Can Natural Language Inference Models Perform Natural Logic Reasoning?

Atticus Geiger
Stanford Symbolic Systems Program
atticusg@stanford.edu

Abstract

Deep learning models for semantics are generally evaluated using naturalistic corpora. However adversarial testing experiments have recently challenged this paradigm, exposing the need for a broad and deep range of evaluations tasks. We help to address this need by constructing an artificial natural language inference task that evaluates a model's ability to perform natural logic reasoning. We find that standard neural models fail, with a continuous bag of words achieving comparable performance to complex deep learning models. Only task specific models that jointly compose premise and hypothesis sentences are able to achieve high performance, and even these models do not solve the task perfectly.

1 Introduction

Evaluations of deep learning approaches to semantics generally rely on corpora of naturalistic examples, using comparative performance metrics as a proxy for the underlying capacity of the models to learn rich meaning representations and find generalized solutions. From this persepctive, the fact that state-of-the-art neural models achieve human level performance on challenging high quality corpora such as the Stanford Natural Inference Corpus (SNLI) and the Stanford Question Answer Dataset (SQuAD) would lead to the conclusion that question answering and natural language inference (NLI) are solved by deep learning (Bowman et al., 2015a; Rajpurkar et al., 2016, 2018). However, adversarial testing experiments have exposed neural models to have brittle, local solutions that fail to generalize to data very similar to that shown in training. While well-known mathematical results show that neural networks are sufficient in terms of theoretical capacity, it remains an empirical question whether a given model architecture can achieve the desired representations and solutions in practice (Cybenko, 1989). We need a broad and deep range of evaluation methods to fully characterize the capacities of these models. We contribute to this line of research here, probing the generalization capabilities of natural language inference learning models by developing a task to evaluate the ability of these models to perform natural logic reasoning.

1.1 Natural Logic

Formal semantics traditionally adopts a truth conditional view of meaning, where a natural language sentence is translated into a logical form that characterizes a set of worlds where the sentence is true (Montague, 1973). However, there is a complementary approach to semantics, where inferential relations between sentences are taken to be primary rather than truth conditions. It is this alternative view that is taken in the natural logic program (Sánchez-Valencia, 1991; MacCartney, 2009; van Benthem, 2008; Icard and Moss, 2013). From a computational perspective, the relational view of meaning has the advantage of explicating how inference is performed, while the truth conditional view only implicitly defines inferential relations between sentences (MacCartney and Manning, 2009, 2007; Bowman, 2016). Because natural logic provides a particular solution to inference, we can investigate whether learning models are able to achieve this solution. In Sections 2 and 3, we argue the essence of natural logic reasoning is the recursive composition of intermediate representations up a tree structure and in Sections 4, 5, and 6 we investigate whether neural models are capable of such reasoning.

1.2 Generalization and Learning

It is intuitive that training and testing on the same data would only test a learning model's memorization ability and therefore is neither challenging nor interesting. As such, models are evaluated on their ability to learn a solution that generalizes to data unseen in training. We typically evaluate this ability by arbitrarily setting aside some percentage of available labelled inputs to evaluate fully trained models on. However, we argue in this thesis that arbitrarily setting aside samples for testing can be less challenging and interesting than desired. While more challenging generalization tasks have certainly been developed, such as training on one genre of the MultiNLI corpus and testing on another or testing on sentences longer than those seen in training, we believe that these experiments often lack the motivation for why a generalization task should be possible and an explicit research question that is being operationalized (Williams et al., 2017; Lake and Baroni, 2017). Should we expect models trained on fiction prose to generalize to data from 9/11 reports? When we find out a model is or isn't able to perform this task, what question is being answered? We certainly believe that a diverse array of generalization tasks are the right direction for model evaluation, however we also believe such tasks should be well motivated and, when possible, precisely formulated.

To introduce our conceptual framework for thinking about generalization tasks, we will argue that the task of training and testing on the same data operationalizes the question *Can some learning model memorize data?* We characterize this task by identifying the simplest model able to perform it. Here is a baseline learning model that achieves a perfect solution to the task: during training, this model simply constructs a map from inputs to outputs and during testing, when provided an input the model returns the mapped to output. The baseline model demonstrates that this task is possible and is transparent

in design, allowing us to use it as a scientific tool. This baseline model has an inductive bias to memorize training examples and it achieves perfect performance when trained and tested on the same data. As such, we would expect any model with the capability of memorizing training data to achieve perfect performance when trained and tested on the same data, and if perfect performance were not achieved, we would consider that evidence the answer to the question *Can this model memorize data?* is no.

This is a toy example of the following conceptual framework. Begin with a high level research question about whether learning models have some capability. Formulate a baseline learning model with an inductive bias that captures the capability being investigated. Construct a learning task that the baseline learning model achieves a perfect solution to. Test the ability of your chosen learning model to perform this task, and if your model fails to find a perfect solution, then it is evidenced to lack this capability.

In this thesis, we operationalize the question *Can some NLI learning model perform natural logic reasoning?* We eschew a binary answer, instead pursuing a nuanced, graded one grounded in empirical results. In Section 2 we present a baseline learning model that learns to recursively compose representations up a tree structure. In section 3, we argue that this recursive composition is the essence of natural logic reasoning. We leverage this conception of natural logic to develop a theoretical motivation for the NLI generalization task in Section 4 that answers our question at hand. In Section 5 we present our neural models and experimental protocol and in Section 6 we find that standard sequential, tree structured, and attention based neural models fail this task, with only task specific models that jointly composes premises and hypotheses achieving performance comparable to our baseline.

1.3 Related Work

We consider this thesis to be situated in a growing literature that explores the capability of learning models through targeted generalization tasks. One area of research in this vein is the adversarial testing of neural models trained on large scale naturalistic corpora, where training examples are systematically perturbed and then tested on. In computer vision, it is common to adversarially train on artificially noisy examples to create a more robust model (Goodfellow et al., 2015; Szegedy et al., 2014). However, in the case of question answering, Jia and Liang (2017) have demonstrated that training on one perturbation of data does not result in generalization to similar perturbations, revealing a need for models with stronger generalization capabilities. Similarly , models trained on SNLI have been exposed to have signifigant holes in their knowledge of lexical and compositional semantics (Glockner et al., 2018; Nie et al., 2018; Dasgupta et al., 2018). These are only some examples of the growing number of experiments that suggest that natural language understanding is far from solved by new deep learning models (Poliak et al., 2018; Gururangan et al., 2018; Tsuchiya, 2018).

Another area of the literature this thesis belongs to is work evaluating the logical reasoning capabilities of learning models using artificially generated inference tasks.

The work here focuses on a natural logic theory of first order logical reasoning, however there is also similar work focusing on a possible world semantics of propositional logic (Evans et al., 2018). Bowman et al. (2015b) conduct similar artificial experiments to those shown here. The most significant difference between our work and that of Bowman et al. (2015b) is our guiding research question. We construct a generalization task that isolates the ability to preform natural logic reasoning, while Bowman et al. (2015b) arbitrarily set aside examples for testing resulting in a far simpler task. Veldhoen and Zuidema (2018) analyze models trained on the tasks in Bowman et al. (2015b). They argue the trained models fail to discover the kind of global solution we would expect if they learned a natural logic system. We compare and contrast our work with Bowman et al. (2015b) and Veldhoen and Zuidema (2018) in Section 6.5.

2 Compositionality and Generalization

To operationalize the question *Can some NLI learning model perform natural logic reasoning?* we will define a baseline learning model that performs natural logic reasoning and then construct constrained generalization task this baseline learns perfectly. In this section we define a baseline learning model with an inductive bias to learn an algorithm that recursively composes intermediate representations up a tree structure. We explore how this baseline learning model can be used develop a generalization task that isolates a models ability to perform such recursive composition. We conclude by providing an algorithm that constructs data for this task. In Section 3, we will argue that this recursive composition up a tree structure is the essence of natural logic reasoning.

2.1 Composition Trees

We begin by introducing the concept of a composition tree. A composition tree describes how to recursively compose elements from an input space up a tree structure to produce an element in an output space. Our baseline learning model will construct a composition tree using training data.

Definition 1. (Composition Tree) Let T be an ordered tree whose leaf nodes are denoted by the set N_{leaf} and whose non-leaf nodes are denote by the set $N_{\text{non-leaf}}$. We may also refer to $N = N_{\text{leaf}} \cup N_{\text{non-leaf}}$, the set of all nodes in T. Let Dom be a map on N that assigns some set to each node, called the *domain* of the node. Let Func be a map on $N_{\text{non-leaf}}$ that assigns a function to each non-leaf node satisfying the following property: For any $a \in N_{\text{non-leaf}}$ with left-to-right ordered children $c_1, \ldots c_m$, we have that $\text{Func}(a) : \text{Dom}(c_1) \times \cdots \times \text{Dom}(c_m) \rightarrow \text{Dom}(a)$. We refer to the tuple $C = (T, \text{Dom}, \text{Func})$ as a *composition tree*. The *input space* of this composition tree is the cartesian product $\mathcal{I}_C = \text{Dom}(l_1) \times \cdots \times \text{Dom}(l_k)$ where $l_1, \ldots l_k$ are the leaf nodes in left-to-right order, and the *output space* is $\mathcal{O}_C = \text{Dom}(r)$ where r is the root node.

A composition tree $C = (T, \text{Dom}, \text{Func})$ realizes a function $F : \mathcal{I}_C \rightarrow \mathcal{O}_C$

in the following way: For any input $x \in \mathcal{I}_C$, this function is given by $F(x) = \texttt{compose}(C, r, x)$ where r is the root node of T and `compose` is defined recursively in Algorithm 1. This algorithm uses helper functions $\texttt{children}(a, C)$, which returns the left-to-right ordered children of node a in tree T, and $\texttt{index}(a, C)$, which returns index of a leaf according to left-to-right ordering.

Data: A composition tree C, a node $a \in N$, and an input $x \in \mathcal{I}_C$
Result: An output from $\text{Dom}(a)$

function `compose` (C, a, x)
  if $a \in N_{leaf}$ **then**
    $i \leftarrow \texttt{index}(a, C)$
    return x_i
  else
    $c_1, \ldots c_m \leftarrow \texttt{children}(a, C)$
    return $\text{Func}(a)(\texttt{compose}(C, c_1, x), \ldots\ \texttt{compose}(C, c_m, x))$
  end

Algorithm 1: Recursive composition up a tree

For a given $x \in \mathcal{I}_C$ and $a \in N_{\text{non-leaf}}$ with children $c_1, \ldots, c_m$, we say that the element of $\text{Dom}(c_1) \times \cdots \times \text{Dom}(c_m)$ that is input to $\text{Func}(a)$ during the computation of $F(x) = \texttt{compose}(C, r, x)$ is the input realized at $\text{Func}(a)$ on x and o, the element of $\text{Dom}(a)$ that is output by $\text{Func}(a)$, is the output realized at $\text{Func}(a)$ on x. If $l_{i_1}, \ldots, l_{i_k}$ are the leaf nodes descended from a, then we say that $(x_{i_1}, \ldots, x_{i_k}) \in \text{Dom}(l_{i_1}) \times \cdots \times \text{Dom}(l_{i_k})$ is a partial input that realizes output o at $\text{Func}(a)$. At a high level, $\texttt{compose}(C, a, x)$ finds the output realized at a node a by computing node a's function $\text{Func}(a)$ with the outputs realized at node a's children as inputs. This recursion bottoms out when the components of x are provided as the output realized at leaf nodes.

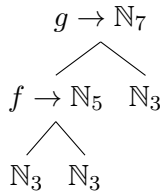

Figure 1: A composition tree that realizes a function adding three numbers from $\mathbb{N}_3$.

