}_{\text{max+count}}$) does not constrain the `find` operation sufficiently—the model has freedom to optimize the paired objective by learning to incorrectly ground to the max-event mention for both the original and constructed question's `find` operation. This analysis reveals that augmented paired examples are most useful when they form enough indirect connections between different types of instances to densely characterize the decision boundary around the latent decisions.

5.6 Related Approaches

Our approach generalizes a few previous methods for learning via paired examples. For learning to ground tokens to image regions, Gupta et al. (2020b) enforce contrastive grounding between the original and a negative token; this is equivalent to using an appropriate S in our framework. A few approaches (Minervini and Riedel, 2018; Li et al., 2019; Asai and Hajishirzi, 2020) use an additional objective on final outputs to enforce domain-specific consistency between paired examples; this is a special case of our framework where S is used on the outputs (y_i, y_j), instead of the latent decisions.

More generally, the challenge in learning models for complex problems can be viewed as the emergence of artificially simple decision boundaries due to data sparsity and the presence of spurious dataset biases (Gardner et al., 2020). To counter data sparsity, data augmentation techniques have been proposed to provide a compositional inductive bias to the model (Chen et al., 2020; Andreas, 2020) or induce consistent outputs (Ribeiro et al., 2019; Asai and Hajishirzi, 2020). However, their

applicability is limited to problems where the end-task supervision for the augmented examples can be easily inferred. To counter dataset biases, model-based data pruning (Bras et al., 2020) and subsampling (Oren et al., 2020) have been proposed. All these techniques modify the training-data distribution to remove a model's propensity to find artificially simple decision boundaries, whereas we modify the training objective to try to accomplish the same goal. Ensemble-based training methodology (Clark et al., 2019; Stacey et al., 2020) has been proposed to learn models robust to dataset artifacts; however, they require prior knowledge about the kind of artifacts present in the data.

Our approach, in spirit, is related to a large body of work on learning structured latent variable models. For example, prior work has incorporated indirect supervision via constraints (Graca et al., 2007; Chang et al., 2007; Ganchev et al., 2010) or used negative examples with implausible latent structures (Smith and Eisner, 2005; Chang et al., 2010). These approaches use auxiliary objectives on a single training instance or global conditions on posterior distributions, whereas our training objective uses *paired examples*.

5.7 Summary

In thi chapter, we proposed a method to leverage *paired examples*—instances that share internal substructure—to provide a richer training signal to latent decisions in compositional model architectures. We provide a general formulation of this technique which should be applicable to a broad range of models. To validate this technique, we present a case study on our text-based neural module networks described in the previous chapters, showing how to apply the general formulation to a specific task. We explore three methods to acquire paired examples in this setting and empirically show that our approach leads to substantially better in- and out-of-distribution generalization of a neural module network in complex compositional question answering. We also show that using our paired objective leads to improved prediction of latent decisions. A lot of recent work is exploring the use of closely related instances for improved evaluation and training. Ours is one of the first works to show substantial improvements by modifying the training objective to try to make better use of the local decision surface. These results should encourage more work exploring this direction.

Chapter 6

Enforcing Consistency in Weakly Supervised Semantic Parsing

In this chapter, we shift our focus from question answering over text to weakly-supervised semantic parsing. In this setting, only (utterance, denotation) supervision is available and the semantic parse is treated as a structured latent variable that needs to be inferred from the denotation alone. A key challenge in training semantic parsers using such weak supervision is the presence of *spurious programs*—incorrect representations that evaluate to the correct denotation. This is analogous to the issue we saw in the previous chapters where, the absence of supervision for the module outputs in NMN and the fact that incorrect module outputs can still lead to the correct answer, makes learning difficult. In this chapter, we explore whether we can use our idea of enforcing consistency in the latent semantic parse of related utterances to improve learning in a weakly-supervised Seq2Seq-based semantic parser. This chapter is based on work originally described in Gupta et al. (2021a).

6.1 Introduction

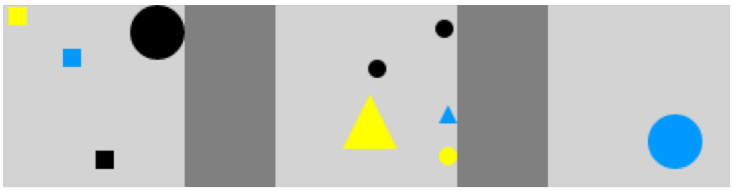
Semantic parsers map a natural language utterance into an executable meaning representation, called a logical form or program (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005). Till now,

they have formed a crucial component of question understanding in our neural module network for reasoning over text. In NMNs, the semantic parse is composed of learnable neural-network modules to execute over unstructured context. Traditionally, the programs in semantic parsers are composed of predicates in a logical language with pre-defined execution, and are executed against a structured-context (e.g., database, graph, etc.) to produce a denotation (e.g., answer) for the input utterance.

Methods for weakly-supervised semantic parsing, i.e., training semantic parsers from only (utterance, denotation) supervision have been developed (Clarke et al., 2010; Liang et al., 2011; Berant et al., 2013); however, training from such weak supervision is challenging. The semantic parse of the utterance is treated as a structured latent variable in such models. The parser needs to search for the correct program from an exponentially large space, and the presence of *spurious programs*—incorrect representations that evaluate to the correct denotation—greatly hampers learning. Several strategies have been proposed to mitigate this issue Guu et al. (2017); Liang et al. (2018); Dasigi et al. (2019). Typically these approaches consider a single input utterance at a time and explore ways to score programs.

In this chapter, we use our idea of paired training to encourage consistency between the output programs of related natural language utterances to mitigate the issue of spurious programs. Consider related utterances, *There are two boxes with three yellow squares* and *There are three yellow squares*, both containing the phrase *three yellow squares*. Ideally, the correct programs for the utterances should contain similar sub-parts that corresponds to the shared phrase. To incorporate this intuition during search, we propose a consistency-based reward to encourage programs for related utterances that share sub-parts corresponding to the shared phrases . By doing so, the model is provided with an additional training signal to distinguish between programs based on their consistency with programs predicted for related utterances.

We also show the importance of designing the logical language in a manner such that the ground-truth programs for related utterances are consistent with each other. Such consistency in the logical language would facilitate the consistency-based training proposed above, and encourage the semantic parser to learn generalizable correspondence between natural language and program tokens. We



x : There is a yellow object above a black object y : True

z_1 : objExists(black(bottom(allObjs)))

z_2 : objExists(yellow(above(black(allObjs))))

z_3 : objExists(yellow(top(allObjs)))

z_4 : objExists(black(top(allObjs)))

x' : There are 2 boxes with a yellow object above a black object

z' : boxCountEq(2, boxFilter(allBoxes,yellow(above(black))))

Figure 16: Utterance x and its program candidates z_1-z_4, all of which evaluate to the correct denotation (`True`). z_2 is the correct interpretation; other programs are *spurious*. Related utterance x' shares the phrase *yellow object above a black object* with x. Our consistency reward would score z_2 the highest since it maps the shared phrase most similarly compared to z'.

perform experiments on the Natural Language Visual Reasoning dataset (NLVR; Suhr et al., 2017), and find that in the previously proposed logical language, the use of macros leads to inconsistent interpretations of a phrase depending on its context. We propose changes to this language such that a phrase in different contexts can be interpreted by the same program parts.

6.2 Background

Natural Language Visual Reasoning (NLVR) dataset contains human-written natural language utterances, where each utterance is paired with 4 synthetically-generated images. Each (utterance, image) pair is annotated with a binary truth-value denotation. Each image is divided into three *boxes*, where each box contains 1-8 *objects*. Each object has four properties: *position*, *color*, *shape*, and *size*. A structured representation of each image is also provided, which we use in this paper.

Weakly supervised iterative search parser We use the semantic parser of Dasigi et al. (2019) as our base-parser, which is a grammar-constrained encoder-decoder with attention model from Krishnamurthy et al. (2017). It learns to map a natural language utterance x into a program z such that it evaluates to the correct denotation $y = [\![z]\!]^r$ when executed against the structured image representa-

tion r. Dasigi et al. (2019) use a manually-designed, typed, variable-free, functional query language for NLVR.

Given a dataset of triples (x_i, c_i, y_i), where x_i is an utterance, c_i is the set of images associated to it, and y_i is the set of corresponding denotations, their approach iteratively alternates between two phases to train the parser: Maximum marginal likelihood (MML) and a Reward-based method (RBM). In MML, for an utterance x_i, the model maximizes the marginal likelihood of programs in a given set of logical forms Z_i, all of which evaluate to the correct denotation. The set Z_i is constructed either by performing a heuristic search, or generated from a trained semantic parser.

The reward-based method maximizes the (approximate) expected value of a reward function $\mathcal{R}$.

$$\max_{\theta} \sum_{\forall i} \mathbb{E}_{\tilde{p}(z_i|x_i;\theta)} \mathcal{R}(x_i, z_i, c_i, y_i) \tag{6.1}$$

Here, $\tilde{p}$ is the *re-normalization* of the probabilities assigned to the programs on the beam, and the reward function $\mathcal{R} = 1$ if z_i evaluates to the correct denotation for all images in c_i, or 0 otherwise. Please refer Dasigi et al. (2019) for details.

6.3 Consistency Reward for Programs

Consider the utterance x = *There is a yellow object above a black object* in Figure 16. There are many program candidates decoded in search that evaluate to the correct denotation. Most of them are *spurious*, i.e., they do not represent the meaning of the utterance and only coincidentally evaluate to the correct output. The semantic parser is expected to distinguish between the correct program and spurious ones by identifying correspondence between parts of the utterance and the program candidates. Consider a related utterance x' = *There are 2 boxes with a yellow object above a black object*. The parser should prefer programs for x and x' which contain similar sub-parts corresponding to the shared phrase p = *yellow object above a black object*. That is, the parser should be consistent in its interpretation of a phrase in different contexts. To incorporate this intuition during program search, we propose an additional reward to programs for an utterance that are consistent with programs for

a related utterance.

Specifically, consider two related utterances x and x' that share a phrase p. We compute a reward for a program candidate z of x based on how similarly it maps the phrase p as compared to a program candidate z' of x'. To compute this reward we need (a) *relevant program parts* in z and z' that correspond to the phrase p, and (b) a *consistency reward* that measures consistency between those parts.

(a) Relevant program parts Our semantic parser (from Krishnamurthy et al. (2017)) outputs a linearized version of the program $z = [z^1, \ldots, z^T]$, decoding one action at a time from the logical language. At each time step, the parser predicts a normalized attention vector over the tokens of the utterance, denoted by $[a_1^t, \ldots, a_N^t]$ for the z^t action. Here, $\sum_{i=1}^{N} a_i^t = 1$ and $a_i^t \geq 0$ for $i \in [1, N]$. We use these attention values as a relevance score between a program action and the utterance tokens. Given the phrase p with token span $[m, n]$, we identify the relevant actions in z as the ones whose total attention score over the tokens in p exceeds a heuristically-chosen threshold $\tau = 0.6$.

$$A(z, p) = \left\{ z^t \;\middle|\; t \in [1, T] \text{ and } \sum_{i=m}^{n} a_i^t \geq \tau \right\} \tag{6.2}$$

This set of program actions $A(z, p)$ is considered to be generated due to the phrase p. For example, for utterance *There is a yellow object above a black object*, with program `objExists(yellow(above(black(allObjs))))`, this approach could identify that for the phrase *yellow object above a black object* the actions corresponding to the functions `yellow`, `above`, and `black` are relevant.

(b) Consistency reward Given a related program z' and its relevant action set $A(z', p)$, we define the consistency reward $S(z, z', p)$ as the F1 score for the action set $A(z, p)$ when compared to $A(z', p)$. Since we only consider a single paired phrase between x and x', the consistency reward between the programs z and z' can be written as $S(z, z', p)$.

As we do not know the gold program for x', we decode top-K program candidates using beam-search

and discard the ones that do not evaluate to the correct denotation. We denote this set of programs by Z'_c. Now, to compute a consistency reward $\mathcal{C}(x, z, x')$ for the program z of x,we take a weighted average of $S(z, z')$ for different $z' \in Z'_c$ where the weights correspond to the probability of the program z' as predicted by the parser.

$$\mathcal{C}(x, z, x') = \sum_{z' \in Z'_c} \tilde{p}(z'|x'; \theta) S(z, z') \tag{6.3}$$

Consistency reward based parser Given x and a related utterance x', we use $\mathcal{C}(x, z, x')$ as an additional reward in Eq. 6.1 to upweight programs for x that are consistent with programs for x'.

$$\max_{\theta} \sum_{\forall i} \mathbb{E}_{\tilde{p}(z_i|x_i;\theta)} \left[\mathcal{R}(x_i, z_i, c_i, y_i) + \mathcal{C}(x_i, z_i, x'_i) \right] \tag{6.4}$$

This consistency-based reward pushes the parser's probability mass towards programs that have consistent interpretations across related utterances, thus providing an additional training signal over simple denotation accuracy.

