# Compositional and Modular Models for Reasoning over Text
## Ph.D. Thesis Proposal

Nitish Gupta

Department of Computer and Information Science

University of Pennsylvania

`nitishg@seas.upenn.edu`

March 9, 2020

*Ph.D. Thesis Advisor:*
Prof. Dan Roth

*Ph.D. Thesis Committee:*
Prof. Mitch Marcus (Chair)
Prof. Lyle Ungar
Prof. Chris Callison-Burch
Prof. Luke Zettlemoyer

## Abstract

In the last decade, deep artificial neural network models have become ubiquitous and have shown to achieve surprisingly exceptional performance on various natural language processing tasks. Despite such successes, several studies have shown that these models fail embarrassingly on seemingly trivial problems. The black-box nature of such models makes it difficult to interpret and debug their decision making process. In this thesis, I focus on developing modular and compositional models that reason over natural language, specifically models that read and answer questions against text as context. My work focuses on models that provide an understanding of the question semantics in terms of a formal executable parse which is composed of learnable modules that can perform reasoning over open domain text. Such models are desirable for various reasons – (a) being inherently compositional in nature, such models should be better able to capture the compositional nature of language and reasoning in general, which should result in accurate models, (b) the structured parse of the question and the outputs of intermediate modules make the model's decision making process interpretable and debuggable, (c) the modular nature of the model allows for transfer of supervision and reasoning capability across various domains and tasks. Until now, we have shown how models that perform natural language and symbolic reasoning over text can be designed and trained using end-goal supervision [2, 3]. We also showed that the use of auxiliary losses and external supervision leads to better performance. While such compositional models should be interpretable, we have also shown that it is difficult to achieve interpretability when trained in an end-to-end manner [1]. We have proposed a systematic and quantitative evaluation of interpretability and introduced ways to

achieve it. Going forward, I am focussing on the following directions – (a) extending the model to handle a broader class of linguistic constructions that require a symbolic interpretation, (b) understanding and improving systematic generalization in such models, and (c) achieving transfer of reasoning capability across domains and tasks by sharing modules and supervision. The biggest challenge in pursuing this direction, and also what excites me the most, is the tension between the neatness of formal logic and fuzziness of reasoning over natural language.

# Relevant Publications

The following publications contain work related to this thesis proposal:

1. (†) Sanjay Subramanian*, Ben Bogin*, Nitish Gupta*, Tomer Wolfson, Sameer Singh, Jonathan Berant and Matt Gardner. Achieving Interpretability in Compositional Neural Networks. *In submission (ACL 2020).*

2. (†) Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh and Matt Gardner. Neural Module Networks for Reasoning over Text. In *ICLR*, April 2020.

3. (†) Nitish Gupta and Mike Lewis. Neural Compositional Denotational Semantics for Question Answering. In *EMNLP*, November 2018.

4. Shyam Upadhyay, Nitish Gupta and Dan Roth. Joint Multilingual Supervision for Cross-lingual Entity Linking. In *EMNLP*, November 2018.

5. Nitish Gupta, Sameer Singh and Dan Roth. Entity Linking via Joint Encoding of Types, Descriptions, and Context. In *EMNLP*, September 2017.

(†) works discussed in this proposal.

# Contents

# 1 Introduction

We humans are able to seamlessly reason about the world we live and answer questions about it without any trouble. Along with using visual perception to learn about how the world functions, another most common way humans gather new information and improve their understanding of the world is by asking questions and getting answers. No wonder much work in the the natural language processing community has focussed on developing question answering systems (Voorhees, 1999; Kwok et al., 2001), i.e., getting machines to answer questions posed in natural language given some relevant background knowledge. Developing question answering systems is not only important to fill human information needs as they also provide a natural setup to probe a system's understanding of language and the world in general.

While the scope of "question answering" is infinte; in theory, one can ask questions about any and everything under the sun, we focus on a setup where questions need to be answered against a passage of text that contains the relevant information. This allows for measuring an agent's capability of understanding language while making miminal assumptions about the world knowledge that is required by the system in order to find the correct answer. Developing models to answer such questions poses a variety of challenges – a system needs to be able to understand the semantics of the question, represent the world being described in the context, and should be able to perform reasoning under this representation to find the answer. Consider the question *"Which country has the highest per capita carbon dioxide emission?"*, an agent would need to understand the semantics of the question and decompose it into multiple simpler but interrelated problems, and answer the original question by composing the solution to these sub-problems. For example, an agent could perform the following operations — locate the *"countries"* mentioned in the context, for each one of them find their respective *"per capita carbon dioxide emission"*, compute the maximum value amongst these and provide as answer the country with this emission value. Solving these sub-problems would require the system to understand the concept of *"countries"* and locate its instantiations in text, tackle various linguistic variations in which *"per capita carbon dioxide emission"* of these countries might be mentioned, and perform symbolic reasoning to find the *highest* value. Similarly, the question *"What was the longest gap between two Radiohead albums?"* is underspecified and poses different challenges; amongst many, it requires the system to infer that the linguistic construction *"longest gap"* in the context of two *albums* refers to a time-span measured in years where these years are the *release dates* of *Radiohead albums*.

Previous research that aimed to solve such problems can broadly be classified into three threads, semantic parsing, machine reading comprehension, and neural module networks. Semantic parsing, rooted in formal semantics, aims to map a natural language utterance (e.g. question, instruction, etc.) to a logical meaning representation. In the context of question answering, this meaning representation is usually an *executable logical form*, that can be thought of as a program, which can be executed against some representation of the world to get the desired output. Explicit modeling of *compositionality* in the meaning representation makes semantic parsing a desirable approach to solve such problems. One major drawback of semantic parsing is that it bypasses the imporant questions of learning how to represent the world (say, a text passage), and hence its usage is limited to modalities where execution can be deterministically defined (e.g. structured databases). Over the last decade, with the

advent of large-scale neural network models for natural language processing, black-box neural models for question answering have emerged (Seo et al., 2018; Yu et al., 2018; Devlin et al., 2018). Such models exploit the expressive representational capacity of neural networks to learn a non-explicit *"meaning representation"* of language, expressed as continuous vectors, and find the answer without resorting to explicit compositional semantics. While such models have shown extremely good performance on standard benchmarks for machine reading comprehension, there have also been several studies showcasing the brittleness of these approaches (Jia and Liang, 2017; Ribeiro et al., 2018; Feng et al., 2018). Furthermore, the *black-box* nature of such models makes it difficult to interpret its decision making process. Recently proposed neural module networks (NMNs; Andreas et al., 2016) tries to combine the best of both worlds, the explicit modeling of question semantics in terms of a formal meaning representation with the power of neural networks for representation learning. Like semantic parsing, it maps the question in to a logical form, but here this logical form is composed of predicates that are not fixed functions but rather functions with learnable parameters (or modules). These modules learn to percieve the context and solve different basic understanding tasks and can be composed to perform higher-order reasoning. While extremely promising, this approach has mainly been applied to question answering in synthetic visual domains and has not been applied to reasoning against text as context.