2.2 A Composition Tree for Addition

Now a short example. Consider $+ : \mathbb{N}_3 \times \mathbb{N}_3 \times \mathbb{N}_3 \to \mathbb{N}_7$ where $\mathbb{N}_i$ is the set containing the natural numbers 0 through $i - 1$ and $+(x, y, z) = (x + y) + z$. We define a composition tree that realizes $+$. Let T_+ be a binary tree with three leaf nodes. Let Dom_+ map the root node to $\mathbb{N}_7$, the children nodes to $\mathbb{N}_3$, and the other node to $\mathbb{N}_5$.

We define $f : \mathbb{N}_3 \times \mathbb{N}_3 \to \mathbb{N}_5$ to be $f(x, y) = x + y$ and $g : \mathbb{N}_5 \times \mathbb{N}_3 \to \mathbb{N}_7$ to be $g(x, y) = x + y$. Let Func_+ map the root node to g and the other non-leaf node to f. The composition tree $(T_+, \text{Dom}_+, \text{Func}_+)$ realizes $+\!\!+$. A visual representation of the composition tree $(T_+, \text{Dom}_+, \text{Func}_+)$ is shown in Figure 1, where a leaf node a is labeled with $\text{Dom}(a)$ and a non-leaf node a is labeled with $\text{Func}(a) \to \text{Dom}(a)$. To compute $+\!\!+(0, 1, 2)$ according to algorithm 1, we begin with the leaf nodes where the three recursive base cases are reached and 0, 1, and 2 are returned. Then we go up one level of the recursive call stack and return $f(0, 1) = 0 + 1 = 1$. We say that the partial input $(0, 1)$ realizes 1 at f. Finally, we move to final level and return $g(1, 2) = 1 + 2 = 3$. We say that $(1, 2)$ is the input realized at g on $(0, 1, 2)$ and that 3 is the output realized at g on $(0, 1, 2)$. This final output 3 is equal to $+\!\!+(0, 1, 2)$.

2.3 A Baseline Learning Model

We now present a baseline learning model in Algorithm 2 that learns a function by constructing a composition tree. Note that constructing a composition tree is equivalent to learning the function that tree realizes, as once the composition tree is created, Algorithm 1 computes the realized function. Because this model constructs a composition tree, it has an inductive bias to recursively compute intermediate representations up a tree structure. At a high level, it constructs a full composition tree when provided with the tree structure and training data that provides a value at every node in the tree by looping through training data inputs and memorizing the output realized at each intermediate function for a given input. As such, any learning model we compare to this baseline model should be provided the outputs realized at every node during training.

2.4 A Fair Generalization Task

Our goal is to create highly constrained training data sets that our baseline learning model learns a perfect solution on. Then we will have generalization tasks that are difficult, but fair relative to our baseline. We define a training data set to be fair with respect to a function if the presented baseline learning model constructs a composition tree that realizes the function. Training data that is fair must expose every intermediate function of a composition tree to every possible intermediate input, allowing the baseline model to learn a global solution.

Definition 2. (A Property Sufficient for Fairness) A property of a training data set $\mathcal{D}$ and tree T that is sufficient for fairness with respect to a function F is that there exists a composition tree $C = (T, \text{Dom}, \text{Func})$ realizing F such that for any $a \in N_{\text{non-leaf}}$ and for any input i to $\text{Func}(a)$ there exists $(x, Y) \in D$ where i is the input realized at $\text{Func}(a)$ on x.

This property is obviously sufficient for fairness, as our baseline model would simply construct C. For our purposes, we only need sufficiency, because, as it turns out, there are highly constrained training data sets that have this property. There are also

Data: An ordered tree T and a set of training data $\mathcal{D}$ containing pairs (x, Y) where x is an input and Y is a function defined on $N_{\text{non-leaf}}$ providing labels at every node of T.

Result: A composition tree $(T, \text{Dom}, \text{Func})$

function `learn` *$(T, \mathcal{D})$*

- Dom, Func= `initialize` (T)
- **for** $(x, Y) \in \mathcal{D}$ **do**
  - `memorize` $(x, Y, T, \text{Dom}, \text{Func}, r)$
- **end**
- **return** $(T, \text{Dom}, \text{Func})$

function `memorize` *$(x, Y, T, Dom, Func, a)$*

- **if** $a \in N_{leaf}$ **then**
  - $i = \texttt{index}(a, T)$
  - $\text{Dom}[a] = \text{Dom}[a] \cup \{x_i\}$
  - **return**
- **else**
  - $\text{Dom}[a] = \text{Dom}[a] \cup \{Y(a)\}$
  - $c_1, \ldots c_m = \texttt{children}(a, T)$
  - $\text{Func}[a][(Y(c_1), \ldots Y(c_m))] = Y(a)$
  - **for** $k = 1 \ldots m$ **do**
    - `memorize` $(x, Y, T, \text{Dom}, \text{Func}, c_k)$
  - **end**
  - **return**
- **end**

Algorithm 2: Given a tree and training data with labels for every node of the tree, this learning model constructs a composition tree. This algorithm uses helper functions $\texttt{children}(a, T)$ and $\texttt{index}(a, T)$ defined in Algorithm 1 as well as $\texttt{initialize}(T)$, which returns Dom, a dictionary mapping N to empty sets and Func, a dictionary mapping $N_{\text{non-leaf}}$ to empty dictionaries.

many unconstrained training data sets that have this property. For example, the entire space of data fulfills this property.

We will now make some basic observations about fair and unfair training sets for our running arithmetic example, to provide an intuition for the contours of this conception of fairness. Observe that for all $z \in \mathbb{N}_3$ the example $+\!\!+(0, 0, z) = (0 + 0) + z = 0 + z = z$ must be included for a training data set to be fair, as this is the only example that provides the function g the input $(0, z)$. Similarly, for all $z \in \mathbb{N}_3$ the example $+\!\!+(2, 2, z) = (2 + 2) + z = 4 + z$ must be included for a training data set to be fair, as this is the only example that provides the function g the input $(4, z)$. Observe that in a fair training data set it must be the case that for all $x, y \in \mathbb{N}_3$, there exists some $z \in \mathbb{N}_3$ such that the example $+\!\!+(x, y, z) = x + y + z$ is in the training data. This is because the function f must be exposed to all possible inputs partial inputs in $\mathbb{N}_3 \times \mathbb{N}_3$.

We stay with our arithmetic example to highlight the insight for how we constrain data while maintaining this sufficiency property. The idea is to restrict the partial inputs that realize the output of a node based on the outputs realized by the node's siblings. For example, we can include $+\!\!+(1, 1, 0) = (1 + 1) + 0 = 2 + 0 = 2$ and $+\!\!+(2, 0, 2) = (2 + 0) + 2 = 2 + 2 = 4$ in a training set and then reserve the examples $+\!\!+(2, 0, 0) = (2 + 0) + 0 = 2 + 0 = 2$ and $+\!\!+(0, 2, 2) = (0 + 2) + 2 = 4 + 0 = 4$ for testing because at every node the input realized by the latter two example has already been realized by one of the first two examples. This means, for a training example, if the output 2 is realized at the intermediate node and a 0 is realized at its sibling, then the partial input that realizes 2 is $(1, 1)$.

2.5 A Composition Tree for Sentiment Analysis

To give the reader further intuition for our notion of fairness and how we constrain datasets, we also present an example using the task of binary sentiment analysis. This task, like addition and, as we will soon show, NLI, can be solved by a composition tree that performs recursive composition of intermediate representations up a tree structure. We can see an exemplar of this idea in the conjunction but, which conjoins two expressions and projects the sentiment of the second expression. For example, the sentence *It was pretty trashy* has a negative sentiment and the sentence *I actually really liked it* has a positive sentiment, and so the sentence *It was pretty trashy, but I actually really liked it* has a positive sentiment. Syntactically, but takes in two expressions as arguments, but in the case of binary sentiment analysis, the conjunction but can be thought of as a function that operates on the sentiment of expressions; it operates on intermediate representations that abstract away from the specific identity of an expression.

We will now provide a composition tree for binary sentiment analysis for expressions of the form *Adj but Neg Adj*, where Neg can be not or the empty string ϵ and Adj can be $good$ or bad. There are eight expressions of this form. Formally, we can define $+$ to be positive sentiment and $-$ to be negative sentiment and think of but as the function $F_{but} : \{good, bad\} \times \{+, -\} \to \{+, -\}$ where $F_{but}(Adj, S_2) = S_2$. We

can think of the word not as being the function $F_{not} : \{good, bad\} \to \{+, -\}$ where $F_{not}(good) = -$ and $F_{not}(bad) = +$[1] and the empty string ϵ as being the function $F_\epsilon : \{good, bad\} \to \{+, -\}$ where $F_\epsilon(good) = +$ and $F_\epsilon(bad) = -$. We present a composition tree for binary sentiment analysis on these eight expressions in Figure 2, where $C_1(Neg, Adj) = F_{Neg}(Adj)$ and $C_2(Adj, but, S) = F_{but}(Adj, S)$.

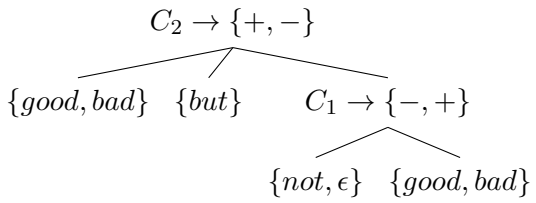

Figure 2: A composition tree that realizes a function performing binary sentiment analysis.

A fair training data set for this binary sentiment analysis task would have the following two properties. First, for any $Adj \in \{good, bad\}$ and any $S \in \{-, +\}$, there is some training example where the input realized at C_2 is (Adj, but, S); the local function word but must be exposed to all four inputs in $\{good, bad\} \times \{+, -\}$ during training. Second, for any $Neg \in \{not, \epsilon\}$ and $Adj \in \{good, bad\}$ there is some training example where the input realized at C_1 is (Neg, Adj); the local function words not and ϵ must both be exposed to all two inputs in $\{good, bad\}$.

The way we create a constrained, but fair training data set is by restricting the way $-$ and $+$ are realized at the intermediate node based on the adjective occurring at the far left leaf node. When the adjective at the far left leaf node is $good$ we realize $-$ with $not\ good$ and we realize $+$ with $not\ bad$. When the adjective at the far left leaf node is bad we realize $-$ with $\epsilon\ bad$ and we realize $+$ with $\epsilon\ good$. This gives us the following constrained, but fair training set containing four of the eight examples:

bad but ϵ good *bad but ϵ bad*

good but not bad *good but not good*

This crucially relies on the fact that there are multiple ways of realizing both $+$ and $-$ at the intermediate node. We are able to expose each local function to all possible inputs with only a fraction of our entire example space.

2.6 Data Generation

We now present Algorithm 3, which generates fair training data of varying difficulties by restricting the way an intermediate output is realized during training based on the outputs realized by sibling nodes. The $ratio$ parameter determines the difficulty

[1]This is an oversimplification that ignores the pragmatic inference that *not bad* has more of a neutral sentiment than a positive sentiment

of the generalization task; the higher the ratio, the more permissive the training set and the easier the task. This algorithm uses a handful of helper functions. The function $children(a)$ returns the left-to-right ordered children of node a and the function $cartesian\_product(L)$ returns the Cartesian product of a list of sets L. The function $sibling\_space(a, c_k)$ returns the set $\text{Dom}(c_1) \times \ldots \text{Dom}(c_{k-1}) \times \text{Dom}(c_{k+1}) \times \text{Dom}(c_m)$ where $c_1, \ldots, c_m$ are the children of a. The function $random\_even\_split(S, D, ratio)$ partitions a set S into P_1 and P_2 where $\frac{|P_1|}{|S|} = ratio$ and returns a function $F : D \to \wp(S)$[2] that satisfies the follow properties: every element in the range of F is non-empty, P_1 is a subset of every element in the range of F, the union of every element in in range of F is S, and the elements of P_2 are randomly and evenly distributed among the elements in the range of F.