6.4 Consistency in Language

The consistency reward (§6.3) makes a key assumption about the logical language in which the utterances are parsed: that the gold programs for utterances sharing a natural language phrase actually correspond to each other. For example, that the phrase *yellow object above a black object* would always get mapped to `yellow(above(black))` irrespective of the utterance it occurs in.

On analyzing the logical language of Dasigi et al. (2019), we find that this assumption does not hold true. Let us look at the following examples:

x_1: *There are items of at least two different colors*

z_1: `objColorCountGrtEq(2, allObjs)`

x_2: *There is a box with items of at least two different colors*

z_2: `boxExists(memberColorCountGrtEq(2,allBoxes))`

Here the phrase *items of at least two different colors* is interpreted differently in the two utterances. In

x_2, a macro function `memberColorCountGrtEq` is used, which internally calls `objColorCountGrtEq` for each *box* in the image. Now consider,

x_3: *There is a tower with exactly one block*

z_3: `boxExists(memberObjCountEq(1,allBoxes))`

x_4: *There is a tower with a black item on the top*

z_4: `objExists(black(top(allObjs)))`

Here the phrase *There is a tower* is interpreted differently: z_3 uses a macro for filtering boxes based on their object count and interprets the phrase using `boxExists`. In the absence of a complex macro for checking *black item on the top*, z_4 resorts to using `objExists` making the interpretation of the phrase inconsistent. These examples highlight that these macros, while they shorten the search for programs, make the language inconsistent.

We make the following changes in the logical language to make it more consistent. Recall from §6.2 that each NLVR image contains 3 boxes each of which contains 1-8 objects. We remove macro functions like `memberColorCountGrtEq`, and introduce a generic `boxFilter` function. This function takes two arguments, a set of *boxes* and a filtering function `f`: `Set[Obj]` $\rightarrow$ `bool`, and prunes the input set of boxes to the ones whose objects satisfies the filter `f`. By doing so, our language is able to reuse the same object filtering functions across different utterances. In this new language, the gold program for the utterance x_2 would be

z_2: `boxCountEq(1, boxFilter(allBoxes, objColorCountGrtEq(2)))`

By doing so, our logical language can now consistently interpret the phrase *items of at least two different colors* using the object filtering function `f`: `objColorCountGrtEq(2)` across both x_1 and x_2. Similarly, the gold program for x_4 in the new logical language would be

z_4: `boxExists(boxFilter(allBoxes, black(top)))`

making the interpretation of *There is a box* consistent with x_3.

Details about our logical language In Figure 17, we show an example utterance with its gold program according to our proposed logical language. We use function composition and function currying to maintain the variable-free nature of our language. For example, action z^7 uses function

composition to create a function from Set[Object] → bool by composing two functions, from Set[Object] → bool and Set[Object] → Set[Object]. Similarly, action z^{11} creates a function from Set[Object] → Set[Object] by composing two functions with the same signature.

Actions z^8 - z^{10} use function currying to curry the 2-argument function objectCountGtEq by giving it one int=2 argument. This results in a 1-argument function objectCountGtEq(2) from Set[Object] → bool.

x: *There is one box with at least 2 yellow squares*
z: boxCountEq(1, boxFilter(allBoxes, objectCountGtEq(2)(yellow(square))))

Program actions for z:
z^1: bool → [<int,[Set[Box]:bool>, int, Set[Box]]
z^2: <int,[Set[Box]:bool> → boxCountEq
z^3: int → 1
z^4: Set[Box] → [<Set[Box],<Set[Obj]:bool>:Set[Box]>, Set[Box], <Set[Obj]:bool>]
z^5: <Set[Box],<Set[Obj]:bool>:Set[Box]> → boxFilter
z^6: Set[Box] → allBoxes
z^7: <Set[Obj]:bool> → [*, <Set[Obj]:bool>, <Set[Obj]:Set[Obj]>]
z^8: <Set[Obj]:bool> → [<int,Set[Obj]:bool>, int]
z^9: <int,Set[Obj]:bool> → objectCountGtEq
z^{10}: int → 2
z^{11}: <Set[Obj]:Set[Obj]> → [*, <Set[Obj]:Set[Obj]>, <Set[Obj]:Set[Obj]>]
z^{12}: <Set[Obj]:Set[Object]> → yellow
z^{13}: <Set[Obj]:Set[Object]> → square

Figure 17: Gold program actions for the utterance *There is one box with at least 2 yellow squares* according to our proposed logical language. The grammar-constrained decoder outputs a linearized abstract-syntax tree of the program in an in-order traversal.

6.5 Experiments

6.5.1 Dataset

We report results on the standard development, public-test, and hidden-test splits of NLVR. The training data contains 12.4k (utterance, image) pairs where each of 3163 utterances are paired with 4 images. Each evaluation set roughly contains 270 unique utterances.

6.5.2 Evaluation Metrics

(1) *Accuracy* measures the proportion of examples for which the correct denotation is predicted. (2) Since each utterance in NLVR is paired with 4 images, a *consistency* metric is used, which measures the proportion of utterances for which the correct denotation is predicted for all associated images. Improvement in this metric is indicative of correct program prediction as it is unlikely for a spurious program to correctly make predictions on multiple images.

6.5.3 Experimental Details

We use the same parser, training methodology, and hyper-parameters as Dasigi et al. (2019). To discover related utterance pairs within the NLVR dataset, we manually identify 11 sets of phrases that commonly occur in NLVR and can be interpreted in the same manner:

1. { COLOR block at the base, the base is COLOR }
2. { COLOR block at the top, the top is COLOR }
3. { COLOR1 object above a COLOR2 object }
4. { COLOR1 block on a COLOR2 block, COLOR1 block over a COLOR2 block }
5. { a COLOR tower }
6. { there is one tower, there is only one tower, there is one box, there is only one box }
7. { there are exactly NUMBER towers, there are exactly NUMBER boxes }
8. { NUMBER different colors }
9. { with NUMBER COLOR items, with NUMBER COLOR blocks, with NUMBER COLOR objects }
10. { at least NUMBER COLOR items, at least NUMBER COLOR blocks, at least NUMBER COLOR objects }

Model	Dev		Test-P		Test-H	
	Acc.	**Cons.**	**Acc.**	**Cons.**	**Acc.**	**Cons.**
Abs. Sup. (Goldman et al., 2018)	84.3	66.3	81.7	60.1	-	-
Abs. Sup. + ReRank (Goldman et al., 2018)	85.7	67.4	84.0	65.0	82.5	63.9
Iterative Search (Dasigi et al., 2019)	85.4	64.8	82.4	61.3	82.9	64.3
+ Logical Language Design (ours)	88.2	73.6	86.0	69.6	-	-
+ Consistency Reward (ours)	**89.6**	**75.9**	**86.3**	**71.0**	**89.5**	**74.0**

Table 9: Performance on NLVR: Design changes in the logical language and consistency-based training, both significantly improve performance. Larger improvements in consistency indicate that our approach efficiently tackles spurious programs.

11. { with NUMBER COLOR SHAPE, are NUMBER COLOR SHAPE, with only NUMBER COLOR SHAPE, are only NUMBER COLOR SHAPE }

In each phrase, we replace the abstract COLOR, NUMBER, SHAPE token with all possible options from the NLVR dataset to create grounded phrases. For example, *black block at the top*, *yellow object above a blue object*. For each set of equivalent grounded phrases, we identify the set of utterances that contains any of the phrase. For each utterance in that set, we pair it with 1 randomly chosen utterance from that set. Overall, we identify related utterances for 1420 utterances (out of 3163) and make 1579 pairings in total; if an utterance contains two phrases of interest, it can be paired with more than 1 utterance.

6.5.4 Baselines

We compare against the state-of-the-art models; Abs. Sup. Goldman et al. (2018) that uses abstract examples, Abs. Sup. + ReRank that uses additional data and reranking, and the iterative search parser of Dasigi et al. (2019).

6.5.5 Results

Table 9 compares the performance of our two proposed methods to enforce consistency in the decoded programs with the previous approaches. We see that changing the logical language to a more consistent one (§6.4) significantly improves performance: the accuracy improves by 2-4% and con-

sistency by 4-8% on the dev. and public-test sets. Additionally, training the parser using our proposed consistency reward (§6.3) further improves performance: accuracy improves by 0.3-0.4% but the consistency significantly improves by 1.4-2.3%.[1] On the hidden-test set of NLVR, our final model improves accuracy by 7% and consistency by 10% compared to previous approaches. Larger improvements in consistency across evaluation sets indicates that our approach to enforce consistency between programs of related utterances greatly reduces the impact of spurious programs.

6.6 Summary

In this chapter, we applied our paired training approach to weakly-supervised semantic parsing to mitigate the issue of spurious programs by enforcing consistency between latent output programs. We proposed two approaches for enforcing consistency. First, a consistency based reward that biases the program search towards programs that map the same phrase in related utterances to similar sub-parts. Such a reward provides an additional training signal to the model by leveraging related utterances. Second, we demonstrate the importance of logical language design such that it facilitates such consistency-based training. The two approaches combined together lead to significant improvements in the resulting semantic parser. Combined with the results of the previous chapter, we show that the idea of using related examples that share internal substructure can be used to provide auxiliary supervision for latent decisions in diverse compositional language understanding problems.

[1] We report average performance across 10 runs trained with different random seeds. All improvements in consistency are statistically significant (p-value < 0.05) based on the stochastic ordering test Dror et al. (2019).

Chapter 7

Neural Compositional Denotational Semantics for Question Answering

Previous chapters used neural seq2seq models for question understanding, by first encoding it is a dense vector using an encoder-LSTM, then decoding a logical form (or program) representation by using a different decoder-LSTM. The denotation for the question is computed by executing this program against the context. Though, this approach aims to model the compositional structure of the question, it differs from formal approaches to compositional semantics that hypothesize that sub-expressions in language can be interpreted independently of their context. Lack of this independence allows the seq2seq models to latch onto unintended correlations and hampers their systematic generalization (Lake and Baroni, 2018). In this chapter, we propose a question understanding approach inspired by formal approaches to semantics. Instead of representing the complete question as a dense representation, each constituent span in the question is represented by a denotation in the context (a knowledge-graph in this case) and a vector that captures ungrounded aspects of meaning. Similar to NMNs, learnable composition modules recursively combine constituent spans, culminating in a grounding for the complete question. One crucial manner in which this approach differs from traditional semantic parsers (and NMNs) is the way in which the modules are defined and parameterized. In semantic parsers, the predicates (or modules) for performing symbolic operations do not contain

any learnable parameters and a different predicate is defined for each symbolic operation, such as union, disjunctions, negations, existence, etc. Therefore, the scope of reasoning that is capable by the model is limited to the predicates that are defined in the logical language. In this approach, we define a small number of higher-order modules which contain their own parameters, but also take as input parameters from language and learn to perform their symbolic operation via data. By defining these higher-order modules, we do not limit the reasoning capability of the model to predefined operations. We demonstrate that this approach can learn a variety of challenging semantic operators, such as quantifiers, disjunctions and composed relations. Learning this model is extremely challenging, and our demonstration can be considered of an exploratory nature on questions against simple knowledge-graphs. This chapter is based on work originally described in Gupta and Lewis (2018).

7.1 Introduction

Compositionality is a mechanism by which the meanings of complex expressions are systematically determined from the meanings of their parts, and has been widely assumed in the study of both artificial and natural languages (Montague, 1973) as a means for allowing speakers to generalize to understanding an infinite number of sentences. Popular seq2seq based semantic parsing approaches use a restricted form of compositionality, typically encoding the utterance word-by-word into a dense vector representation, then decoding a logical form representation from that vector. Such models can fail to generalize from training data in surprising ways (Lake and Baroni, 2018; Finegan-Dollak et al., 2018). In this chapter, we describe a model for question understanding that is inspired by linguistic theories of compositional semantics. Our approach builds a latent tree of interpretable expressions over a question, recursively combining constituents using a small set of neural modules. The questions we consider in this chapter are posed against a knowledge graph (KG) as context.

