University of Toronto, Department of Computer Science
CSC 2501F—Computational Linguistics, Fall 2025

Reading assignment 4

Due date: Electronically by 12:10, Friday 24th October, 2025.
Late write-ups will not be accepted without documentation of a medical or other emergency.
This assignment is worth 5% of your final grade.

What to read

1. Z. Chen and Q. Gao, Monotonicity Marking from Universal Dependency Trees. *Proc. IWCS 2021*, pp. 121–131.

2. Z. Chen, Q. Gao and L.S. Moss, NeuralLog: Natural Language Inference with Joint Neural and Logical Reasoning. *Proc. *SEM 2021*, pp. 78–88.

What to write

Write a *brief* summary of the papers' argumentation, with a critical assessment of their merits.

Some points to consider:

- Neither of these papers manages to define monotonicity. What is it?
- How is the choice of NeuralLog's architecture justified? Do you find the justification convincing?

General requirements: Your write-up should be typed, using 12-point font and 1.5-line spacing; it should fit on one to two sides of a sheet of paper.
Submit electronically using Quercus.

Monotonicity Marking from Universal Dependency Trees

Zeming Chen **Qiyue Gao**
Department of Computer Science and Software Engineering,
Rose-Hulman Institute of Technology
{chenz16, gaoq}@rose-hulman.edu

Abstract

Dependency parsing is a tool widely used in the field of Natural Language Processing and computational linguistics. However, there is hardly any work that connects dependency parsing to monotonicity, which is an essential part of logic and linguistic semantics. In this paper, we present a system that automatically annotates monotonicity information based on Universal Dependency parse trees. Our system utilizes surface-level monotonicity facts about quantifiers, lexical items, and token-level polarity information. We compared our system's performance with existing systems in the literature, including NatLog and ccg2mono, on a small evaluation dataset. Results show that our system outperforms NatLog and ccg2mono.

1 Introduction

The number of computational approaches for Natural Language Inference (NLI) has rapidly grown in recent years. Most of the approaches can be categorized as (1) Systems that translate sentences into first-order logic expressions and then apply theorem proving (Blackburn and Bos, 2005). (2) Systems that use blackbox neural network approaches to learn the inference (Devlin et al., 2019; Liu et al., 2019). (3) Systems that apply natural logic as a tool to make inferences (MacCartney and Manning, 2009; Hu et al., 2020; Angeli et al., 2016; Abzianidze, 2017). Compared to neural network approaches, systems that apply natural logic are more robust, formally more precise, and more explainable. Several systems contributed to the third category (MacCartney and Manning, 2009; Hu et al., 2020; Angeli et al., 2016) to solve the NLI task using monotonicity reasoning, a type of logical inference that is based on word replacement. Below is an example of monotonicity reasoning:

1. (a) **All** <u>students</u>↓ carry a <u>MacBook</u>↑.
   (b) All students carry a <u>laptop</u>.
   (c) All <u>new students</u> carry a MacBook.
2. (a) **Not all** <u>new students</u>↑ carry a laptop.
   (b) Not all <u>students</u> carry a laptop.

As the example shows, the word replacement is based on the polarity mark (arrow) on each word. A monotone polarity (↑) allows an inference from (1a) to (1b), where a more general concept *laptop* replaces the more specific concept *MacBook*. An antitone polarity (↓) allows an inference from (1a) to (1c), where a more specific concept *new students* replaces the more general concept *students*. The direction of the polarity marks can be reversed by adding a downward entailment operator like *Not* which allows an inference from (2a) to (2b). Thus, successful word placement relies on accurate polarity marks. To obtain the polarity mark for each word, an automatic polarity marking system is required to annotate a sentence by placing polarity mark on each word. This is formally called the polarization process. Polarity markings support monotonicity reasoning, and thus are used by systems for Natural Language Inference and data augmentations for language models. (MacCartney and Manning, 2009; Hu et al., 2020; Angeli et al., 2016).

In this paper, we introduce a novel automatic polarity marking system that annotates monotonicity information by applying a polarity algorithm on a universal dependency parse tree. Our system is inspired by ccg2mono, an automatic polarity marking system (Hu and Moss, 2018) used by Hu et al. (2020). In contrast to ccg2mono, which derives monotonicity information from CCG (Lewis and Steedman, 2014) parse trees, our system's polarization algorithm derives monotonicity information using Universal Dependency (Nivre et al., 2016) parse trees. There are several advantages of using UD parsing for polarity marking rather than

CCG parsing. First, UD parsing is more accurate since the amount of training data for UD parsing is larger than those of CCG parsing. The high accuracy of UD parsing should lead to more accurate polarity annotation. Second, UD parsing works for more types of text. Overall, our system opens up a new framework for performing inference, semantics, and automated reasoning over UD representations. We will introduce the polarization algorithm's general steps, a set of rules we used to mark polarity on dependency parse trees, and comparisons between our system and some existing polarity marking tools, including NatLog (MacCartney and Manning, 2009; Angeli et al., 2016) and ccg2mono. Our evaluation focuses on a small dataset used to evaluate ccg2mono (Hu and Moss, 2020). Our system outperforms NatLog and ccg2mono. In particular, our system achieves the highest annotation accuracy on both the token level and the sentence level.

2 Related Work

Universal Dependencies (UD) (Nivre et al., 2016) was first designed to handle language tasks for many different languages. The syntactic annotation in UD mostly relies on dependency relations. Words enter into dependency relations, and that is what UD tries to capture. There are 40 grammatical dependency relations between words, such as nominal subject (**nsubj**), relative clause modifier (**acl:relcl**), and determiner (**det**). A dependency relation connects a headword to a modifier. For example, in the dependency parse tree for *All dogs eat food* (figure 1), the dependency relation **nsubj** connects the modifier *dogs* and the headword *eat*. The system presented in this paper utilizes Universal Dependencies to obtain a dependency parse tree from a sentence. We will explain the details of the parsing process in the implementation section.

There are two relevant systems of prior work: (1) The NatLog (MacCartney and Manning, 2009; Angeli et al., 2016) system included in the Stanford CoreNLP library (Manning et al., 2014); (2) The ccg2mono system (Hu and Moss, 2018). The NatLog system is a natural language inference system, a part of the Stanford CoreNLP Library. NatLog marks polarity to each sentence by applying a pattern-based polarization algorithm to the dependency parse tree generated by the Stanford dependency parser. A list of downward-monotone and non-monotone expressions are defined along

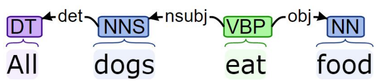


Figure 1: A dependency parse tree for "All dogs eat food."

with an arity and a Tregex pattern for the system to identify if an expression occurred.

The ccg2mono system is a polarity marking tool that annotates a sentence by polarizing a CCG parse tree. The polarization algorithm of ccg2mono is based on van Benthem (1986)'s work and Moss (2012)'s continuation on the soundness of internalized polarity marking. The system uses a marked/order-enriched lexicon and can handle application rules, type-raising, and composition in CCG. The main polarization contains two steps: mark and polarize. For the mark step, the system puts markings on each node in the parse tree from leaf to root. For the polarize step, the system generates polarities to each node from root to leaf. Compared to NatLog, an advantage of ccg2mono is that it polarizes on both the word-level and the constituent level.

3 Universal Dependency to Polarity

3.1 Overview

Our system's polarization algorithm contains three steps: (1) Universal Dependency Parsing, which transforms a sentence to a UD parse tree, (2) Binarization, which converts a UD parse tree to a binary UD parse tree, and (3) Polarization, which places polarity marks on each node in a binary UD parse tree.

3.2 Binarization

To preprocess the dependency parse graph, we designed a binarization algorithm that can map each dependency tree to an s-expression (Reddy et al., 2016). Formally, an s-expression has the form (exp1 exp2 exp3), where exp1 is a dependency label, and both exp2 and exp3 are either (1) a word such as *eat*; or (2) an s-expression such as (**det** *all dogs*). The process of mapping a dependency tree to an s-expression is called binarization. Our system represents an s-expression as a binary tree. A binary tree has a root node, a left child node, and a right child node. In representing an s-expression, the root node can either be a single word or a dependency label. Both the left and the right child nodes can either be a sub-binary-tree, or null. The

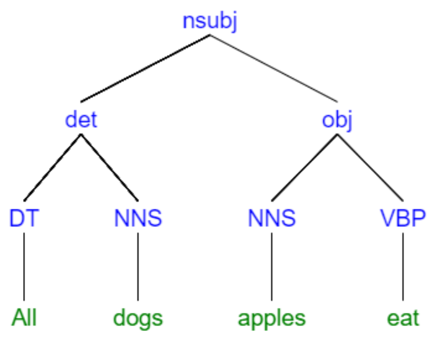


Figure 2: A binarized dependency parse tree for "All dogs eat apples."

system always puts the modifiers on the left and the headwords on the right. For example, the sentence *All dogs eat apples* has an s-expression

$$(\textbf{nsubj}\ (\textbf{det}\ \textit{All dogs})\ (\textbf{obj}\ \textit{eat apples}))$$

and can be shown as a binary tree in figure 2. In the left sub-tree (*All dogs*), the dependency label **det** will be the root node, the modifier *all* will be the left child, and the headword *dogs* will be the right child.

Our binarization algorithm employs a dependency relation hierarchy to impose a strict traversal order from the root relation to each leaf word. The hierarchy allows for an ordering on the different modifier words. For example, in the binary dependency parse tree (**nsubj** (**det** All dogs) (**obj** eat apples)), the nominal subject (**nsubj**) goes above the determiner (**det**) in the tree because **det** is lower than **nsubj** in the hierarchy. We originally used the binarization hierarchy from Reddy et al. (2016)'s work, and later extended it with additional dependency relations such as oblique nominal (**obl**) and expletive (**expl**). Table 1 shows the complete hierarchy where the level-id indicates a relation's level in the hierarchy. The smaller a relation's level-id is, the higher that relation is in the hierarchy.

relation	level-id	relation	level-id
conj-sent	0	obl:tmod	50
advcl-sent	1	obl:npmod	50
advmod-sent	2	cop	50
case	10	det	55
mark	10	det:predet	55
expl	10	acl	60
discourse	10	acl:relcl	60
nsubj	20	appos	60
csubj	20	conj	60
nsubj:pass	20	conj-np	60
conj-vp	25	conj-adj	60
ccomp	30	obj	60
advcl	30	iobj	60
advmod	30	cc	70
nmod	30	amod	75
nmod:tmod	30	nummod	75
nmod:npmod	30	compound	80
nmod:poss	30	compound:prt	80
xcomp	40	fixed	80
aux	40	conj-n	90
aux:pass	40	conj-vb	90
obl	50	flat	100

Table 1: Universal Dependency relation hierarchy. The smaller a relation's level-id is, the higher that relation is in the hierarchy.