Consider a node a with children $c_1, \ldots, c_m$ and leaf node descendants $l_1, \ldots, l_n$. Let $l_1^k, \ldots, l_{n_k}^k$ be the leaf node descendants of c_k. We define:

$$\mathcal{C} = \text{Dom}(c_1) \times \cdots \times \text{Dom}(c_m) \qquad \mathcal{L} = \text{Dom}(l_1) \times \cdots \times \text{Dom}(l_n)$$

$$\mathcal{C}_k = \text{Dom}(c_1) \times \ldots \text{Dom}(c_{k-1}) \times \text{Dom}(c_{k+1}) \times \text{Dom}(c_m)$$

$$\mathcal{L}_k = \text{Dom}(l_1^k) \times \cdots \times \text{Dom}(l_{n_k}^j)$$

The output of $generate\_inputs(T, \text{Dom}, \text{Func}, a, ratio)$ is a function $F_a : \text{Dom}(a) \to \wp(\mathcal{L})$ where $F_a(o)$ is a set of partial inputs that realize o at $\text{Func}(a)$. This function is recursively constructed in the following way:

For each child c_k, we call $generate\_inputs(T, \text{Dom}, \text{Func}, c_k, ratio)$ which outputs a function $F_{c_k} : \text{Dom}(c_k) \to \wp(\mathcal{L}_k)$ where $F_{c_k}(o)$ is a set of partial inputs that realize o at $\text{Func}(c_k)$.

Then for each $i_k \in \text{Dom}(c_k)$ we construct a function:

$$F_{i_k} = random\_even\_split(F_{c_k}(i_k), \mathcal{C}_k, ratio)$$

where $F_{i_k} : \mathcal{C}_k \to \wp(\mathcal{L}_k)$ and $F_{i_k}(j)$ is a set of partial inputs that realize i_k at $\text{Func}(c_k)$. The $ratio$ parameter tells us that there is a $P \subseteq F_{c_k}(i_k)$ where $\frac{|P|}{|F_{c_k}(i_k)|} = ratio$ and for all $j \in \mathcal{C}$ we have $P \subseteq F_{i_k}(j)$. At a high level, the function F_{i_k} maps a combination of outputs realized at $c_1, \ldots, c_{k-1}, c_{k+1}, \ldots, c_m$ to a set of partial inputs realizing i_k; it describes how i_k is realized based on the outputs realized by the siblings of c_k.

Then for every $i \in \mathcal{C}$, where $i = (i_1, \ldots, i_m)$, we compute the following cross product:

$$\mathcal{P}_i = F_{i_1}(i_2, \ldots, i_m) \times F_{i_2}(i_1, i_3, \ldots, i_m) \times \cdots \times F_{i_m}(i_1, \ldots, i_{m-1})$$

where P_i is a set of partial inputs realizing the input i at a.The output of $generate\_inputs(T, \text{Dom}, \text{Func}, a, ratio)$ is the function $F_a : \text{Dom}(a) \to \wp(\mathcal{L})$ where:

$$F_a(o) = \bigcup_{i \in \mathcal{C}} \mathcal{P}_i[\text{Func}(a)(i) = o]$$

[2] $\wp$ is the powerset function.

Data: A composition tree $(T, \text{Dom}, \text{Func})$ and $ratio$ a number between 0 and 1 inclusive.

Result: A training data set $\mathcal{D}$ that is fair with respect to our baseline model

function *generate_inputs*$(T, Dom, Func, a, ratio)$

- **if** $a \in N_{leaf}$ **then**
  - equivalence_classes = Dict()
  - **for** $i \in Dom(a)$ **do**
    - equivalence_classes[i] = {i}
  - **end**
  - **return** equivalence_classes
- **else**
  - $c_1, \ldots c_m \leftarrow \texttt{children}(a)$
  - equivalence_classes = Dict()
  - $\mathcal{C} = \text{Dom}(c_1) \times \text{Dom}(c_2) \cdots \times \text{Dom}(c_m)$
  - **for** $(i_1, i_2, \ldots, i_m) \in \mathcal{C}$ **do**
    - new_class = List()
    - **for** $k = 1 \ldots m$ **do**
      - new_class.append(
      - random_even_split(
      - generate_inputs$(T, \text{Dom}, \text{Func}, c_k)[i_k]$, sibling_space(a, k), $ratio)[i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_m]$)
    - **end**
    - equivalence_classes[Func$(a)(i_1, \ldots, i_m)$] =
    - equivalence_classes[Func$(a)(i_1, \ldots, i_m)$] $\cup$
      - cartesian_produce(new_class)
  - **end**
  - **return** equivalence_classes
- **end**

Algorithm 3: This model generates a training data set that is fair with respect to our baseline model. The output of $generate\_inputs(T, \text{Dom}, \text{Func}, a, ratio)$ is a function mapping elements of $\text{Dom}(a)$ to sets of partial inputs that realize the element.

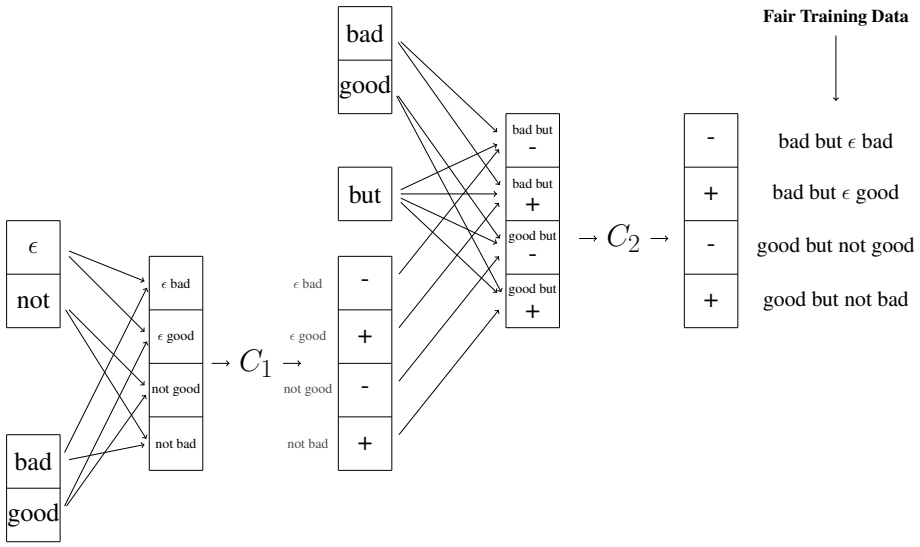

Figure 3: A visual representation of Algorithm 3 (left) generating a fair training data set (right) using the composition tree for binary sentiment analysis in Figure 2.

This recursion bottoms out when $generate\_inputs$ is called on a leaf node l and returns a function $F_l : \text{Dom}(l) \to \wp(\text{Dom}(l))$ where $F_l(o) = \{o\}$.

The output of $generate\_inputs(T, \text{Dom}, \text{Func}, r, ratio)$ is a function $F_r : \mathcal{O} \to \wp(\mathcal{I})$ where $F_r(o)$ is a set of inputs that realize o at $\text{Func}(r)$. The training data is:

$$\bigcup_{o \in \mathcal{O}} F_r(o)$$

The remaining input space is the testing data. If a tree has only root and leaf nodes, then this algorithm has no recursive calls and the entire input space is the training data.

2.7 A Fair Dataset for Sentiment Analysis

In Figure 2, we present a visualization of Algorithm 3 generating a fair training data set for our binary sentiment analysis example. We now walk through this visualization to provide an intuition for the algorithm. Let l_1, l_2, l_3, l_4 be the leaf nodes ordered left-to-right, r be the root node, and a be the other node of the tree for sentiment analysis. Let $\mathbb{B} = \{+, -\}$, $Adj = \{good, bad\}$, $Neg = \{not, \epsilon\}$, and $Conj = \{but\}$.

We now walk through the visualization. There are four recursive base cases, where $generate\_inputs$ is called on the leaf nodes l_1, l_2, l_3, l_4 and functions $F_{l_1} : Adj \to \wp(Adj)$, $F_{l_2} : Conj \to \wp(Conj)$, $F_{l_3} : Neg \to \wp(Neg)$, and $F_{l_4} : Adj \to \wp(Adj)$ are returned where $F_{l_1}(A) = F_{l_4}(A) = \{A\}$, $F_{l_2}(but) = \{but\}$, and $F_{l_3}(N) = \{N\}$ with $A \in Adj$ and $N \in Neg$.

When $generate\_inputs$ is called on a, for each $N \in Neg$ and $A \in Adj$ we construct functions:

$$F_N = random\_even\_split(F_{l_3}(N), Adj, ratio)$$

$$F_A = random\_even\_split(F_{l_4}(A), Neg, ratio)$$

Given an adjective $A \in Adj$ realized at l_3, the function $F_N : Adj \to \wp(Neg)$ tells us to realize the negation item $N \in Neg$ at node l_3 with the partial input $F_N(A) = \{N\}$. Given a negation item $N \in Neg$ realized at l_4, the function $F_A : Neg \to \wp(Adj)$ tells us to realize an adjective $A \in Adj$ with the partial input $F_A(N) = \{A\}$. At this point, there are not multiple ways to realize values, so these functions aren't interesting.

The four functions F_A and F_N where $A \in Adj$ and $N \in Neg$ are represented by the arrows at the far left of the diagram in Figure 3.

Then for every $(N, A) \in Neg \times Adj$, we compute the following cross product

$$\mathcal{P}_{(N,A)} = F_N(A) \times F_A(N)$$

where $P_{(N,A)} = \{(N, A)\}$ is a set of partial inputs realizing the input (N, A) at a. The output of $generate\_inputs(T, \text{Dom}, \text{Func}, a, ratio)$ is the function $F_a : \mathbb{B} \to \wp(Neg \times Adj)$ where:

$$F_a(o) = \bigcup_{(N,A) \in Neg \times Adj} \mathcal{P}_{(N,A)}[C_1(N, A) = o]$$

This function maps an element in $\mathbb{B}$ to a set of partial inputs that realizes that element:

$$F_a(+) = \{(not, bad), (\epsilon, good)\} \quad F_a(-) = \{(\epsilon, bad), (not, good)\}$$

When $generate\_inputs$ is called on r, for but and each $A \in Adj$ and $B \in \mathbb{B}$ we construct functions:

$$F_{but} = random\_even\_split(F_{l_2}(but), Adj \times \mathbb{B}, ratio)$$

$$F_A = random\_even\_split(F_{l_1}(A), Conj \times \mathbb{B}, ratio)$$

$$F_B = random\_even\_split(F_a(b), Adj \times Conj, ratio)$$

Given an adjective sentiment pair $(A, B) \in Adj \times \mathbb{B}$ realized at l_1 and a, the function $F_{but} : Adj \times \mathbb{B} \to \wp(Conj)$ tells us to realize the conjunction but at node l_2 with the partial input $F_{but}(A, B) = \{but\}$. Given a conjunction sentiment pair $(but, B) \in Conj \times \mathbb{B}$ realized at l_2 and a, the function $F_A : Conj \times \mathbb{B} \to \wp(Adj)$ tells us to realize an adjective $A \in Adj$ with the partial input $F_A(N) = \{A\}$. There are still not multiple ways to realize these values, so these functions aren't interesting.

Given a conjunction adjective pair $(A, but) \in Adj \times Conj$ realized at l_1 and l_2, the function $F_B : Adj \times Conj \to \wp(Neg \times Adj)$ tells us to realize a sentiment $B \in \mathbb{B}$ with the partial input $F_B(A, but)$. There are actually two ways to realize each of the two elements $+, - \in \mathbb{B}$, so this where we are actually able to constrain our final training data set. For the two elements in $Adj \times Conj$, we randomly assign a single way to realize $+$ and a single way to realize $-$ to one element and then assign the other way to

realize $+$ and the other way to realize $-$ to the other element. Below are the assignments that we make in Figure 3.

$$F_+(good, but) = \{(not, bad)\} \quad F_+(bad, but) = \{(\epsilon, good)\}$$
$$F_-(good, but) = \{(not, good)\} \quad F_-(bad, but) = \{(\epsilon, bad)\}$$

The five functions F_{but}, F_A, and F_B where $A \in adj$ and $B \in \mathbb{B}$ are represented by the arrows at the far right of the diagram in Figure 3.

