Neural Compositional Denotational Semantics for Question Answering

Nitish Gupta*
University of Pennsylvania
Philadelphia, PA
nitishg@cis.upenn.edu

Mike Lewis
Facebook AI Research
Seattle, WA
mikelewis@fb.com

Abstract

Answering compositional questions requiring multi-step reasoning is challenging. We introduce an end-to-end differentiable model for interpreting questions about a knowledge graph (KG), which is inspired by formal approaches to semantics. Each span of text is represented by a denotation in a KG, together with a vector that captures ungrounded aspects of meaning. Learned composition modules recursively combine constituents, culminating in a grounding for the complete sentence which answers the question. For example, to interpret "*not green*", the model represents "*green*" as a set of KG entities and "*not*" as a trainable ungrounded vector—and then uses this vector to parameterize a composition function to performs a complement operation. For each sentence, we build a parse chart subsuming all possible parses, allowing the model to jointly learn both the composition operators and output structure by gradient descent from end-task supervision. The model learns a variety of challenging semantic operators, such as quantifiers, disjunctions and composed relations, and infers latent syntactic structure. The model also generalizes well to longer sentences than seen in its training data, in contrast to LSTM, semantic parsing, and RelNet baselines.

1 Introduction

Compositionality is a mechanism by which the meanings of complex expressions are systematically determined from the meanings of their parts, and has been widely assumed in the study of both artificial and natural languages (Montague, 1973) as a means for allowing speakers to generalize to understanding an infinite number of sentences. Popular neural network approaches to question answering use a restricted form of compositionality, typically encoding a sentence word-by-word, and then executing the complete sentence encoding against a knowledge source (Perez et al., 2017). Such models can fail to generalize from training data in surprising ways. Inspired by linguistic theories of compositional semantics, we instead build a latent tree of interpretable expressions over a sentence, recursively combining constituents using a small set of neural modules. Our model outperforms RNN encoders, particularly when test questions are longer than training questions.

Our approach resembles Montague semantics, in which a tree of interpretable expressions is built over the sentence, with nodes combined by a small set of composition functions. However, both the structure of the sentence and the composition functions are learned by end-to-end gradient descent. To achieve this, we define the parametric form of small set of composition modules, and then build a parse chart over each 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 (similarly to Maillard et al. (2017)). The trees induced by our model are linguistically plausible, in contrast to prior work on structure learning from semantic objectives (Williams et al., 2017).

Typical neural approaches to grounded question answering first encode a question with a recurrent neural network (RNN), and then evaluate the encoding against an encoding of the knowledge source (for example, a knowledge graph or image) (Santoro et al., 2017). In contrast to classical approaches to compositionality, constituents of complex expressions are not given explicit interpretations in isolation. For example, in *Which cubes are large or green?*, an RNN encoder will not explic-

*Work done while interning with Facebook AI Research.

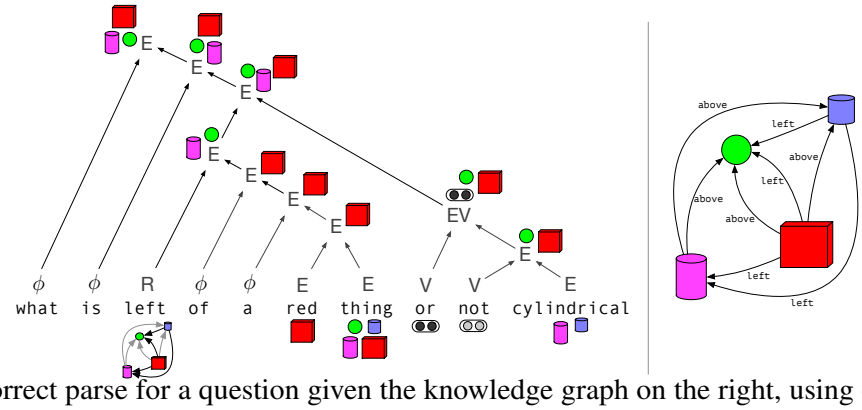


Figure 1: A correct parse for a question given the knowledge graph on the right, using our model. We show the type for each node, and its denotation in terms of the knowledge graph. The words *or* and *not* are represented by vectors, which parameterize composition modules. The denotation for the complete question represents the answer to the question. Nodes here have types E for sets of entities, R for relations, V for ungrounded vectors, EV for a combination of entities and a vector, and ϕ for semantically vacuous nodes. While we show only one parse tree here, our model builds a parse chart subsuming all trees.

itly build an interpretation for the phrase *large or green*. We show that such approaches can generalize poorly when tested on more complex sentences than they were trained on. Our approach instead 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 the previous example, *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.