Our approach resembles Montague semantics, in which a tree of interpretable expressions is built over the sentence, with nodes combined by a small set of composition functions. However, both the structure of the sentence and the composition functions are learned by end-to-end gradient descent. To achieve this, we define the parametric form of small set of composition modules, and then build a parse chart over each question subsuming all possible trees. Each node in the chart represents a

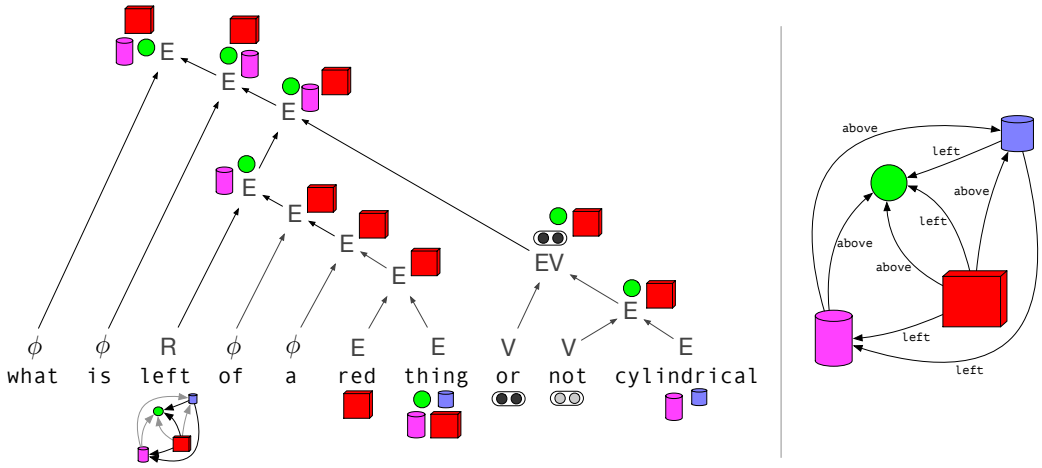


Figure 18: A correct parse for a question given the knowledge graph on the right, using our model. We show the type for each node, and its denotation in terms of the knowledge graph. The words *or* and *not* are represented by vectors, which parameterize composition modules. The denotation for the complete question represents the answer to the question. Nodes here have types E for sets of entities, R for relations, V for ungrounded vectors, EV for a combination of entities and a vector, and ϕ for semantically vacuous nodes. While we show only one parse tree here, our model builds a parse chart subsuming all trees.

span of text with a distribution over groundings (in terms of booleans and knowledge graph nodes and edges), as well as a vector representing aspects of the meaning that have not yet been grounded. The representation for a node is built by taking a weighted sum over different ways of building the node (similar to Maillard et al. (2017)). Our approach imposes independence assumptions that give a linguistically motivated inductive bias. In particular, it enforces that phrases are interpreted independently of surrounding words, allowing the model to generalize naturally to interpreting phrases in different contexts. Experiments on two datasets—one generated synthetically from a pre-defined grammar, and one containing questions written by humans (Referring Expressions (FitzGerald et al., 2013))—demonstrates the feasbility of our approach.

7.2 Model Overview

Our task is to answer a question $q = w_{1..|q|}$, with respect to a Knowledge Graph (KG) consisting of nodes $\mathcal{E}$ (representing entities) and labelled directed edges $\mathcal{R}$ (representing relationship between entities). In our task, answers are either booleans, or specific subsets of nodes from the KG.

Our model builds a parse for the sentence, in which phrases are grounded in the KG, and a small set of composition modules are used to combine phrases, resulting in a grounding for the complete question sentence that answers it. For example, in Figure 18, the phrases *not* and *cylindrical* are interpreted as a function word and an entity set, respectively, and then *not cylindrical* is interpreted by computing the complement of the entity set.

The node at the root of the parse tree is the answer to the question. Our model answers questions by:

(a) Grounding individual tokens in a KG, that can either be grounded as particular sets of entities and relations in the KG, as ungrounded vectors, or marked as being semantically vacuous. For each word, we learn parameters that are used to compute a distribution over semantic types and corresponding denotations in a KG (§ 7.3.1).

(b) Combining representations for adjacent phrases into representations for larger phrases, using trainable neural composition modules (§ 7.3.2). This produces a denotation for the phrase.

(c) Assigning a binary-tree structure to the question sentence, which determines how words are grounded, and which phrases are combined using which modules. We build a parse chart subsuming all possible structures, and train a parsing model to increase the likelihood of structures leading to the correct answer to questions. Different parses leading to a denotation for a phrase of type t are merged into an expected denotation, allowing dynamic programming (§ 7.4).

(d) Answering the question, with the most likely grounding of the phrase spanning the sentence.

7.3 Compositional Semantics

7.3.1 Semantic Types

Our model classifies spans of text into different semantic types to represent their meaning as explicit denotations, or ungrounded vectors. All phrases are assigned a distribution over semantic types. The semantic type determines how a phrase is grounded, and which composition modules can be used to

combine it with other phrases. A phrase spanning $w_{i..j}$ has a denotation $[\![w_{i..j}]\!]^t_{KG}$ for each semantic type t. For example, in Figure 18, *red* corresponds to a set of entities, *left* corresponds to a set of relations, and *not* is treated as an ungrounded vector.

The semantic types we define can be classified into three broad categories.

Grounded Semantic Types: Spans of text that can be fully grounded in the KG.

1. **Entity** (**E**): Spans of text that can be grounded to a set of entities in the KG, for example, *red sphere* or *large cube*. **E**-type span grounding is represented as an attention value for each entity, $[p_{e_1}, \ldots, p_{e_{|\mathcal{E}|}}]$, where $p_{e_i} \in [0, 1]$. This can be viewed as a soft version of a logical set-valued denotation, which we refer to as a soft entity set.

2. **Relation** (**R**): Spans of text that can be grounded to set of relations in the KG, for example, *left of* or *not right of or above*. **R**-type span grounding is represented by a soft adjacency matrix $A \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{E}|}$ where $A_{ij} = 1$ denotes a directed edge from $e_i \rightarrow e_j$.

3. **Truth** (**T**): Spans of text that can be grounded with a Boolean denotation, for example, *Is anything red?*, *Is one ball green and are no cubes red?*. **T**-type span grounding is represented using a real-value $p_{true} \in [0, 1]$ that denotes the probability of the span being true.

Ungrounded Semantic Types: Spans of text whose meaning cannot be grounded in the KG.

1. **Vector** (**V**): This type is used for spans representing functions that cannot yet be grounded in the KG (e.g. words such as *and* or *every*). These spans are represented using 4 different real-valued vectors v_1-$v_4 \in \mathbb{R}^2$-$\mathbb{R}^5$, that are used to parameterize the composition modules described in §7.3.2.

2. **Vacuous** (ϕ): Spans that are considered semantically vacuous, but are necessary syntactically, e.g. *of* in *left of a cube*. During composition, these nodes act as identity functions.

Partially-Grounded Semantic Types: Spans of text that can only be partially grounded in the knowledge graph, such as *and red* or *are four spheres*. Here, we represent the span by a combination of a grounding and vectors, representing grounded and ungrounded aspects of meaning respectively. The grounded component of the representation will typically combine with another fully grounded representation, and the ungrounded vectors will parameterize the composition module. We define 3 semantic types of this kind: **EV**, **RV** and **TV**, corresponding to the combination of entities, relations and boolean groundings respectively with an ungrounded vector. Here, the word represented by the vectors can be viewed as a binary function, one of whose arguments has been supplied.

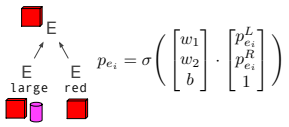
7.3.2 Composition Modules

Next, we describe how we compose phrase representations (from § 7.3.1) to represent larger phrases. We define a small set of composition modules, that take as input two constituents of text with their corresponding semantic representations (grounded representations and ungrounded vectors), and outputs the semantic type and corresponding representation of the larger constituent. The composition modules are parameterized by the trainable word vectors. These can be divided into several categories:

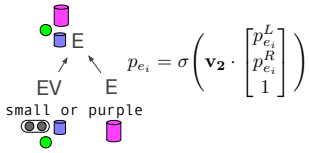
Composition modules resulting in fully grounded denotations: Described in Figure 19.

Composition with ϕ-typed nodes: Phrases with type ϕ are treated as being semantically transparent identity functions. Phrases of any other type can combined with these nodes, with no change to their type or representation.

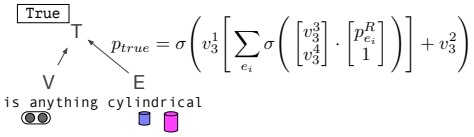
Composition modules resulting in partially grounded denotations: We define several modules that combine fully grounded phrases with ungrounded phrases, by deterministically taking the union of the representations, giving phrases with partially grounded representations (§ 7.3.1). These modules are useful when words act as binary functions; here they combine with their first argument. For example, in Fig. 18, *or* and *not cylindrical* combine to make a phrase containing both the vectors for *or* and the entity set for *not cylindrical*.



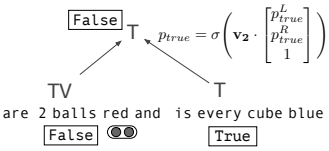
E + **E** $\rightarrow$ **E**: This module performs a function on a pair of soft entity sets, parameterized by the model's global parameter vector $[w_1, w_2, b]$ to produce a new soft entity set. The composition function for a single entity's resulting attention value is shown. Such a composition module can be used to interpret compound nouns and entity appositions. For example, the composition module shown above learns to output the intersection of two entity sets.



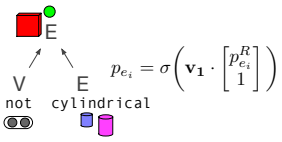
EV + **E** $\rightarrow$ **E**: This module combines two soft entity sets into a third set, parameterized by the v_2 word vector. This composition function is similar to a linear threshold unit and is capable of modeling various mathematical operations such as logical conjunctions, disjunctions, differences etc. for different values of v_2. For example, the word *or* learns to model set union.



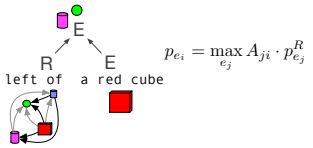
V + **E** $\rightarrow$ **T**: This module maps a soft entity set onto a soft boolean, parameterized by word vector (v_3). The module counts whether a sufficient number of elements are in (or out) of the set. For example, the word *any* should test if a set is non-empty.



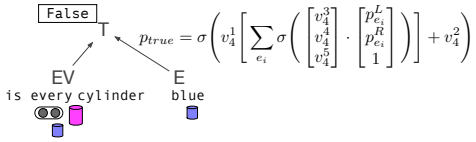
TV + **T** $\rightarrow$ **T**: This module maps a pair of soft booleans into a soft boolean using the v_2 word vector to parameterize the composition function. Similar to **EV** + **E** $\rightarrow$ **E**, this module facilitates modeling a range of boolean set operations. Using the same functional form for different composition functions allows our model to use the same ungrounded word vector (v_2) for compositions that are semantically analogous.



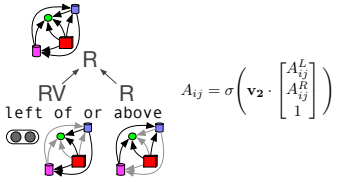
V + **E** $\rightarrow$ **E**: This module performs a function on a soft entity set, parameterized by a word vector, to produce a new soft entity set. For example, the word *not* learns to take the complement of a set of entities. The entity attention representation of the resulting span is computed by using the indicated function that takes the $v_1 \in \mathbb{R}^2$ vector of the **V** constituent as a parameter argument and the entity attention vector of the **E** constituent as a function argument.



R + **E** $\rightarrow$ **E**: This module composes a set of relations (represented as a single soft adjacency matrix) and a soft entity set to produce an output soft entity set. The composition function uses the adjacency matrix representation of the **R**-span and the soft entity set representation of the **E**-span.



EV + **E** $\rightarrow$ **T**: This module combines two soft entity sets into a soft boolean, which is useful for modelling generalized quantifiers. For example, in *is every cylinder blue*, the module can use the inner sigmoid to test if an element e_i is in the set of cylinders ($p^L_{e_i} \approx 1$) but not in the set of blue things ($p^R_{e_i} \approx 0$), and then use the outer sigmoid to return a value close to 1 if the sum of elements matching this property is close to 0.