The algorithm described thus far assumes that the $ratio$ parameter is 0. If we were to have a ratio of 1, then the functions F_+ and F_- would be as follows:

$$F_+(good, but) = \{(not, bad), (\epsilon, good)\} \quad F_+(bad, but) = \{(not, bad), (\epsilon, good)\}$$
$$F_-(good, but) = \{(not, good), (\epsilon, bad)\} \quad F_-(bad, but) = \{(not, good), (\epsilon, bad)\}$$

Then for every $(A, but, B) \in Adj \times Conj \times \mathbb{B}$, we compute the following cross product:

$$\mathcal{P}_{(A,but,B)} = F_A(but, B) \times F_{but}(A, B) \times F_B(A, but)$$

where $P_{(A,but,B)}$ is a set of partial inputs realizing (A, but, B). The output of $generate\_inputs(T, \text{Dom}, \text{Func}, r, ratio)$ is the function $F_r : \{+, -\} \to \wp(Adj \times Conj \times Neg \times Adj)$ where:

$$F_r(o) = \bigcup_{(A,but,B) \in Adj \times Conj \times \mathbb{B}} \mathcal{P}_{(A,but,B)}[C_2(A, but, B) = o]$$

This function maps an element in $\{+, -\}$ to a set of full inputs that realizes that element.

The training data shown in Figure 3 is:

$$\bigcup_{o \in \mathcal{O}} F_r(o)$$

3 Natural Logic

In Section 2 we presented a baseline model that recursively composes intermediate representations up a tree structure and provided an algorithm to create highly constrained generalization tasks on which this baseline model performs perfectly. We will now argue that this recursive composition captures the essence of natural logic reasoning. In this section we begin by introducing relevant natural logic theory and the NatLog model. We then extend the presented theory to include joint projectivity, allowing for new inferences involving quantifiers. We conclude by recasting the sequences of edits in the NatLog model as composition on aligned semantic parse trees by showing how to create composition trees for inference on sets of linguistic expressions. We point the reader to MacCartney and Manning (2007, 2009) for considerably more depth and detail and to van Benthem (2008) for a history.

By demonstrating that natural logic reasoning can be viewed as a composition tree algorithm, we will be able to make use of Algorithm 3 to create a highly constrained NLI generalization task that operationalizes our question *Can some NLI learning model perform natural logic reasoning?*, which we do in Section 4.

3.1 Relations and Projectivity

We begin by introducing $\mathcal{B}$ the set of seven basic semantic relations in Figure 4. These relations can hold between any linguistic expressions of the same type if the expressions are interpreted as sets. We define REL to be the function that outputs the relation between any two expressions. We will consider ϵ to be a type polymorphic empty string with unique relations such as: $REL(blue, \epsilon) = \sqsubset$, $REL(not, \epsilon) = \ \hat{}$, $REL(does, \epsilon) = \equiv$. We also introduce the join operator $\bowtie$ in Figure 5. This operator composes relations from $\mathcal{B}$ in the sense that if xR_1y and yR_2z then $x(R_1 \bowtie R_2)z$. Often the output of the join operator is the union of several relations; this is represented in Figure 5 by multiple relations being in a single cell. We approximate these unions with $\#$.

symbol	name	example	set theoretic definition
$x \equiv y$	equivalence	$couch \equiv sofa$	$x = y$
$x \sqsubset y$	forward entailment	$crow \sqsubset bird$	$x \subset y$
$x \sqsupset y$	reverse entailment	$bird \sqsupset crow$	$x \supset y$
$x \hat{} y$	negation	$human \hat{} nonhuman$	$x \cap y = \emptyset \wedge x \cup y = U$
$x \mid y$	alternation	$cat \mid dog$	$x \cap y = \emptyset \wedge x \cup y \neq U$
$x \smile y$	cover	$animal \smile nonhuman$	$x \cap y \neq \emptyset \wedge x \cup y = U$
$x \# y$	independence	$hungry \# hippo$	(all other cases)

Figure 4: Seven basic semantic relations

$\bowtie$	$\equiv$	$\sqsubset$	$\sqsupset$	$\wedge$	$\mid$	$\smile$	#
$\equiv$	$\equiv$	$\sqsubset$	$\sqsupset$	$\wedge$	$\mid$	$\smile$	#
$\sqsubset$	$\sqsubset$	$\sqsubset$	$\equiv\sqsubset\sqsupset$ | #	$\mid$	$\mid$	$\equiv\sqsubset\wedge\smile$ | #	$\sqsubset$ | #
$\sqsupset$	$\sqsupset$	$\equiv\sqsubset\sqsupset\smile$ #	$\sqsupset$	$\smile$	$\mid\sqsupset\smile$ #	$\smile$	$\sqsupset\smile$ #
$\wedge$	$\wedge$	$\smile$	$\mid$	$\equiv$	$\sqsupset$	$\sqsubset$	#
$\mid$	$\mid$	$\sqsubset\mid\smile$ #	$\mid$	$\sqsubset$	$\equiv\sqsubset\sqsupset$ | #	$\sqsubset$	$\sqsubset$ | #
$\smile$	$\smile$	$\smile$	$\sqsupset\mid\smile$ #	$\sqsupset$	$\sqsupset$	$\equiv\sqsubset\sqsupset\smile$ #	$\sqsupset\smile$ #
#	#	$\sqsubset\smile$ #	$\sqsupset\mid$ #	#	$\sqsupset\mid$ #	$\sqsubset\smile$ #	$\equiv\wedge\sqsupset\sqsubset\smile\mid$ #

Figure 5: The join operator

We also introduce the concept of projectivity. The projectivity signature of a semantic function f is $P_f : \mathcal{B} \rightarrow \mathcal{B}$ where if the relation between A and B is R the relation between $f(A)$ and $f(B)$ is $P_f(R)$. Let $\mathcal{P}$ be the set of all possible projectivity signatures. We provide the projectivity signatures of quantifiers and negation in Figure 6. The projectivity signature of the empty string is the identity function on $\mathcal{B}$.

	Projectivity for first argument							Projectivity for second argument						
	$\equiv$	$\sqsubset$	$\sqsupset$	$\wedge$	$\mid$	$\smile$	#	$\equiv$	$\sqsubset$	$\sqsupset$	$\wedge$	$\mid$	$\smile$	#
every	$\equiv$	$\sqsupset$	$\sqsubset$	$\mid$	#	$\mid$	#	$\equiv$	$\sqsubset$	$\sqsupset$	$\mid$	$\mid$	#	#
some	$\equiv$	$\sqsubset$	$\sqsupset$	$\smile$	#	$\smile$	#	$\equiv$	$\sqsubset$	$\sqsupset$	$\smile$	#	$\smile$	#
not	$\equiv$	$\sqsupset$	$\sqsubset$	$\wedge$	$\smile$	$\mid$	#	-	-	-	-	-	-	-

Figure 6: The projectivity signatures for negation and the quantifiers *every* and *some* from MacCartney and Manning (2007, 2009).

3.2 NatLog

We now concisely summarize the NatLog model to give the reader intuition for natural logic proofs. At a high level, NatLog finds a sequence of edits that transforms a premise into a hypothesis, computes the relations between the sentences in the edit sequence using lexical relations and projectivity signatures, and then composes the relations between the sentences using $\bowtie$ to find the relation between the premise and hypothesis. The three *atomic edits* in the NatLog system are the substitution, insertion, or deletion of a subexpression in a compound linguistic expression. For simplicity, we only have the edit substitution and consider insertion and deletion as substitution of the empty string for a non-empty subexpression and vice versa, respectively.

Here is a description of the NatLog model:

1. Find a sequence of atomic edits $\langle e_1, ..., e_n \rangle$ which transforms p into h: thus $h = (e_n \circ ... \circ e_1)(p)$. Let us define $x_0 = p, x_n = h$ and $x_i = e_i(x_{i-1})$ for $i \in [1, n]$.

2. For each atomic edit $e_i = SUB(A, B)$:

   (a) Determine $REL(A, B)$.

(b) Project $REL(A, B)$ upward through the semantic parse tree of expression x_{i-1} to find $REL(x_{i-1}, x_i)$.

3. Join the semantic relations across the sequence of edits to find the semantic relation between p and h: $REL(x_0, x_1) \bowtie ... \bowtie REL(x_{n-1}, x_n)$

3.3 Joint Projectivity

The natural logic described thus far is less expressive than first order logic. For example, it is unable to derive De Morgan's laws for quantifiers. We provide a small extension to the natural logic theory of MacCartney and Manning (2009) by introducing joint projectivity signatures, which allow for new inferences involving quantifiers. The joint projectivity signature of a pair of semantic functions f and g is $P_{f/g} : \mathcal{B} \to \mathcal{B}$ where if the relation between A and B is R the relation between $f(A)$ and $g(B)$ is $P_{f/g}(R)$.

We begin by analyzing how the NatLog system handles computing the relation between some *f A* and *g B*. As can be seen from Figure 7, there are two edit paths between *f A* and *g B* so according to the NatLog system, the relation between *f A* and *g B* is the stronger of the relations from the two edit paths, namely the stronger relation of $P_f(\text{REL}(X, Y)) \bowtie \text{REL}\ (f, g)$ and $\text{REL}\ (f, g) \bowtie P_g(\text{REL}(X, Y))$.

i	x_i	e_i	$REL(e_i)$	$REL(x_{i-1}, x_i)$	$REL(x_0, x_i)$
0	*f X*				
1	*f Y*	SUB(X, Y)	REL(X,Y)	$P_f(\text{REL}(X, Y))$	$P_f(\text{REL}(X, Y))$
2	*g Y*	SUB(f, g)	REL(f,g)	REL(f,g)	$P_f(\text{REL}(X, Y)) \bowtie \text{REL}\ (f, g)$
i	x_i	e_i	$REL(e_i)$	$REL(x_{i-1}, x_i)$	$REL(x_0, x_i)$
0	*f X*				
1	*g Y*	SUB(f, g)	REL(f,g)	REL(f,g)	$\text{REL}\ (f, g)$
2	*f Y*	SUB(X, Y)	REL(X,Y)	$P_f(\text{REL}(X, Y))$	$\text{REL}\ (f, g) \bowtie P_g(\text{REL}(X, Y))$

Figure 7: Two NatLog edit paths for computing the relation between *f A* and *g B*

However, this fails in the case of *some dog eats* and *every dog does not eat*. We see in Figure 8 that both of the edit paths output a # relation. However, it is the case that *some dog eats* ˆ *every dog does not eat*. This demonstrates the need for a joint projectivity signature that directly captures such relations.

i	x_i	e_i	$REL(e_i)$	$REL(x_{i-1}, x_i)$	$REL(x_0, x_i)$
0	*some dog eats*				
1	*some dog does not eat*	SUB(ϵ, *does not*)	ˆ	$\smile$	$\smile$
2	*every dog does not eat*	SUB(*some*, *every*)	$\sqsupset$	$\sqsupset$	$\smile \bowtie \sqsupset = \#$
i	x_i	e_i	$REL(e_i)$	$REL(x_{i-1}, x_i)$	$REL(x_0, x_i)$
0	*some dog eats*				
1	*every dog eats*	SUB(*some*, *every*)	$\sqsupset$	$\sqsupset$	$\sqsupset$
2	*every dog does not eat*	SUB(ϵ, *does not*)	ˆ	|	$\sqsupset \bowtie \| = \#$

Figure 8: Two NatLog edit paths for computing the relation between *some dog eats* and *every dog does not eat*

The joint projectivity signatures of $every$ and $some$ are provided in Figure 9. With these joint projectivity signatures, the natural logic is able to derive De Morgan's laws. We define the function PROJ to be the function that outputs the joint projectivity signature between two semantic functions. Observe that the joint projectivity between a semantic function and itself is equivalent to the projectivity of that semantic function.