Another perspective on our work is as a method for learning layouts of Neural Module Networks (NMNs) (Andreas et al., 2016b). Work on NMNs has focused on construction of the structure of the network, variously using rules, parsers and reinforcement learning (Andreas et al., 2016a; Hu et al., 2017). Our end-to-end differentiable model jointly learns structures and modules by gradient descent.

Our model is a new combination of classical and neural methods, which maintains the interpretability and generalization behaviour of semantic parsing, while being end-to-end differentiable.

2 Model Overview

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

Our model builds a parse for the sentence, in which phrases are grounded in the KG, and a small set of composition modules are used to combine phrases, resulting in a grounding for the complete question sentence that answers it. For example, in Figure 1, the phrases *not* and *cylindrical* are interpreted as a function word and an entity set, 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. We describe a model that 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 (§ 3.1).

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

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

3 Compositional Semantics

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 1, *red* corresponds to a set of entities, *left* corresponds to a set of relations, and *not* is treated as an ungrounded vector.

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

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

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

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

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

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

1. **Vector** (**V**): This type is used for spans representing functions that cannot yet be grounded in the KG (e.g. words such as *and* or *every*). These spans are represented using 4 different real-valued vectors v_1-$v_4 \in \mathbb{R}^2$-$\mathbb{R}^5$, that are used to parameterize the composition modules described in §3.2.

2. **Vacuous** ($\boldsymbol{\phi}$): 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 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.

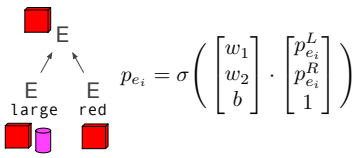
3.2 Composition Modules

Next, we describe how we compose phrase representations (from § 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:

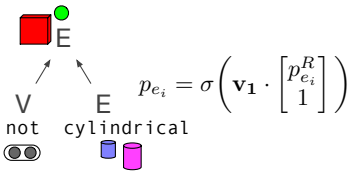
Composition modules resulting in fully grounded denotations: Described in Figure 2.

Composition with $\boldsymbol{\phi}$-typed nodes: Phrases with type $\boldsymbol{\phi}$ are treated as being semantically transparent identity functions. Phrases of any other type can combined with these with no change to their type or representation.

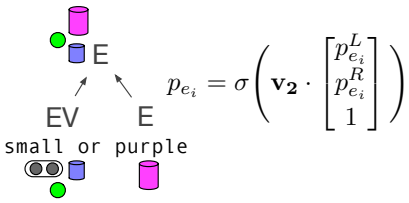
Composition modules resulting in partially grounded denotations: We define several modules that combine fully grounded phrases with ungrounded phrases, by deterministically taking the union of the representations, giving phrases with partially grounded representations (§ 3.1). These modules are useful when words act as binary functions; here they combine with their first argument. For example, in Fig. 1, *or* and *not cylindrical* combine to make a phrase containing both the vectors for *or* and the entity set for *not cylindrical*.



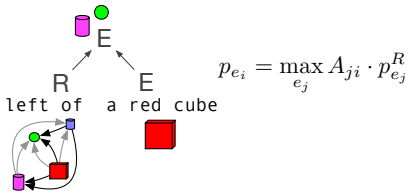
E + E → E: This module performs a function on a pair of soft entity sets, parameterized by the model's global parameter vector $[w_1, w_2, b]$ to produce a new soft entity set. The composition function for a single entity's resulting attention value is shown. Such a composition module can be used to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