## 1.1   Overview of current work

In this thesis, we aim to borrow ideas from these research directions and take steps towards developing models that are able to perform reasoning over text. Specifically, we would like the models we design to have the following desiderata; (a) the model should explicitly model compositionality in language and reasoning, i.e., the model structure should imitate the linguistic and reasoning structure closely, (b) the model's decision making process should be interpretable to some level that allows for understanding the model behavior and opens up possibilities for debugging, (c) the model should be modular, i.e. composed of operators that are resuable; this should allow for transfer of supervision and primitive reasoning capability across various domains and tasks.

In Chaper 1, we present a neural module network (NMN) for answering compositional questions that require multiple steps of reasoning against text as context. We introduce modules that are capable of performing both shallow reasoning over a paragraph of text and symbolic reasoning (such as arithmetic, sorting, counting) over numbers and dates in a probabilistic and differentiable manner. Our model which builds off of NMNs fulfils all desiderata as explained above but learning such a model using only weak question-answer supervision is extremely challenging. We additionally show how problem decomposition allows for using auxiliary objectives to aid learning.

While such models are inherently interpretable by the means of the question program and the intermediate outputs of program execution, we show in Chapter 2 that learning from end-goal supervision does not guarantee that the modules outputs are faithful to the logical meaning representation of the question. We find that due to the extreme expressivity of neural models, the modules do not learn their intended behavior when the only learning supervision is from the end-task. We outline few methods to alleviate this issue; providing auxiliary supervision for module outputs, designing the formal language such that expected

linguistic phenonmena have corresponding semantic predicates, and providing the correct inductive bias to the model through careful module architecture design.

For question parsing we use Seq2Seq (Sutskever et al., 2014; Bahdanau et al., 2015) models. These have become standard practice to parse natural language utterances into logical forms (Dong and Lapata, 2016; Krishnamurthy et al., 2017; Iyer et al., 2018) even though it has also been shown that these parsers can fail to generalize in surprising ways (Finegan-Dollak et al., 2018; Bahdanau et al., 2019). In Chapter 3, we present an approach for semantic parsing that resembles Montague semantics (Montague, 1973), in which a tree of interpretable expressions is built over the utterance. The nodes in the parse tree are combined using semantic-composition functions (modules) whose parameters are jointly learned with the parser. Though similar in essence to NMNs, by building a parse-tree over the input utterance, this approach imposes a linguistically motivated independence assumption where phrases are interpreted independently from their surrounding words. Such inductive bias helps the model generalize to interpreting phrases in novel contexts. This work provides an evidence in a synthetically constructed knowledge-graph domain that it is possible to jointly learn semantic-composition functions and a semantic parser using just utterance-denotation as supervision.

## 1.2   Overview of future work

Our contributions so far have focussed on developing a compositional and modular model for question answering against text that is capable of performing symbolic reasoning. The ideas presented until now can be extended and improved in various ways. For the rest of the thesis we plan to focus on these directions.

**Extending reasoning capabilities**  - The model we present in Chapter 1 has limited capability in terms of handling linguistic constructions that require a symbolic interpretation. For example, the modules we currently define cannot handle determiners such as *"most"* in *"Did Bernie Sanders win the most votes?"* or *"more than half"* in *"Who scored more than half the runs?"*. We would like to push forward in this direction and design differentiable modules such that our model is capable of handling a diverse range of operators (for example, such generalized quantifiers).

**Transfer learning**  - One key advantage of modular models is the capability to reuse modules in novel contexts, domains, and tasks. We would like to pursue this direction and and as a starting step try to develop a single question answering system that can be trained and evaluated on multiple benchmarks silmultaneously, for example, on datasets in the Open Reading Benchmark (ORB; Dua et al., 2019a).

**Systematic Generalization**  - Though it seems natural that explicitly compositional models should be able to generalize better than black-box models, several works (Bahdanau et al., 2019; Bahdanau et al., 2020) have shown that it is not necessarily true. We would like to investigate this in the context of questions we tackle and improve model generalization

on recently introduced stress test-sets (Gardner et  al., 2019) and challenging evaluation settings (Finegan-Dollak et  al., 2018).

# 2 Neural module networks for reasoning over text

Answering complex compositional questions against text is challenging since it requires a comprehensive understanding of both the question semantics and the text against which the question needs to be answered. Consider the question in Figure 1; a model needs to understand the compositional reasoning structure of the questions, perform accurate information extraction from the passage (eg. extract *lengths*, *kickers*, etc. for the *field goals* and *touchdowns*), and perform symbolic reasoning (eg. counting, sorting, etc.).

Semantic parsing techniques 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. These modules are designed to perform basic reasoning tasks and can be composed to perform complex reasoning over unstructured knowledge. 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.

In this chapter we present a neural module network for question answering against text as context. We find that 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. Jointly learning the parser and executor using only QA supervision is also extremely challenging (§2.1.2).

We introduce neural modules to perform reasoning over text using distributed representations, and perform symbolic reasoning, such as arithmetic, sorting, comparisons, and counting (§2.2). 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.

We also show that the challenges arising in learning from end-task QA supervision can be alleviated with an auxiliary loss 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 (§2.3.1). Additionally, we show that providing heuristically-obtained supervision for question programs and outputs for intermediate modules in a program (§2.3.2) for a small subset of the training data (5–10%) is sufficient for accurate learning.

We experiment on 21,800 questions from the recently proposed DROP dataset (Dua et al., 2019b) that are heuristically chosen based on their first n-gram such that they are covered by our designed modules. This is a significantly-sized subset that poses a wide variety of reasoning challenges and allows for controlled development and testing of models. We show that our model, which has interpretable intermediate outputs by design, significantly outperforms state-of-the-art black box models on this dataset. We conclude with a discussion of the challenges of pushing NMNs to the entire DROP dataset, where some questions require reasoning that is hard to design modules for.
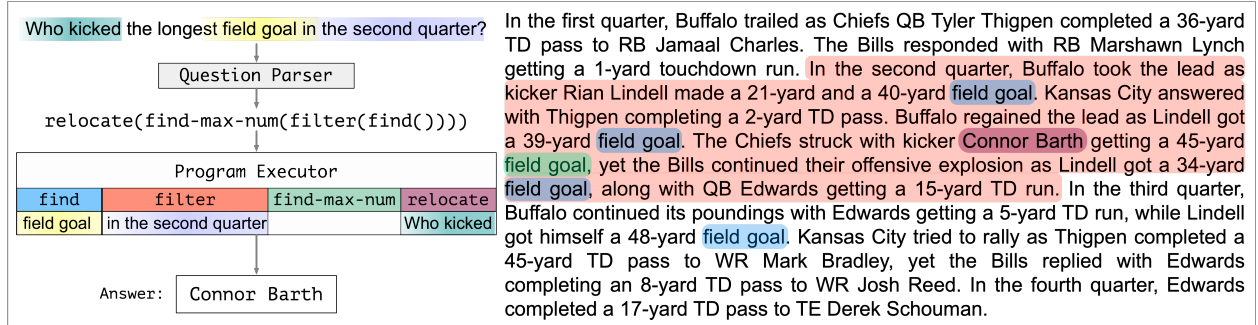
Figure 1: **Model Overview**: Given a question, our model parses it into a program composed of neural modules. This program is executed against the context to compute the final answer. The modules operate over soft attention values (on the question, passage, numbers, and dates). For example, `filter` 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*.