The joint projectivity signatures between $every$ and $some$ are provided in Figure 9 along with the joint projectivity signatures between ε and not. We mark where the natural logic of MacCartney and Manning (2009) is extended. The remaining joint projectivity signatures between the quantifiers *every*, *some*, *no*, and *not every* can be determined by composing the joint projectivity signatures of *every* and *some* with the joint projectivity signatures between *not* and ϵ, where we parse *no* as *not some*.

	Projectivity for first argument							Projectivity for second argument						
	≡	⊏	⊐	^	|	⌣	#	≡	⊏	⊐	^	|	⌣	#
some/every	⊐	⊐	⊐	#	#	#	#	⊐	#	⊐	^	|	⌣	#
every/some	⊏	⊏	⊏	#	#	#	#	⊏	⊏	#	^	|	⌣	#
every/every	≡	⊐	⊏	|	#	|	#	≡	⊏	⊐	|	|	#	#
some/some	≡	⊏	⊐	⌣	#	⌣	#	≡	⊏	⊐	⌣	#	⌣	#
not/ε	^	⌣	|	≡	⊐	⊏	#	-	-	-	-	-	-	-
ε*/not*	^	⌣	|	≡	⊏	⊐	#	-	-	-	-	-	-	-
not/not	≡	⊐	⊏	^	⌣	|	#	-	-	-	-	-	-	-
ε/ε	≡	⊏	⊐	^	|	⌣	#	-	-	-	-	-	-	-

Figure 9: The four joint projectvity signatures between quantifiers *every* and *some* and the four joint projectivity signatures between ε and *not*. Boxed relations would not be computed in the natural logic of MacCartney and Manning (2009)

3.4 Compositionality and Natural Logic

The concept of linguistic compositionality has been informally defined by Partee (1984) as the meaning of an expression being a function of the meanings of its parts and of the way they are syntactically combined. Natural logic provides an analogous concept of compositionality: the relation of two expressions is a function of the relations and projectivity of their parts and of the way they are syntactically combined. The NatLog system computes inference using a sequence of edits, and we now recast this computation as composition on aligned semantic parse trees in Figure 10.

We now connect natural logic with composition trees, which recursively compose intermediate representations up a tree structure. When inference is performed on a

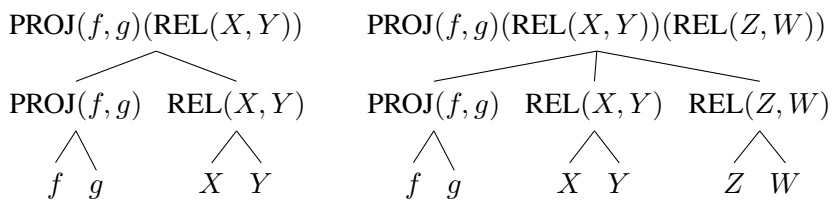

Figure 10: Natural logic inference cast as composition on aligned semantic parse trees. We show inference on $f\ X$ and $g\ Y$, where f and g are one place semantic functions, (left) and $f\ X\ Z$ and $g\ Y\ W$, where f and g are two place semantic functions (right).

pair of expressions using natural logic reasoning, the nature of the recursive composition is determined by the projectivity signatures of semantic functions. These semantic functions operate on the semantic relations between aligned subexpressions, which are intermediate representations that abstract away from the specific identity of the aligned subexpressions. The tree structure is determined by a shared semantic parse tree between the two expressions.

These composition trees are *aligned* in the following sense: they jointly composes lexical items from the premise and hypothesis. The leaf nodes come in sibling pairs where one sibling is a lexical item from the premise and the other is a lexical item from the hypothesis. If both leaf nodes in a sibling pair have domains containing lexical items that are semantic functions, then their parent node domain contains the joint projectivity signatures between those semantic functions. Otherwise the parent node domain contains the semantic relations between the lexical items in the two sibling node domains. The root captures the overall semantic relation between the premise and the hypothesis, while the remaining non-leaf nodes represent intermediate phrasal relations.

For example, consider the set of expressions {*fun, not fun, very fun, not very fun*}. A visualization of a composition tree for inference on these expressions is shown in Figure 11, where a leaf node a is labeled with $\text{Dom}(a)$ and a non-leaf node a is labeled with $\text{Func}(a) \to \text{Dom}(a)$. Now that natural logic reasoning has been formulated in the language of composition tree, we know our baseline learning model can learn inference and we can use it to create fair NLI generalization tasks.

A fair training data set for NLI would have the following properties. For any aligned semantic function f and g with unknown joint projectivity signature $P_{f/g}$ and for any semantic relation $R \in \mathcal{B}$, there is some training example where the semantic function $P_{f/g}$ is exposed to the semantic relation R. A fair training data set exposes each local function to all possible inputs. Our baseline learning model locally memorizes how semantic relations are projected through aligned semantic functions. Because there are many different aligned subexpressions that are in the same semantic relations, we are able to create constrained, but fair training data sets for NLI that our baseline model learns a perfect solution from.

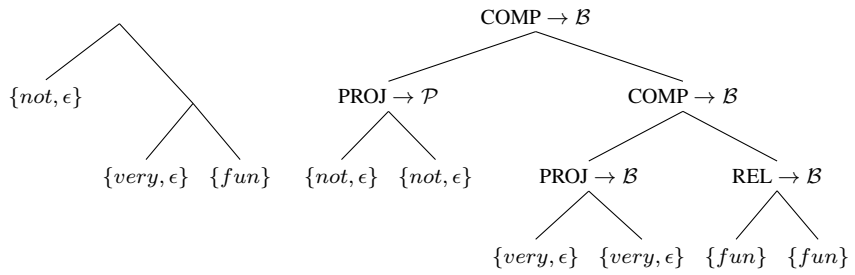

Figure 11: A shared semantic parse tree (left) and composition tree for inference on {*fun, not fun, very fun, not very fun*} (right).

4 An Artificial NLI Task

Now that we have connected composition trees and natural logic, we are able to construct a generalization task that operationalizes our question at hand *Can some NLI learning model perform natural logic reasoning?* In this section we present a method for generating artificial data sets for the task of natural language inference. Our method is built around an interpreted formal grammar that generates sentences containing two quantifiers, modifiers, and negation. The sentences from this grammar are deterministically translated into first-order logic, and an off-the-shelf theorem prover is used to generate labels for NLI examples. In this setting, we have control over all aspects of the generated data set and complete visibility into how different systems handle specific classes of example.

4.1 Data Generation

Our fragment G consists of sentences of the form:

$$\mathrm{Q_S\ Adj_S\ N_S\ Neg\ Adv\ V\ Q_O\ Adj_O\ N_O}$$

where $\mathrm{N_S}$ and $\mathrm{N_O}$ are nouns, V is a verb, $\mathrm{Adj_S}$ and $\mathrm{Adj_O}$ are adjectives, and Adv is an adverb. Neg is *does not*, and $\mathrm{Q_S}$ and $\mathrm{Q_O}$ can be *every*, *not every*, *some*, or *no*; in each of the remaining categories, there are 100 words. Additionally, $\mathrm{Adj_S}$, $\mathrm{Adj_O}$, Adv, and Neg can be the empty string, which is represented in the data by a unique token. Semantic scope is fixed by surface order, with earlier elements scoping over later ones.

For NLI, we define the set of premise–hypothesis pairs $\mathcal{S} \subset G \times G$ such that $(s_p, s_h) \in \mathcal{S}$ iff the non-identical non-empty nouns, adjectives, verbs, and adverbs with identical positions in s_p and s_h are in the $\#$ relation. This constraint on $\mathcal{S}$ trivializes the task of determining the lexical relations between adjectives, nouns, adverbs, and verbs, since the relation is $\equiv$ where the two aligned elements are identical and otherwise $\#$. Furthermore, it follows that distinguishing contradictions from entailments is trivial. The only sources of contradictions are negation and the negative quantifiers *no* and *not every*. Consider $(s_p, s_h) \in \mathcal{S}$ and let C be the number of times negation or a negative quantifier occurs in s_p and s_h. If s_p contradicts s_h, then C is odd; if s_p entails s_h, then C is even.

We constrain the open-domain vocabulary[3] to trivialize their contributions. This stresses models with learning interactions between logically complex function words; we trivialize the task of lexical semantics to isolate the task of compositional semantics. We also do not have multiple morphological forms, use artificial tokens that do not correspond to English words, and collapse *do not* and *not every* to single tokens too further simply the task and isolate a model's ability to perform compositional logical reasoning.

Our corpora use the three-way labeling scheme of *entailment*, *contradiction*, and *neutral*. To assign these labels, we translate each premise–hypothesis pair into first-order logic and use Prover9 (McCune, 2005–2010). If the logical form of the premise entails the logical form of the hypothesis, then the label is $entailment$. If the logical form of the premise entails the negation of the logical form of the hypothesis, then the label is $contradiction$. Otherwise, the label is $neutal$. We assume no expression is empty or universal and encode these assumptions as additional premises. This label generation process implicitly assumes the relation between unequal nouns, verbs, adjectives, and adverbs is independence.

For NLI corpora, we create samples from $\mathcal{S}$ in which, for a given example, every adjective–noun and adverb–verb pair across the premise and hypothesis is equally likely to have the relation $\equiv$, $\sqsubset$, $\sqsupset$, or $\#$. Without this balancing, any given adjective–noun and adverb–verb pair across the premise and hypothesis has more than a 99% chance of being in the independence relation. Even with this step, 98% of the sentence pairs are neutral, so we again sample to create corpora that are balanced across the three NLI labels. This balancing of labels justifies our use of an accuracy metric rather than an F1 score.

4.2 A Composition Tree for NLI

We now present a composition tree for inference on $\mathcal{S}$ in Figure 12. This tree was constructed in an identical manner to the composition tree in Section 3.4, except we trimmed the domain of every node so that the function of every node is surjective. Observe that pairs of expressions containing quantifiers can be in any of the seven basic semantic relations; even with the contributions of open-class lexical items trivialized, the level of complexity remains high, and all of it emerges from semantic composition, rather than from lexical relations.

The sets Adj_S, N_S, Adj_O, N_O,Adv, and V each have 100 of their respective open class lexical items. The set Q is $\{some, every, no, not\ every\}$ and the set Neg is $\{\epsilon, not\}$. $\mathcal{Q}$ is the set of 16 joint projectivity signatures between the quantifiers *some*, $every$, no, and *not every*, $\mathcal{N}$ is the set of 4 joint projectivity signatures between the empty string ϵ and no, and $\mathcal{A}$ is the set of 4 projectivity signatures between the empty string and an intersective adjective or adverbs in the $\equiv$ or $\#$ relation. These joint projectivity signatures were exhaustively determined by the authors by hand, using the

[3]Open-domain vocabulary meaning nouns, verbs, adjecitves, and adverbs

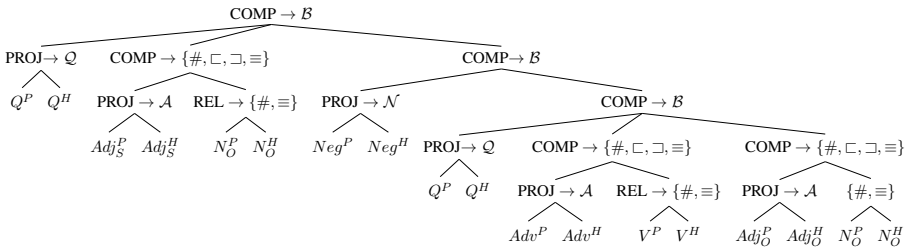

Figure 12: A composition tree for inference on $\mathcal{S}$ our set of examples. The superscripts P and H refer to premise and hypothesis. We use the seven basic semantic relations of MacCartney and Manning (2009): $\mathcal{B} = \{\#$ = independence, $\sqsubset$ = entailment, $\sqsupset$ = reverse entailment, $|$ = alternation, $\smile$ = cover, $\hat{}$ = negation, $\equiv$ = equivalence$\}$. The functions PROJ, COMP, and REL and the set $\mathcal{B}$ are defined in Section 3.4.

joint projectivity signatures between negation items and quantifiers provided in Section 3 and the projectivity signature that MacCartney and Manning (2009) ascribes typical intersective modifiers, which is $\{\equiv:\equiv, \sqsupset:\sqsupset, \sqsubset:\sqsubset, \hat{}: |, | : |, \smile: \#, \# : \#\}$. [4]

We implemented this composition tree to ensure our data is labelled correctly and that our joint projectivity signatures are correct. The labels using our first order logic theorem prover coincided perfectly with the labels from our composition tree.