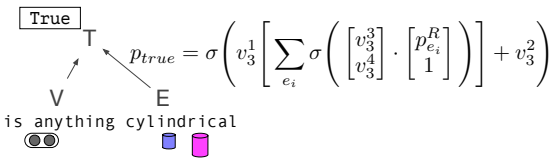
V + E → E: This module performs a function on a soft entity set, parameterized by a word vector, to produce a new soft entity set. For example, the word *not* learns to take the complement of a set of entities. The entity attention representation of the resulting span is computed by using the indicated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



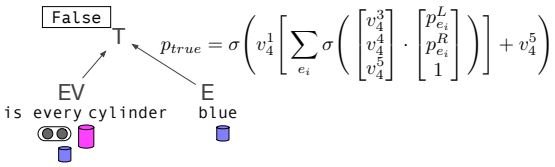
EV + E → E: This module combines two soft entity sets into a third set, parameterized by the v_2 word vector. This composition function is similar to a linear threshold unit and is capable of modeling various mathematical operations such as logical conjunctions, disjunctions, differences etc. for different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



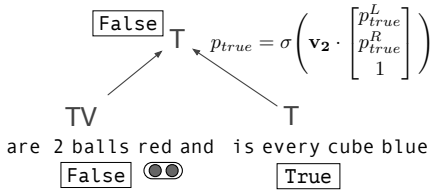
R + E → E: This module composes a set of relations (represented as a single soft adjacency matrix) and a soft entity set to produce an output soft entity set. The composition function uses the adjacency matrix representation of the **R**-span and the soft entity set representation of the **E**-span.



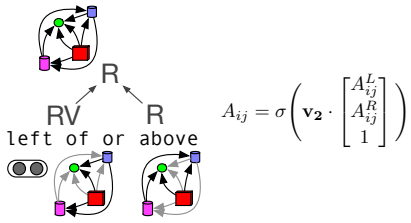
V + E → T: This module maps a soft entity set onto a soft boolean, parameterized by word vector (v_3). The module counts whether a sufficient number of elements are in (or out) of the set. For example, the word *any* should test if a set is non-empty.



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



TV + T → T: This module maps a pair of soft booleans into a soft boolean using the v_2 word vector to parameterize the composition function. Similar to **EV + E → 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.



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

Figure 2: Composition Modules that compose two constituent span representations into the representation for the combined larger span, using the indicated equations.

4 Parsing Model

Here, we describe how our model classifies question tokens into semantic type spans and computes their representations (§ 4.1), and recursively uses the composition modules defined above to parse the question into a soft latent tree that provides the answer (§ 4.2). The model is trained end-to-end using only question-answer supervision (§ 4.3).

4.1 Lexical Representation Assignment

Each token in the question sentence is assigned a distribution over the semantic types, and a grounded representation for each type. Tokens can only be assigned the **E**, **R**, **V**, and $\boldsymbol{\phi}$ types. For example, the token *cylindrical* in the question in Fig. 1 is assigned a distribution over the 4 semantic types (one shown) and for the **E** type, its representation is the set of *cylindrical* entities.

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

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

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

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

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

   For example, in Fig. 1, the word *red* will be grounded as all the red entities.

2. **R**-Type Representation: Each relation $r \in \mathcal{R}$, is represented using $v_r \in \mathbb{R}^d$. For each token w_i, we compute a distribution over relations, and then use this to compute the *expected* adjacency matrix that forms the **R**-type representation for this token.

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

   e.g. the word *left* in Fig. 1 is grounded as the subset of edges with label 'left'.

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

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

4.2 Parsing Questions

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

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

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

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

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

$$\psi^t_{i,j} = \sum_{k=i}^{j-1} \sum_{\substack{(t_1+t_2\rightarrow t) \\ \in \text{Modules}}} \psi^{t_1+t_2\rightarrow t}_{i,k,j}$$

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

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

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

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

Answer Grounding: By recursively computing the phrase semantic-type potentials and representations, we can infer the semantic type distribution of the complete question sentence (Eq. 1) and the resulting grounding for different semantic type 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:

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

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

4.3 Training Objective

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

$$\mathcal{L} = \sum_i \log p(\mathrm{a}^i|\mathrm{q}^i, \mathrm{KG}^i) \tag{3}$$

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

5 Dataset

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

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

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

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

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 things that are above a sphere metallic?*. These questions require more multi-step reasoning.