Algorithm 1 Binarization

```
1:  root ← GET_ROOT_NODE($\mathcal{G}$)
2:  $\mathcal{T}$ ← COMPOSE(root)
3:  return $\mathcal{T}$
4:
5:  function COMPOSE(node):
6:      $\mathcal{C}$ ← GET_CHILDREN(node)
7:      $\mathcal{C}_s$ ← SORT_BY_PRIORITY($\mathcal{C}$)
8:      if | $\mathcal{C}_s$ | == 0 then
9:          $\mathcal{B}$ ← BINARYDEPENDENCYTREE()
10:         $\mathcal{B}$.val = node
11:         return $\mathcal{B}$
12:     else
13:         top ← $\mathcal{C}$.pop()
14:         $\mathcal{B}$ ← BINARYDEPENDENCYTREE()
15:         $\mathcal{B}$.val = RELATE(top, node)
16:         $\mathcal{B}$.left = COMPOSE(top)
17:         $\mathcal{B}$.right = COMPOSE(node)
18:         return $\mathcal{B}$
19:     end if
20: end function
```

3.3 Polarization

The polarization algorithm places polarities on each node of a UD parse tree based on a lexicon of polarization rules for each dependency relation and some special words. Our polarization algorithm is similar to the algorithms surveyed by Lavalle-Martínez et al. (2018). Like the algorithm of Sanchez (1991), our algorithm computes polarity from leaves to root. One difference our algorithm has is that often, the algorithm computes polarity following a left-to-right inorder traversal (left⟶root⟶right) or a right-to-left inorder traversal (right⟶root⟶left) in additional to the top-down traversal. In our algorithm, each node's polarity depends both on its parent node and its sibling node (left side or right side), which is different from algorithms in Lavalle-Martínez et al. (2018)'s paper. Our algorithm is deterministic, and thus never fails.

The polarization algorithm takes in a binarized UD parse tree $\mathcal{T}$ and a set of polarization rules, both dependency-relation-level ($\mathcal{L}$) and word-level ($\mathcal{W}$). The algorithm outputs a polarized UD parse tree $\mathcal{T}^*$ such that (1) each node is marked with

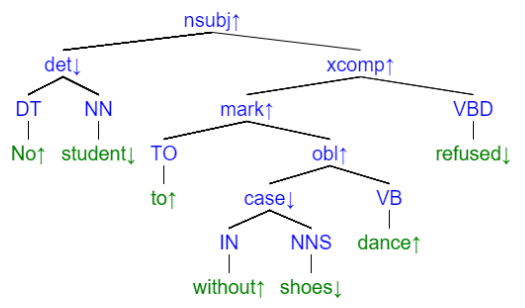


Figure 3: Visualization of a polarized binary dependency parse tree for a triple negation sentence *No student refused to dance without shoes.*

a polarity of either monotone (↑), antitone (↓), or no monotonicity information (=), (2) both $\mathcal{T}$ and $\mathcal{T}^*$ have the same universal dependency structure except the polarity marks. Figure 3 shows a visualization of the binary dependency parse tree after polarization completes. The general steps of the polarization start from the root node of the binary parse tree. The system will get the corresponding polarization rule from the lexicon according to the root node's dependency relation. In each polarization rule, the system applies the polarization rule and then continues the above steps recursively down the left sub-tree and the right sub-tree. Each polarization rule is composed from a set of basic building blocks include rules for negation, equalization, and monotonicity generation. When the recursion reaches a leaf node, which is an individual word in a sentence, a set of word-based polarization rules will be retrieved from the lexicon, and the system polarizes the nodes according to the rule corresponding to a particular word. More details about word-based polarization rules will be covered in section 3.4.2, Polarity Generation. An overview of the polarization algorithm and a general scheme of the implementation for dependency-level polarization rules are shown in Algorithm 2.

3.4 Polarization Rules

Our polarization algorithm contains a lexicon of polarization rules corresponding to each dependency relation. Each polarization rule is composed from a set of building blocks divided into three categories: negation rules, equalization rules, and monotonicity generation rules. The generation rules will generate three types of monotonicity: monotone (↑), antitone (↓), and no monotonicity information (=) either by initialization or based on the words.

Algorithm 2 Polarization

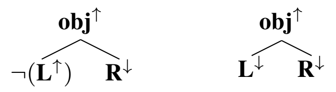
Input: $\mathcal{T}$: binary dependency tree
$\mathcal{L}$: dependency-level polarization rules
$\mathcal{W}$: word-level polarization rules
Output: $\mathcal{T}^*$: polarized binary dependency tree

```
1:  if $\mathcal{T}$.is_tree then
2:      relation ← $\mathcal{T}$.val
3:      POLARIZATION_RULE(.) ← $\mathcal{L}$[relation]
4:      POLARIZATION_RULE($\mathcal{T}$)
5:  end if
6:
7:  ▷ General scheme of a polarization rule's implementation
    for a dependency relation
8:  function POLARIZATION_RULE($\mathcal{T}$)
9:      ▷ Initialize or inherit polarities
10:     if $\mathcal{T}$.mark ≠ NULL then
11:         $\mathcal{T}$.right.mark = $\mathcal{T}$.mark
12:         $\mathcal{T}$.left.mark = $\mathcal{T}$.mark
13:     else
14:         $\mathcal{T}$.right.mark = ↑
15:         $\mathcal{T}$.left.mark = ↑
16:     end if
17:
18:     ▷ Polarize sub-trees
19:     POLARIZATION($\mathcal{T}$.left)
20:     POLARIZATION($\mathcal{T}$.right)
21:     ▷ Or, for relations like nsubj:
22:     ▷ POLARIZATION($\mathcal{T}$.right)
23:     ▷ POLARIZATION($\mathcal{T}$.left)
24:
25:     ▷ Apply negation and equalization rules
26:     if NEGATE is applicable then
27:         NEGATE($\mathcal{T}$)
28:     end if
29:     if EQUALIZE is applicable then
30:         EQUALIZE($\mathcal{T}$)
31:     end if
32:
33:     ▷ Apply word-level rules
34:     if not $\mathcal{T}$.is_tree and $\mathcal{T}$.val ∈ $\mathcal{W}$.keys then
35:         WORD_RULE(.) ← $\mathcal{W}$[$\mathcal{T}$.val]
36:         WORD_RULE($\mathcal{T}$)
37:     end if
38: end function
```

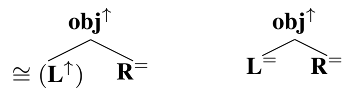
3.4.1 Building Blocks

Negation and Equalization The negation rule and the equalization rule are used by several core dependency relations such as **nmod**, **obj**, and **acl:recl**. Both negation and equalization have two ways of application: backward or top-down. A backward negation rule is triggered by a downward polarity (↓) on the right node of the tree (marked below as R), flipping every node's polarity under the left node (marked below as L). Similarly, a backward equalization rule is triggered by a no monotonicity information polarity (=) on the tree's right node, and it marks every node under the left node as =. Examples for trees before and after applying a backward and forward negation and equalization are shown as follows:

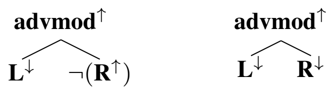
- Backward Negation:



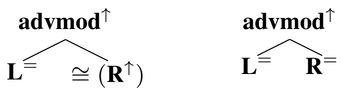
- Backward Equalization:



- Forward Negation:

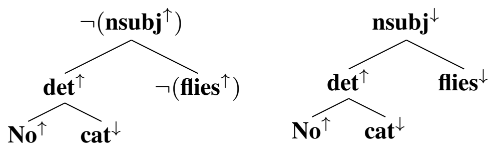


- Forward Equalization:



where ¬ means negation and ≅ means equalization.

A top-down negation is used by the polarization rule like determiner (**det**) and adverbial modifier (**advmod**). It starts at the parent node of the current tree, and flips the arrow on each node under that parent node excluding the current tree. This top-down negation is used by **det**, **case**, and **advmod** when a negation operators like *no*, *not*, or *at-most* appears. Below is an example of a tree before and after applying the top-down negation:



Polarity Generation The polarity is generated by words. During the polarization, the polarity can change based on a particular word that can promote the polarity governing the part of the sentence to which it belongs. These words include quantifiers and verbs. For the monotonicity from quantifiers, we follow the monotonicity profiles listed in the work done by Icard III and Moss (2014) on monotonicity, which built on van Benthem (1986). Additionally, to extend to more quantifiers, we observed polarization results generated by ccg2mono. Overall, we categorized the quantifiers as follows:

- Universal Type

  Every ↓ ↑ Each ↓ ↑ All ↓ ↑

- Negation Type

  No ↓ ↓ Less than ↓ ↓ At most ↓ ↓

- Exact Type

  Exactly n = = The = ↑ This = ↑

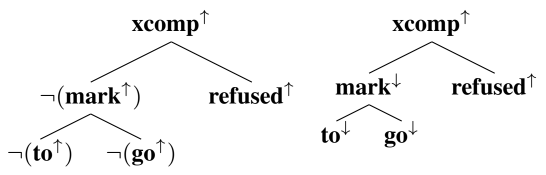
- Existential Type

  Some ↑ ↑ Several ↑ ↑ A, An ↑ ↑

- Other Type

  Most = ↑ Few = ↓

Where the first mark is the monotonicity for the first argument after the quantifier and the second mark is the monotonicity for the second argument after the quantifier. For verbs, there are upward entailment operators and downward entailment operators. Verbs that are downward entailment operators, such as *refuse*, promote an antitone polarity, which will negate its dependents. For example, for the phrase *refused to go*, *refused* will promote an antitone polarity, which negates *to dance*:



In addition to quantifiers and verbs, some other words also change the monotonicity of a sentence. For example, words like *not*, *none*, and *nobody* promote an antitone polarity. Our system also handles material implications with the form *if x then y*. Based on Moss (2012), the word *if* promotes an antitone polarity in the antecedent and positive polarity in the consequent. For background on monotonicity and semantics, see van Benthem (1986), Keenan and Faltz (1984), and also Karttunen (2012).

3.4.2 Dependency Relation Rules

Each dependency relation has a corresponding polarization rule. All the rules start with initializing the starting node as upward monotone polarity (↑). Alternatively, if the starting node has a polarity marked, each child node will inherit the root node's polarity. Each rule's core part is a combination of the default rules and monotonicity generation rules. In this section, we will briefly show three major types of dependency relation rules in the polarization algorithm. The relative clause modifier relation will represent rules for modifier relations. The determiner relation rule will represent rules containing monotonicity generation rules. The Object and open clausal complement rule will represent rules containing word-level polarization rules.