5 Models and Tasks

In this section we describe six NLI learning models and the two kinds of generalization tasks we evaluate them on.

5.1 Models

We consider six different model architectures:

CBoW Premise and hypothesis are represented by the average of their respective word embeddings (continuous bag of words).

LSTM Encoder Premise and hypothesis are processed as sequences of words using a recurrent neural network (RNN) with LSTM cells, and the final hidden state of each serves as its representation (Hochreiter and Schmidhuber, 1997; Elman, 1990; Bowman et al., 2015a).

TreeNN Premise and hypothesis are processed as trees, and the semantic compsition function is a single layer feed forward network (Socher et al., 2011b,a). The value of the root node is the semantic representation in each case.

Attention LSTM An LSTM RNN with word-by-word attention (Rocktäschel et al., 2015).

[4] In our case, the only relations being input to the adjective and adverb signatures are $\#$ and $\equiv$, so it doesn't matter what we treat there signatures, since $\#$ and $\equiv$ are always mapped to themselves.

CompTreeNN The premise and hypothesis are processed as a single aligned tree, following the structure of the composition tree in figure 12. The semantic compsition function is a single layer feed forward network (Socher et al., 2011b,a). The value of the root node is the semantic representation of the premise and hypothesis together.

CompTreeNTN Identical to the CompTreeNN model, except a neural tensor network is used as a composition function (Socher et al., 2013).

For the first three models, the premise and hypothesis representations are concatenated. For the CompTreeNN, CompTreeNTN and Attention LSTM, there is just a single representation of the pair. In all cases, the premise-hypothesis representation is fed through two hidden layers and a softmax layer.

All models are initialized with random 100-dimensional word vectors and optimized using Adam (Kingma and Ba, 2014). A grid hyperparameter search was run over dropout values of $[0, 0.1, 0.2, 0.3]$ on the output and keep layers of LSTM cells, learning rates of $[1\mathrm{e}{-2}, 3\mathrm{e}{-3}, 1\mathrm{e}{-3}, 3\mathrm{e}{-4}]$, L2 regularization values of $[0, 1\mathrm{e}{-4}, 1\mathrm{e}{-3}, 1\mathrm{e}{-2}]$ on all weights, and activation functions relu and $\tanh$. Each hyperparamter setting was run for three epochs and parameters with the highest development set score were used to fully train six models.

5.2 Tasks

We will evaluate these models on two learning tasks. All random sampling is balanced across adjective-noun and adverb-verb relations as well as across the three NLI labels as described in section 4.1.

We refer to the first task as the standard generalization task, where a training set contains 500000 randomly sampled examples and the test and development sets each contain 10000 distinct randomly sampled examples. We call this task standard because arbitrarily setting aside examples for testing and development is standard practice. We consider this task simple, because there is a near trivial learning model that performs perfectly on the standard generalization task. During training, for each example this model computes the relations between open class lexical items and then maps a tuple containing the semantic relations between open class lexical items and the identity and location of quantifiers and negation to an output label. It abstracts away the identity of open class lexical items and then memorizes training examples. Since there are far more open class lexical items than there are quantifiers and negation, the only difference between training and testing data is in these open class lexical items. Specifically, every possible combination of open class lexical relations, quantifiers, and negation are seen during training, so this learning model achieves a perfect solution. Then we understand this standard generalization task to test no more than the ability of models to learn lexical relations and then construct large look up tables; no interesting logical reasoning is necessary to achieve perfect performance on this task.

sentence	verb phrase	modifier phrase	single words
Every tall kid ϵ happily kicks every ϵ rock	happily kicks every ϵ rock	happily kicks	tall $\equiv$ tall
entails	$\sqsubset$	$\sqsubset$	kid $\equiv$ kid
No tall kid does not ϵ kicks some large rock	ϵ kicks some large rock	ϵ kicks	happily $\sqsubset$ ϵ
negation phrase	modifier phrase	modifier phrase	single words
ϵ happily kicks every ϵ rock	tall kid	ϵ rock	kicks $\equiv$ kicks
$\mid$	$\equiv$	$\sqsupset$	ϵ $\sqsupset$ large
does not ϵ kicks some large rock	tall kid	large rock	rock $\equiv$ rock

Figure 13: For the natural logic generalization task, for any example sentence pair (top left) the neural models are trained using the a weighted sum of the error on 12 prediction tasks shown above.

We refer to the second task as the natural logic generalization task, where Algorithm 3 is ran with the composition tree from Figure 12 with some ratio R to generate $\mathcal{S}_{train}$ a subset of $\mathcal{S}$. The training set contains 500000 examples randomly sampled from $\mathcal{S}_{train}$ and the test and development sets each contain 10000 distinct examples randomly sampled from $\bar{\mathcal{S}}_{train}$. Remember that even the hardest generalization task (which is when R=0) adheres to our criteria of fairness, and so we should expect a model that performs natural logic reasoning to have perfect performance on this task just as our baseline model does. For this reason, we consider this to be our primary generalization task. However, we will evaluate models on generalization tasks where R ranges from 0 to 1 to create a gradation of difficulty and expose nuanced differences in generalization capabilities. We experimentally verified that our generated datasets adhere to our standard of fairness.

The training data sets for the natural logic generalization task are only fair if the outputs realized at every non-leaf node are provided during training just as they are in our baseline learning model. For our neural models, we accomplish this by predicting semantic relations for every subexpression pair in the scope of a node in the tree in Figure 12 and summing the loss of the predictions together. We do not do this for the nodes labeled PROJ $\rightarrow \mathcal{Q}$ or PROJ $\rightarrow \mathcal{N}$, as the function PROJ is a bijection at these nodes and no intermediate representations are created. For any example sentence pair the neural models are trained using the a weighted sum of the error on 12 prediction tasks shown in Figure 13. The 12 errors are weighted to regularize the loss according to the length of the expressions being predicted on.

6 Results and Analysis

6.1 The Standard Generalization Task

Our central finding on the standard generalization task is that only the CompTreeNN and CompTreeNTN perform perfectly. Even the Attention LSTM, which has the space to learn lexical alignments, fails to find optimal solutions. What is special about the CompTreeNN and CompTreeNTN are their ability to abstract away the identity of specific lexical items by computing and propagating semantic relations. When stressed

with even our small fragment of natural language's complexity, the other models lose the identity of open-class lexical items as the sentence embeddings are recursively constructed. This results in systematic errors, which is surprising considered the low difficulty of this task.

Table 1 summarizes the performance of each model on the standard generalization task. Figure 14 shows dev-set model performance througout training. The CBoW, LSTM, Attention LSTM, and TreeNN all jump quickly to $\approx 94\%$, and slightly increase performance plateauing at very good but not perfect performance ($\approx 96\%$). When trained further, the train set is overfit and test performance returns to $\approx 94\%$. Only the CompTreeNN and CompTreeNTN are able to perform perfectly.

Model	Train	Dev	Test	Informative Open-class Subset
CBoW	96.29 ± 0.30	95.4 ± 0.2	95.06 ± 0.22	89.24 ± 0.79
LSTM Encoder	96.05 ± 0.29	95.83 ± 0.14	95.61 ± 0.21	26.90 ± 1.44
TreeNN	96.20 ± 0.17	96.19 ± 0.15	95.99 ± 0.11	31.09 ± 2.91
Attention LSTM	97.50 ± 2.69	95.98 ± 2.23	95.82 ± 2.16	35.69 ± 35.98
CompTreeNN	99.85 ± 0.07	99.87 ± 0.06	99.85 ± 0.12	98.05 ± 1.02
CompTreeNTN	99.99 ± 0.00	99.99 ± 0.00	99.95 ± 0.00	99.78 ± 0.00

Table 1: Mean accuracy of 5 runs on the standard generalization task, with 95% confidence intervals. 'Informative Open-class Subset' are the Test set examples labeled neutral solely due to the independence relation between open-class lexical items.

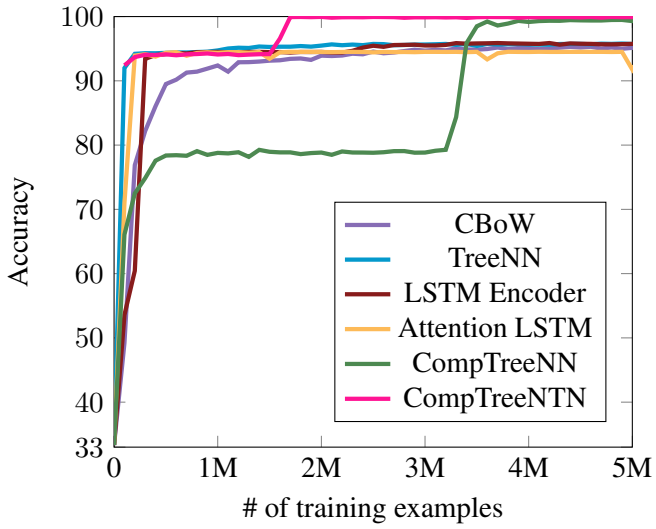

Figure 14: Model performance on the standard generalization task throughout training.

6.2 A Shared Suboptimal Solution

We noted above that the CBoW, LSTM, Attention LSTM, and TreeNN models get stuck at $\approx 94\%$ accuracy early in training. Because of the highly controlled way that we generate our data sets, we can pinpoint exactly why this happens. The LSTM, Attention LSTM, and TreeNN models find the same suboptimal solution: they learn the identity

and order of quantifiers, the importance of negation, and whether or not adjectives and adverbs are empty, but they are unable to make use of the specific identity of non-empty nouns, verbs, adjectives, and adverbs. As a result, they make systematic errors on neutral examples that would be contradictions or entailments if aligned open-class lexical were equal (see Table 1). The following collapsing is performed:

$$\begin{aligned} \textit{Every Swiss baker madly rubs some rock} &\Rightarrow \textit{every } Adj_S\ N_S\ \textit{Adv V some } N_O \\ \textit{neutral} &\Rightarrow \textit{entailment} \\ \textit{Every wild baker sells some rock} &\Rightarrow \textit{every } Adj_S\ N_S\ \ \textit{V some } N_O \end{aligned}$$

As a result of this collapsing, this looks like an entailment relation, because the only difference is the deletion of the adverb, which expands the scope of the universal quantification. However, the relation between these sentences is in fact neutral.

For the LSTM Encoder and TreeNN models, there is a natural explanation for why these errors are made. These models separately bottleneck the information from the premise and hypothesis into two 100-dimensional embeddings before a prediction is made using the concatenation of these embeddings. The function words are, like function words in natural data, very complex and very frequent as compared to open-class items. The stress of learning them seems to nullify the models' ability to store the precise identity of up to six open-class items per example, each drawn from a lexicon of 100 items. Both these models make minor improvements on this solution to achieve $\approx 96\%$. The LSTM sometimes notices when object nouns and adjectives differ across the premise and hypothesis, and the TreeNN model is able to do so with subject nouns and adjectives. These are precisely the lexical items whose contributions to the sentence embeddings are most recent, emphasizing the architectural nature of this problem.