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

6 Experiments

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

6.1 Experimentation Setting

We describe training details, and the baselines.

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

[1] Johnson et al. 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 1: **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.

Baseline Models: We compare to the following baselines. **(a)** Models that assume linear structure of language, and encode the question using linear RNNs—LSTM (NO KG), LSTM, BI-LSTM, and a RELATION-NETWORK (Santoro et al., 2017) augmented model. [2] **(b)** Models that assume tree-like structure of language. We compare two variants of Tree-structured LSTMs (Zhu et al., 2015; Tai et al., 2015)—TREE-LSTM, that is trained on pre-parsed questions, and TREE-LSTM(UNSUP.), an unsupervised Tree-LSTM model (Maillard et al., 2017). For GENX, we also use an end-to-end semantic parsing model from Pasupat and Liang (2015). Finally, to isolate the contribution of the proposed denotational-semantics model, we train our model on pre-parsed questions. Note that, all LSTM based models only have access to the entities of the KG but not the relationship information between them. See Appendix C for details.

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 2: **Results for Complex Questions** (CLEVRGEN): All baseline models fail to generalize to questions requiring longer chains of reasoning than 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.

6.2 Experiments

Short Questions Performance: Table 1 shows that our model perfectly answers all test questions, demonstrating that it can learn challenging semantic operators and induce parse trees from end task supervision. Performance drops when using external parser, showing that our model learns an effective syntactic model for this domain. The RELATION NETWORK also achieves good performance, particularly on questions involving relations. LSTM baselines work well on questions not involving relations.[3]

Complex Questions Performance: Table 2 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 1, which were trained on short questions. All baselines achieve close to random performance, despite high accuracy for shorter questions. This shows the challenges in generalizing RNN encoders beyond their training data. In contrast, the strong inductive bias from our model structure allows it to generalize well to complex questions.

Performance on Human-generated Language: Table 3 shows the performance of our model on complex human-generated queries in GENX. Our approach outperforms strong LSTM and semantic parsing baselines, despite the semantic parser's use

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

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

Model	Accuracy
LSTM (NO KG)	0.0
LSTM	64.9
BI-LSTM	64.6
TREE-LSTM	43.5
TREE-LSTM (UNSUP.)	67.7
SEMPRE	48.1
Our Model (Pre-parsed)	67.1
Our Model	73.7

Table 3: **Results for Human Queries** (GENX) Our model outperforms LSTM and semantic parsing models on complex human-generated queries, showing it is robust enough to work on natural language. Better performance than TREE-LSTM (UNSUP.) shows the efficacy in representing subphrases using explicit denotations. Our model also improves upon using an external parser showing that it learns better syntax of the question.

of hard-coded operators. These results suggest that our method represents an attractive middle ground between minimally structured and highly structured approaches to interpretation. Our model learns to interpret operators such as *except* that were not considered during development. This shows that our model can learn to parse human language, which contains greater lexical and structural diversity than synthetic questions. Trees induced by the model are linguistically plausible (see Appendix D).

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

Discussion: While our model shows promising results, there is significant potential for future work. Performing exact inference over large KGs is likely to be intractable, so approximations such as KNN search, beam search, feature hashing or parallelization may be necessary. In most real-world scenarios there is single KG, such as Freebase, for which each entity can be represented using a separate low-dimensional embedding. To deal with the issue of cold-start, techniques proposed by recent work (Verga et al., 2017; Gupta et al., 2017) that explore representing entities as composition of its different properties, such as, types, description etc. can be used. In this work, the modules were designed in a way to provide good inductive bias for the kind of composition we expected them to model. For example, $\mathbf{EV} + \mathbf{E} \rightarrow \mathbf{E}$ is modeled as a linear composition function making it easy to represent words such as *and* and *or*. These modules can be exchanged with any other function with the same 'type signature', with different tradeoffs — for example, more general feed-forward networks with greater representation capacity would be needed to represent a linguistic expression equivalent to *xor*. Similarly, more module types would be required to handle certain constructions—for example, a multiword relation such as *much larger than* needs a $\mathbf{V} + \mathbf{V} \rightarrow \mathbf{V}$ module. This is an exciting line of work for future research.