Algorithm 3 Polarize_acl:relcl

```
Input: $\mathcal{T}$: binary dependency sub-tree
Output: $\mathcal{T}^*$: polarized binary dependency sub-tree

1:  if $\mathcal{T}$.mark $\neq$ NULL then
2:      $\mathcal{T}$.right.mark = $\mathcal{T}$.mark
3:  else
4:      $\mathcal{T}$.right.mark = $\uparrow$
5:  end if
6:  $\mathcal{T}$.left.mark = $\uparrow$
7:
8:  POLARIZE($\mathcal{T}$.right)
9:  POLARIZE($\mathcal{T}$.left)
10:
11: if $\mathcal{T}$.right.mark == $\downarrow$ then
12:     NEGATE($\mathcal{T}$.left)
13: else if $\mathcal{T}$.right.mark == $=$ then
14:     EQUALIZE($\mathcal{T}$.left)
15: end if
```

Algorithm 4 Polarize_det

```
Input: $\mathcal{T}$: binary dependency sub-tree
       $\mathcal{D}$: determiner mark dictionary
Output: $\mathcal{T}^*$: polarized binary dependency sub-tree

1:  det_type $\leftarrow$ GET_DET_TYPE($\mathcal{T}$.left)
2:  if $\mathcal{T}$.mark $\neq$ NULL then
3:      $\mathcal{T}$.left.mark = $\mathcal{T}$.mark
4:  else
5:      $\mathcal{T}$.left.mark = $\uparrow$
6:  end if
7:
8:  $\mathcal{T}$.right.mark = $\mathcal{D}$[det_type]
9:  POLARIZE($\mathcal{T}$.right)
10:
11: if det_type == negation then
12:     NEGATE($\mathcal{T}$.parent)
13: end if
```

Relative Clause Modifier For the relative clause modifier relation (**acl:relcl**), the relative clause depends on the noun it modifies. First, the polarization will first be performed on both the left and right nodes, and then, depending on the polarity of the right node, a negation or an equalization rule will be applied. The algorithm first applies a top-down inheritance if the root already has its polarity marked; otherwise, it initializes the left and right nodes as monotone. The algorithm polarizes both the left and right nodes. Next, the algorithm checks the right node's polarity. If the right node is marked as antitone, a backward negation is applied. Alternatively, if the right node is marked as no monotonicity information, a backward equalization is applied. During the experiments, we noticed that if the root node is marked antitone, and the left node inherits that, a negation later will cause a double negation, producing incorrect polarity marks. To avoid this double negation, we exclude the left node from the top-down inheritance rule by initializing the left node directly with a monotone mark. The rule for **acl:relcl** also applies to the adverbial clause modifier (**advcl**) and the clausal modifier of noun (**acl**). An overview of the algorithm is shown in Algorithm 3.

Determiner For the determiner relation (**det**), each different determiner can assign a new monotonicity to the noun it modifies. First, the algorithm performs a top-down inheritance on the left node if the root already has polarity marked. Next, the algorithm assigns the polarity for the noun depending on the determiner's type. For example, if the determiner is a universal quantifier, an antitone polarity is assigned to the right node. For negation quantifiers like *no*, its right node also receives an antitone polarity. Thus, a top-down negation is applied at the determiner relation tree's parent. Algorithm 4 shows an overview of the algorithm.

Object and Open Clausal Complement For the object relation (**obj**) and the open clausal complement relation **xcomp**, both the verb and the noun would inherit the monotonicity from the parent in the majority of cases. The inheritance procedure is the same as the one used in **acl:relcl**'s rule. Similarly, after the inheritance, the rule will polarize both the right sub-tree and the left sub-tree. Differently, since **obj** and **xcomp** both have a verb under the relation, they require a word-level polarization rule that will check the verb determine if the verb is a downward entailment operator, which prompts an antitone monotonicity. The algorithm takes in a dictionary that contains a list of verbs and their

Algorithm 5 Polarize_obj

```
Input: $\mathcal{T}$: binary dependency sub-tree
Output: $\mathcal{T}^*$: polarized binary dependency sub-tree

1:  if $\mathcal{T}$.mark $\neq$ NULL then
2:      $\mathcal{T}$.right.mark = $\mathcal{T}$.mark
3:  else
4:      $\mathcal{T}$.right.mark = $\uparrow$
5:  end if
6:  $\mathcal{T}$.left.mark = $\uparrow$
7:
8:  POLARIZE($\mathcal{T}$.right)
9:  POLARIZE($\mathcal{T}$.left)
10:
11: ▷ Word-level polarization rule for downward entailment
    operators
12: if IS_DOWNWARD_OPERATOR($\mathcal{T}$.right.mark) then
13:     NEGATE($\mathcal{T}$.left)
14: end if
15:
```

implicatives. The dictionary is generated from the implicative verb dataset made by Ross and Pavlick (2019). If a verb is a downward entailment operator, which has a negative implicative, the rule will apply a negation rule on the left sub-tree to flip each node's arrow in the left sub-tree. An overview of the algorithm is shown in Algorithm 5.

4 Comparison to Existing Systems

We conducted several preliminary comparisons to two existing systems. First, we compared to NatLog's monotonicity annotator. Natlog's annotator also uses dependency parsing. The polarization algorithm does pattern-based matching for finding occurrences of downward monotonicity information, and the algorithm only polarizes on word-level. In contrast, our system uses a tree-based polarization algorithm that polarizes both on word-level polarities and constituent level polarities. Our intuition is that the Tregex patterns used in NatLog is not as common or as easily understandable as the binary tree structure, which is a classic data structure wildly used in the filed of computer science.

According to the comparison on a list of sentences, NatLog's annotator does not perform as well as our system. For example, for a phrase *the rabbit*, *rabbit* should have a polarity with no monotonicity information (=). However, NatLog marks *rabbit* as a monotone polarity (↑). NatLog also incorrectly polarizes sentences containing multiple negations. For example, for a triple negation sentence, *No newspapers did not report no bad news*, NatLog gives: *No*$^{\uparrow}$ *newspapers*$^{\downarrow}$ *did*$^{\downarrow}$ *not*$^{\downarrow}$ *report*$^{\uparrow}$ *no*$^{\uparrow}$ *bad*$^{\uparrow}$ *news*$^{\uparrow}$. This result has incorrect polarity marks on multiple words, where *report*, *bad*, *news* should be ↓, and *no* should be ↑. Both of the scenarios above can be handled correctly by our system.

Comparing to ccg2mono, our algorithm shares some similarities to its polarization algorithm. Both of the systems polarize on a tree structure and rely on a lexicon of rules, and they both polarize on the word-level and the constituent level. One difference is that ccg2mono's algorithm contains two steps, the first step puts markings on each node, and the second step puts polarities on each node. Our system does not require the step of adding markings and only contains the step of adding polarities on each node.

Our system has multiple advantages over ccg2mono. For parsing, our system uses UD parsing, which is more accurate than CCG parsing used by ccg2mono due to a large amount of training data. Also, our system covers more types of text than ccg2mono because UD parsing works for a variety of text genres such as web texts, emails, reviews, and even informal texts like Twitter tweets. (Silveira et al., 2014; Zeldes, 2017; Liu et al., 2018). Our system can also work for more languages than ccg2mono since UD parsing supports more languages than CCG parsing.

Overall, our system delivers more accurate polarization than ccg2mono. Many times the CCG parser makes mistakes and leads to polarization mistakes later on. For example, in the annotation *The*$^{\downarrow}$ *market*$^{\downarrow}$ *is*$^{\downarrow}$ *not*$^{\downarrow}$ *impossible*$^{\downarrow}$ *to*$^{\downarrow}$ *navigate*$^{\downarrow}$, ccg2mono incorrectly marks every word as ↓. Our system, on the other hand, uses UD parsing which has higher parsing accuracy than CCG parsing, and thus leads to fewer polarization mistakes compared to ccg2mono. For the expression above, our system correctly polarizes it as *The*$^{\uparrow}$ *market*$^{=}$ *is*$^{\uparrow}$ *not*$^{\uparrow}$ *impossible*$^{\downarrow}$ *to*$^{\uparrow}$ *navigate*$^{\uparrow}$.

Our system also handles multi-word quantifiers better than ccg2mono. For example, for a multi-word quantifier expression like *all of the dogs*, ccg2mono mistakenly marks *dogs* as =. Our system, however, can correctly mark the expression: *all*$^{\uparrow}$ *of*$^{\uparrow}$ *the*$^{\uparrow}$ *dogs*$^{\downarrow}$.

Moreover, the core of ccg2mono does not include aspects of verbal semantics of downward-entailing operators like *forgot* and *regret* (Moss and Hu, 2020). For example ccg2mono's polarization for *Every*$^{\uparrow}$ *member*$^{\downarrow}$ *forgot*$^{\uparrow}$ *to*$^{\uparrow}$ *attend*$^{\uparrow}$ *the*$^{\uparrow}$ *meeting*$^{=}$ is not correct because it fails to flip the polarity of *to attend the*. In contrast, our system produces a correct result: *Every*$^{\uparrow}$ *member*$^{\downarrow}$ *forgot*$^{\uparrow}$ *to*$^{\downarrow}$ *attend*$^{\downarrow}$ *the*$^{\downarrow}$ *meeting*$^{=}$.

All three systems have difficulty polarizing sentences containing numbers. A scalar number **n**'s monotonicity information is hard to determine because it can presenter different contexts: a single number **n**, without additional quantifiers or adjectives, can either mean *at least* **n**, *at most* **n**, *exactly* **n**, and *around* **n**. These contexts are syntactically hard to identify for a dependency parser or a CCG parser because it would require pragmatics and some background knowledge which the parsers do not have. For example, in the sentence *A dog ate 2 rotten biscuits*, the gold label for *2* is = which indicates that the context is "exactly 2". However, our system marks this as "↓ since it considers the

sentence	type
More$^{\uparrow}$ dogs$^{\uparrow}$ than$^{\uparrow}$ cats$^{\downarrow}$ sit$^{=}$	comparative
Less$^{\uparrow}$ than$^{\uparrow}$ 5$^{\uparrow}$ people$^{\downarrow}$ ran$^{\downarrow}$	less-than
A$^{\uparrow}$ dog$^{\uparrow}$ who$^{\uparrow}$ ate$^{\uparrow}$ two$^{=}$ rotten$^{\uparrow}$ biscuits$^{\uparrow}$ was$^{\uparrow}$ sick$^{\uparrow}$ for$^{\uparrow}$ three$^{\downarrow}$ days$^{\downarrow}$	number
Every$^{\uparrow}$ dog$^{\downarrow}$ who$^{\downarrow}$ likes$^{\downarrow}$ most$^{\downarrow}$ cats$^{=}$ was$^{\uparrow}$ chased$^{\uparrow}$ by$^{\uparrow}$ at$^{\uparrow}$ least$^{\uparrow}$ two$^{\downarrow}$ of$^{\uparrow}$ them$^{\uparrow}$	every:most:at-least
Even$^{\uparrow}$ if$^{\uparrow}$ you$^{\downarrow}$ are$^{\downarrow}$ addicted$^{\downarrow}$ to$^{\downarrow}$ cigarettes$^{\downarrow}$ you$^{\uparrow}$ can$^{\uparrow}$ smoke$^{\uparrow}$two$^{\downarrow}$ a$^{\uparrow}$ day$^{\uparrow}$	conditional:number

Table 2: Example sentences in Hu and Moss (2020)'s evaluation dataset

context as "at least 2", which is different from the gold label.