RV + **R** $\rightarrow$ **R**: This module composes a pair of soft set of relations to a produce an output soft set of relations. For example, the relations *left* and *above* are composed by the word *or* to produce a set of relations such that entities e_i and e_j are related if either of the two relations exists between them. The functional form for this composition is similar to **EV** + **E** $\rightarrow$ **E** and **TV** + **T** $\rightarrow$ **T** modules.

Figure 19: Composition Modules that compose two constituent span representations into the representation for the combined larger span, using the indicated equations.

7.4 Parsing Model

Here, we describe how our model classifies question tokens into semantic type spans and computes their representations (§ 7.4.1), and recursively uses the composition modules defined above to parse the question into a soft latent tree that provides the answer (§ 7.4.2). The model is trained end-to-end using only question-answer supervision (§ 7.4.3).

7.4.1 Lexical Representation Assignment

Each token in the question sentence is assigned a distribution over the semantic types, and a grounded representation for each type. Tokens can only be assigned the **E**, **R**, **V**, and $\boldsymbol{\phi}$ types. For example, the token *cylindrical* in the question in Fig. 18 is assigned a distribution over the 4 semantic types (one shown) and for the **E** type, its representation is the set of *cylindrical* entities.

Semantic Type Distribution for Tokens: To compute the semantic type distribution, our model represents each word w, and each semantic type t using an embedding vector; $v_w, v_t \in \mathbb{R}^d$. The semantic type distribution is assigned with a softmax:

$$p(t|w_i) \propto \exp(v_t \cdot v_{w_i})$$

Grounding for Tokens: For each of the semantic type, we need to compute their representations:

1. **E**-Type Representation: Each entity $e \in \mathcal{E}$, is represented using an embedding vector $v_e \in \mathbb{R}^d$ based on the concatenation of vectors for its properties. For each token w, we use its word vector to find the probability of each entity being part of the **E**-Type grounding:

   $$p^w_{e_i} = \sigma(v_{e_i} \cdot v_w) \;\; \forall \, e_i \in \mathcal{E}$$

   For example, in Fig. 18, the word *red* will be grounded as all the red entities.

2. **R**-Type Representation: Each relation $r \in \mathcal{R}$, is represented using an embedding vector $v_r \in$

$\mathbb{R}^d$. For each token w_i, we compute a distribution over relations, and then use this to compute the *expected* adjacency matrix that forms the **R**-type representation for this token.

$$p(r|w_i) \propto \exp(v_r \cdot v_{w_i})$$
$$A^{w_i} = \sum_{r \in \mathcal{R}} p(r|w_i) \cdot A_r$$

e.g. the word *left* in Fig. 18 is grounded as the subset of edges with label 'left'.

3. **V**-Type Representation: For each word $w \in \mathcal{V}$, we learn four vectors $v_1 \in \mathbb{R}^2, v_2 \in \mathbb{R}^3, v_3 \in \mathbb{R}^4, v_4 \in \mathbb{R}^5$, and use these as the representation for words with the **V**-Type.

4. $\boldsymbol{\phi}$-Type Representation: Semantically vacuous words that do not require a representation.

7.4.2 Parsing Questions

To learn the correct structure for applying composition modules, we use a simple parsing model. We build a parse-chart over the question encompassing all possible trees by applying all composition modules, similar to a standard CRF-based PCFG parser using the CKY algorithm. Each node in the parse-chart, for each span $w_{i..j}$ of the question, is represented as a distribution over different semantic types with their corresponding representations.

Phrase Semantic Type Potential ($\psi^t_{i,j}$): The model assigns a score, $\psi^t_{i,j}$, to each $w_{i..j}$ span, for each semantic type **t**. This score is computed from all possible ways of forming the span $w_{i..j}$ with type **t**. For a particular composition of span $w_{i..k}$ of type $\mathbf{t_1}$ and $w_{k+1..j}$ of type $\mathbf{t_2}$, using the $\mathbf{t_1} + \mathbf{t_2} \rightarrow \mathbf{t}$ module, the composition score is:

$$\psi^{t_1+t_2 \rightarrow t}_{i,k,j} = \psi^{t_1}_{i,k} \cdot \psi^{t_2}_{k+1,j} \cdot e^{\theta \cdot f^{t_1+t_2 \rightarrow t}(i,j,k|q)}$$

where θ is a trainable vector and $f^{t_1+t_2 \rightarrow t}(i,j,k|q)$ is a simple feature function. Features consist of a conjunction of the composition module type and: the words before (w_{i-1}) and after (w_{j+1}) the span, the first (w_i) and last word (w_k) in the left constituent, and the first (w_{k+1}) and last (w_j) word

in the right constituent.

The final **t**-type potential of $w_{i..j}$ is computed by summing scores over all possible compositions:

$$\psi_{i,j}^{t} = \sum_{k=i}^{j-1} \sum_{\substack{(t_1+t_2 \to t) \\ \in \text{Modules}}} \psi_{i,k,j}^{t_1+t_2 \to t}$$

Combining Phrase Representations ($[\![w_{i..j}]\!]_{KG}^{t}$): To compute $w_{i..j}$'s **t**-type denotation, $[\![w_{i..j}]\!]_{KG}^{t}$, we compute an expected output representation from all possible compositions that result in type **t**.

$$[\![w_{i..j}]\!]_{KG}^{t} = \frac{1}{\psi_{i,j}^{t}} \sum_{k=i}^{j-1} [\![w_{i..k..j}]\!]_{KG}^{t}$$

$$[\![w_{i..k..j}]\!]_{KG}^{t} = \sum_{\substack{(t_1+t_2 \to t) \\ \in \text{Modules}}} \psi_{i,k,j}^{t_1+t_2 \to t} \cdot [\![w_{i..k..j}]\!]_{KG}^{t_1+t_2 \to t}$$

where $[\![w_{i..j}]\!]_{KG}^{t}$, is the **t**-type representation of the span $w_{i..j}$ and $[\![w_{i..k..j}]\!]_{KG}^{t_1+t_2 \to t}$ is the representation resulting from the composition of $w_{i..k}$ with $w_{k+1..j}$ using the $\mathbf{t_1} + \mathbf{t_2} \to \mathbf{t}$ composition module.

Answer Grounding: By recursively computing the phrase semantic-type potentials and representations, we can infer the semantic type distribution of the complete question and the resulting grounding for different semantic types t, $[\![w_{1..|q|}]\!]_{KG}^{t}$.

$$p(t|q) \propto \psi(1, |q|, t) \tag{7.1}$$

The answer-type (boolean or subset of entities) for the question is computed using:

$$t^* = \underset{t \in \mathbf{T},\mathbf{E}}{\text{argmax}} \; p(t|q) \tag{7.2}$$

The corresponding grounding is $[\![w_{1..|q|}]\!]_{KG}^{t^*}$, which answers the question.

7.4.3 Training Objective

Given a dataset $\mathcal{D}$ of (question, answer, knowledge-graph) tuples, $\{\mathrm{q}^i, \mathrm{a}^i, \mathrm{KG}^i\}_{i=1}^{i=|\mathcal{D}|}$, we train our model to maximize the log-likelihood of the correct answers. We maximize the following objective:

$$\mathcal{L} = \sum_i \log p(\mathrm{a}^i | \mathrm{q}^i, \mathrm{KG}^i) \tag{7.3}$$

Further details regarding the training objective are given in Appendix A.

7.5 Experimental Details

7.5.1 Dataset

We experiment with two datasets, 1) Questions generated based on the CLEVR (Johnson et al., 2017) dataset, and 2) Referring Expression Generation (GenX) dataset (FitzGerald et al., 2013), both of which feature complex compositional queries.

CLEVRGEN: We generate a dataset of question and answers based on the CLEVR dataset (Johnson et al., 2017), which contains knowledge graphs containing attribute information of objects and relations between them.

We generate a new set of questions as existing questions contain some biases that can be exploited by models.[1] We generate 75K questions for training and 37.5K for validation. Our questions test various challenging semantic operators. These include conjunctions (e.g. *Is anything red and large?*), negations (e.g. *What is not spherical?*), counts (e.g. *Are five spheres green?*), quantifiers (e.g. *Is every red thing cylindrical?*), and relations (e.g. *What is left of and above a cube?*). We create two test sets:

1. **Short Questions:** Drawn from the same distribution as the training data (37.5K).

[1] Johnson et al. (2017) found that many spatial relation questions can be answered only using absolute spatial information, and many long questions can be answered correctly without performing all steps of reasoning. We employ some simple tests to remove trivial biases from our dataset.

2. **Complex Questions:** Longer questions than the training data (22.5K). This test set contains the same words and constructions, but chained into longer questions. For example, it contains questions such as *What is a cube that is right of a metallic thing that is beneath a blue sphere?* and *Are two red cylinders that are above a sphere metallic?* Solving these questions require more multi-step reasoning.

Referring Expressions (GenX) (FitzGerald et al., 2013): This dataset contains human-generated queries, which identify a subset of objects from a larger set (e.g. *all of the red items except for the rectangle*). It tests the ability of models to precisely understand human-generated language, which contains a far greater diversity of syntactic and semantic structures. This dataset does not contain relations between entities, and instead only focuses on entity-set operations. The dataset contains 3920 questions for training, 600 for development and 940 for testing. Our modules and parsing model were designed independently of this dataset, and we re-use hyperparameters from CLEVRGEN.

7.5.2 Training Details

Training the model is challenging since it needs to learn both good syntactic structures and the complex semantics of neural modules—so we use Curriculum Learning (Bengio et al., 2009) to pre-train the model on an easier subset of questions. Appendix B contains the details of curriculum learning and other training details.

7.5.3 Baseline Models

We compare to the following baselines. **(a)** Models that assume linear structure of language, and encode the question using linear RNNs—LSTM (No KG), LSTM, Bi-LSTM, and a Relation-Network (Santoro et al., 2017) augmented model. [2] **(b)** Models that assume tree-like structure of language. We compare two variants of Tree-structured LSTMs (Zhu et al., 2015; Tai et al., 2015)—Tree-LSTM, that operates on pre-parsed questions, and Tree-LSTM(Unsup.), an unsupervised Tree-LSTM model (Maillard et al., 2017) that learns to jointly parse and represent the sentence. For GenX, we also use an

[2] We use this baseline only for CLEVRGEN since GenX does not contain relations.

Model	Boolean Questions	Entity Set Questions	Relation Questions	Overall
LSTM (No KG)	50.7	14.4	17.5	27.2
LSTM	88.5	99.9	15.7	84.9
Bi-LSTM	85.3	99.6	14.9	83.6
Tree-LSTM	82.2	97.0	15.7	81.2
Tree-LSTM (Unsup.)	85.4	99.4	16.1	83.6
Relation Network	85.6	89.7	97.6	89.4
Our Model (Pre-parsed)	94.8	93.4	70.5	90.8
Our Model	99.9	100	100	99.9

Table 10: Results for Short Questions (CLEVRGEN): Performance of our model compared to baseline models on the Short Questions test set. The LSTM (No KG) has accuracy close to chance, showing that the questions lack trivial biases. Our model almost perfectly solves all questions showing its ability to learn challenging semantic operators, and parse questions only using weak end-to-end supervision.

end-to-end semantic parsing model from Pasupat and Liang (2015). Finally, to isolate the contribution of the proposed denotational-semantics model, we train our model on pre-parsed questions. Note that, all LSTM based models only have access to the entities of the KG but not the relationship information between them. See Appendix C for details.

7.6 Results

Our experiments investigate the ability of our model to understand complex synthetic and natural language queries, learn interpretable structure, and generalize compositionally. We also isolate the effect of learning the syntactic structure and representing sub-phrases using explicit denotations.

Short Questions Performance: Table 10 shows that our model perfectly answers all test questions, demonstrating that it can learn challenging semantic operators and induce parse trees from end task supervision. Performance drops when using external parser, showing that our model learns an effective syntactic model for this domain. The Relation Network also achieves good performance, particularly on questions involving relations. LSTM baselines work well on questions not involving relations.[3]

[3]Relation questions are out of scope for these models.

Model	Non-relation Questions	Relation Questions	Overall
LSTM (No KG)	46.0	39.6	41.4
LSTM	62.2	49.2	52.2
Bi-LSTM	55.3	47.5	49.2
Tree-LSTM	53.5	46.1	47.8
Tree-LSTM (Unsup.)	64.5	42.6	53.6
Relation Network	51.1	38.9	41.5
Our Model (Pre-parsed)	94.7	74.2	78.8
Our Model	81.8	85.4	84.6

Table 11: Results for Complex Questions (CLEVRGEN): All baseline models fail to generalize well to questions requiring longer chains of reasoning than those seen during training. Our model substantially outperforms the baselines, showing its ability to perform complex multi-hop reasoning, and generalize from its training data.