7 Related Work

Many approaches have been proposed to perform question-answering against structured knowledge sources. *Semantic parsing* models have learnt structures over pre-defined discrete operators, to produce logical forms that can be executed to answer the question. Early work trained using gold-standard logical forms (Zettlemoyer and Collins, 2005; Kwiatkowski et al., 2010), whereas later efforts have only used answers to questions (Liang et al., 2011; Krishnamurthy and Kollar, 2013; Pasupat and Liang, 2015). A key difference is that our model must learn semantic operators from data, which may be necessary to model the fuzzy meanings of function words like *many* or *few*.

Another similar line of work is neural program induction models, such as Neural Programmer (Neelakantan et al., 2016) and Neural Symbolic Machine (Liang et al., 2017). These models learn to produce programs composed of predefined operators using weak supervision to answer questions against semi-structured tables.

Neural module networks have been proposed for learning semantic operators (Andreas et al., 2016b) for question answering. This model assumes that the structure of the semantic parse is given, and must only learn a set of operators. Dynamic Neural Module Networks (D-NMN) extend this approach by selecting from a small set of candidate module structures (Andreas et al., 2016a). We instead learn a model over all possible structures.

Our work is most similar to N2NMN (Hu et al., 2017) model, which learns both semantic operators

and the layout in which to compose them. However, optimizing the layouts requires reinforcement learning, which is challenging due to the high variance of policy gradients, whereas our approach is end-to-end differentiable.

8 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. Future work should explore scaling the model to other question answering tasks, using more general composition modules, and introducing additional module types.

Acknowledgement

We would like to thank Shyam Upadhyay, XYZ, and the anonymous EMNLP reviewers for their helpful suggestions.

References

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016a. Learning to compose neural networks for question answering. In *HLT-NAACL*.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016b. Neural module networks. In *CVPR*, pages 39–48.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM.

Nicholas FitzGerald, Yoav Artzi, and Luke S. Zettlemoyer. 2013. Learning distributions over logical forms for referring expression generation. In *EMNLP*.

Nitish Gupta, Sameer Singh, and Dan Roth. 2017. Entity linking via joint encoding of types, descriptions, and context. In *EMNLP*.

Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. 2017. Learning to reason: End-to-end module networks for visual question answering. *CoRR*, abs/1704.05526.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. 2017. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*.

Dan Klein and Christopher D Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting of the association for computational linguistics*.

Jayant Krishnamurthy and Thomas Kollar. 2013. Jointly learning to parse and perceive: Connecting natural language to the physical world. *TACL*, 1:193–206.

Tom Kwiatkowski, Luke S. Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *EMNLP*.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *ACL*.

Percy Liang, Michael I Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 590–599. Association for Computational Linguistics.

Jean Maillard, Stephen Clark, and Dani Yogatama. 2017. Jointly learning sentence embeddings and syntax with unsupervised tree-lstms. *CoRR*, abs/1705.09189.

Richard Montague. 1973. The proper treatment of quantification in ordinary English. In K. J. J. Hintikka, J. Moravcsic, and P. Suppes, editors, *Approaches to Natural Language*, pages 221–242. Reidel, Dordrecht.

Arvind Neelakantan, Quoc V. Le, Martín Abadi, Andrew McCallum, and Dario Amodei. 2016. Learning a natural language interface with neural programmer. *CoRR*, abs/1611.08945.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *ACL*.

Ethan Perez, Harm de Vries, Florian Strub, Vincent Dumoulin, and Aaron C. Courville. 2017. Learning visual reasoning without strong priors. *CoRR*, abs/1707.03017.

Adam Santoro, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter W. Battaglia, and Timothy P. Lillicrap. 2017. A simple neural network module for relational reasoning. In *NIPS*.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *ACL*.

Patrick Verga, Arvind Neelakantan, and Andrew McCallum. 2017. Generalizing to unseen entities and entity pairs with row-less universal schema. In *EACL*.

Adina Williams, Andrew Drozdov, and Samuel R Bowman. 2017. Learning to parse from a semantic objective: It works. is it syntax? *arXiv preprint arXiv:1709.01121*.