## 2.1 Neural Module Networks

Consider the question *"Who kicked the longest field goal in the second quarter?"* in Figure 1. 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.

Neural module networks (NMN) capture this intuition naturally, which makes them a good fit to solve reasoning problems like these. A NMN would parse such a question into an executable program, such as `relocate(find-max-num(filter(find())))`, whose execution against the given paragraph yields the correct 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 `relocate` module should find "who kicked" the *field goal* corresponding to its input span.

### 2.1.1 Components of a NMN for Text

**Modules.** To perform natural language and symbolic reasoning 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 §2.2.

**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.,

9

2018) model. Here $n$ and $m$ are the number of tokens in the question and the paragraph, respectively.

**Question Parser.** We use an encoder-decoder model with attention to map the question into an executable program. Similar to 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. The output of the decoder is a linearized abstract syntax tree (in an in-order traversal).

**Learning.** We define our model probabilistically, i.e., for any given program $\mathbf{z}$, we can compute the likelihood of the gold-answer $p(y^*|\mathbf{z})$. Combined with the likelihood of the program under the question-parser model $p(\mathbf{z}|q)$, we can maximize the marginal likelihood of the answer by enumerating all possible programs; $J = \sum_{\mathbf{z}} p(y^*|\mathbf{z})p(\mathbf{z}|q)$. Since the space of all programs is intractable, we run beam search to enumerate top-K programs and maximize the approximate marginal-likelihood.

### 2.1.2 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.

**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. Additionally, joint learning is challenging as prediction errors from one component lead to incorrect training of the other. E.g., if the parser predicts the program `relocate(find())` for the question in Fig. 1, 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.

## 2.2 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 the main contributions of our work is introducing differentiable modules that perform reasoning over text and symbols in a probabilistic manner. Table 1 gives an overview of representative modules and §2.2.2 describes them in detail.

| 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 |
| relocate | 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 §2.2.1.

### 2.2.1 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. [1]
- **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.

---

[1] We extract numbers and dates as a pre-processing step.

### 2.2.2   Neural Modules for Question Answering

The question and paragraph contextualized embeddings ($\mathbf{Q}$ and $\mathbf{P}$) 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 §2.1.1. We describe some representative modules here; refer to appendix A.1 for details about other modules.

`find(Q) → P`   This module is used to ground attended question tokens to similar tokens in the paragraph (e.g., "field goal" in Figure 1). 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 $\mathbf{A}$, weighed by the input question attention, $P = \sum_i Q_i \cdot \mathbf{A}_{i:} \in \mathbb{R}^m$. $\mathbf{A}$ 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 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 elementwise multiplication.

`find-num(P) → 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, $\mathbf{S}^{\text{num}}_{i,j} = \mathbf{P}_{i:}^T \mathbf{W}_{\text{num}} \mathbf{P}_{n_j:}$, 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. $\mathbf{A}^{\text{num}}_{i:} = \text{softmax}(\mathbf{S}^{\text{num}}_{i:})$. 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 $N = [0.5, 0.3, 0.2]$.

`count(P) → 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 $p(c) \propto \exp\left(-(c-c_v)^2/2v^2\right) \ \forall c \in [0, 9]$. Pretraining this module by generating synthetic data of attention and count values helps.

`compare-num-lt(P1, P2) → 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, $\texttt{P}_{out} = p(N_1 < N_2) * \texttt{P}_1 + p(N_2 < N_1) * \texttt{P}_2$. 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.

`find-max-num(P)` $\to$ `P`, `find-min-num(P)` $\to$ `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, $T^{\max} \in \mathbb{R}^{N_{\text{tokens}}}$ (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: $\bar{P}_i = \sum_j T_j^{\max}/T_j \cdot P_i \cdot \mathbf{A}^{\text{num}}_{ij}$.

**Computing $T^{\mathbf{max}}$:** 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 $T^{\max}$.

## 2.3   Auxiliary Supervision

As mentioned in §2.1.2, 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 (§2.3.1); and (b) provide heuristically-obtained supervision for question program and intermediate module output (§2.3.2) for a subset of questions (5–10%).

### 2.3.1   Unsupervised Auxiliary Loss for IE

The `find-num`, `find-date`, and `relocate` 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

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

We compute a similar loss for the date-attention map $\mathbf{A}^{\mathrm{date}}$ ($H_{\mathrm{loss}}^{\mathrm{d}}$) and the relocate-map $\mathbf{R}$ ($H_{\mathrm{loss}}^{\mathrm{r}}$). The final auxiliary loss is $H_{\mathrm{loss}} = H_{\mathrm{loss}}^{\mathrm{n}} + H_{\mathrm{loss}}^{\mathrm{d}} + H_{\mathrm{loss}}^{\mathrm{r}}$.

### 2.3.2   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 DROP, 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). 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.

## 2.4 Experiments

### 2.4.1 Dataset

We perform experiments on a portion of the recently released DROP dataset (Dua et al., 2019b), 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.

### 2.4.2 Results

We compare to publicly available best performing models: NAQANet (Dua et al., 2019b), NABERT+ (Kinley and Lin, 2019), TAG-NABERT+ (Efrat et al., 2019), and MTMSN (Hu et al., 2019), all trained on the same data as our model.

**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 SoTA 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.

| 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**.

| 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.
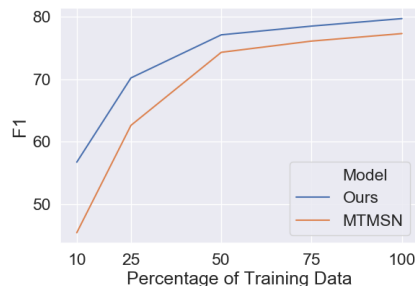


(b) **Performance with less training data**: Our model performs significantly better than the baseline with less training data, showing the efficacy of explicitly modeling compositionality.

Figure 2: Effect of auxiliary losses and the size of training data on model performance.

**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; it outperforms MTMSN on only some of the runs. 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.

**Effect of Additional Supervision.** Figure 2a 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.

## 2.5  Limitations and Future Directions

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. §2.1.2). 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. 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, mimicking BERT-QA-style models, 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.

## 2.6  Conclusion

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. Our proposed future research aims to continue to bridge these reasoning gaps.