Complex Questions Performance: Table 11 shows results on complex questions, which are constructed by combining components of shorter questions. These require complex multi-hop reasoning, and the ability to generalize robustly to new types of questions. We use the same models as in Table 10, which were trained on short questions. All baselines achieve close to random performance, despite high accuracy for shorter questions. This shows the challenges in generalizing RNN encoders beyond their training data. In contrast, the strong inductive bias from our model structure allows it to generalize well to complex questions. Our model outperforms Tree-LSTM (Unsup.) and the version of our model that uses pre-parsed questions, showing the effectiveness of explicit denotations and learning the syntax, respectively.

Performance on Human-generated Language: Table 12 shows the performance of our model on complex human-generated queries in GenX. Our approach outperforms strong LSTM and semantic parsing baselines, despite the semantic parser's use of hard-coded operators. These results suggest that our method represents an attractive middle ground between minimally structured and highly structured approaches to interpretation. Our model learns to interpret operators such as *except* that were not considered during development. This shows that our model can learn to parse human language, which contains greater lexical and structural diversity than synthetic questions. Trees induced by the model are linguistically plausible (see Appendix D).

Model	Accuracy
LSTM (No KG)	0.0
LSTM	64.9
Bi-LSTM	64.6
Tree-LSTM	43.5
Tree-LSTM (Unsup.)	67.7
Sempre	48.1
Our Model (Pre-parsed)	67.1
Our Model	73.7

Table 12: Results for Human Queries (GenX): Our model outperforms LSTM and semantic parsing models on complex human-generated queries, showing it is robust to work on natural language. Better performance than Tree-LSTM (Unsup.) shows the efficacy in representing sub-phrases using explicit denotations. Our model also performs better without an external parser, showing the advantages of latent syntax.

Error Analysis: We find that most model errors are due to incorrect assignments of structure, rather than semantic errors from the modules. For example, in the question *Are four red spheres beneath a metallic thing small?*, our model's parse composes *metallic thing small* into a constituent instead of composing *red spheres beneath a metallic thing* into a single node. Future work should explore more sophisticated parsing models.

Discussion: While our model shows promising results, there is significant potential for future work. Performing exact inference over large KGs is likely to be intractable, so approximations such as KNN search, beam search, feature hashing or parallelization may be necessary. To model the large number of entities in KGs such as Freebase, techniques proposed by recent work (Verga et al., 2017; Gupta et al., 2017) that explore representing entities as composition of its properties, such as, types, description etc. could be used. The modules in this work were designed in a way to provide good inductive bias for the kind of composition we expected them to model. For example, $\mathbf{EV} + \mathbf{E} \rightarrow \mathbf{E}$ is modeled as a linear composition function making it easy to represent words such as *and* and *or*. These modules can be exchanged with any other function with the same 'type signature', with different trade-offs—for example, more general feed-forward networks with greater representation capacity would be needed to represent a linguistic expression equivalent to *xor*. Similarly, more module types would be required to handle certain constructions—for example, a multiword

relation such as *much larger than* needs a $\mathbf{V} + \mathbf{V} \rightarrow \mathbf{V}$ module. Significant effort would be needed to design such modules, that take language as parameters, for textual reasoning. These are all exciting directions for future research.

7.7 Summary

In this chapter, we introduced a model for compositional question understanding that combines ideas from compositional semantics with end-to-end learning of composition operators and structure. We demonstrated that the model is able to learn a number of complex composition operators from end task supervision, and showed that the linguistically motivated inductive bias imposed by the structure of the model allows it to generalize well beyond its training data. Future work should explore scaling the model to more realistic questions by relaxing the independence assumptions, using more general composition modules, and introducing additional module types.

Chapter 8

Conclusion

The thesis began by motivating the need for developing language understanding systems that are capable of representing and reasoning about the compositional nature of language to tackle complex language-related tasks and generalize to novel problem instances in a manner humans do. We discussed how the dominant paradigm of training large monolithic neural network models using huge amounts of labeled input-output pairs for complex tasks is problematic and ignores the issue of compositional processing. We argued that due to the extremely high-dimensional nature of natural language, such data will always contain *spurious correlations*, that these expressive neural networks will latch onto, leading to models that will fail to generalize.

In Chapter 3, we present a machine learning model with a modular architecture for answering complex, compositional questions against natural language text. Our approach, based on neural module networks (Andreas et al., 2016), leverages the compositional structure provided by formal semantics by mapping the question into a formal meaning representation that dictates the reasoning structure required to answer the question. This is combined with the representational capacity of neural networks—we design an inventory of learnable neural modules to perform various atomic language understanding and symbolic reasoning tasks (e.g., arithmetic, sorting, comparisons, counting) in a human-interpretable and differentiable manner. The question's meaning representation guides the

structure in which the modules are composed yielding a question-dependent model architecture. The modules are designed to operate independently with the idea that they can be freely composed to perform novel complex reasoning that is never observed during training. Module differentiability makes the model end-to-end differentiable and allows for learning via end-task supervision. The structured architecture of the model allows us to inject background knowledge and intermediate module-output supervision via auxiliary training objectives. Our experiments on the DROP (Dua et al., 2019) dataset demonstrates the feasibility of our approach.

In Chapter 4, we study the implications of learning from just end-task supervision. We show that having a modular architecture, like the one in the previous chapter, is not enough to induce the correct problem decomposition when learning from just input-output pairs. The output supervision provides a very weak indirect signal for what the correct output of the modules should be, and is insufficient to induce correct module behavior. We introduce the concept of *module faithfulness*—whether a module learns to perform its intended task—and propose systematic evaluation metrics to quantify module correctness in both textual and visual reasoning models. We collect annotations to carry out this study, and find that in the absence of any intermediate module output supervision, the modules *do not* learn to perform their intended operation. The composition of modules forms an expressive neural neural network without any constraints on the intermediate outputs, and as a result, the modules do not learn to separate out the sub-tasks intended for them only using end-task supervision. We also show that modules trained in such manner do not compose freely in novel contexts which in turn hurts generalization.

In Chapter 5, to address the limitations of learning from end-task supervision, we propose a new training paradigm that leverages *paired examples*—training instances that share internal substructure—to provide indirect supervision to the intermediate outputs of a model that performs explicit problem decomposition. In this approach, when two related training examples share internal substructure, we add an additional training objective to encourage consistency between the latent outputs resulting from the shared substructure. This objective does not require external supervision for the values of the latent decisions, yet provides an additional training signal to that provided by individual train-

ing examples. We apply this technique to our modular question-answering model by enforcing that similar question-program subtrees for different questions, yield the same output from partial model execution. For example, for *How many field goals were scored?* and *What is the longest field goal?*, we will encourage that the output of the sub-computation '*what are the field goals*' yields the same output for both the questions. We also explore different ways of acquiring paired training examples. Our experiments on the DROP dataset demonstrate that this training paradigm when applied to our model results in significantly better in- and out-of distribution generalization, and also significantly improves the learning of modules to perform their intended tasks, according to the evaluation proposed in the previous chapter.

In Chapter 6, we apply our paired training paradigm to another language understanding task in which the model predicts a structured, human-interpretable latent output—semantic parsing in a weakly supervised setting. The key challenge in learning semantic parsers from weak supervision is the prevalence of *spurious programs*—programs that do not represent the meaning of the utterance yet execute to the correct output. Existence of such programs makes the learning quite challenging since there is no way to distinguish such programs from the correct ones. We apply our paired training approach to a seq2seq semantic parser to reward programs that map the same phrase in paired inputs to the same sub-parts in their respective programs. This ensures that a phrase in different utterances is consistently interpreted, thus allowing the parser to induce the correct utterance decomposition and learn generalizable correspondence between natural language and program tokens. On the Natural Language Visual Reasoning (NLVR) dataset Suhr et al. (2017) we demonstrate that this approach yields significant improvements over previous best approaches.

In Chapter 7, we propose a fully-grounded question understanding approach that resembles Montague semantics (Montague, 1973). This approach differs from the approaches used throughout this dissertation in two key ways. Firstly, in the seq2seq based semantic parsers used previously, there are no predictions for explicit meaning representations of the sub-expressions of the question, i.e., parts of the question are not interpreted independently of their context. Lack of this independence limits a seq2seq model's ability to process language compositionally, and allows the model to

latch onto unintended correlations, which hampers their systematic generalization (Lake and Baroni, 2018). Instead, in the approach proposed in this chapter, each constituent span in the question is represented by a denotation in the context (a knowledge-graph in this case) and a vector that captures ungrounded aspects of meaning. Subsequently, learnable composition modules recursively combine constituent spans, culminating in a denotation for the complete question. The other key difference in this approach is how the modules are parameterized. Similar to previous approaches, modules contain learnable parameters, but in this approach some modules can also take 'language' parameters as argument to perform different operations using the same module. For example, previously we would define two separate modules `add(N, N)` $\rightarrow$ `N` and `subtract(N, N)` $\rightarrow$ `N` to carry out the two operations. In this model, we can define a single module `arithmetic(N, N, word)` $\rightarrow$ `N` to carry out different operations by using the parameters associated with the `word` to change the operational form of the module. This allows us to define a small number of compositional modules and let the data guide the learning of semantic operators required to solve the problems. We demonstrate the efficacy of this approach on data generated from a pre-defined grammar and on human-written utterances from the Referring Expressions (FitzGerald et al., 2013) dataset, and show that it leads to significantly better systematic generalization. The approach we describe has limitations which limits its direct usage in a real-world setting, but is an interesting direction that warrants future research.

8.1 Summary of Contributions

The principal contributions of this work can be summarized as follows:

1. We present a modular approach, based on neural module networks (NMNs; Andreas et al., 2016), for answering compositional questions against open-domain natural language paragraphs. We design modules for performing basic natural language understanding operations (e.g., grounding entities/events in paragraph, predicate-argument extraction) and discrete symbolic operations on numbers and dates (e.g., counting, min/max, arithmetic, comparisons) in a probabilistic and differentiable manner. This leads to an end-to-end differentiable model that can be trained from end-task supervision, and is capable of performing complex reasoning in an interpretable manner.

2. We demonstrate the implications of learning from just end-task supervision in compositional structured models (e.g., neural module networks (NMNs)). We develop systematic evaluation techniques for measuring the correctness of intermediate outputs (e.g., outputs of modules) in compositional models for textual and visual reasoning. Our evaluations demonstrate that models do not learn to perform the correct intermediate reasoning tasks from just weak end-task supervision. Furthermore, training in such manner hurts interpretability and generalization.

3. We design a new training paradigm that improves learning in compositional structured models by leveraging groups of *related training examples* that share internal substructure. By enforcing consistency in the latent decisions corresponding to the shared substructure, this approach induces the correct compositional processing and is able to provide additional indirect supervision beyond what is provided by individual examples. We demonstrate that this leads to improved performance in neural module networks and semantic parsing when trained in weakly-supervised settings.

4. We provide a fully-grounded question understanding approach for answering compositional questions against knowledge-graphs. This approach presents a significant departure from Seq2Seq methods for semantic parsing by recursively building an explicit tree containing meaning representations for the sub-expressions of the question. Further, the composition modules designed for this approach are parameterized by language mitigating the need to manually define discrete operators. The model learns a variety of challenging semantic operators (e.g., quantifiers, disjunctions and composed relations) in a data-driven manner.

8.2 Future Directions

In many aspects, the research done in this dissertation is exploratory in nature. This exploration of encouraging compositional processing in language understanding systems has revealed several challenges and interesting directions for future research. We briefly describe a few of them below.

Context-conditional planning One key drawback of our approach is that the reasoning plan for a question (question $\rightarrow$ program) is predicted independently of the information presented by the context. This is limiting since different operations might be needed to answer the same question depending on the available information. On the contrary, a human would possibly approach this problem by generating multiple potential plans and estimating their feasibility on-the-fly by performing partial executions. Developing such planning-based question-answering models should be explored.