Luke S Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *UAI*.

Xiaodan Zhu, Parinaz Sobihani, and Hongyu Guo. 2015. Long short-term memory over recursive structures. In *International Conference on Machine Learning*, pages 1604–1612.

A Training Objective

Given a dataset $\mathcal{D}$ of (question, answer, knowledge-graph) tuples, $\{\mathrm{q}^i, \mathrm{a}^i, \mathrm{KG}^i\}_{i=1}^{i=|\mathcal{D}|}$, we train our model to maximize the log-likelihood of the correct answers. Answers are either booleans ($a \in \{0, 1\}$), or specific subsets of entities ($a = \{e_j\}$) from the KG. We denote the semantic type of the answer as a_t. The model's answer is found by taking the complete question representation, containing a distribution over types and the representation for each type. We maximize the following objective:

$$\mathcal{L} = \sum_i \log p(\mathrm{a}^i | \mathrm{q}^i, \mathrm{KG}^i) \quad (4)$$

$$= \sum_i \mathcal{L}_b^i + \mathcal{L}_e^i \quad (5)$$

where $\mathcal{L}_b^i$ and $\mathcal{L}_e^i$ are respectively the objective functions for questions with boolean answers and entity set answers.

$$\mathcal{L}_b^i = \mathbb{1}_{a_t^i = \mathbf{T}} \Big[\log(p_{true})^{a^i} (1 - p_{true})^{(1-a^i)} \Big] \quad (6)$$

$$\mathcal{L}_e^i = \frac{\mathbb{1}_{a_t^i = \mathbf{E}}}{|\mathcal{E}^i|} \Big[\log \prod_{e_j^i \in a^i} p_{e_j^i} \prod_{e_j^i \notin a^i} (1 - p_{e_j^i}) \Big] \quad (7)$$

We also add L_2-regularization for the scalar parsing features introduced in § 4.2.

B Training Details

Representing Entities: Each entity in CLEVR-GEN and GENX datasets consists of 4 attributes. For each attribute-value, we learn an embedding vector and concatenate these vectors to form the representation for the entity.

Training details: For curriculum learning, for the CLEVRGEN dataset we use a 2-step schedule where we first train our model on simple attribute match (*What is a red sphere?*), attribute existence (*Is anything blue?*) and boolean composition (*Is anything green and is anything purple?*) questions and in the second step on all questions. For GENX we use a 5-step, question-length based schedule, where we first train on shorter questions and eventually on all questions.

We tune hyper-parameters using validation accuracy on the CLEVRGEN dataset, and use the same hyper-parameters for both datasets. We train using SGD with a learning rate of 0.5, a mini-batch size of 4, and regularization constant of 0.3. When assigning the semantic type distribution to the words at the leaves, we add a small positive bias of $+1$ for $\boldsymbol{\phi}$-type and a small negative bias of -1 for the **E**-type score before the softmax. Our trainable parameters are: question word embeddings (64-dimensional), relation embeddings (64-dimensional), entity attribute-value embeddings (16-dimensional), four vectors per word for **V**-type representations, parameter vector θ for the parsing model that contains six scalar feature scores per module per word, and the global parameter vector for the **E+E→E** module.

C Baseline Models

C.1 LSTM (No KG)

We use a LSTM network to encode the question as a vector q. We also define three other parameter vectors, t, e and b that are used to predict the answer-type $P(a = \mathbf{T}) = \sigma(q \cdot t)$, entity attention value $p_{e_i} = \sigma(q \cdot e)$, and the probability of the answer being True $p_{true} = \sigma(q \cdot b)$.

C.2 LSTM

Similar to LSTM (NO RELATION), the question is encoded using a LSTM network as vector q. Similar to our model, we learn entity attribute-value embeddings and represent each entity as the concatenation of the 4 attribute-value embeddings, v_{e_i}. Similar to LSTM (NO RELATION), we also define the t parameter vector to predict the answer-type. The entity-attention values are predicted as $p_{e_i} = \sigma(v_{e_i} \cdot q)$. To predict the probability of the boolean-type answer being true, we first add the entity representations to form $b = \sum_{e_i} v_{e_i}$, then make the prediction as $p_{true} = \sigma(q \cdot b)$.