# 3 Achieving interpretibaility in compositional models

## 3.1 Introduction

To solve problems as shown in Figure 3 from NLVR2, both models that assume an intermediate structure (Andreas et al., 2016; Jiang and Bansal, 2019) and models without such structure (Tan and Bansal, 2019; Hu et al., 2019; Min et al., 2019) have been proposed. Our Text-NMN proposed in Chapter 2 is an example of a structured model for reasoning against text. While good performance can be obtained without a structured representation, an advantage of structured approaches is that the reasoning process is more *interpretable*. In this chapter we study the interpretability of NMNs and propose ways to improve it.
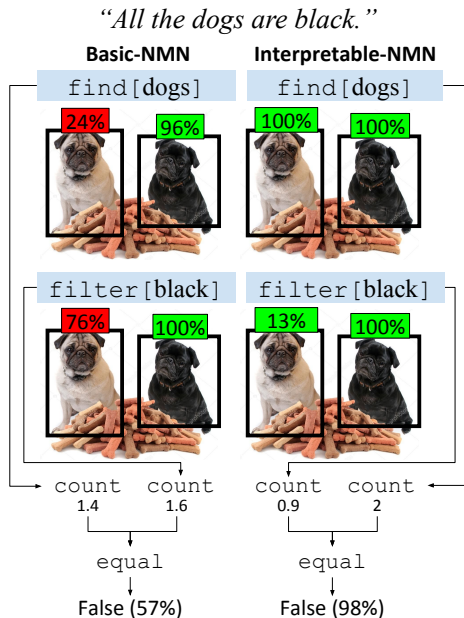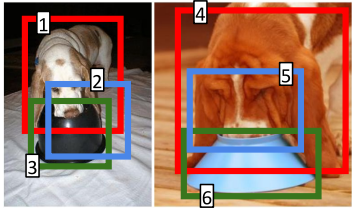


Figure 3: An example for a visual reasoning problem where both the Basic and Interpretable NMNs produce the correct answer. The Basic NMN, however, fails to give meaningful intermediate outputs for the `find` and `filter` modules, whereas our improved interpretable-NMN assigns correct probabilities in all cases. Boxes are green if probabilities are as expected, red otherwise.

In a NMN, since module parameters are typically learned from end-task supervision only, it is possible that the model will solve the task, but the modules will not perform the reasoning steps as intended. In Figure 3, a *basic NMN* predicts the correct answer `False`, but does not correctly locate one of the *dogs* in the image. This is undesirable, since the motivation for NMNs in this case is interpretability. While recent work (Hu et al., 2018; Jiang and Bansal, 2019), and even ours in Chapter 2, has shown that one can obtain good performance when using NMNs, interpretability was mostly evaluated through qualitative analysis, rather than systematically evaluating the intermediate outputs of each module.

In this chapter we provide three primary contributions regarding interpretability in NMNs. First, we propose the concept of *module-wise interpretability* – a systematic evaluation of

**Figure 4:** 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).

individual module performance that judges whether they have learned their intended operations, and define metrics to quantify this for both textual and visual reasoning (§3.2). Second, we provide strategies for improving module-wise interpretability in NMNs (§3.3). Specifically, (a) we demonstrate how module architecture affects interpretability, (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 interpretability.

Empirically, we show on both NLVR2 (Suhr et al., 2018) and DROP (Dua et al., 2019b) that training a NMN using end-goal supervision, even using *gold programs*, does *not* yield module-wise interpretability which can be improved using techniques proposed in this chapter. Figure 3 shows an example where our approach (*Interpretable-NMN*) results in much more sensible module outputs as compared to the *Basic-NMN*.

## 3.2   Module-wise Interpretability

We introduce the concept of *module-wise interpretability*, aimed at providing an evaluation of module performance in a trained NMN. Module-wise interpretability evaluates whether each module has correctly learned its intended operation by judging the correctness of the *intermediate* module outputs. For example, in Figure 4 (top), a model would be judged module-wise interpretable if the outputs of all the modules, `find`, `relocate`, and `with_relation`, are correct.

We provide gold programs when evaluating interpretability, to not conflate interpretability with parser accuracy.

**Measuring interpretability in Text-NMN**   As shown in Chapter 2, each module in Text-NMN produces a distribution over passage tokens which represents the selected spans in a soft manner. In order to measure module-wise interpretability of Text-NMN, we annotated the set of spans that should be output by each module in the gold program. Ideally, modules

like `find`, `filter`, etc., should predict high probability for tokens that appear in the gold spans and *zero* probability for other tokens. For example, the relevant spans for each module are shown in Figure 4 (bottom).

Concretely, we use a metric akin to cross-entropy loss to measure the deviation of the predicted module output $p_{\text{att}}$ from the gold spans $S = [s_1, s_2, \ldots, s_N]$. Here each span $s_i = (t^i_{\text{start}}, t^i_{\text{end}})$ is annotated as the start and end tokens. Interpretability for a module is measured by:

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

Here, lower cross-entropy corresponds to better interpretability of a module.

**Measuring interpretability in Visual-NMN**    Similarly, we define a quantitative meausure of interpretability in Visual-NMN based on the overlap between gold and predicted bounding-boxes that should be output by each module in the program. For brevity, we omit the details and refer the reader to the accompanying paper.

## 3.3   Improving Interpretability in NMNs

Module-wise interpretability 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 interpretability of NMNs across these dimensions.

### 3.3.1   Choice of modules

**Textual reasoning**    In the context of Text-NMN (on DROP), we study the effect of formal language (module choice) 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 consist of 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 complex tasks that were not intended for them, hurting

interpretability. 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.

**Visual reasoning**    The `count` module always appears in NLVR2 as one of the top-level modules (see Figures 3 and 4). 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.

We study how the architectural design of such a high-level module affects the learning of other modules in the model. We try three different architectures for the `count` module with varying expressivity and see how it affects learning. **Layer-count module** uses a linear projection from image attention, followed by a softmax. This architecture explicitly uses the visual features giving it greater expressivity compared to simpler methods. 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 any visual features and simply computes the sum of bounding box probabilities. Being parameter-less, this architecture provides direct feedback to descendant modules. However, such a simple functional-form does not ignore low-probability boxes and bounding boxes that overlap. **Graph-count module** (Zhang et al., 2018) is a middle ground between both approaches; it does not use visual features, but learns to ignore overlapping and low-confidence bounding boxes while introducing only a minimal number of parameters. Such an architecture should provide the required inductive bias for accurate learning of the task and also should propagate gradients in a manner that encourages correct learning of *descendant* modules.

### 3.3.2   Supervising module output

In Chapter 2 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 used an auxiliary objective that encourages these modules to output the "gold" numbers and dates according to the heuristic supervision. In this chapter we evaluate the effect of such supervision on the interpretability of both the supervised modules, as well as other modules that are trained jointly.