Paired Training for Pre-trained Language Models In the last couple of years, large pre-trained language models for contextualized text representations have shown extremely impressive performance on a variety of benchmark NLP tasks (Devlin et al., 2019; Brown et al., 2020). At the same time, the compositional generalization capabilities of these models have been brought under scrutiny by various studies (Gardner et al., 2020; Oren et al., 2020; Akula et al., 2020). In Chapters 5 and 6 we show that our paired training paradigm can induce compositional processing in modular architectures without requiring additional supervision. It will be interesting to explore the idea of paired training to devise unsupervised objectives for pre-training on unlabeled data to provide language models with the required compositional inductive bias.

Diverse reasoning phenomena Our question-answering model in Chapter 3 is limited in its reasoning capability by the inventory of modules defined in the model. While it is capable of handling a diverse set of phenomena, many interesting phenomena are yet to be explored. Performing reasoning that requires key-value representations, and modeling challenging linguistic quantifiers (e.g., generalized quantifiers) that require set-theoretic reasoning, in a differentiable manner, over distributed representations of text, is important but also quite challenging. Similarly, exploring the idea presented in Chapter 7 of defining higher-order modules that can be parameterized by language to learn semantic operations in a data-driven manner in real-world settings is an interesting direction for future research. Such approaches can allow scaling models with modular architectures to reason about a diverse range of challenging phenomena without combinatorially blowing up the space of logical meaning representations or requiring significant human effort.

The work presented in this dissertation explores the idea of compositional reasoning in language understanding systems. This is an intriguing direction with a wide variety of questions that remain to be explored and answered. We believe that to develop AI agents with human-like capabilities, they need to be able to understand the world in a compositional manner.

APPENDIX

A.1 Auxiliary Supervision in NMN

In this section, we describe how the auxiliary supervision is derived for training our QA model in Chapter 3.

Auxiliary Question Parse Supervision For questions with parse supervision $\mathbf{z}^*$, we decouple the marginal likelihood into two maximum likelihood objectives, $p(\mathbf{z}^*|q)$ and $p(y^*|\mathbf{z}^*)$. We also add a loss for the decoder to attend to the tokens in the question attention supervision when predicting the relevant modules. The question attention supervision is provided as a mutli-hot vector $\alpha^* \in \{0,1\}^n$. The loss against the predicted attention vector α is, $Q_{\text{loss}} = -\sum_{i=1}^{n} \alpha_i^* \log \alpha_i$. Since the predicted attention is a normalized distribution, the objective increases the sum of log-probabilities of the tokens in the supervision.

The following patterns are used to extract the question parse supervision for the training data:

1. *what happened first SPAN1 or SPAN2?*

   `span(compare-date-lt(find(), find()))`: with `find` attentions on SPAN1 and SPAN2, respectively. Use `compare-date-gt`, if *second* instead of *first*.

2. *were there fewer SPAN1 or SPAN2?*

   `span(compare-num-lt(find(), find()))`: with `find` attentions on SPAN1 and SPAN2, respectively. Use `compare-num-gt`, if *more* instead of *fewer*.

3. *how many yards was the longest {touchdown / field goal}?*

   `find-num(find-max-num(find()))`: with `find` attention on *touchdown* / *field goal*. For *shortest*, the `find-min-num` module is used.

4. *how many yards was the longest {touchdown / field goal} SPAN ?*

   `find-num(find-max-num(filter(find())))`: with `find` attention on *touchdown* / *field goal* and

filter attention on all SPAN tokens.

5. *how many {field goals, touchdowns, passes} were scored SPAN?*

   `count(filter(find()))`: with `find` attention on *{field goals, touchdowns, passes}* and `filter` attention on SPAN.

6. *who {kicked, caught, threw, scored} SPAN?*

   `span(relocate(filter(find())))`: with `relocate` attention on *{kicked, caught, threw, scored}*, `find` attention on *{touchdown / field goal}*, and `filter` attention on all other tokens in the SPAN.

Heuristic Intermediate Module Output Supervision As mentioned in Section 4.3, we heuristically find supervision for the output of the `find-num` and `find-date` module for a subset of questions that already contain question program supervision. These are as follows:

1. *how many yards was the longest/shortest {touchdown, field goal}?*

   We find all instances of touchdown/field goal in the passage and assume that the number appearing closest should be an output of the `find-num` module.

2. *what happened first EVENT1 or EVENT2?*

   Similar to above, we perform fuzzy matching to find the instance of EVENT1 and EVENT2 in the paragraph and assume that the closest dates should be the output of the two `find-date` module calls made by the `compare-date-lt` module in the gold program.

3. *were there fewer SPAN1 or SPAN2?*

   This is exactly the same as previous for `find-num` module calls by `compare-num-lt`.

A.2 Measuring Faithfulness in Visual-NMN

A.2.1 Numerators of Precision and Recall

As stated in Section 4.3.1, for a given module type and a given example, precision is defined as the number of matched proposed bounding boxes divided by the number of proposed bounding boxes to

which the module assigns a probability more than 0.5. Recall is defined as the number of matched annotated bounding boxes divided by the number of annotated bounding boxes. Therefore, the numerators of the precision and the recall need not be equal. In short, the reason for the discrepancy is that there is no one-to-one alignment between annotated and proposed bounding boxes. To further illustrate why we chose not to have a common numerator, we will consider two sensible choices for this shared numerator and explain the issues with them.

One choice for the common numerator is the number of matched proposed bounding boxes. If we were to keep the denominator of the recall the same, then the recall would be defined as the number of matched proposed bounding boxes divided by the number of annotated bounding boxes. Consider an example in which there is a single annotated bounding box that is aligned with five proposed bounding boxes. When this definition of recall is applied to this example, the numerator would exceed the denominator. Another choice would be to set the denominator to be the number of proposed bounding boxes that are aligned with some annotated bounding box. In the example, this approach would penalize a module that gives high probability to only one of the five aligned proposed bounding boxes. However, it is not clear that a module giving high probability to all five proposed boxes is more faithful than a module giving high probability to only one bounding box (e.g. perhaps one proposed box has a much higher IOU with the annotated box than the other proposed boxes). Hence, this choice for the numerator does not make sense.

Another choice for the common numerator is the number of matched annotated bounding boxes. If we were to keep the denominator of the precision the same, then the precision would be defined as the number of matched annotated bounding boxes divided by the number of proposed bounding boxes to which the module assigns probability more than 0.5. Note that since a single proposed bounding box can align with multiple annotated bounding boxes, it is possible for the numerator to exceed the denominator.

Thus, these two choices for a common numerator have issues, and we avoid these issues by defining the numerators of precision and recall separately.

A.3 Details about Modules

See Table 13 for details. First five contain parameters, the rest are deterministic. The implementation of `count` shown here is the Sum-count version; please see Section 4.4 for a description of other count module varieties and a discussion of their differences. 'B' denotes the Boolean type, which is a probability value ([0..1]). 'N' denotes the Number type which is a probability distribution. $K = 72$ is the maximum count value supported by our model. To obtain probabilities, we first convert each Normal random variable X to a categorical distribution over $\{0, 1, ..., K\}$ by setting $\Pr[X = k] = \Phi(k + 0.5) - \Phi(k - 0.5)$ if $k \in \{1, 2, ..., K - 1\}$. We set $\Pr[X = 0] = \Phi(0.5)$ and $\Pr[X = K] = 1 - \Phi(K - 0.5)$. Here $\Phi(\cdot)$ denotes the cumulative distribution function of the Normal distribution. W_1, W_2 are weight vectors with shapes $2h \times 1$ and $h \times 1$, respectively. Here $h = 768$ is the size of LXMERT's representations. b_1 is a scalar weight. MLP denotes a two-layer neural network with a GeLU activation Hendrycks and Gimpel (2016) between layers. x denotes a question representation, and v_i denotes encodings of objects in the image. x and v_i have shape $h \times |\mathcal{B}|$, where $|\mathcal{B}|$ is the number of proposals. p denotes a vector of probabilities for each proposal and has shape $1 \times |\mathcal{B}|$. $\odot$ and $[;]$ represent elementwise multiplication and matrix concatenation, respectively. The expressions for the mean and variance in the division module are based on the approximations in Seltman (2018). The macros execute a given program on the two input images. `in-at-least-one-image` macro returns true iff the program returns true when executed on at least one of the images. `in-each-image` returns true iff the program returns true when executed on both of the images. `in-one-other-image` takes two programs and returns true iff one program return true on left image and second program returns true on right image, or vice-versa.

A.4 Significance tests

We perform a paired permutation test to test the hypothesis H_0: *NMN w/ Graph-count + decont. + pretraining* has the same inherent faithfulness as *NMN w/ Layer-count*. We follow the procedure described by Ventura (2007), which is similar to tests described by Yeh (2000) and Noreen (1989). Specifically, we perform $N_{total} = 100,000$ trials in which we do the following. For every example,

Module	Output	Implementation
$\texttt{find}[q_{att}]$	p	$W_1^T([x;v]) + b_1$
$\texttt{filter}[q_{att}](p)$	p	$p \odot (W_1^T([x;v]) + b_1)$
$\texttt{with-relation}[q_{att}](p_1, p_2)$	p	$\max(p_2) p_1 \odot \text{MLP}([x; v_1; v_2])$
$\texttt{project}[q_{att}](p)$	p	$\max(p)\texttt{find}(q_{att}) \odot \text{MLP}([W_2; v_1; v_2])$
$\texttt{count}(p)$	N	$\texttt{number}\left(\sum(p), \sigma^2\right)$
$\texttt{exist}(p)$	B	greater-equal(p, 1)
$\texttt{greater-equal}\ (a : \text{N}, b : \text{N})$	B	greater(a, b) + equal(a, b)
$\texttt{less-equal}\ (a : \text{N}, b : \text{N})$	B	less(a, b) + equal(a, b)
$\texttt{equal}(a : \text{N}, b : \text{N})$	B	$\sum_{k=0}^{K} \Pr[a = k] \Pr[b = k]$
$\texttt{less}(a : \text{N}, b : \text{N})$	B	$\sum_{k=0}^{K} \Pr[a = k] \Pr[b > k]$
$\texttt{greater}(a : \text{N}, b : \text{N})$	B	$\sum_{k=0}^{K} \Pr[a = k] \Pr[b < k]$
$\texttt{and}(a : \text{B}, b : \text{B})$	B	a*b
$\texttt{or}(a : \text{B}, b : \text{B})$	B	a+b-a*b
$\texttt{number}(m : \text{F}, v : \text{F})$	N	$\text{Normal}(\text{mean} = m, \text{var} = v)$
$\texttt{sum}(a : \text{N}, b : \text{N})$	N	$\texttt{number}\ (a_{mean} + b_{mean}, a_{var} + b_{var})$
$\texttt{difference}(a : \text{N}, b : \text{N})$	N	$\texttt{number}\ (a_{mean} - b_{mean}, a_{var} + b_{var})$
$\texttt{division}(a : \text{N}, b : \text{N})$	N	$\texttt{number}\ \left(\frac{a_{mean}}{b_{mean}} + \frac{b_{var} a_{mean}}{b_{mean}^3}, \frac{a_{mean}^2}{b_{mean}^2}\left(\frac{a_{var}}{a_{mean}^2} + \frac{b_{var}}{b_{mean}^2}\right)\right)$
$\texttt{intersect}(p_1, p_2)$	p	$p_1 \cdot p_2$
$\texttt{discard}(p_1, p_2)$	p	$\max(p_1 - p_2, 0)$
$\texttt{in-left-image}(p)$	p	p s.t. probabilities for right image are 0
$\texttt{in-right-image}(p)$	p	p s.t. probabilities for left image are 0
in-at-least-one-image	B	macro (see caption)
in-each-image	B	macro (see caption)
in-one-other-image	B	macro (see caption)

Table 13: Implementations of modules for NLVR2 NMN.

with probability $1/2$ we swap the F_1 scores obtained by the two models for that example. Then we check whether the difference in the aggregated F_1 scores for the two models is at least as extreme as the original difference in the aggregated F_1 scores of the two models. The p-value is given by $\frac{N_{exceed}}{N_{total}}$, where N_{exceed} is the number of trials in which the new difference is at least as extreme as the original difference.