C.3 Tree-LSTM

Training the Tree-LSTM model requires pre-parsed sentences for which we use a binary constituency tree generating PCFG parser (Klein and Manning, 2003). We input the pre-parsed question to the Tree-LSTM to get the question embedding q. The rest of the model is same the LSTM model above.

C.4 Relation Network Augmented Model

The original formulation of the relation network module is as follows:

$$RN(q, KG) = f_\phi\left(\sum_{i,j} g_\theta(e_i, e_j, q)\right) \quad (8)$$

where e_i, e_j are the representations of the entities and q is the question representation from an LSTM network. The output of the Relation Network module is a scalar score value for the elements in the answer vocabulary. Since our dataset contains entity-set valued answers, we modified the module in the following manner.

We concatenate the entity-pair representations with the representations of the pair of relations between them[4]. We use the RN-module to produce an output representation for each entity as:

$$RN_{e_i} = f_\phi\left(\sum_{j} g_\theta(e_i, e_j, r^1_{ij}, r^2_{ij}, q)\right) \quad (9)$$

Similar to the LSTM baselines, we define a parameter vector t to predict the answer-type, and compute the vector b to compute the probability of the boolean type answer being true.

To predict the entity-attention values, we use a separate attribute-embedding matrix to first generate the output representation for each entity, e^{out}_i, then predict the output attention values as follows:

$$p_{e_i} = \sigma\left(RN_{e_i} \cdot e^{out}_i\right) \quad (10)$$

We tried other architectures as well, but this modification provided the best performance on the validation set. We also tuned the hyper-parameters and found the setting from Santoro et al. (2017) to work the best based on validation accuracy. We used a different 2-step curriculum to train the RELATION NETWORK module, in which we replace the Boolean questions with the relation questions in the first-schedule and jointly train on all questions in the subsequent schedule.

[4]In the CLEVR dataset, between any pair of entities, only 2 directed relations, *left* or *right*, and *above* or *beneath* are present.

C.5 SEMPRE

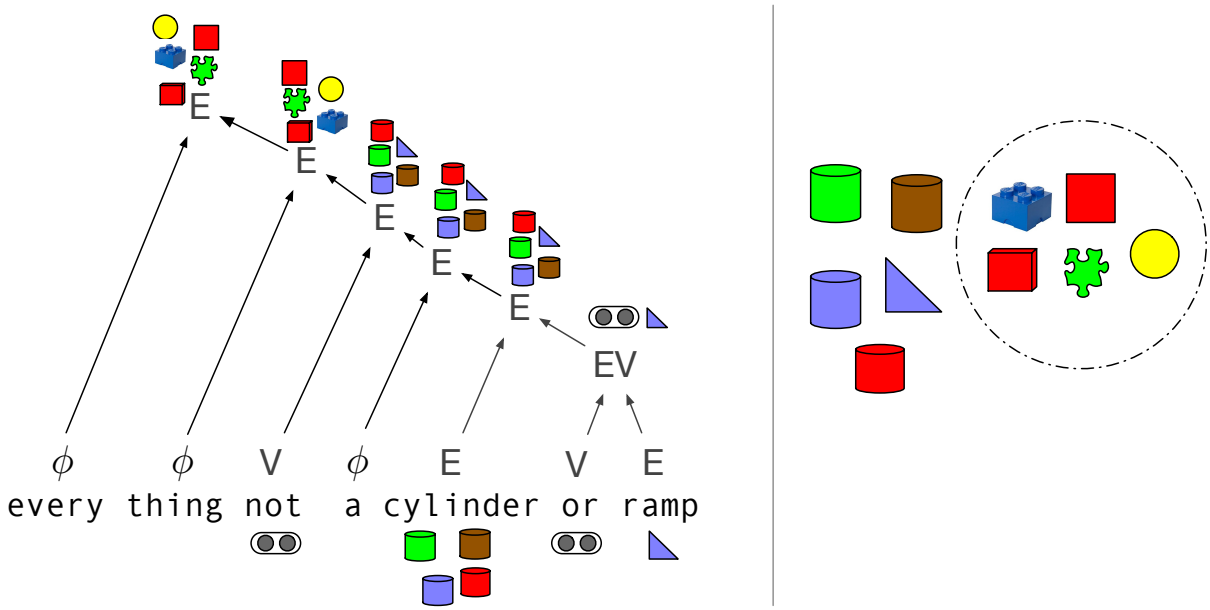
The semantic parsing model from (Pasupat and Liang, 2015) answers natural language queries for semi-structured tables. The answer is a denotation as a list of cells in the table. To use the SEMPRE framework, we convert the KGs in the GENX to tables as follows:

1. Each table has the first row (header) as: $\mid ObjId \mid P1 \mid P2 \mid P3 \mid P4 \mid$
2. Each row contains an object id, and the 4 property-attribute values in cells.
3. The answer denotation, i.e. the objects selected by the human annotators is now represented as a list of object ids.

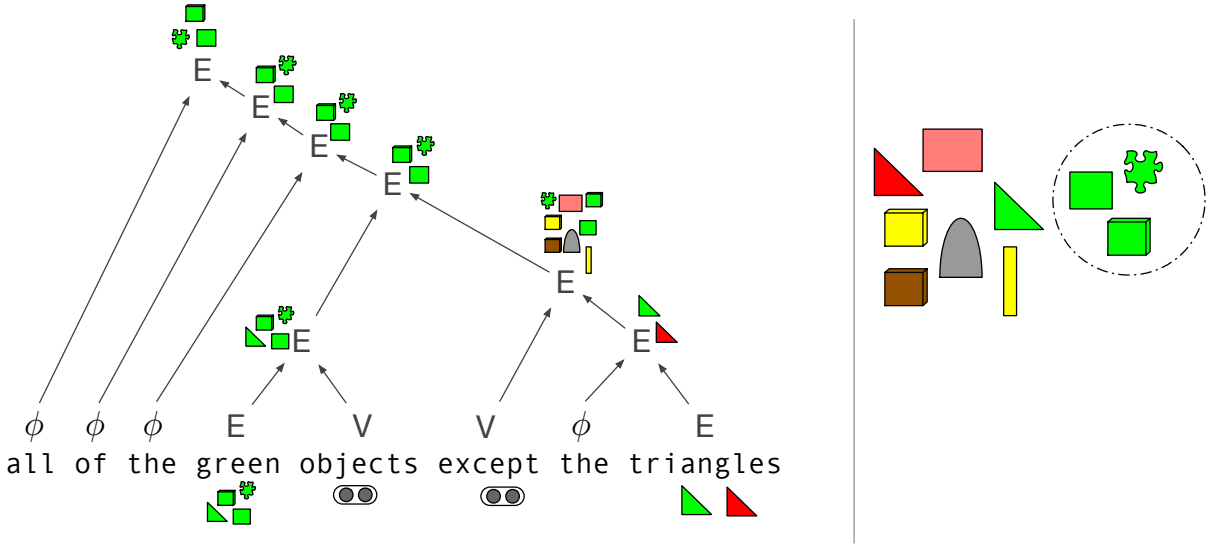
After converting the the KGs to tables, SEMPRE framework can be trivially used to train and test on the GENX dataset. We tune the number of epochs to train for based on the validation accuracy and find 8 epochs over the training data to work the best. We use the default setting of the other hyper-parameters.

D Example Output Parses

In Figure 3, we show example queries and their highest scoring output structure from our learned model for GENX dataset.



(a) An example output from our learned model showing that our model learns to correctly parse the questions sentence, and model the relevant semantic operator; *or* as a set union operation, to generate the correct answer denotation. It also learns to cope with lexical variability in human language; *triangle* being referred to as *ramp*.



(b) An example output from our learned model that shows that our model can learn to correctly parse human-generated language into relatively complex structures and model semantic operators, such as *except*, that were not encountered during model development.

Figure 3: **Example output structures from our learned model**: Examples of queries from the GENX dataset and the corresponding highest scoring tree structures from our learned model. The examples shows that our model is able to correctly parse human-generated language and jointly learn to model semantic operators, such as set unions, negations etc.