We similarly find auxiliary supervision for modules in Visual-NMN and pretrain modules using that. We show that such pre-training helps learning of modules. Details are ommited for brevity.

### 3.3.3   Decontextualized word representations

We show the effect of decontextualized question token representations only for Visual-NMN and omit details here. Please refer the accompanying paper.

## 3.4 Experiments

We demonstrate that training NMNs using end-task supervision only does not yield module-wise interpretability both for visual and textual reasoning. We then show that the methods proposed in this chapter are crucial for achieving interpretability and how different design choices affect it.

**Experimental setup**   We use the Text-NMN introduced in Chapter 2 which is evaluated on the complete development set of DROP which does not contain any program supervision. Module-wise interpretability is measured on 100 manually-labeled questions from the development set, which are annotated with gold programs and the passage spans. We perform similar evaluation for Visual-NMN that we skip from this document for brevity.

**Interpretability evaluation**   As seen in Table 3, when trained on DROP using question-program supervision, the model achieves an $F_1$ score of 62.7 and an interpretability score of 6.8. When adding supervision for intermediate modules, we find that the module-wise interpretability score improves to 5.1. Similar to Visual-NMN, this shows that supervising intermediate modules in a program leads to better interpretability.

To analyze how choice of modules affects interpretability, we train without sorting and comparison modules (`find-max-num`, `num-compare`, etc.). We find that while performance improves slightly, interpretability deteriorates significantly to 8.4, showing that modules that perform atomic reasoning are crucial for interpretability. When trained without program supervision, removing `extract-answer` improves interpretability ($10.4 \rightarrow 9.0$) but at the cost of model performance ($63.4 \rightarrow 60.8$ $F_1$). This shows that such a black-box module encourages performing compositional reasoning in a non-interpretable manner, but can improve performance by overcoming the limitations of pre-defined modules.

| Model | Performance ($F_1$ Score) | Overall Interp. (cross-entropy* ↓) | Module-wise Interpretability* (↓) | | | | |
|---|---|---|---|---|---|---|---|
| | | | find | filter | relocate | min-max† | find-arg† |
| Text-NMN w/o prog-sup | | | | | | | |
|   w/ `extract-answer` | 63.4 | 10.4 | 14.1 | 11.8 | 3.0 | 4.2 | 11.6 |
|   w/o `extract-answer` | 60.8 | **9.0** | 11.3 | 10.9 | **0.9** | 2.7 | 10.5 |
| Text-NMN w/ prog-sup | | | | | | | |
|   no auxiliary sup | 62.7 | 6.8 | 7.7 | **5.8** | **0.9** | **1.7** | 8.7 |
|   w/o sorting & comparison | **63.8** | 8.4 | 9.8 | 7.4 | 1.0 | 2.2 | 10.7 |
|   w/ module-output-sup | **63.8** | **5.1** | **6.1** | 6.1 | **0.9** | 2.0 | **6.9** |

Table 3: Interpretability and performance scores for various NMNs on DROP. *lower is better. †min-max is average interpretability of `find-min-num` and `find-max-num`; find-arg of `find-num` and `find-date`.

## 3.5 Conclusion

In this chapter we introduce the concept of *module-wise interpretability*; a systematic evaluation of interpretability in neural module networks (NMNs) for visual and textual reasoning. We show that naïve training of NMNs does not produce interpretable modules and propose

several techniques to improve module-wise interpretability in NMNs. We show how our approach leads to much higher module-wise interpretability at a low cost to performance. As future work we propose to study how such module-wise interpretability in NMNs affects their systematic generalization.

# 4 Neural Compositional Denotational Semantics for Question Answering

## 4.1 Introduction

In our model in chapter 2 and 3 the parser uses Seq2Seq (w/ attention) encoder-decoder architecture to encode the question as a dense vector and then decode a program (in a grammar-constrained manner) using this vector with attention to input tokens. Since the program-decoding is not tied to the input utterance explicitly, as opposed to parsers used before (Zettlemoyer and Collins, 2005; Artzi and Zettlemoyer, 2013; Pasupat and Liang, 2015), such an architecture does not guarantee generalization to novel combinations of sub-utterances. Additionally, our model's symbolic reasoning capability is limited to operations that are pre-defined in the module-set of the model. This does not allow for automatic discovery of new discrete operations based on question-denotation supervision. In this chapter we present a model that pushes in both these directions.

Inspired by linguistic theories of compositional semantics, we propose a parser that builds a latent tree of interpretable expressions over a sentence, recursively combining constituents using a small set of neural modules. Our approach resembles Montague semantics (Montague, 1973), 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 sentence subsuming all possible trees. Each node in the chart represents a span of text with a distribution over groundings (in terms of booleans and knowledge base 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.

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. In our model, *large or green* will be represented as a particular set of entities in a knowledge graph, and be intersected with the set of entities represented by the *cubes* node. We show that this inductive bias allows our model to generalize well when tested on more complex sentences than it was trained on. We also show that our model learns a variety of challenging semantic operators, such as quantifiers, disjunctions and composed relations, without being explictly defined to do so.

## 4.2 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

Figure 5: 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 $\phi$ for semantically vacuous nodes. While we show only one parse tree here, our model builds a parse chart subsuming all trees.

the complete question sentence that answers it. For example, in Figure 5, 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 (§ 4.3.1).

**(b)** Combining representations for adjacent phrases into representations for larger phrases, using trainable neural composition modules (§ 4.3.2). 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 (§ 4.4).

26

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

## 4.3  Compositional Semantics

### 4.3.1  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 5, *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. *We omit details for brevity and refer the reader to the paper (Gupta and Lewis, 2018) for details.*

**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.*

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.

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?.*

**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 §4.3.2.

2. **Vacuous ($\phi$)**: 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.

### 4.3.2   Composition Modules

Next, we describe how we compose phrase representations (from § 4.3.1) 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 6. *For some reason, this figure appears at the end of the proposal document; I will fix it.*

**Composition with $\phi$-typed nodes:** Phrases with type $\phi$ 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 (§ 4.3.1). These modules are useful when words act as binary functions; here they combine with their first argument. For example, in Fig. 5, *or* and *not cylindrical* combine to make a phrase containing both the vectors for *or* and the entity set for *not cylindrical*.

## 4.4   Parsing Model

Here, we describe how our model classifies question tokens into semantic type spans and computes their representations (§ 4.4.1), and recursively uses the composition modules defined above to parse the question into a soft latent tree that provides the answer (§ 4.4.2). The model is trained end-to-end using only question-answer supervision.