5 Experiment

Dataset We obtained the small evaluation dataset used in the evaluation of ccg2mono (Hu and Moss, 2020) from its authors. The dataset contains 56 hand-crafted English sentences, each with manually annotated monotonicity information. The sentences cover a wide range of linguistic phenomena such as quantifiers, conditionals, conjunctions, and disjunctions. The dataset also contains hard sentences involving scalar numbers. Some example sentences from the dataset are shown in Table 2.

Dependency Parser In order to obtain a universal dependency parse tree from a sentence, we utilize a parser from Stanza (Qi et al., 2020), a Python natural language analysis package made by Stanford. The neural pipeline in Stanza allow us to use pretrained neural parsing models to generate universal dependency parse trees. To achieve optimal performance, we trained two neural parsing models: one parsing model trained on Universal Dependency English GUM corpus (Zeldes, 2017). The pretrained parsing model achieved 90.0 LAS (Zeman et al., 2018) evaluation score on the testing data.

Experiment Setup We evaluated the polarization accuracy on both the token level and the sentence level, in a similar fashion to the evaluation for part-of-speech tagging (Manning, 2011). For both levels of accuracy, we conducted one evaluation on all tokens (*acc(all-tokens)* in Table 3) and another one on key tokens including content words (nouns, verbs, adjectives, adverbs), determiners, and numbers (*acc(key-tokens)* in Table 3). The key tokens contain most of the useful monotonicity information for inference. In token-level evaluation, we counted the number of correctly annotated tokens for *acc(all-tokens)* or the number of correctly annotated key tokens for *acc(key-tokens)*. In sentence-level evaluation, we counted the number of correct sentences. A correct sentence has all tokens correctly annotated for *acc(all-tokens)* or all key tokens correctly annotated for *acc(key-tokens)*. We also evaluated our system's robustness on the token level. We followed the robustness metric for evaluating multi-class classification tasks, which uses precision, recall, and F1 score to measure a system's robustness. We calculated these three metrics for each polarity label: monotone(↑), antitone(↓), and None or no monotonicity information(=). The robustness evaluation is also done both on all tokens and on key tokens.

	Token-level		
system	NatLog	ccg2mono	ours
acc(all-tokens)	69.9	76.0	**96.5**
acc(key-tokens)	68.1	78.0	**96.5**
	Sentence-level		
system	NatLog	ccg2mono	ours
acc(all-tokens)	28.0	44.6	**87.5**
acc(key-tokens)	28.6	50.0	**89.2**

Table 3: This table shows the polarity annotation accuracy on the token level and the sentence level for three systems: NatLog, ccg2mono, and our system. The token level accuracy counts the number of correctly annotated tokens, and the sentence level accuracy counts the number of correctly annotated sentences. Two types of accuracy are used. For *acc(all-tokens)*, all tokens are evaluated. For *acc(key-tokens)*, only key tokens (content words + determiners + numbers) are evaluated.

6 Evaluation

Table 3 shows the performance of our system, compared with NatLog and ccg2mono. Our evaluation process is the same as Hu and Moss (2020). From Table 3, we first observe that our system consistently outperforms ccg2mono and NatLog on both the token level and the sentence level. For accuracy on the token level, our system has the highest accuracy for the evaluation on all tokens (96.5) and the highest accuracy for the evaluation on key tokens (96.5). Our system's accuracy on key tokens is higher than the accuracy on all tokens, which demonstrates our system's good performance on polarity annotation for tokens that are more signif-

	All Tokens								
system	**NatLog**			**ccg2mono**			**ours**		
Polarity	Monotone	Antitone	None	Monotone	Antitone	None	Monotone	Antitone	None
precision	71.4	43.5	70.7	86.0	75.6	58.0	**97.6**	**96.5**	**91.7**
recall	87.3	15.9	63.9	77.8	78.3	74.6	**97.2**	**89.4**	**87.3**
F1-score	78.6	23.3	67.1	81.7	76.9	65.3	**97.4**	**97.6**	**89.4**
	Key Tokens								
system	**NatLog**			**ccg2mono**			**ours**		
Polarity	Monotone	Antitone	None	Monotone	Antitone	None	Monotone	Antitone	None
precision	68.7	70.9	42.1	85.2	78.7	62.7	**96.9**	**96.4**	**94.2**
recall	88.6	61.5	14.0	80.3	79.3	73.7	**97.9**	**98.5**	**86.0**
F1-score	77.4	65.9	21.1	82.7	79.0	67.7	**97.4**	**97.4**	**89.9**

Table 4: Token level robustness comparison between NatLog, ccg2mono, and our system. The robustness score is evaluated both on all tokens and on key tokens (content words + determiners + numbers). For each of the three polarities: monotone($\uparrow$), antitone($\downarrow$), and None or no monotonicity information($=$), the relative precision, recall and F1 score are calculated.

icant to monotonicity inference. For accuracy on the sentence level, our system again has the highest accuracy for the evaluation on all tokens (87.5) and the highest accuracy for the evaluation on key tokens (89.2). Such results suggest that our system can achieve good performance on determining the monotonicity of the sentence constituents. Overall, the evaluation validates that our system has higher polarity annotation accuracy than existing systems. We compared our annotations to ccg2mono's annotation and observed that of all the tokens in the 56 sentences, if ccg2mono annotates it correctly, then our system also does so. This means, our system's polarization covers more linguistic phenomena than ccg2mono. Table 4 shows the robustness score of our system and the two existing systems. Our systems has much higher precision and recall on all three polarity labels than the other two systems. For the F1 score, our system again has the highest points over the other two systems. The consistent and high robustness scores show that our system's performance is much more robust on the given dataset than existing systems.

7 Conclusion and Future Work

In this paper, we have demonstrated our system's ability to automatically annotate monotonicity information (polarity) for a sentence by conducting polarization on a universal dependency parse tree. The system operates by first converting the parse tree to a binary parse tree and then marking polarity on each node according to a lexicon of polarization rules. The system produces accurate annotations on sentences involving many different linguistic phenomena such as quantifiers, double negation, relative clauses, and conditionals. Our system had better performance on polarity marking than existing systems including ccg2mono (Hu and Moss, 2018) and NatLog (MacCartney and Manning, 2009; Angeli et al., 2016). Additionally, by using UD parsing, our system offers many advantages. Our system supports a variety of text genres and can be applied to many languages. In general, this paper opens up a new framework for performing inference, semantics, and automated reasoning over UD representations.

For future work, an inference system can be made that utilizes the monotonicity information annotated by our system, which is similar to the MonaLog system (Hu et al., 2020). Several improvements can be made to the system to obtain more accurate annotations. One improvement would be to incorporate pragmatics to help determine the monotonicity of a scalar number.

Acknowledgements

This research is advised by Dr. Lawrence Moss from Indiana University and Dr. Michael Wollowski from Rose-hulman Institute of Technology. We thank their helpful advises and feedback on this research. We also thank the anonymous reviewers for their insightful comments.

References

Lasha Abzianidze. 2017. LangPro: Natural language theorem prover. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 115–120, Copenhagen, Denmark. Association for Computational Linguistics.

Gabor Angeli, Neha Nayak, and Christopher D. Manning. 2016. Combining natural logic and shallow reasoning for question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 442–452, Berlin, Germany. Association for Computational Linguistics.

Johan van Benthem. 1986. *Essays in Logical Semantics*, volume 29 of *Studies in Linguistics and Philosophy*. D. Reidel Publishing Co., Dordrecht.

P. Blackburn and Johan Bos. 2005. Representation and inference for natural language - a first course in computational semantics. In *CSLI Studies in Computational Linguistics*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Hai Hu, Qi Chen, Kyle Richardson, Atreyee Mukherjee, Lawrence S Moss, and Sandra Kübler. 2020. MonaLog: a lightweight system for natural language inference based on monotonicity. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2020*, pages 319–329.

Hai Hu and Larry Moss. 2018. Polarity computations in flexible categorial grammar. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 124–129, New Orleans, Louisiana. Association for Computational Linguistics.

Hai Hu and Lawrence S. Moss. 2020. An automatic monotonicity annotation tool based on ccg trees. In *Second Tsinghua Interdisciplinary Workshop on Logic, Language, and Meaning: Monotonicity in Logic and Language*.

Thomas F. Icard III and Lawrence S. Moss. 2014. Recent progress on monotonicity. In *Linguistic Issues in Language Technology, Volume 9, 2014 - Perspectives on Semantic Representations for Textual Inference*. CSLI Publications.

Lauri Karttunen. 2012. Simple and phrasal implicatives. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics - Volume 1: Proceedings of the Main Conference and the Shared Task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, SemEval '12, page 124–131, USA. Association for Computational Linguistics.

Edward L. Keenan and Leonard M. Faltz. 1984. *Boolean Semantics for Natural Language*. Springer.

J. Lavalle-Martínez, M. Montes y Gómez, L. Pineda, Héctor Jiménez-Salazar, and Ismael Everardo Bárcenas Patiño. 2018. Equivalences among polarity algorithms. *Studia Logica*, 106:371–395.

Mike Lewis and Mark Steedman. 2014. A* CCG parsing with a supertag-factored model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 990–1000, Doha, Qatar. Association for Computational Linguistics.

Wei Liu, Lei Li, Zuying Huang, and Yinan Liu. 2019. Multi-lingual Wikipedia summarization and title generation on low resource corpus. In *Proceedings of the Workshop MultiLing 2019: Summarization Across Languages, Genres and Sources*, pages 17–25, Varna, Bulgaria. INCOMA Ltd.

Yijia Liu, Yi Zhu, Wanxiang Che, Bing Qin, Nathan Schneider, and Noah A. Smith. 2018. Parsing tweets into Universal Dependencies. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 965–975, New Orleans, Louisiana. Association for Computational Linguistics.