GLOSSARY

Chapter 3	
q	input natural language question
p	natural language paragraph provided as context
z	predicted logical form representation of q
y	predicted output answer for the question q
$[\![z]\!]$	output of the execution of z (= y)
$\mathbf{Q}$	contextualized token representations of the question
$\mathbf{P}$	contextualized token representations of the question
J	maximum marginal likelihood training objective
$\mathbf{R}$	attention map for predicate-argument extraction
$\mathbf{A}^{\text{num}}$	attention map from tokens to number tokens
$\mathbf{A}^{\text{date}}$	attention map from tokens to date tokens
$H^{\text{r}}_{\text{loss}}$	unsupervised auxiliary loss for argument attention map
$H^{\text{n}}_{\text{loss}}$	unsupervised auxiliary loss for number attention map
$H^{\text{d}}_{\text{loss}}$	unsupervised auxiliary loss for date attention map
H_{loss}	final unsupervised auxiliary loss ($H^{\text{n}}_{\text{loss}} + H^{\text{d}}_{\text{loss}} + H^{\text{r}}_{\text{loss}}$)
Chapter 4	
x	input natural language utterance (question or statement)
z	logical meaning representation of x as an executable program
y	output denotation by executing z (answer or truth-value)
$\mathcal{B}$	set of bounding boxes identified in an image
$\boldsymbol{p}$	output probability for each bounding box $\in [0, 1]^{\lvert\mathcal{B}\rvert}$
Chapter 5	
x	input example
z	computation tree for x (e.g., $z = f(g(x), h(x))$)

$[\![z]\!]$	output of the execution of z
y	output for x, i.e., $y = [\![z]\!]$
x_i	a particular training example
x_j	paired example for x_i
$g(x_i)$	internal computation for x_i paired with x_j
$g(x_j)$	internal computation for x_j shared with x_i
$S([\![g(x_i)]\!], [\![g(x_j)]\!])$	consistency metric between $g(x_i)$ and $g(x_j)$
$\mathcal{L}_{\text{paired}}$	paired training objective based on the consistency metric

Chapter 6

x	input natural language utterance
c	set of images provided as context with x
y	gold output supervision for x (truth-value in NLVR)
z	logical meaning representation of x as an executable program
$[\![z]\!]$	output of the execution of z
$\mathcal{R}(x, z, c, y)$	reward function to score z
x'	paired utterance related to x
p	shared phrase between x and x'
$A(z, p)$	set of relevant program actions in z corresponding to p
$S(z, z', p)$	consistency reward between z and z' based on shared phrase p
$C(x, z, x')$	consistency reward for program z of x

Chapter 7

q	input natural language question
$w_{1..\|q\|}$	tokens in the question
$[\![q]\!]$	output denotation of q (= $[\![w_{1..\|q\|}]\!]$)
KG	knowledge-graph associated with q
$\mathcal{E}$	set of entities (nodes) in the KG
$\mathcal{R}$	relationships between entities (directed labeled edges) in the KG

$p(t\|w_i)$	semantic type distribution for a token
$\psi_{i,j}^t$	semantic t-type potential for phrase $w_{i..j}$
$[\![w_{i..j}]\!]^t$	t-type denotation for $w_{i..j}$
$\mathcal{L}$	maximum log-likelihood training objective

BIBLIOGRAPHY

A. Agrawal, J. Lu, S. Antol, M. Mitchell, C. L. Zitnick, D. Parikh, and D. Batra. VQA: Visual Question Answering. *International Journal of Computer Vision*, 2015.

A. R. Akula, S. Gella, Y. Al-Onaizan, S. Zhu, and S. Reddy. Words aren't enough, their order matters: On the Robustness of Grounding Visual Referring Expressions. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

J. Andreas. Good-Enough Compositional Data Augmentation. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Learning to Compose Neural Networks for Question Answering. In *Proc. of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, San Diego, California, 2016. URL `https://www.aclweb.org/anthology/N16-1181`.

Y. Artzi and L. Zettlemoyer. Bootstrapping Semantic Parsers from Conversations. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011.

Y. Artzi and L. Zettlemoyer. Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions. *Transactions of the Association for Computational Linguistics*, 2013.

A. Asai and H. Hajishirzi. Logic-Guided Data Augmentation and Regularization for Consistent Question Answering. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR*, abs/1409.0473, 2015.

D. Bahdanau, H. D. Vries, T. J. O'Donnell, S. Murty, P. Beaudoin, Y. Bengio, and A. C. Courville. CLOSURE: Assessing Systematic Generalization of CLEVR Models. In *ViGIL@NeurIPS*, 2019.

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proc. of the International Conference on Machine Learning (ICML)*, 2009.

J. Berant and P. Liang. Semantic Parsing via Paraphrasing. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2014.

J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on Freebase from Question-Answer pairs. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

R. L. Bras, S. Swayamdipta, C. Bhagavatula, R. Zellers, M. E. Peters, A. Sabharwal, and Y. Choi. Adversarial Filters of Dataset Biases. In *Proc. of the International Conference on Machine Learning (ICML)*, 2020.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, Am, a Askell, S, hini Agarwal, A. Herbert-Voss, G. Krüger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McC, lish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners. *ArXiv*, abs/2005.14165, 2020.

Q. Cai and A. Yates. Large-scale Semantic Parsing via Schema Matching and Lexicon Extension. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2013.

B. Carpenter. Type-Logical Semantics. 1997.

M.-W. Chang, L. Ratinov, and D. Roth. Guiding Semi-Supervision with Constraint-Driven Learning. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 280–287, Prague, Czech Republic, 6 2007. Association for Computational Linguistics. URL `http://cogcomp.org/papers/ChangRaRo07.pdf`.

M.-W. Chang, V. Srikumar, D. Goldwasser, and D. Roth. Structured Output Learning with Indirect Supervision. In *Proc. of the International Conference on Machine Learning (ICML)*, 2010. URL `http://cogcomp.org/papers/CSGR10.pdf`.

X. Chen, C. Liang, A. W. Yu, D. Zhou, D. Song, and Q. V. Le. Neural Symbolic Reader: Scalable Integration of Distributed and Symbolic Representations for Reading Comprehension. In *Proc. of the International Conference on Learning Representations*, 2020.

K. Cho, B. V. Merrienboer, Çaglar Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

C. Clark, M. Yatskar, and L. Zettlemoyer. Don't Take the Easy Way Out: Ensemble Based Methods for Avoiding Known Dataset Biases. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

J. Clarke, D. Goldwasser, M.-W. Chang, and D. Roth. Driving Semantic Parsing from the World's Response. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*, 7 2010. URL `http://cogcomp.org/papers/CGCR10.pdf`.

Y. Cui, Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu. Attention-over-Attention Neural Networks for Reading Comprehension. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

P. Dasigi, M. Gardner, S. Murty, L. Zettlemoyer, and E. Hovy. Iterative Search for Weakly Supervised Semantic Parsing. In *Proc. of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

Y. Deng, Y. Kim, J. T. Chiu, D. Guo, and A. M. Rush. Latent Alignment and Variational Attention. In *NeurIPS*, 2018.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*, 2019.

B. Dhingra, H. Liu, Z. Yang, W. Cohen, and R. Salakhutdinov. Gated-Attention Readers for Text Comprehension. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017. URL https://www.aclweb.org/anthology/P17-1168.

L. Dong and M. Lapata. Language to Logical Form with Neural Attention. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016.

R. Dror, S. Shlomov, and R. Reichart. Deep dominance-how to properly compare deep neural models. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

X. Du, J. Shao, and C. Cardie. Learning to Ask: Neural Question Generation for Reading Comprehension. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner. DROP: A Reading Comprehension Benchmark Requiring Discrete Reasoning Over Paragraphs. In *NAACL-HLT*, 2019.

C. Dyer, A. Kuncoro, M. Ballesteros, and N. A. Smith. Recurrent Neural Network Grammars. In *HLT-NAACL*, 2016.

E. S. A. Efrat and M. Shoham. Tag-based Multi-Span Extraction in Reading Comprehension. 2019. URL https://github.com/eladsegal/project-NLP-AML.

C. Finegan-Dollak, J. K. Kummerfeld, L. Zhang, K. Ramanathan, S. Sadasivam, R. Zhang, and D. Radev. Improving Text-to-SQL Evaluation Methodology. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018. URL https://www.aclweb.org/anthology/P18-1033.

N. FitzGerald, Y. Artzi, and L. S. Zettlemoyer. Learning Distributions over Logical Forms for Referring Expression Generation. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

K. Ganchev, J. Graça, J. Gillenwater, and B. Taskar. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research (JMLR)*, 2010.

M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. F. Liu, M. E. Peters, M. Schmitz, and L. S. Zettlemoyer. AllenNLP: A Deep Semantic Natural Language Processing Platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, 2018.

M. Gardner, J. Berant, H. Hajishirzi, A. Talmor, and S. Min. Question Answering is a Format; When is it Useful? *ArXiv*, abs/1909.11291, 2019.

M. Gardner, Y. Artzi, V. Basmova, J. Berant, B. Bogin, S. Chen, P. Dasigi, D. Dua, Y. Elazar, A. Gottumukkala, N. Gupta, H. Hajishirzi, G. Ilharco, D. Khashabi, K. Lin, J. Liu, N. F. Liu, P. Mulcaire, Q. Ning, S. Singh, N. A. Smith, S. Subramanian, R. Tsarfaty, E. Wallace, A. Zhang,

and B. Zhou. Evaluating Models' Local Decision Boundaries via Contrast Sets. In *Findings of EMNLP*, 2020.

O. Goldman, V. Latcinnik, U. Naveh, A. Globerson, and J. Berant. Weakly-supervised Semantic Parsing with Abstract Examples. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

D. Goldwasser and D. Roth. Learning from Natural Instructions. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011. URL http://cogcomp.org/papers/GoldwasserRo11(2).pdf.

J. V. Graca, K. Ganchev, and B. Taskar. Expectation maximization and posterior constraints. In *NIPS*, 2007.

N. Gupta and M. Lewis. Neural Compositional Denotational Semantics for Question Answering. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

N. Gupta, S. Singh, and D. Roth. Entity Linking via Joint Encoding of Types, Descriptions, and Context. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017. URL http://cogcomp.org/papers/GuptaSiRo17.pdf.

N. Gupta, K. Lin, D. Roth, S. Singh, and M. Gardner. Neural Module Networks for Reasoning over Text. In *Proc. of the International Conference on Learning Representations*, 2020a. URL https://cogcomp.seas.upenn.edu/papers/GLRSG20.pdf.

N. Gupta, S. Singh, and M. Gardner. Enforcing Consistency in Weakly-Supervised Semantic Parsing. In *Under Review*, 2021a.

N. Gupta, S. Singh, M. Gardner, and D. Roth. Paired Examples as Indirect Supervision in Latent Decision Models. *arXiv preprint arXiv:2104.01759*, 2021b. URL https://arxiv.org/abs/2104.01759.

T. Gupta, A. Vahdat, G. Chechik, X. Yang, J. Kautz, and D. Hoiem. Contrastive Learning for Weakly Supervised Phrase Grounding. *Proc. of the European Conference on Computer Vision (ECCV)*, 2020b.

S. Gururangan, S. Swayamdipta, O. Levy, R. Schwartz, S. R. Bowman, and N. A. Smith. Annotation Artifacts in Natural Language Inference Data. In *NAACL-HLT*, 2018.

K. Guu, P. Pasupat, E. Liu, and P. Liang. From Language to Programs: Bridging Reinforcement Learning and Maximum Marginal Likelihood. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

K. Hermann, T. Kociský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching Machines to Read and Comprehend. In *NIPS*, 2015.

J. Herzig and J. Berant. Span-based Semantic Parsing for Compositional Generalization. *ArXiv*, abs/2009.06040, 2020.

M. J. Hosseini, H. Hajishirzi, O. Etzioni, and N. Kushman. Learning to solve arithmetic word problems with verb categorization. In *Proc. of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2014.

M. Hu, Y. Peng, Z. Huang, and D. Li. A Multi-Type Multi-Span Network for Reading Comprehension that Requires Discrete Reasoning. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 804–813, 2017.

D. A. Hudson and C. D. Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6700–6709, 2019.

S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer. Mapping Language to Code in Programmatic Context. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Y. Ji, C. Tan, S. Martschat, Y. Choi, and N. A. Smith. Dynamic Entity Representations in Neural Language Models. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

R. Jia and P. Liang. Adversarial Examples for Evaluating Reading Comprehension Systems. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. Girshick. CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *Proc. of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2013.

R. J. Kate, Y. W. Wong, and R. Mooney. Learning to Transform Natural to Formal Languages. In *Proc. of the Conference on Artificial Intelligence (AAAI)*, 2005.

D. Khashabi, T. Khot, A. Sabharwal, P. Clark, O. Etzioni, and D. Roth. Question Answering via Integer Programming over Semi-Structured Knowledge. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2016. URL `http://cogcomp.org/papers/KKSCER16.pdf`.