### 4.4.1 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 $\mathbf{E}$, $\mathbf{R}$, $\mathbf{V}$, and $\boldsymbol{\phi}$ types. For example, the token *cylindrical* in the question in Fig. 5 is assigned a distribution over the 4 semantic types (one shown) and for the $\mathbf{E}$ 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 assign their denotation in a fairly simple manner. $\mathbf{E}$-Type representation is assigned by locally normalizing the dot-product score between a token's word embedding and the entity vector, for each entity in the KG. $\mathbf{R}$-Type representation is an expected adjacency matrix computed under the token-relation affinity distribution. This distribution is obtained by normalizing the dot-product score between the token's and relation's embedding. $\mathbf{V}$-Type representation are four additional word vectors that are learned for each word in the vocabulary. $\boldsymbol{\phi}$-type is vacuous and does not require a representation. Please refer to the accompanying paper (Gupta and Lewis, 2018) for details.

### 4.4.2 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. We omit details for brevity; please refer to the accompanying paper (Gupta and Lewis, 2018) for details.

**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 $t$, $[\![w_{1..|q|}]\!]^t_{KG}$.

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

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

$$t^* = \operatorname*{argmax}_{t \in \mathbf{T}, \mathbf{E}} \; p(t|q) \tag{2}$$

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

## 4.5 Experiments

**Dataset** 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.[2] 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).

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.

We also experiment with Referring Expression Generation (GenX) dataset (FitzGerald et al., 2013) and refer the reader to our paper to read about experiments related to it.

**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. **(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. 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.

**Short Questions Performance:** Table 4 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.

**Complex Questions Performance:** Table 5 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 4, which were trained on short questions. All baselines achieve

---

[2]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.

| 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 4: **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.

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.

| 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 5: **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. Analysis suggests that most errors from our model are due to assigning incorrect structures, rather than mistakes by the composition modules.

## 4.6 Conclusion

We have introduced a model for answering questions requiring compositional reasoning 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. The indepdence assumptions in the paper presented here are too strict for general usage. As future work we propose to minimally relax these assumptions and extend the parser that would be applicable to a wider variety of questions.

# 5 Proposed Future Work

Our contributions so far have focussed on developing a compositional and modular model for question answering against text that is capable of performing symbolic reasoning. The ideas presented until now can be extended and improved in innumerable ways. We focus on the following directions for the rest of the thesis.

## 5.1 Background on Question Decomposition Meaning Represenation (QDMR)

Much of our planned future work is based off of the recently proposed resource on Question Decomposition Meaning Represenation (QDMR; Wolfson et al., 2020) which we explain in this subsection. This resoures aims at defining a formalism, QDMR, for representing the meaning of questions that is agnostic to any context and is based on *question decomposition*. QDMR (Figure 7) represents complex questions as a sequence of simpler questions that can be executed in that order to answer the original question. These atomic questions can be thought of as operations performed by a single predicate in a formal semantic parse or a database query (e.g. in SQL or SPARQL). Unlike semantic parsing though, atomic questions in QDMR are expressed in natural language with the introduction of only a small set of *formal operators*. Such usage of natural language in the meaning representation allows for representing a much diverse range of questions as compared to lossy logical meaning representation. As expected, the choice of using high-entropy natural language as the basis for meaning representation has its drawbacks.

## 5.2 Extending reasoning capabilities

The reasoning capability of the model we present in Chapter 1 is limited by the kind of modules we define; it cannot answer questions that require semantic operations for which modules do not exist.

**Problem** Our model lacks in modeling various linguistic constructions that would require a symbolic treatment. For example, our model cannot represent determiners such as *"most"* in *"Which player scored the most touchdowns?"* or *"more than half"* in *"Who scored more than half the field goals?"*. As one extention to our model, we would like to push forward in the direction of trying to design differentiable modules that are capable of handling such generalized quantifiers (Mostowski, 1957).

**Potential solution** Modeling determiners such as *"most"*, *"fewest"*, *"more than 20"* etc. in a differentiable manner is challenging. It would require us to represent, in a probabilistic manner, the context as sets of elements and properties associated with the elements of the sets, i.e., some kind of a key-value representation. For example, it would require to represent the unique players who scored touchdown passes (keys) and how many each did (values) in a probabilistic manner. Such a representation would in turn require one to identify atomic semantic units (e.g. event mentions, entitiy mentions, etc.) from the context which would

be needed to identify keys and values. This is hard to achieve when working with natural language text without any strict and lossy definition of events, entities, etc. and withot supervision of such kind. Additionally, it would also require identifying equivalence between these semantic units (mentions of entities and events) to cluster mentions of the same concept (for example, to identify unique keys). The model presented in Chapter 1 made a design choice of having tokens as the basic units of representation which restricts trying to model such quantifiers. As the next step we are working on having representing module inputs and outputs as logical-sets of text-spans and hope to be able to model more challenging quantifiers with such a represenation.

Since learning a NMN using just question-answer supervision is challenging (Chapter 1), we decide to use QDMR annotations for the DROP dataset as gold question-program supervision which allows to focus on improving program execution and modeling such challenging semantic operators.

## 5.3 Transfer learning

To achieve machine reading in the true sense, we need to develop models in a way such that a single trained model can be applied to answer a diverse set of questions from various domains. Over the last few years, many reading comprehension datasets have been proposed in various domains (MRQA (Fisch et al., 2019) provides a representative list) that test for a variety of phenonmenon ranging from simple paraphrase-matching and entity typing to entity tracking and implicative reasoning. Even then the community largely hasn't focussed on developing a single model that could read and answer questions from various domains, say all the reading comprehension datasets developed until now. Note that, recently MultiQA (Talmor and Berant, 2019) and submissions to MRQA (Fisch et al., 2019) did push in this direction but all models were based on fine-tuning a large pretrained language model on multiple datasets.

One key advantage of modular models is the capability to reuse modules in novel contexts, domains, and tasks. We would like to pursue this direction and and as a starting step try to develop a single question answering system that can be trained and evaluated on multiple benchmarks, for example, datasets in Open Reading Benchmark (ORB; Dua et al., 2019a) silmultaneously. Decomposing the question into multiple atomic questions (via structured programs) would also allow us to study what kind of reasoning primitives can be easily transfered to other domains and which cannot.

## 5.4 Systematic Generalization

There is a growing realization in the NLP community that the traditional supverised learning paradigm, where i.i.d. train/test splits are used to measure model generalization, is not the ideal setup to test NLP models, espcially given the growing size of the models and training datasets that are used. It has been shown in various studies that models find shortcuts and exploit dataset artifacts to achieve high test-set performance but infact fail in surprising ways on seemingly simple inputs. Instead, we would like our models to systematically generalize, i.e., actually understand the phenonmena they are trained for and be robust to distributional shifts as long as it results in test instances that are close enough to the training instances.

Though it seems natural that explicitly compositional models should be able to generalize better than black-box models, several works (Bahdanau et al., 2019; Bahdanau et al., 2020) have shown that it is not necessarily true. In Chapter 2 we saw that training such compositional models using only end-task supervision does not guarantee that the modules learn to behave in a faithful manner. Such ill-behaved modules would hamper model's generalization when used in unique configurations. We would like to investigate and improve systematic generalization of our model using the recently introduced stress test-sets (Gardner et al., 2019). Similarly, we are also studying systematic generalization of semantic parsers (extending work from Finegan-Dollak et al., 2018) and planning to design parsing models that are mix of Seq2Seq parsers with parsers that build a logical form over the input (Zettlemoyer and Collins, 2005; Pasupat and Liang, 2015; Gupta and Lewis, 2018).