Bill MacCartney and Christopher D. Manning. 2009. An extended model of natural logic. In *Proceedings of the Eight International Conference on Computational Semantics*, pages 140–156, Tilburg, The Netherlands. Association for Computational Linguistics.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland. Association for Computational Linguistics.

Christopher D. Manning. 2011. Part-of-speech tagging from 97linguistics? In *CICLing*.

L. Moss. 2012. The soundness of internalized polarity marking. *Studia Logica*, 100:683–704.

Lawrence S. Moss and Hai Hu. 2020. Syllogistic logics with comparative adjectives. Unpublished ms., Indiana University.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666, Portorož, Slovenia. European Language Resources Association (ELRA).

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational*

Linguistics: System Demonstrations, pages 101–108, Online. Association for Computational Linguistics.

Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. 2016. Transforming dependency structures to logical forms for semantic parsing. *Transactions of the Association for Computational Linguistics*, 4:127–140.

Alexis Ross and Ellie Pavlick. 2019. How well do NLI models capture verb veridicality? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2230–2240, Hong Kong, China. Association for Computational Linguistics.

V. Sanchez. 1991. Studies on natural logic and categorial grammar.

Natalia Silveira, Timothy Dozat, Marie-Catherine de Marneffe, Samuel Bowman, Miriam Connor, John Bauer, and Christopher D. Manning. 2014. A gold standard dependency corpus for English. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*.

Amir Zeldes. 2017. The GUM corpus: Creating multilayer resources in the classroom. *Language Resources and Evaluation*, 51(3):581–612.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. CoNLL 2018 shared task: Multilingual parsing from raw text to Universal Dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium. Association for Computational Linguistics.

NeuralLog: Natural Language Inference with Joint Neural and Logical Reasoning

Zeming Chen[†*] **Qiyue Gao**[†] **Lawrence S. Moss**[‡]
[†]Rose-Hulman Institute of Technology, Terre Haute, IN, USA
[‡]Indiana University, Bloomington, IN, USA
{chenz16,gaoq}@rose-hulman.edu
{lmoss}@indiana.edu

Abstract

Deep learning (DL) based language models achieve high performance on various benchmarks for Natural Language Inference (NLI). And at this time, symbolic approaches to NLI are receiving less attention. Both approaches (symbolic and DL) have their advantages and weaknesses. However, currently, no method combines them in a system to solve the task of NLI. To merge symbolic and deep learning methods, we propose an inference framework called NeuralLog, which utilizes both a monotonicity-based logical inference engine and a neural network language model for phrase alignment. Our framework models the NLI task as a classic search problem and uses the beam search algorithm to search for optimal inference paths. Experiments show that our joint logic and neural inference system improves accuracy on the NLI task and can achieve state-of-art accuracy on the SICK and MED datasets.

1 Introduction

Currently, many NLI benchmarks' state-of-the-art systems are exclusively deep learning (DL) based language models (Devlin et al., 2019; Lan et al., 2020; Liu et al., 2020; Yin and Schütze, 2017). These models often contain a large number of parameters, use high-quality pre-trained embeddings, and are trained on large-scale datasets, which enable them to handle diverse and large test data robustly. However, several experiments show that DL models lack generalization ability, adopt fallible syntactic heuristics, and show exploitation of annotation artifacts (Glockner et al., 2018; McCoy et al., 2019; Gururangan et al., 2018). On the other hand, there are logic-based systems that use symbolic reasoning and semantic formalism to solve NLI (Abzianidze, 2017; Martínez-Gómez et al., 2017; Yanaka et al., 2018; Hu et al., 2020). These systems show high precision on complex inferences involving difficult linguistic phenomena and present logical and explainable reasoning processes. However, these systems lack background knowledge and do not handle sentences with syntactic variations well, which makes them poor competitors with state-of-the-art DL models. Both DL and logic-based systems show a major issue with NLI models: they are too one-dimensional (either purely DL or purely logic), and no method has combined these two ap-

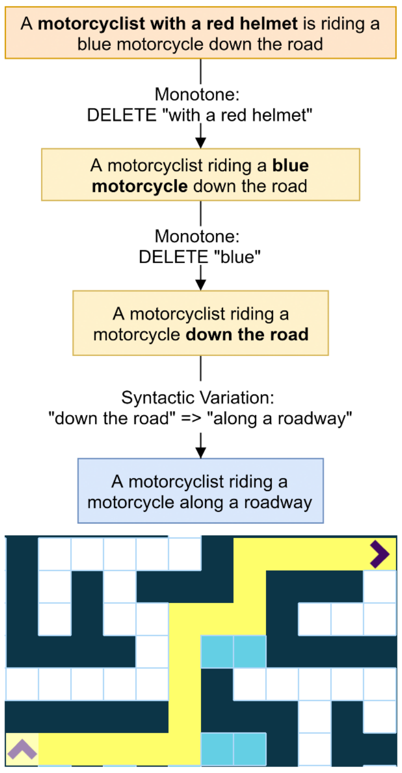

A **motorcyclist with a red helmet** is riding a blue motorcycle down the road
Monotone: DELETE "with a red helmet"
A motorcyclist riding a **blue motorcycle** down the road
Monotone: DELETE "blue"
A motorcyclist riding a motorcycle **down the road**
Syntactic Variation: "down the road" => "along a roadway"
A motorcyclist riding a motorcycle along a roadway

Figure 1: Analogy between path planning and an entailment inference path from the premise *A motorcyclist with a red helmet is riding a blue motorcycle down the road* to the hypothesis *A motorcyclist is riding a motorbike along a roadway*.

[*]The first two authors have equal contribution

proaches together for solving NLI.

This paper makes several contributions, as follows: first, we propose a new framework in section 3 for combining logic-based inference with deep-learning-based network inference for better performance on conducting natural language inference. We model an NLI task as a path-searching problem between the premises and the hypothesis. We use beam-search to find an optimal path that can transform a premise to a hypothesis through a series of inference steps. This way, different inference modules can be inserted into the system. For example, DL inference modules will handle inferences with diverse syntactic changes and logic inference modules will handle inferences that require complex reasoning. Second, we introduce a new method in section 4.3 to handle syntactic variations in natural language through sequence chunking and DL based paraphrase detection. We evaluate our system in section 6 by conducting experiments on the SICK and MED datasets. Experiments show that joint logical and neural reasoning show state-of-art accuracy and recall on these datasets.

2 Related Work

Perhaps the closest systems to NeuralLog are Yanaka et al. (2018), MonaLog (Hu et al., 2020), and Hy-NLI (Kalouli et al., 2020). Using Martínez-Gómez et al. (2016) to work with logic representations derived from CCG trees, Yanaka et al. (2018) proposed a framework that can detect phrase correspondences for a sentence pair, using natural deduction on semantic relations and can thus extract various paraphrases automatically. Their experiments show that assessing phrase correspondences helps improve NLI accuracy. Our system uses a similar methodology to solve syntactic variation inferences, where we determine if two phrases are paraphrases. Our method is rather different on this point, since we call on neural language models to detect paraphrases between two sentences. We feel that it would be interesting to compare the systems on a more theoretical level, but we have not done the comparison in this paper.

NeuralLog inherits the use of polarity marking found in MonaLog (Hu et al., 2020). (However, we use the dependency-based system of Chen and Gao (2021) instead of the CCG-based system of Hu and Moss (2018).) MonaLog did propose some integration with neural models, using BERT when logic failed to find entailment or contradiction. We are doing something very different, using neural models to detect paraphrases at several levels of "chunking". In addition, the exact algorithms found in Sections 3 and 4 are new here. In a sense, our work on alignment in NLI goes back to MacCartney and Manning (2009) where alignment was used to find a chain of edits that changes a premise to a hypothesis, but our work uses much that simply was not available in 2009.

Hy-NLI is a hybrid system that makes inferences using either symbolic or deep learning models based on how linguistically challenging a pair of sentences is. The principle Hy-NLI followed is that deep learning models are better at handling sentences that are linguistically less complex, and symbolic models are better for sentences containing hard linguistic phenomena. Although the system integrates both symbolic and neural methods, its decision process is still separate, in which the symbolic and deep learning sides make decisions without relying on the other side. Differently, our system incorporates logical inferences and neural inferences as part of the decision process, in which the two inference methods rely on each other to make a final decision.

3 Method

3.1 NLI As Path Planning

The key motivation behind our architecture and inference modules is that the Natural Language Inference task can be modeled as a path planning problem. Path planning is a task for finding an optimal path traveling from a start point to a goal containing a series of actions. To formulate NLI as path planning, we define the **premise** as the **start state** and the **hypothesis** as the **goal** that needs to be reached. The classical path planning strategy applies expansions from the start state through some search algorithms, such as depth-first-search or Dijkstra search, until an expansion meets the goal. In a grid map, two types of action produce an expansion. The vertical action moves up and down, and the horizontal action moves left and right. Similarly, language inference also contains these two actions. Monotonicity reasoning is a vertical action, where the monotone inference moves up and simplifies a sentence, and the antitone inference moves down and makes a sentence more specific. Syntactic variation and synonym replacement are horizontal actions. They change the form of a sentence while maintaining the original mean-

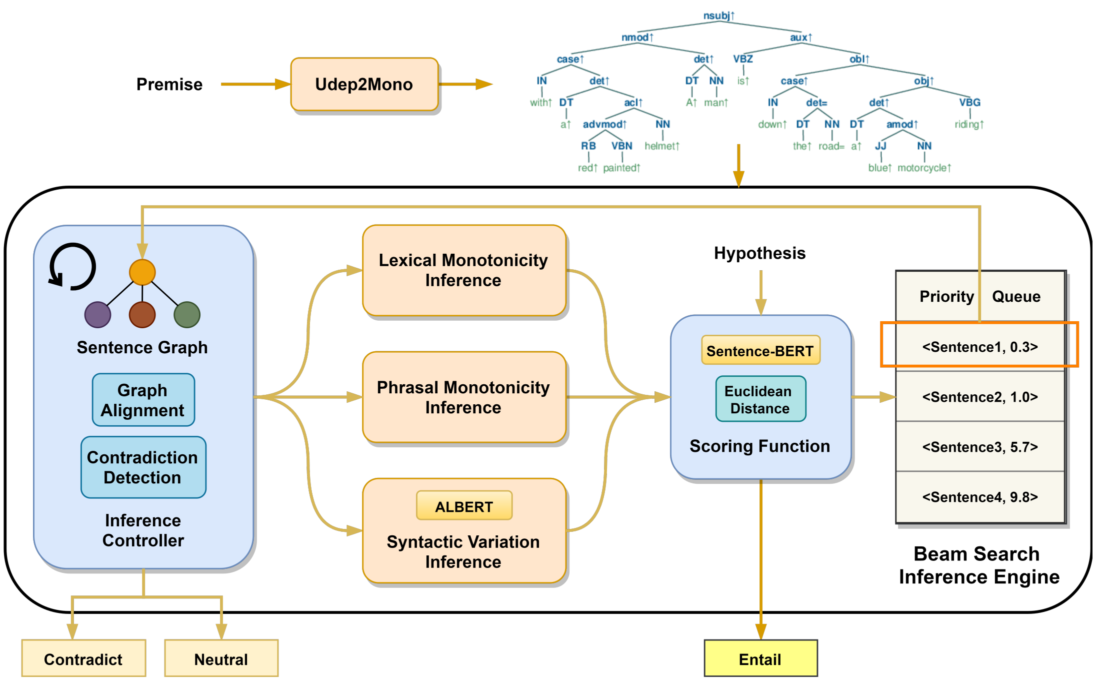


Figure 2: Overview system diagram of NeuralLog.

ing. Then, similar to path planning, we can continuously make inferences from the premise using a search algorithm to determine if the premise entails the hypothesis by observing whether one of the inferences can reach the hypothesis. If the hypothesis is reached, we can connect the list of inferences that transform a premise to a hypothesis to be the optimal path in NLI, a valid reasoning chain for entailment.