D. Khashabi, T. Khot, A. Sabharwal, and D. Roth. Question Answering as Global Reasoning over Semantic Abstractions. In *Proc. of the Conference on Artificial Intelligence (AAAI)*, 2018. URL http://cogcomp.org/papers/2018_aaai_semanticilp.pdf.

T. Khot, N. Balasubramanian, E. Gribkoff, A. Sabharwal, P. Clark, and O. Etzioni. Exploring Markov Logic Networks for Question Answering. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

C. Kiddon, L. Zettlemoyer, and Y. Choi. Globally Coherent Text Generation with Neural Checklist Models. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

J. Kinley and R. Lin. NABERT+ : Improving Numerical Reasoning in Reading Comprehension. 2019. URL https://github.com/raylin1000/drop-bert.

K. Korrel, D. Hupkes, V. Dankers, and E. Bruni. Transcoding compositionally: using attention to find more generalizable solutions. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

K. Krishna and M. Iyyer. Generating Question-Answer Hierarchies. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, 123(1):32–73, 2017.

J. Krishnamurthy, O. Tafjord, and A. Kembhavi. Semantic Parsing to Probabilistic Programs for Situated Question Answering. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016. URL http://aclweb.org/anthology/D/D16/D16-1016.pdf.

J. Krishnamurthy, P. Dasigi, and M. Gardner. Neural Semantic Parsing with Type Constraints for Semi-Structured Tables. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

A. Kumar, O. Irsoy, P. Ondruska, M. Iyyer, J. Bradbury, I. Gulrajani, V. Zhong, R. Paulus, and R. Socher. Ask Me Anything: Dynamic Memory Networks for Natural Language Processing. In *Proc. of the International Conference on Machine Learning (ICML)*, 2016.

N. Kushman, L. Zettlemoyer, R. Barzilay, and Y. Artzi. Learning to automatically solve algebra word problems. In *Proc. of the Annual Meeting of the Association of Computational Linguistics (ACL)*, 2014.

T. Kwiatkowski, L. S. Zettlemoyer, S. Goldwater, and M. Steedman. Inducing Probabilistic CCG Grammars from Logical Form with Higher-Order Unification. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2010.

T. Kwiatkowski, L. Zettlemoyer, S. Goldwater, and M. Steedman. Lexical Generalization in CCG

Grammar Induction for Semantic Parsing. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011.

T. Kwiatkowski, E. Choi, Y. Artzi, and L. Zettlemoyer. Scaling Semantic Parsers with On-the-Fly Ontology Matching. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. P. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee, K. Toutanova, L. Jones, M. Kelcey, M.-W. Chang, A. M. Dai, J. Uszkoreit, Q. Le, and S. Petrov. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics*, 2019.

C. C. T. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the Web. In *WWW '01*, 2001.

A. Lai and J. Hockenmaier. Illinois-LH: A Denotational and Distributional Approach to Semantics. In *SemEval@COLING*, 2014.

G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy. RACE: Large-scale ReAding Comprehension Dataset From Examinations. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

B. M. Lake. Compositional generalization through meta sequence-to-sequence learning. In *NeurIPS*, 2019.

B. M. Lake and M. Baroni. Generalization without Systematicity: On the Compositional Skills of Sequence-to-Sequence Recurrent Networks. In *Proc. of the International Conference on Machine Learning (ICML)*, 2018.

M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

T. Li, V. Gupta, M. Mehta, and V. Srikumar. A Logic-Driven Framework for Consistency of Neural Models. In *EMNLP/IJCNLP*, 2019.

C. Liang, M. Norouzi, J. Berant, Q. V. Le, and N. Lao. Memory Augmented Policy Optimization for Program Synthesis and Semantic Parsing. In *NeurIPS*, 2018.

P. Liang. Lambda Dependency-Based Compositional Semantics. *arXiv*, abs/1309.4408, 2013. URL `http://arxiv.org/abs/1309.4408`.

P. Liang, M. I. Jordan, and D. Klein. Learning dependency-based compositional semantics. In *Proc. of the Annual Meeting of the Association of Computational Linguistics (ACL)*, 2011.

W. Ling, P. Blunsom, E. Grefenstette, K. Hermann, T. Kociský, F. Wang, and A. Senior. Latent

Predictor Networks for Code Generation. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016.

I. R. L. Logan, N. F. Liu, M. E. Peters, M. Gardner, and S. Singh. Barack's Wife Hillary: Using Knowledge-Graphs for Fact-Aware Language Modeling. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

J. Maillard, S. Clark, and D. Yogatama. Jointly Learning Sentence Embeddings and Syntax with Unsupervised Tree-LSTMs. *CoRR*, abs/1705.09189, 2017. URL `http://arxiv.org/abs/1705.09189`.

J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, and J. Wu. The Neuro-Symbolic Concept Learner: Interpreting Scenes Words and Sentences from Natural Supervision. In *Proc. of the International Conference on Learning Representations*, 2019.

B. McCann, N. Keskar, C. Xiong, and R. Socher. The Natural Language Decathlon: Multitask Learning as Question Answering. *ArXiv*, abs/1806.08730, 2018.

S. Miller, D. Stallard, R. J. Bobrow, and R. Schwartz. A Fully Statistical Approach to Natural Language Interfaces. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 1996.

S. Min, E. Wallace, S. Singh, M. Gardner, H. Hajishirzi, and L. Zettlemoyer. Compositional Questions Do Not Necessitate Multi-hop Reasoning. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

P. Minervini and S. Riedel. Adversarially Regularising Neural NLI Models to Integrate Logical Background Knowledge. In *CoNLL*, 2018.

R. Montague. The proper treatment of quantification in ordinary english. In P. Suppes, J. Moravcsik, and J. Hintikka, editors, *Approaches to Natural Language*. 1973.

E. Noreen. *Computer-Intensive Methods for Testing Hypotheses: An Introduction*. Wiley, 1989. ISBN 9780471611363. URL `https://books.google.com/books?id=kinvAAAAMAAJ`.

I. Oren, J. Herzig, N. Gupta, M. Gardner, and J. Berant. Improving Compositional Generalization in Semantic Parsing. In *Findings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020. URL `https://cogcomp.seas.upenn.edu/papers/OHGGB20.pdf`.

P. Pasupat and P. Liang. Compositional Semantic Parsing on Semi-Structured Tables. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *NAACL-HLT*, 2018.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21:140:1–140:67, 2020.

P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100, 000+ Questions for Machine Comprehension of Text. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2015. URL `http://dl.acm.org/citation.cfm?id=2969239.2969250`.

M. T. Ribeiro, C. Guestrin, and S. Singh. Are red roses red? evaluating consistency of question-answering models. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

M. Richardson, C. J. C. Burges, and E. Renshaw. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013*, pages 193–203, 2013.

A. S. Ross, M. C. Hughes, and F. Doshi-Velez. Right for the Right Reasons: Training Differentiable Models by Constraining their Explanations. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

S. Roy and D. Roth. Solving General Arithmetic Word Problems. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015. URL `http://cogcomp.org/papers/arithmetic.pdf`.

A. M. Rush, S. Chopra, and J. Weston. A Neural Attention Model for Sentence Summarization. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

M. Sachan and E. Xing. Learning to Solve Geometry Problems from Natural Language Demonstrations in Textbooks. In **SEM*, 2017.

M. Sachan, K. A. Dubey, E. Xing, and M. Richardson. Learning Answer-Entailing Structures for Machine Comprehension. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015.

A. Santoro, D. Raposo, D. G. T. Barrett, M. Malinowski, R. Pascanu, P. W. Battaglia, and T. P. Lillicrap. A simple neural network module for relational reasoning. In *NIPS*, 2017.

H. Seltman. Approximations for Mean and Variance of a Ratio, 2018. URL `http://www.stat.cmu.edu/~hseltman/files/ratio.pdf`.

M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional Attention Flow for Machine Comprehension. In *International Conference on Learning Representations (ICLR)*, 2017.

N. Smith and J. Eisner. Contrastive estimation: Training log-linear models on unlabeled data. In *Proc. of the Annual Meeting of the Association of Computational Linguistics (ACL)*, 2005.

J. Stacey, P. Minervini, H. Dubossarsky, S. Riedel, and T. Rocktäschel. There is Strength in Numbers: Avoiding the Hypothesis-Only Bias in Natural Language Inference via Ensemble Adversarial Training. *ArXiv*, abs/2004.07790, 2020.

M. Steedman. The syntactic process. In *Language, speech, and communication*, 2004.

S. Subramanian*, B. Bogin*, N. Gupta*, T. Wolfson, S. Singh, J. Berant, and M. Gardner. Obtaining Faithful Interpretations from Compositional Neural Networks. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020. URL `https://cogcomp.seas.upenn.edu/papers/SBGWSBG20.pdf`.

A. Suhr, M. Lewis, J. Yeh, and Y. Artzi. A Corpus of Natural Language for Visual Reasoning. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

A. Suhr, S. Zhou, A. Zhang, I. Zhang, H. Bai, and Y. Artzi. A Corpus for Reasoning about Natural Language Grounded in Photographs. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019. URL `https://www.aclweb.org/anthology/P19-1644`.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

K. S. Tai, R. Socher, and C. D. Manning. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015.

A. Talmor and J. Berant. The Web as a Knowledge-Base for Answering Complex Questions. In *Proc. of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018. URL `https://www.aclweb.org/anthology/N18-1059`.

H. Tan and M. Bansal. LXMERT: Learning Cross-Modality Encoder Representations from Transformers. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019. URL `https://www.aclweb.org/anthology/D19-1514`.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. ukasz Kaiser, and I. Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*. 2017. URL `http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf`.

D. Ventura. *CS478 Paired Permutation Test Overview*, 2007. URL `http://axon.cs.byu.edu/Dan/478/assignments/permutation_test.php`. Accessed April 29, 2020.

P. Verga, A. Neelakantan, and A. McCallum. Generalizing to Unseen Entities and Entity Pairs with Row-less Universal Schema. In *Proc. of the Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2017.

O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. E. Hinton. Grammar as a Foreign Language. In *NIPS*, 2015a.

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

E. Voorhees. The TREC-8 Question Answering Track Report. In *Proc. of the Text Retrieval Conference (TREC)*, 1999.

A. Wang, K. Cho, and M. Lewis. Asking and Answering Questions to Evaluate the Factual Consistency of Summaries. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

S. I. Wang, P. Liang, and C. Manning. Learning Language Games through Interaction. In *Association for Computational Linguistics (ACL)*, 2016.

J. Weston, A. Bordes, S. Chopra, and T. Mikolov. Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks. *arXiv: Artificial Intelligence*, 2016.

S. Wiegreffe and Y. Pinter. Attention is not not Explanation. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019. URL `https://www.aclweb.org/anthology/D19-1002`.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.

T. Winograd. Understanding natural language. *Cognitive psychology*, 1972.

T. Wolfson, M. Geva, A. Gupta, M. Gardner, Y. Goldberg, D. Deutch, and J. Berant. Break it down: A question understanding benchmark. *Transactions of the Association for Computational Linguistics (TACL)*, 2020. URL `https://arxiv.org/abs/2001.11770`.

Y.-W. Wong and R. Mooney. Learning for semantic parsing with statistical machine translation. In *Proc. of the Annual Meeting of the North American Association of Computational Linguistics (NAACL)*, 2006.

W. Woods. The lunar sciences natural language information system. *BBN report*, 1972.

A. Yeh. More accurate tests for the statistical significance of result differences. In *Proceedings of the 18th conference on Computational linguistics-Volume 2*. Association for Computational Linguistics, 2000.

P. Yin and G. Neubig. A syntactic neural model for general-purpose code generation. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

J. M. Zelle and R. J. Mooney. Learning to parse database queries using inductive logic proramming. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1996.

L. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

L. Zettlemoyer and M. Collins. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference of EMNLP-CoNLL*, 2007.

T. Zhang*, V. Kishore*, F. Wu*, K. Q. Weinberger, and Y. Artzi. BERTScore: Evaluating Text Generation with BERT. In *Proc. of the International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkeHuCVFDr.

Y. Zhang, J. Hare, and A. Prügel-Bennett. Learning to Count Objects in Natural Images for Visual Question Answering. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B12Js_yRb.

X. Zhu, P. Sobihani, and H. Guo. Long short-term memory over recursive structures. In *Proc. of the International Conference on Machine Learning (ICML)*, 2015.