# References

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2016). Neural module networks. In *CVPR*.

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

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*.

Bahdanau, D., de Vries, H., O'Donnell, T. J., Murty, S., Beaudoin, P., Bengio, Y., and Courville, A. (2020). CLOSURE: Assessing Systematic Generalization of CLEVR Models. In *ICLR*.

Bahdanau, D., Murty, S., Noukhovitch, M., Nguyen, T. H., de Vries, H., and Courville, A. C. (2019). Systematic Generalization: What Is Required and Can It Be Learned? In *ICLR*.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *ICML*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dong, L. and Lapata, M. (2016). Language to Logical Form with Neural Attention. In *ACL*.

Dua, D., Gottumukkala, A., Talmor, A., Singh, S., and Gardner, M. (2019a). Orb: An open reading benchmark for comprehensive evaluation of machine reading comprehension. *ArXiv*, abs/1912.12598.

Dua, D., Wang, Y., Dasigi, P., Stanovsky, G., Singh, S., and Gardner, M. (2019b). Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *NAACL-HLT*.

Efrat, A., Segal, E., and Shoham, M. (2019). Tag-based multi-span extraction in reading comprehension.

Feng, S., Wallace, E., Grissom, A., Iyyer, M., Rodriguez, P., and Boyd-Graber, J. L. (2018). Pathologies of neural models make interpretation difficult. In *EMNLP*.

Finegan-Dollak, C., Kummerfeld, J. K., Zhang, L., Ramanathan, K., Sadasivam, S., Zhang, R., and Radev, D. (2018). Improving text-to-sql evaluation methodology. In *ACL*.

Fisch, A., Talmor, A., Jia, R., Seo, M., Choi, E., and Chen, D. (2019). MRQA 2019 shared task: Evaluating generalization in reading comprehension. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*.

FitzGerald, N., Artzi, Y., and Zettlemoyer, L. S. (2013). Learning distributions over logical forms for referring expression generation. In *EMNLP*.

Gardner, M. et al. (2019). Evaluating NLP Models via Contrast Sets. *In submission*.

Gupta, N. and Lewis, M. (2018). Neural compositional denotational semantics for question answering. In *EMNLP*.

Hu, M., Peng, Y., Huang, Z., and Li, D. (2019). A multi-type multi-span network for reading comprehension that requires discrete reasoning. In *EMNLP*.

Hu, R., Andreas, J., Darrell, T., and Saenko, K. (2018). Explainable neural computation via stack neural module networks. In *ECCV*.

Hu, R., Andreas, J., Rohrbach, M., Darrell, T., and Saenko, K. (2017). Learning to reason: End-to-end module networks for visual question answering. In *ICCV*.

Iyer, S., Konstas, I., Cheung, A., and Zettlemoyer, L. (2018). Mapping language to code in programmatic context. In *EMNLP*.

Jia, R. and Liang, P. (2017). Adversarial examples for evaluating reading comprehension systems. In *EMNLP*.

Jiang, Y. and Bansal, M. (2019). Self-Assembling Modular Networks for Interpretable Multi-Hop Reasoning. In *EMNLP*.

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., and Girshick, R. B. (2017). CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. *CVPR*.

Kinley, J. and Lin, R. (2019). Nabert+ : Improving numerical reasoning in reading comprehension.

Krishnamurthy, J., Dasigi, P., and Gardner, M. (2017). Neural semantic parsing with type constraints for semi-structured tables. In *EMNLP*.

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

Liang, P. S., Jordan, M. I., and Klein, D. (2011). Learning dependency-based compositional semantics. *Computational Linguistics*.

Maillard, J., Clark, S., and Yogatama, D. (2017). Jointly Learning Sentence Embeddings and Syntax with Unsupervised Tree-LSTMs. *Natural Language Engineering*.

Min, S., Wallace, E., Singh, S., Gardner, M., Hajishirzi, H., and Zettlemoyer, L. (2019). Compositional questions do not necessitate multi-hop reasoning. In *ACL*.

Montague, R. (1973). The Proper Treatment of Quantification in Ordinary English.

Mostowski, A. (1957). On a generalization of quantifiers.

Pasupat, P. and Liang, P. (2015). Compositional semantic parsing on semi-structured tables. In *ACL*.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2018). Semantically equivalent adversarial rules for debugging nlp models. In *ACL*.

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

Seo, M., Kembhavi, A., Farhadi, A., and Hajishirzi, H. (2018). Bidirectional attention flow for machine comprehension. In *ICLR*.

Suhr, A., Zhou, S., Zhang, I., Bai, H., and Artzi, Y. (2018). A Corpus for Reasoning About Natural Language Grounded in Photographs. In *ACL*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *NIPS*.

Tai, K. S., Socher, R., and Manning, C. D. (2015). Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *ACL*.

Talmor, A. and Berant, J. (2019). Multiqa: An empirical investigation of generalization and transfer in reading comprehension. In *ACL*.

Tan, H. H. and Bansal, M. (2019). Lxmert: Learning cross-modality encoder representations from transformers. In *EMNLP/IJCNLP*.

Voorhees, E. M. (1999). The trec-8 question answering track report. In *TREC*.

Wolfson, T., Geva, M., Gupta, A., Gardner, M., Goldberg, Y., Deutch, D., and Berant, J. (2020). Break it down: A question understanding benchmark. *Transactions of the Association for Computational Linguistics*.

Yu, A. W., Dohan, D., Luong, M.-T., Zhao, R., Chen, K., Norouzi, M., and Le, Q. V. (2018). Qanet: Combining local convolution with global self-attention for reading comprehension. In *ICLR*.

Zelle, J. M. and Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*.

Zettlemoyer, L. and Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *UAI*.

Zhang, Y., Hare, J., and Prügel-Bennett, A. (2018). Learning to Count Objects in Natural Images for Visual Question Answering. In *International Conference on Learning Representations*.

Zhu, X., Sobihani, P., and Guo, H. (2015). Long short-term memory over recursive structures. In *ICML*.

# A  Appendix

## A.1  Modules for question answering

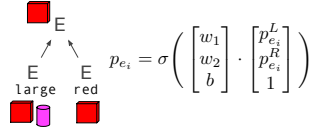Here we describe other modules in our model not described in §2.2.2

`filter(Q, P) → 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. 1). 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 $\sigma$ is the *sigmoid* non-linearity function. The output is a normalized masked input paragraph attention, $P_{\text{filtered}} = \text{normalize}(M \circ P)$.