Figure 1 shows an analogy between an optimal path for the classical grid path planning problem and an example of an optimal inference path for NLI. On the top, we have a reasoning process for natural language inference. From the premise, we can first delete the modifier *with a red helmet*, then delete *blue* to get a simplified sentence. Finally, we can paraphrase *down the road* to *along a roadway* in the premise to reach the hypothesis and conclude the entailment relationship between these two sentences.

3.2 Overview

Our system contains four components: (1) a polarity annotator, (2) three sentence inference modules, (3) a search engine, and (4) a sentence inference controller. Figure 2 shows a diagram of the full system. The system first annotates a sentence with monotonicity information (polarity marks) using Udep2Mono (Chen and Gao, 2021). The polarity marks include monotone (↑), antitone (↓), and no monotonicity information (=) polarities. Next, the polarized parse tree is passed to the search engine. A beam search algorithm searches for the optimal inference path from a premise to a hypothesis. The search space is generated from three inference modules: lexical, phrasal, and syntactic variation. Through graph alignment, the sentence inference controller selects a inference module to apply to the premise and produce a set of new premises that potentially form entailment relations with the hypothesis. The system returns **Entail** if an inference path is found. Otherwise, the controller will determine if the premise and hypothesis form a contradiction by searching for counter example signatures and returns **Contradict** accordingly. If neither **Entail** nor **Contradict** is returned, the system returns **Neutral**.

3.3 Polarity Annotator

The system first annotates a given premise with monotonicity information using Udep2Mono, a polarity annotator that determines polarization of all constituents from universal dependency trees. The annotator first parses the premise into a binarized universal dependency tree and then conducts polarization by recursively marks polarity on each node . An example can be *Every*$^{\uparrow}$ *healthy*$^{\downarrow}$ *person*$^{\downarrow}$ *plays*$^{\uparrow}$ *sports*$^{\uparrow}$.

3.4 Search Engine

To efficiently search for the optimal inference path from a premise $\mathcal{P}$ to a hypothesis $\mathcal{H}$, we use a beam search algorithm which has the advantage of reducing search space by focusing on sentences with higher scores. To increase the search efficiency and accuracy, we add an inference controller that can guide the search direction.

Scoring In beam search, a priority queue $\mathcal{Q}$ maintains the set of generated sentences. A core operation is the determination of the highest-scoring generated sentence for a given input under a learned scoring model. In our case, the maximum score is equivalent to the minimum distance:

$$\mathbf{y}^{\star} = \arg\max_{\mathrm{s}\in\mathcal{S}} \mathrm{score}(\mathrm{s}, \mathcal{H})$$
$$\mathbf{y}^{\star} = \arg\min_{\mathrm{s}\in\mathcal{S}} \mathrm{dist}(\mathrm{s}, \mathcal{H})$$

where $\mathcal{H}$ is the hypothesis and $\mathcal{S}$ is a set of generated sentences produced by the three (lexical, phrasal, syntactic variation) inference modules. We will present more details about these inference modules in section 4. We formulate the distance function as the Euclidean distance between the sentence embeddings of the premise and hypothesis. To obtain semantically meaningful sentence embeddings efficiently, we use Reimers and Gurevych (2019)'s language model, Sentence-BERT (SBERT), a modification of the BERT model. It uses siamese and triplet neural network structures to derive sentence embeddings which can be easily compared using distance functions.

3.5 Sentence Inference Controller

In each iteration, the search algorithm expands the search space by generating a set of potential sentences using three inference modules: (1) lexical inference, (2) phrasal inference, and (3) syntactic variation inference. To guide the search engine to select the most applicable module, we designed a inference controller that can recommend which of the labels the overall algorithm should proceed with. For example, for a premise *All animals eat food* and a hypothesis *All dogs eat food*, only a lexical inference of *animals* to *dogs* would be needed. Then, the controller will apply the lexical inference to the premise, as we discuss below.

3.5.1 Sentence Representation Graph

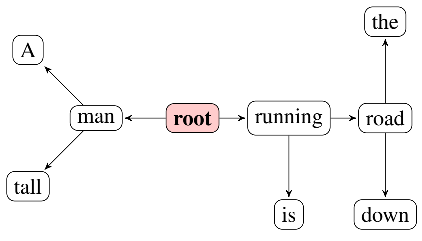
The controller makes its decision based on graph-based representations for the premise and the hypothesis. We first build a sentence representation graph from parsed input using Universal Dependencies. Let $\mathcal{V} = \mathcal{V}_m \cup \mathcal{V}_c$ be the set of vertices of a sentence representation graph, where $\mathcal{V}_m$ represents the set of modifiers such as *tall* in Figure 5, and V_c represents the set of content words (words that are being modified) such as *man* in Figure 5. While content words in $\mathcal{V}_c$ could modify other content words, modifiers in $\mathcal{V}_m$ are not modified by other vertices. Let $\mathcal{E}$ be the set of directed edges in the form $\langle v_c, v_m \rangle$ such that $v_m \in \mathcal{V}_m$ and $v_c \in \mathcal{V}_c$. A sentence representation graph is then defined as a tuple $\mathrm{G} = \langle \mathcal{V}, \mathcal{E} \rangle$. Figure 3a shows an example graph.

3.5.2 Graph Alignment

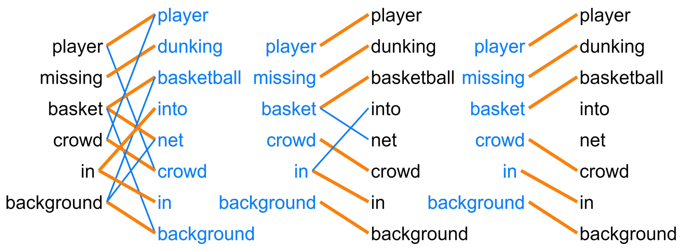
To observe the differences between two sentences, we rely on graph alignment between two sentence representation graphs. We first align nodes from subjects, verbs and objects, which constitutes what we call a component level. Define G_p as the graph for a premise and G_h as the graph for a hypothesis. Then, $\mathcal{C}_p$ and $\mathcal{C}_h$ are component level nodes from the two graphs. We take the Cartesian product $\mathcal{C}_p \times \mathcal{C}_h = \{(\mathrm{c}_p, \mathrm{c}_h) : \mathrm{c}_p \in \mathcal{C}_p, \mathrm{c}_h \in \mathcal{C}_h\}$. In the first round, we recursively pair the child nodes of each c_p to child nodes of each c_h. We compute word similarity between two child nodes c_p^i and c_h^i and eliminate pairs with non-maximum similarity. We denote the new aligned pairs as a set $\mathcal{A}^*$. At the second round, we iterate through the aligned pairs in $\mathcal{A}^*$. If multiple child nodes from the first graph are paired to a child node in the second graph, we only keep the pair with maximum word similarity. In the final round, we perform the same check for each child node in the first graph to ensure that there are no multiple child nodes from the second graph paired to it. Figure 3b shows a brief visualization of the alignment process.

3.5.3 inference Module Recommendation

After aligning the premise graph $\mathcal{G}_p$ with hypothesis graph $\mathcal{G}_h$, the controller checks through each node in the two graphs. If a node does not get aligned, the controller considers to delete the node or insert it depending on which graph the node belongs to and recommends phrasal inference. If a node is different from its aligned node, the controller recommends lexical inference. If additional lexical or phrasal inferences are detected under this node, the controller decides that there is a more complex transition under this node and rec-



(a) Sentence representation graph



(b) Graph alignment visualization

Figure 3: (a) A sentence representation graph for *A tall man is running down the road.* (b) Visualization for the graph alignment. The lines between two words represent their similarity. The orange lines are the pairs with maximum similarities for a blue word. Through bi-directional alignment, we eliminate word pairs with non-maximum similarity and gets the final alignment pairs.

ommends a syntactic variation.

3.5.4 Contradiction Detection

We determine whether the premise and the hypothesis contradict each other inside the controller by searching for potential contradiction transitions from the premise to the hypothesis. For instance, a transition in the scope of the quantifier ($a \longrightarrow$ *no*) from the same subject could be what we call a contradiction signature (possible evidence for a contradiction). With all the signatures, the controller decides if they can form a contradiction as a whole. To avoid situations when multiple signatures together fail to form a complete contradiction, such as double negation, the controller checks through the contradiction signatures to ensure a contradiction. For instance, in the verb pair (*not remove*, *add*), the contradiction signature *not* would cancel the verb negation contradiction signature from *remove* to *add* so the pair as a whole would not be seen as a contradiction. Nevertheless, other changes from the premise to the hypothesis may change the meaning of the sentence. Hence, our controller would go through other transitions to make sure the meaning of the sentence does not change when the contradiction sign is valid. For example, in the neutral pair P: *A person is eating* and H: *No tall person is eating*, the addition of *tall* would be detected by our controller. But the aligned word of the component it is applied to, *person* in P, has been marked downward monotone. So this transition is invalid. This pair would then be classified as neutral.

For P2 and H2 in Figure 4, the controller notices the contradictory quantifier change around the subject *man*. The subject *man* in P2 is upward monotone so the deletion of *tall* is valid. Our controller also detects the meaning transition from

signature type	example
quantifier negation	**no** dogs $\Longrightarrow$ **some** dogs
verb negation	is **eating** $\Longrightarrow$ is **not eating**
noun negation	**some people** $\Longrightarrow$ **nobody**
action contradiction	is **sleeping** $\Longrightarrow$ is **running**
direction contradiction	The **turtle** is following the **fish** $\Longrightarrow$ The **fish** is following the **turtle**

Table 1: Examples of contradiction signatures.

down the road to *inside the building*, which affects the sentence's meaning and cancels the previous contradiction signature. The controller thus will not classify P2 and H2 as a pair of contradiction.