The Attention LSTM has a high variance. On some runs, it gets stuck at $\approx 94\%$ test accuracy, and others it achieves $\approx 97\%$. This gives some hope for attention. However, in all runs, performance on examples with informative open-class lexical items is no higher than 60%, and the systematic errors remain. It is surprising that the Attention LSTM show this limitation, as it was designed to overcome the problem of representational bottlenecks by allowing interaction between the lexical items in the premise and hypothesis. However, Rocktäschel et al. (2015) anticipate this failure: "attention can fail, for example when two sentences and their words are entirely unrelated". Our data set pinpoint this weakness.

The CBoW model also separately bottle necks the premise and hypothesis, but directly averaging word vectors does not dilute the information from open class lexical items to the degree of the TreeNN and LSTM model. This explains its higher performance of $\approx 90\%$ on examples with informative open-class lexical items. However, this model can not make use of word order, resulting in a total inability to learn certain classes of examples and its overall performance being the lowest of all the models.

The CompTreeNN and CompTreeNTN avoid this bottleneck by design, since they mixes the premise and hypothesis via a strict word-by-word alignment. This makes their

learning task much simpler: it need only determine these local relations and propagate them. Observe in Figure 14 that during training both these models reach a stable local optimum before solving the task perfectly. Interestingly, the CompTreeNTN discovers the suboptimal solution found by the other four models, before ascending to 100%. The CompTreeNN model spends a sustained period of time hovering just below 80%. We conjecture that the other four models learn many local solutions in parallel resulting in a rapid descent to a local minimum that can be escaped only by learning to form an intermediate representation of two open class lexical items, while the CompTreeNN model more slowly develops a single global solution that transitions to a local minimum before achieving the global minimum.

6.3 The Natural Logic Generalization Task

Our central finding on the natural logic generalization task is that our four standard neural models fail the task completely, and while the CompTreeNN and CompTreeNTN achieve high performance, neither are able to solve the task perfectly. Surprisingly, of the four standard models, the continuous bag of words model achieves the highest average performance on the task, evidencing that the other more complex architectures do not contribute to the ability to perform natural logic reasoning.

Model	Train	Dev	Test
CBoW	88.04 ± 0.68	54.18 ± 0.17	53.99 ± 0.27
TreeNN	67.01 ± 12.71	54.01 ± 8.4	53.73 ± 8.36
LSTM encoder	98.43 ± 0.41	53.14 ± 2.45	52.51 ± 2.78
Attention LSTM	73.66 ± 9.97	47.52 ± 0.43	47.28 ± 0.95
CompTreeNN	99.65 ± 0.42	80.17 ± 7.53	80.21 ± 7.71
CompTreeNTN	99.92 ± 0.08	90.45 ± 2.48	90.32 ± 2.71

Table 2: Mean accuracy of 5 runs on the natural logic generalization task, with 95% confidence intervals. The training data was generated by running Algorithm 3 with $ratio = 0$ and the composition tree from Figure 12. These models are trained on the intermediate predictions described in Section 5.2 and shown in Figure 13

Table 2 summarizes the performance of each model on the natural logic generalization task when $ratio$ is set to 0 and Figure 15 shows how model performance evolves throughout training. Neither of our task specific models achieve perfect performance, however CompTreeNN outperforms our four standard neural models by $\approx 30\%$ and the CompTreeNTN improves on this by another $\approx 10\%$. The jump in performance when a feed forward composition function is replaced with a neural tensor network is similarly seen in the task of sentiment analysis, where it was found that neural tensor layers are more capable of modeling negation flipping positive sentiment and reducing the inten-

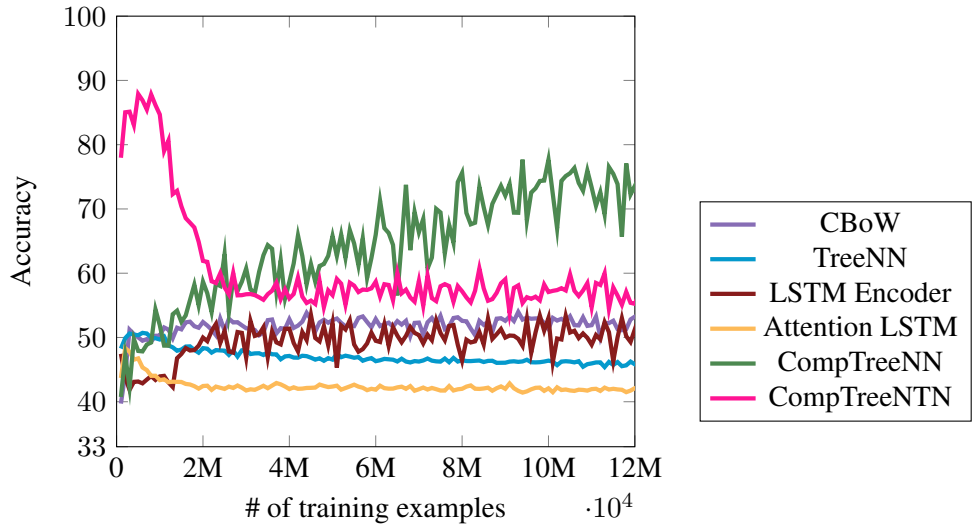

Figure 15: Model performance on the natural logic generalization task throughout training.

sity of negative sentiment as well as the conjunction *but* projecting the sentiment of the sentence following it (Socher et al., 2013). This increase in performance leads us to believe their may be some other composition function that solves this task perfectly. Both the CompTreeNN and CompTreeNTN have high 95% confidence intervals, indicating the models are volatile and sensitive to random initialization. This is reflected in the jagged, jumpy training lines in Figure 15, where, curiously, the CompTreeNN model is the only model that doesn't peak in the first two million training examples, showing steady improvement throughout training.

Surprisingly, our four standard models are largely similar in their performance, with CBoW, the simplest model, achieving the highest mean performance of the group. However, there is a notable caveat to this. Notice that the TreeNN has a large 95% interval. This is because on one of the five runs the TreeNN achieved a test accurracy of 65.76%, indicating this model may have more potential than the other three.

We look to Figure 16 to find the performance of models as the $ratio$ parameter is increased and observe the impact of being trained on the intermediate predictions described in Section 5.2 and shown in Figure 13. We can see that the CompTreeNN and CompTreeNTN models both rapidly ascend to perfect performance as $ratio$ increases and, as we would expect, these two models are significantly aided by being trained on intermediate predictions. The four standard neural models continue to have largely undifferentiated performance for all but the highest values of $ratio$, where CBoW falls behind the other three models. Oddly, the TreeNN and CBoW model have a dip in performance when $ratio$ is 0.5; perhaps they overfit this training dataset getting stuck at a suboptimal solution. On the easiest generalization tasks, being trained on intermediate predictions enable the LSTM encoder, Attentions LSTM, and TreeNN models to all achieve perfect performance and overcome the suboptimal solution they fall prey to on

the standard generalization task that is presented in Section 6.2. This makes sense, as these models are being trained explicitly on lexical relations.

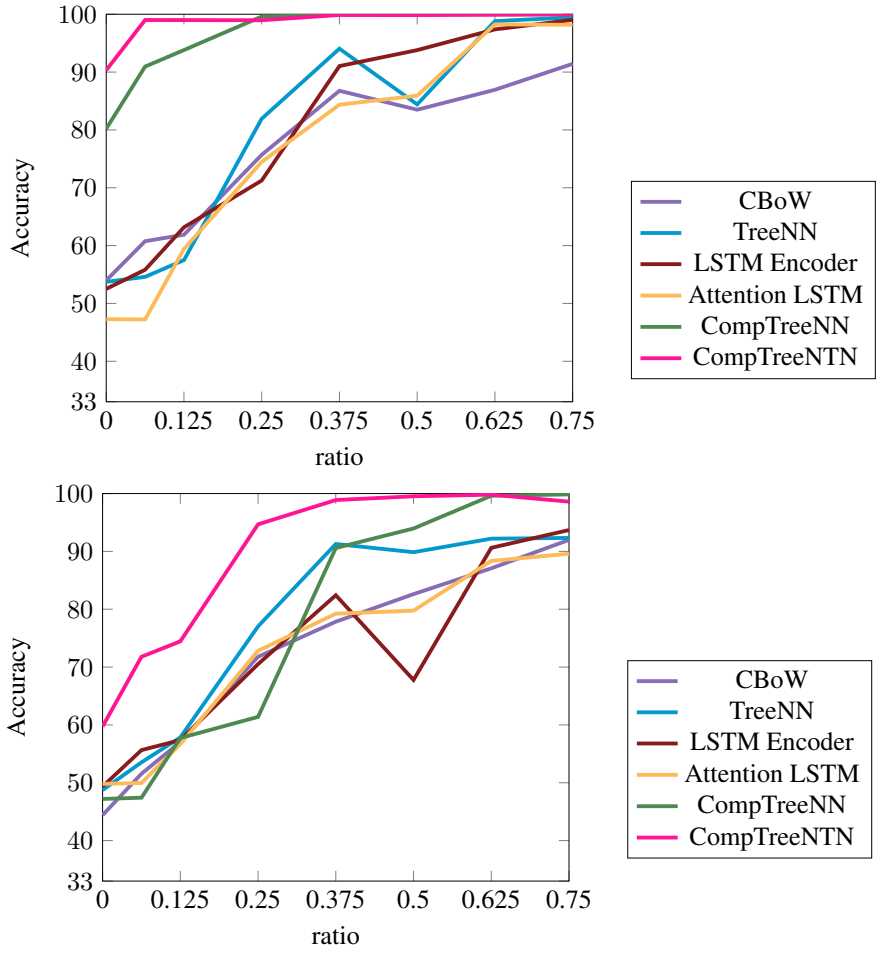

Figure 16: Mean accuracy of 5 runs on the natural logic generalization task over a gradation of difficulties. The training data was generated by running Algorithm 3 on the composition tree from Figure 12 with $ratio$ taking on values between 0 and 1. The top graph shows models that are trained on the intermediate predictions described in Section 5.2 and shown in Figure 13. The bottom graph shows models that are not trained on these intermediate predictions.

6.4 The Problem is Architecture

One might worry that these results represent a failure to optimize these models properly. These systematic errors are not an issue of low dimensionality; the trends by epoch and final results are virtually identical with 200-dimensional rather than 100-dimensions representations.

The reason these standard neural models fail to perform natural logic reasoning is their architecture. The CBoW, TreeNN, and LSTM Encoder models all separately bottle beck the premise and hypothesis sentences into two sentence vector embeddings, so the only place interactions between the two sentences can occur are in the two hidden

layers before the softmax layer. As we have argued thoroughly throughout this thesis, the essence of natural logic reasoning is recursive composition up a tree structure where the premise and hypothesis are composed jointly, and so these models fail miserably at the natural logic generalization task. The Attention LSTM model has an architecture that can align and combine lexical items from the premise and hypothesis, but it cannot perform this process recursively and also fails. The CompTreeNN and CompTreeNTN have this recursive tree structure encoded as hard alignments in their architecture, resulting in high performance. Perhaps in future work, a general purpose model will be developed that can learn to perform this recursive composition with out a hard encoded tree structure.