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

`time-diff(P1, P2) → 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 $p(t_d) = \sum_{i,j} \mathbb{1}_{(d_i - d_j = t_d)} D_1^i D_2^j$.
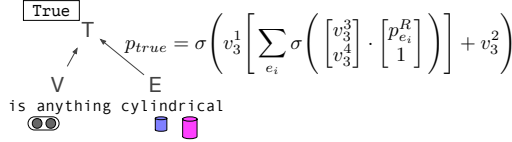
`span(P) → S`    This module is used to convert a paragraph attention into a contiguous answer span and only appears as the outermost module in a program. The module outputs two probability distributions, $P_s$ and $P_e \in \mathbb{R}^m$, denoting the probability of a token being the start and end of a span, respectively. This module is implemented similar to the `count` module.

$$p_{e_i} = \sigma\left(\begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} \cdot \begin{bmatrix} p_{e_i}^L \\ p_{e_i}^R \\ 1 \end{bmatrix}\right)$$

$\mathbf{E} + \mathbf{E} \rightarrow \mathbf{E}$: 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.



$$p_{e_i} = \sigma\left(\mathbf{v_2} \cdot \begin{bmatrix} p_{e_i}^L \\ p_{e_i}^R \\ 1 \end{bmatrix}\right)$$
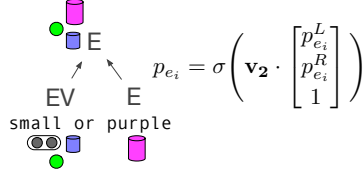
$\mathbf{EV} + \mathbf{E} \rightarrow \mathbf{E}$: This module combines two soft entity sets into a third set, parameterized by the $v_2$ word vector. This composition function is similar to a linear threshold unit and is capable of modeling various mathematical operations such as logical conjunctions, disjunctions, differences etc. for different values of $v_2$. For example, the word *or* learns to model set union.



$$p_{true} = \sigma\left(v_3^1\left[\sum_{e_i}\sigma\left(\begin{bmatrix} v_3^3 \\ v_3^4 \end{bmatrix} \cdot \begin{bmatrix} p_{e_i}^R \\ 1 \end{bmatrix}\right)\right] + v_3^2\right)$$

$\mathbf{V} + \mathbf{E} \rightarrow \mathbf{T}$: 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.
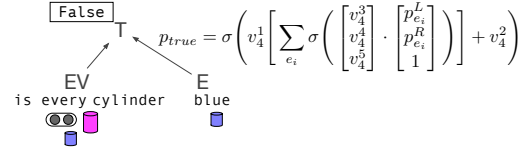


$$p_{true} = \sigma\left(\mathbf{v_2} \cdot \begin{bmatrix} p_{true}^L \\ p_{true}^R \\ 1 \end{bmatrix}\right)$$
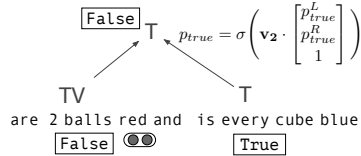
$\mathbf{TV} + \mathbf{T} \rightarrow \mathbf{T}$: This module maps a pair of soft booleans into a soft boolean using the $v_2$ word vector to parameterize the composition function. Similar to $\mathbf{EV} + \mathbf{E} \rightarrow \mathbf{E}$, this module facilitates modeling a range of boolean set operations. 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.



$$p_{e_i} = \sigma\left(\mathbf{v_1} \cdot \begin{bmatrix} p_{e_i}^R \\ 1 \end{bmatrix}\right)$$
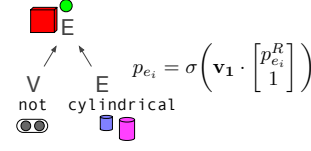
$\mathbf{V} + \mathbf{E} \rightarrow \mathbf{E}$: This module performs a function on a soft entity set, parameterized by a word vector, to produce a new soft entity set. For example, the word *not* learns to take the complement of a set of entities. The entity attention representation of the resulting span is computed by using the indicated function that takes the $v_1 \in \mathbb{R}^2$ vector of the $\mathbf{V}$ constituent as a parameter argument and the entity attention vector of the $\mathbf{E}$ constituent as a function argument.



$$p_{e_i} = \max_{e_j} A_{ji} \cdot p_{e_j}^R$$
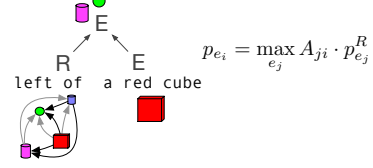
$\mathbf{R} + \mathbf{E} \rightarrow \mathbf{E}$: This module composes a set of relations (represented as a single soft adjacency matrix) and a soft entity set to produce an output soft entity set. The composition function uses the adjacency matrix representation of the $\mathbf{R}$-span and the soft entity set representation of the $\mathbf{E}$-span.



$$p_{true} = \sigma\left(v_4^1\left[\sum_{e_i}\sigma\left(\begin{bmatrix} v_4^3 \\ v_4^4 \\ v_4^5 \end{bmatrix} \cdot \begin{bmatrix} p_{e_i}^L \\ p_{e_i}^R \\ 1 \end{bmatrix}\right)\right] + v_4^2\right)$$

$\mathbf{EV} + \mathbf{E} \rightarrow \mathbf{T}$: 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 ($p_{e_i}^L \approx 1$) but not in the set of blue things ($p_{e_i}^R \approx 0$), 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.



$$A_{ij} = \sigma\left(\mathbf{v_2} \cdot \begin{bmatrix} A_{ij}^L \\ A_{ij}^R \\ 1 \end{bmatrix}\right)$$
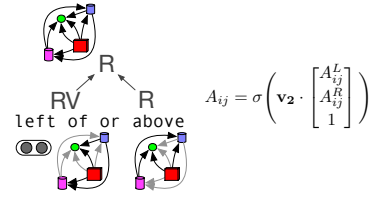
$\mathbf{RV} + \mathbf{R} \rightarrow \mathbf{R}$: This module composes a pair of soft set of relations to a produce an output soft set of relations. For example, the relations *left* and *above* are composed by the word *or* to produce a set of relations such that entities $e_i$ and $e_j$ are related if either of the two relations exists between them. The functional form for this composition is similar to $\mathbf{EV} + \mathbf{E} \rightarrow \mathbf{E}$ and $\mathbf{TV} + \mathbf{T} \rightarrow \mathbf{T}$ modules.

Figure 6: Composition Modules that compose two constituent span representations into the representation for the combined larger span, using the indicated equations.

1. Return papers
2. Return keywords of #1
3. Return the number of #1 for each #2
**4. Return #2 where #3 is highest**

(a) QDMR of *"What is the keyword, which has been contained by the most number of papers?"*

1. **Return touchdowns**
2. Return the yards of #1
3. Return the highest of #2

(b) QDMR of *"How many yards was the longest touchdown of the game?"*

Figure 7: Examples of Question Decomposition Meaning Representation (QDMR)