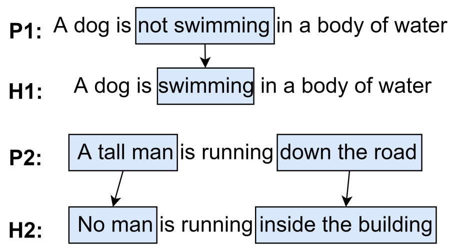

P1: A dog is not swimming in a body of water
H1: A dog is swimming in a body of water
P2: A tall man is running down the road
H2: No man is running inside the building

Figure 4: Example of contradiction signatures. P1 and H1 form a contradiction. P2 and H2 does not form a contradiction because the meaning after the verb *running* has changed.

4 Inference Generation

4.1 Lexical Monotonicity Inference

Lexical inference is word replacement based on monotonicity information for key-tokens including nouns, verbs, numbers, and quantifiers. The system uses lexical knowledge bases including WordNet (Miller, 1995) and ConceptNet (Liu and Singh, 2004). From the knowledge bases, we extract four word sets: hypernyms, hyponyms, synonyms, and antonyms. Logically, if a word has a monotone polarity (↑), it can be replaced by its hypernyms. For example, *swim* ≤ *move*; then *swim* can be replaced with *move*. If a word has an antitone polarity (↓), it can be replaced by its hyponyms. For example, *flower* $\geq$ *rose*. Then, *flower* can be replaced with *rose*. We filter out irrelevant words from the knowledge bases that do not appear in the hypothesis. Additionally, we handcraft knowledge relations for words like quantifiers and prepositions that do not have sufficient taxonomies from knowledge bases. Some handcrafted relations include: *all* = *every* = *each* $\leq$ *most* $\leq$ *many* $\leq$ *several* $\leq$ *some* = *a*, *up* $\perp$ *down*.

4.2 Phrasal Monotonicity Inference

Phrasal replacements are for phrase-level monotonicity inference. For example, with a polarized sentence *A* $^{\uparrow}$ *woman*$^{\uparrow}$ *who*$^{\uparrow}$ *is*$^{\uparrow}$ *beautiful*$^{\uparrow}$ *is*$^{\uparrow}$ *walking*$^{\uparrow}$ *in*$^{\uparrow}$ *the*$^{\uparrow}$ *rain*$^{=}$, the monotone mark $^{\uparrow}$ on *woman* allows an upward inference: *woman* $\sqsupseteq$ *woman who is beautiful*, in which the relative clause *who is beautiful* is deleted. The system follows a set of phrasal monotonicity inference rules. For upward monotonicity inference, modifiers of a word are deleted. For downward monotonicity inference, modifiers are inserted to a word. The algorithm traverses down a polarized UD parse tree, deletes the modifier sub-tree if a node is monotone ($\uparrow$), and inserts a new sub-tree if a node is antitone ($\downarrow$). To insert new modifiers, the algorithm extracts a list of potential modifiers associated to a node from a modifier dictionary. The modifier dictionary is derived from the hypothesis and contains word-modifier pairs for each dependency relation. Below is an example of a modifier dictionary from *There are no beautiful flowers that open at night*:

- **obl**: [head: *open*, mod: *at night*]
- **amod**: [head: *flowers*, mod: *beautiful*]
- **acl:relcl**: [head: *flowers*, mod: *that open at night*]

4.3 Syntactic Variation Inference

We categorize linguistic changes between a premise and a hypothesis that cannot be inferred from monotonicity information as *syntactic variations*. For example, a change from *red rose* to *a rose which is red* is a syntactic variation. Many logical systems rely on handcrafted rules and manual transformation to enable the system to use syntactic variations. However, without accurate alignments between the two sentences, these methods are not robust enough, and thus are difficult to scale up for wide-coverage input.

Recent development of pretrained transformer-based language models are showing state-of-art performance on multiple benchmarks for Natural Language Understanding (NLU) including the task for paraphrase detection (Devlin et al., 2019; Lan et al., 2020; Liu et al., 2020) exemplify phrasal knowledge of syntactic variation. We propose a method that incorporates transformer-based language models to robustly handle syntactic variations. Our method first uses a sentence chunker to decompose both the premise and the hypothesis into chunks of phrases and then forms a Cartesian product of chunk pairs. For each pair, we use a transformer model to calculate the likelihood of a pair of chunks being a pair of paraphrases.

4.3.1 Sequence Chunking

To obtain phrase-level chunks from a sentence, we build a sequence chunker to extract chunks from a sentence using its universal dependency information. Instead of splitting a sentence into chunks, our chunker composes word tokens recursively to form meaningful chunks. First, we construct a sentence representation graph of a premise from the controller. Recall that a sentence representation graph is defined as $\mathrm{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, where $\mathcal{V} = \mathcal{V}_m \cup \mathcal{V}_c$ is the set of modifiers ($\mathcal{V}_m$) and content words ($\mathcal{V}_c$), and $\mathcal{E}$ is the set of directed edges. To generate the chunk for a content word in $\mathcal{V}_c$, we arrange its modifiers, which are nodes it points to, together with the content word by their word orders in the original sentence to form a word chain. Modifiers that make the chain disconnected are discarded because they are not close enough to be part of the chunk. For instance, the chunk from the verb *eats* in the sentence *A person eats the food carefully* would not contain its modifier *carefully* because they are separated by the object *the food*. If the sentence is stated as *A person carefully eats the food*, *carefully* now is next to *eat* and it would be included in the chunk of the verb *eat*. To obtain chunks for a sentence, we iterate through each main component node, which is a node for subject, verb, or object, in the sentence's graph representation and construct verb phrases by combining verbs' chunks with their paired objects' chunks. There are cases when a word modifies other words and gets modified in the same time. They often occur when a chunk serves as a modifier. For example, in *The woman in a pink dress is dancing*, the phrase *in a pink dress* modifies *woman* whereas *dress* is modified by *in*, *a* and *pink*. Then edges from *dress* to *in*, *a*, *pink* with the edge from *woman* to *dress* can be drawn. Chunks *in a pink dress* and *the woman in a*

Type	Premise	Hypothesis
Verb Phrase Variation	Two men are standing near the water and are **holding fishing poles**	Two men are standing near the water and are **holding tools used for fishing**
Noun Phrase Variation	A man with climbing equipment is hanging from **rock which is vertical and white**	A man with equipment used for climbing is hanging from a **white, vertical rock**.

Table 2: Examples of phrasal alignments detected by the syntactic variation module

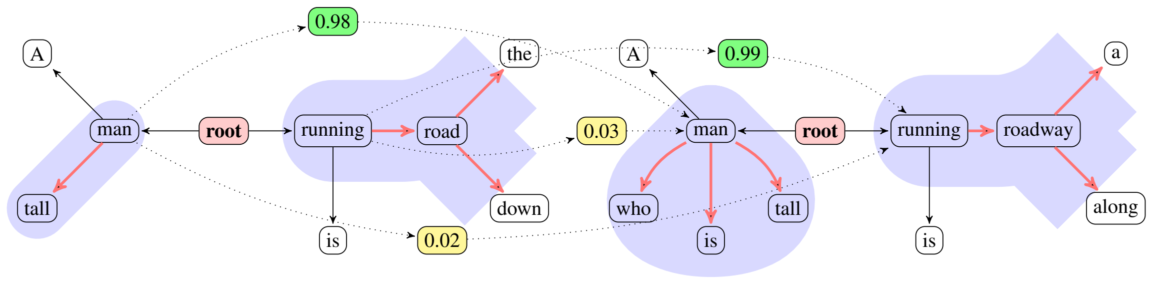


Figure 5: A graph representation of the monolingual phrase alignment process. Here the left graph represents the premise: *A tall man is running down the road*. The right graph represents the hypothesis *A man who is tall is running along a roadway*. The blue region represents phrase chunks extracted by the chunker from the graph. An alignment score is calculated for each pair of chunks. The pair ⟨ *tall man*, *man who is tall* ⟩ is a pair of paraphrases, and thus has a high alignment score (0.98). The pair ⟨ *tall man*, *running along a road way* ⟩ has two unrelated phrases, and thus has a low alignment score(0.03).

pink dress will be generated for *dress* and *woman* respectively.

4.3.2 Monolingual Phrase Alignment

After the chunker outputs a set of chunks from a generated sentence and from the hypothesis, the system selects chunk pairs that are aligned by computing an alignment score for each pair of chunks. Formally, we define $\mathcal{C}_s$ as the set of chunks from a generated sentence and $\mathcal{C}_h$ as the set of chunks from the hypothesis. We build the Cartesian product from $\mathcal{C}_s$ and $\mathcal{C}_h$, denoted $\mathcal{C}_s \times \mathcal{C}_h$. For each chunk pair $(\mathrm{c}_{si}, \mathrm{c}_{hj}) \in \mathcal{C}_s \times \mathcal{C}_h$, we compute an alignment score $\boldsymbol{\alpha}$:

$$\mathbf{y}_{\langle \mathbf{c_{si}}, \mathbf{c_{hi}} \rangle} = \mathrm{ALBERT.forward}(\langle \mathbf{c_{si}}, \mathbf{c_{hi}} \rangle)$$
$$\boldsymbol{\alpha}_{\langle \mathbf{c_{si}}, \mathbf{c_{hi}} \rangle} = \mathrm{p}(\mathbf{c_{si}} \mid \mathbf{c_{hj}})$$
$$\boldsymbol{\alpha}_{\langle \mathbf{c_{si}}, \mathbf{c_{hi}} \rangle} = \frac{\exp^{\mathbf{y}_{\langle \mathbf{c_{si}}, \mathbf{c_{hi}} \rangle_0}}}{\sum_{j=1}^{2} \exp^{\mathbf{y}_{\langle \mathbf{c_{si}}, \mathbf{c_{hi}} \rangle_j}}}$$

If $\boldsymbol{\alpha} > 0.85$, the system records this pair of phrases as a pair of syntactic variation. To calculate the alignment score, we use an ALBERT (Lan et al., 2020) model for the paraphrase detection task, fine tuned on the Microsoft Research Paraphrase Corpus (Dolan and Brockett, 2005). We first pass the chunk pair to ALBERT to obtain the logits. Then we apply a softmax function to the logits to get the final probability. A full demonstration of the alignment between chunks is shown in Figure 5.