6.5 Comparative Analysis

Our approach to data set generation resembles that of Bowman et al. (2015b), who conduct a range of artificial language experiments. In one, they create corpora of single-quantified examples and show that simple tree-structured neural networks are able to learn the semantic relations between them when arbitrarily generating a training and testing split. Our work here differs in several ways. First, our work investigates whether learning models are able to learn a natural logic solution to inference, while their work does not isolate a particular solution. This difference in our research question results in Bowman et al. (2015b) only having the standard generalization task with arbitrary training and testing splits, while we have this standard generalization task as well as our more challenging natural logic generalization task. Second, the data we generate here is semantically more complex; we generate sentences with multiple quantifiers and make use of a larger lexicon. This results in standard neural models finding a suboptimal solution on our standard generalization task and these models perfectly solving their standard generalization task. Third, we test a wider range of learning models, including neural models with a tree structure that jointly composes the lexical items in the premise and hypothesis creating a single premise-hypothesis representation.

We feel the most significant of these differences is our generalization task, as we feel the arbitrary train test split used by Bowman et al. (2015b) is susceptible to the same simple learning model described in Section 5.2. This learning model takes a training example, computes the relations between open class lexical items and then memorizes the example. This model would perform perfectly on their data because every possible combination of quantifiers, negation, and lexical relations is essentially guaranteed[5] to be seen in training. This baseline model abstracts away the identity of open class lexical items, but is otherwise identical to the memorization model in Section 1.2. Bowman (2013) attempts to remedy this issue, by conducting generalization tests where certain reasoning patterns are held out during training and presented during testing, however there is no argument presented as to why we should expect these generalizations to be

[5]The dataset is randomly split between training and testing, but this will be true very close to always.

possible. In fact, their generalization tasks SUBCLASS-OUT and PAIR-OUT are unfair by our standards.

Veldhoen and Zuidema (2018) replicate the results of Bowman et al. (2015b) and perform a dimensionality reduction analysis on the sentence representations created by tree networks to investigate how these neural models achieve their solution, which is closely linked to the questions pursued here. They find evidence that neural networks are finding many partial solutions that generalize locally, but no complete solution that generalizes globally. We believe they discovered the insight that a model memorizing quantifiers, negation, and lexical relations can perform perfectly on this task. However, their work only considers the case where semantic relations between sentences are represented by a linear geometric relation, and it is unclear why relations are expected to be represented linearly. They also propose a test to determine whether a solution that generalizes globally is achieved, where networks are tested on examples of sentence equivalence resulting from De Morgan's laws. We believe this is a step in the right direction, as the baseline learning model would fail this generalization task. However, it is not clear whether any model can be expected to perform this generalization task, and our baseline model we develop in Sections 2 and 3 would not meet this high bar so we find this task to be unfair as well.

Finally, we wholeheartedly embrace their rejection of the question *Can learning models learn logical reasoning?* as binary and their belief that this question should be operationalized with a generalization task. We believe that there is no right or wrong generalization task, but that there should be theoretical motivation for why a generalization task is possible and a precise conception of what generalization capabilities a task requires. We must strike a balance between difficulty and possibility. By being explicit about our expectations, we can be explicit about what scientific question we are asking and what generalization capability is being tested. We believe that the random train test split task of Bowman et al. (2015b) tests only the very basic generalization skills of computing lexical relations and memorizing examples, but that the more difficult generalization tasks presented by Bowman (2013) and Veldhoen and Zuidema (2018) lack theoretical motivation for why they are possible and a precise conception of what capability is being tested for. In this work we present a generalization task that is firmly grounded in natural logic theory; our task is possible, but only if a model learns to perform natural logic reasoning.

7 Conclusion and Future Work

We began with a challenging question: *Can some NLI learning model perform natural logic reasoning?* To operationalize the question, we argued that the core of natural logic reasoning is the recursive composition of intermediate representations up a tree structure and presented a baseline learning model with an inductive bias to learn such recursive compositions. We then artificially produce a collection of generalization tasks

with varying difficulty that our presented baseline learning model achieves perfect accuracy on. Table 2 is our answer to the questions *Can the CBoW/TreeNN/LSTM encoder/Attention LSTM/CompTreeNN/CompTreeNTN model perform natural logic reasoning?* No model achieves the high bar of perfect accuracy, and so for no model is the answer a resounding yes. However, the CompTreeNN and CompTreeNTN model greatly out perform the standard neural models and we have high hopes that a simiarly structured model with a different composition function may be able to solve the task perfectly. The answer to this questions for the four standard models is a resounding no. In fact, the CBoW model has comparable performance with our other neural models, demonstrating that while these more complex models are better suited to learn inference on large naturalistic corpora, they are no better at learning natural logic reasoning than a simple neural baseline. This helps us diagnose the problem: the information bottleneck formed by learning separate premise and hypothesis representations. This bottle beck prevents the meaningful interactions between the premise and hypothesis that are at the core of natural logic reasoning. The CompTreeNN and CompTreeNTN are highly task-specific models, so their stand-out performance might be seen as more of a challenge than a victory for deep learning approaches to semantics.

We believe our approach to operationalizing our research question has several strengths. It forces us to refine our high level intuition into a inductive bias; having a baseline learning model with an explicit inductive bias makes our expectations transparent. It makes precise which training data which are fair for a baseline learning algorithm to achieve a perfect global solution, providing a theoretical justification of why a generalization task is fair. It provides us with an graded empirical answer to difficult abstract questions. In future work, we hope to pursue further questions about learning models using this framework.

References

Bowman, Sam (2016), *Modeling natural language semantics in learned representations*. Ph.D. thesis, Stanford University.

Bowman, Samuel R. (2013), "Can recursive neural tensor networks learn logical reasoning?" *CoRR*, abs/1312.6192, URL `http://arxiv.org/abs/1312.6192`.

Bowman, Samuel R., Gabor Angeli, Christopher Potts, and Christopher D. Manning (2015a), "A large annotated corpus for learning natural language inference." In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 632–642, Association for Computational Linguistics, Lisbon, Portugal, URL `http://aclweb.org/anthology/D15-1075`.

Bowman, Samuel R., Christopher Potts, and Christopher D. Manning (2015b), "Recursive neural networks can learn logical semantics." In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, 12–21,

Association for Computational Linguistics, URL http://www.aclweb.org/anthology/W15-4002.

Cybenko, George (1989), "Approximation by superpositions of a sigmoidal function." *Mathematics of Control, Signals and Systems*, 2, 303–314.

Dasgupta, Ishita, Demi Guo, Andreas Stuhlmüller, Samuel J. Gershman, and Noah D. Goodman (2018), "Evaluating compositionality in sentence embeddings." *CoRR*, abs/1802.04302, URL http://arxiv.org/abs/1802.04302.

Elman, Jeffrey L. (1990), "Finding structure in time." *Cognitive Science*, 14, 179–211.

Evans, Richard, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette (2018), "Can neural networks understand logical entailment?" *CoRR*, abs/1802.08535, URL http://arxiv.org/abs/1802.08535.

Glockner, Max, Vered Shwartz, and Yoav Goldberg (2018), "Breaking nli systems with sentences that require simple lexical inferences." In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 650–655, Association for Computational Linguistics, URL http://aclweb.org/anthology/P18-2103.

Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy (2015), "Explaining and harnessing adversarial examples." In *ICLR*.

Gururangan, Suchin, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A. Smith (2018), "Annotation artifacts in natural language inference data." In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 107–112, Association for Computational Linguistics, URL http://aclweb.org/anthology/N18-2017.

Hochreiter, Sepp and Jürgen Schmidhuber (1997), "Long short-term memory." *Neural Comput.*, 9, 1735–1780, URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Icard, Thomas F. and Lawrence S. Moss (2013), "Recent progress on monotonicity." *Linguistic Issues in Language Technology*, 9, 1–31.

Jia, Robin and Percy Liang (2017), "Adversarial examples for evaluating reading comprehension systems." In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2021–2031, Association for Computational Linguistics, Copenhagen, Denmark, URL https://www.aclweb.org/anthology/D17-1215.

Kingma, Diederik P. and Jimmy Ba (2014), "Adam: A method for stochastic optimization." *CoRR*, abs/1412.6980, URL http://arxiv.org/abs/1412.6980.

Lake, Brenden M. and Marco Baroni (2017), "Still not systematic after all these years: On the compositional skills of sequence-to-sequence recurrent networks." *CoRR*, abs/1711.00350, URL http://arxiv.org/abs/1711.00350.

MacCartney, Bill (2009), *Natural Language Inference*. Ph.D. thesis, Stanford University.

MacCartney, Bill and Christopher D. Manning (2007), "Natural logic for textual inference." In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, RTE '07, 193–200, Association for Computational Linguistics, Stroudsburg, PA, USA, URL http://dl.acm.org/citation.cfm?id=1654536.1654575.

MacCartney, Bill and Christopher D. Manning (2009), "An extended model of natural logic." In *Proceedings of the Eight International Conference on Computational Semantics*, 140–156, Association for Computational Linguistics, URL http://www.aclweb.org/anthology/W09-3714.

McCune, W. (2005–2010), "Prover9 and Mace4." http://www.cs.unm.edu/~mccune/prover9/.

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

Nie, Yixin, Yicheng Wang, and Mohit Bansal (2018), "Analyzing compositionality-sensitivity of NLI models." *CoRR*, abs/1811.07033.

Partee, Barbara (1984), "Compositionality." In *Varieties of Formal Semantics* (F. Landman and F. Veltman, eds.), Groningen-Amsterdam studies in semantics, 281–311, Walter de Gruyter GmbH & Company KG, URL https://books.google.com/books?id=mDttAAAAIAAJ.

Poliak, Adam, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme (2018), "Hypothesis only baselines in natural language inference." In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, 180–191, Association for Computational Linguistics, URL http://aclweb.org/anthology/S18-2023.

Rajpurkar, Pranav, Robin Jia, and Percy Liang (2018), "Know what you don't know: Unanswerable questions for squad." *CoRR*, abs/1806.03822.

Rajpurkar, Pranav, Jian Zhang, Konstantin Lopyrev, and Percy Liang (2016), "Squad: 100, 000+ questions for machine comprehension of text." *CoRR*, abs/1606.05250.

Rocktäschel, Tim, Edward Grefenstette, Karl Moritz Hermann, Tomás Kociský, and Phil Blunsom (2015), "Reasoning about entailment with neural attention." *CoRR*, abs/1509.06664, URL http://arxiv.org/abs/1509.06664.

Sánchez-Valencia, V.M.S. (1991), *Studies on Natural Logic and Categorial Grammar*. Universiteit van Amsterdam, URL https://books.google.com/books?id=qH0sAAAAIAAJ.

Socher, Richard, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning (2011a), "Parsing natural scenes and natural language with recursive neural networks." In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, 129–136, Omnipress, USA, URL http://dl.acm.org/citation.cfm?id=3104482.3104499.

Socher, Richard, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning (2011b), "Semi-supervised recursive autoencoders for predicting sentiment distributions." In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, 151–161, Association for Computational Linguistics, Stroudsburg, PA, USA, URL http://dl.acm.org/citation.cfm?id=2145432.2145450.

Socher, Richard, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts (2013), "Recursive deep models for semantic compositionality over a sentiment treebank." In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 1631–1642, Association for Computational Linguistics, Seattle, Washington, USA, URL https://www.aclweb.org/anthology/D13-1170.

Szegedy, Christian, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus (2014), "Intriguing properties of neural networks." In *International Conference on Learning Representations*, URL http://arxiv.org/abs/1312.6199.

Tsuchiya, Masatoshi (2018), "Performance impact caused by hidden bias of training data for recognizing textual entailment." *CoRR*, abs/1804.08117, URL http://arxiv.org/abs/1804.08117.

van Benthem, Johan (2008), "A brief history of natural logic."

Veldhoen, Sara and Willem Zuidema (2018), "Can neural networks learn logical reasoning?" In *Proceedings of the Conference on Logic and Machine Learning in Natural Language (LaML 2017)*, 35–41, Association for Computational Linguistics.

Williams, Adina, Nikita Nangia, and Samuel R. Bowman (2017), "A broad-coverage challenge corpus for sentence understanding through inference." *CoRR*, abs/1704.05426, URL http://arxiv.org/abs/1704.05426.