5 Data

5.1 The SICK Dataset

The SICK (Marelli et al., 2014) dataset is an English benchmark that provides in-depth evaluation for compositional distribution models. There are 10,000 English sentence pairs exhibiting a variety of lexical, syntactic, and semantic phenomena. Each sentence pair is annotated as Entailment, Contradiction, or Neutral. we use the 4,927 test problems for evaluation.

5.2 The MED Dataset

The Monotonicity Entailment Dataset (MED), is a challenge dataset designed to examine a model's ability to conduct monotonicity inference (Yanaka et al., 2019a). There are 5382 sentence pairs in MED, where 1820 pairs are upward inference problems, 3270 pairs are downward inference problems, and 292 pairs are problems with no monotonicity information. MED's problems cover a variety of linguistic phenomena, such as lexical knowledge, reverse, conjunction and disjunction, conditional, and negative polarity items.

6 Evaluation

6.1 Experiment Setup

For Universal Dependency parsing, we follow Chen and Gao (2021)'s framework and use a parser

Model	P	R	acc.
ML/DL-based systems			
BERT (base, uncased)	86.8	85.4	86.7
Yin and Schütze (2017)	–	–	87.1
Beltagy et al. (2016)	–	–	85.1
Logic-based systems			
Abzianidze (2017)	98.0	58.1	81.4
Martínez-Gómez et al. (2017)	97.0	63.6	83.1
Yanaka et al. (2018)	84.2	77.3	84.3
Hu et al. (2020)	83.8	70.7	77.2
Abzianidze (2020)	94.3	67.9	84.4
Hybrid System			
Hu et al. (2020)+BERT	83.2	85.5	85.4
Kalouli et al. (2020)	–	–	86.5
Our System			
NeuralLog (full system)	88.0	**87.6**	**90.3**
− ALBERT-SV	68.9	79.3	71.4
− Monotonicity	74.5	75.1	74.7

Table 3: Performance on the SICK test set

from Stanford's natural language analysis package, Stanza (Qi et al., 2020). In the parser, we use a neural parsing model pretrained on the UD English GUM corpus (Zeldes, 2017) with 90.0 LAS (Zeman et al., 2018) evaluation score. For Sentence-BERT, we selected the BERT-large model pre-trained on STS-B (Cer et al., 2017). For ALBERT, we used textattack's ALBERT-base model pretrained on MRPC from transformers. For word alignment in the controller, we select Řehůřek and Sojka (2010)'s Gensim framework to calculate word similarity from pre-trained word embedding. We evaluated our model on the SICK and MED datasets using the standard NLI evaluation metrics of accuracy, precision, and recall. Additionally, we conducted two ablation tests focusing on analyzing the contributions of the monotonicity inference modules and the syntactic variation module.

6.2 Results

SICK Table 3 shows the experiment results tested on SICK. We compared our performance to several logic-based systems as well as two deep learning based models. As the evaluation results show, our model achieves the state-of-art performance on the SICK dataset. The best logic-based model is Abzianidze (2020) with 84.4 percent accuracy. The best DL-based model is Yin and Schütze (2017) with 87.1 percent accuracy. Our system outperforms both of them with 90.3 percent accuracy. Compare to Hu et al. (2020) + BERT, which also explores a way of combining logic-based methods and deep learning based methods, our system

Model	Up	Down	All
DeComp (Parikh et al., 2016)	71.1	45.2	51.4
ESIM (Chen et al., 2017)	66.1	42.1	53.8
BERT (Devlin et al., 2019)	82.7	22.8	44.7
BERT+ (Yanaka et al., 2019a)	76.0	70.3	71.6
NeuralLog (ours)	**91.4**	**93.9**	**93.4**

Table 4: Results comparing model compared to state-of-art NLI models evaluated on MED. **Up**, **Down**, and **All** stand for the accuracy on upward inference, downward inference, and the overall dataset.

shows higher accuracy with a 4.92 percentage point increase. In addition, our system's accuracy has a 3.8 percentage point increase than another hybrid system, Hy-NLI (Kalouli et al., 2020). The good performance proves that our framework for joint logic and neural reasoning can achieve state-of-art performance on inference and outperforms existing systems.

Ablation Test In addition to the standard evaluation on SICK, we conducted two ablation tests. The results are included in Table 3. First, we remove the syntactic variation module that uses neural network for alignment (−ALBERT-SV). As the table shows, the accuracy drops 18.9 percentage points. This large drop in accuracy indicates that the syntactic variation module plays a major part in our overall inference process. The result also proves our hypothesis that deep learning methods for inference can improve the performance of traditional logic-based systems significantly. Secondly, when we remove the monotonicity-based inference modules (−Monotonicity), the accuracy shows another large decrease in accuracy, with a 15.6 percentage point drop. This result demonstrates the important contribution of the logic-based inference modules toward the overall state-of-the-art performance. Compared to the previous ablation test which removes the neural network based syntactic variation module, the accuracy does not change much (only 3.3 differences). This similar performance indicates that neural network inference in our system alone cannot achieve state-of-art performance on the SICK dataset, and additional guidance and constrains from the logic-based methods are essential parts of our framework. Overall, we believe that the results reveal that both modules, logic and neural, contribute equally to the final performance and are both important parts that are unmovable.

MED Table 4 shows the experimental results tested on MED. We compared to multiple deep learning based baselines. Here, DeComp and ESIM are trained on SNLI and BERT is fine-tuned with MultiNLI. The BERT+ model is a BERT model fine-tuned on a combined training data with the HELP dataset, (Yanaka et al., 2019b), a set of augmentations for monotonicity reasoning, and the MultiNLI training set. Both models were tested in Yanaka et al. (2019a). Overall, our system (NeuralLog) outperforms all DL-based baselines in terms of accuracy, by a significant amount. Compared to BERT+, our system performs better both on upward (+15.4) and downward (+23.6) inference, and shows significant higher accuracy overall (+21.8). The good performance on MED validates our system's ability on accurate and robust monotonicity-based inference.

6.3 Error Analysis

For entailment, a large amount of inference errors are due to an incorrect dependency parse trees from the parser. For example, P: *A black, red, white and pink dress is being worn by a woman*, H: *A dress, which is black, red, white and pink is being worn by a woman*, has long conjunctions that cause the parser to produce two separate trees from the same sentence. Secondly, a lack of sufficient background knowledge causes the system to fail to make inferences which would be needed to obtain a correct label. For example, P: *One man is doing a bicycle trick in midair*, H: *The cyclist is performing a trick in the air* requires the system to know that *a man doing a bicycle trick* is a *cyclist*. This kind of knowledge can only be injected to the system either by handcrafting rules or by extracting it from the training data. For contradiction, our analysis reveals inconsistencies in the SICK dataset. We account for multiple sentence pairs that have the same syntactic and semantic structures, but are labeled differently. For example, P: *A man is folding a tortilla*, H: *A man is unfolding a tortilla* has gold-label **Neutral** while P: *A man is playing a guitar*, H: *A man is not playing a guitar* has gold-label **Contradiction**. These two pair of sentences clearly have similar structures but have inconsistent gold-labels. Both gold-labels would be reasonable depending on whether the two subjects refer to the same entity.

7 Conclusion and Future Work

In this paper, we presented a framework to combine logic-based inference with deep-learning based inference for improved Natural Language Inference performance. The main method is using a search engine and an alignment based controller to dispatch the two inference methods (logic and deep-learning) to their area of expertise. This way, logic-based modules can solve inference that requires logical rules and deep-learning based modules can solve inferences that contain syntactic variations which are easier for neural networks. Our system uses a beam search algorithm and three inference modules (lexical, phrasal, and syntactic variation) to find an optimal path that can transform a premise to a hypothesis. Our system handles syntactic variations in natural sentences using the neural network on phrase chunks, and our system determines contradictions by searching for contradiction signatures (evidence for contradiction). Evaluations on SICK and MED show that our proposed framework for joint logical and neural reasoning can achieve state-of-art accuracy on these datasets. Our experiments on ablation tests show that neither logic nor neural reasoning alone fully solve Natural Language Inference, but a joint operation between them can bring improved performance.

For future work, one plan is to extend our system with more logic inference methods such as those using dynamic semantics (Haruta et al., 2020) and more neural inference methods such as those for commonsense reasoning (Levine et al., 2020). We also plan to implement a learning method that allows the system to learn from mistakes on a training dataset and automatically expand or correct its rules and knowledge bases, which is similar to Abzianidze (2020)'s work.

Acknowledgements

We thank the anonymous reviewers for their insightful comments. We also thank Dr. Michael Wollowski from Rose-hulman Institute of Technology for his helpful feedback on this paper.

References

Lasha Abzianidze. 2017. LangPro: Natural language theorem prover. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 115–120, Copenhagen, Denmark. Association for Computational Linguistics.

Lasha Abzianidze. 2020. Learning as abduction: Trainable natural logic theorem prover for natural language inference. In *Proceedings of the Ninth Joint Conference on Lexical and Computational Seman-*

tics, pages 20–31, Barcelona, Spain (Online). Association for Computational Linguistics.

I. Beltagy, Stephen Roller, Pengxiang Cheng, Katrin Erk, and Raymond J. Mooney. 2016. Representing meaning with a combination of logical and distributional models. *Computational Linguistics*, 42(4):763–808.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2017. Enhanced LSTM for natural language inference. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1657–1668, Vancouver, Canada. Association for Computational Linguistics.

Zeming Chen and Qiyue Gao. 2021. Monotonicity marking from universal dependency trees. *CoRR*, abs/2104.08659.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Max Glockner, 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)*, pages 650–655, Melbourne, Australia. Association for Computational Linguistics.

Suchin Gururangan, 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)*, pages 107–112, New Orleans, Louisiana. Association for Computational Linguistics.

Izumi Haruta, Koji Mineshima, and Daisuke Bekki. 2020. Combining event semantics and degree semantics for natural language inference. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 1758–1764, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Hai Hu, Qi Chen, Kyle Richardson, Atreyee Mukherjee, Lawrence S. Moss, and Sandra Kuebler. 2020. MonaLog: a lightweight system for natural language inference based on monotonicity. In *Proceedings of the Society for Computation in Linguistics 2020*, pages 334–344, New York, New York. Association for Computational Linguistics.

Hai Hu and Larry Moss. 2018. Polarity computations in flexible categorial grammar. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 124–129, New Orleans, Louisiana. Association for Computational Linguistics.

Aikaterini-Lida Kalouli, Richard Crouch, and Valeria de Paiva. 2020. Hy-NLI: a hybrid system for natural language inference. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 5235–5249, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Yoav Levine, Barak Lenz, Or Dagan, Ori Ram, Dan Padnos, Or Sharir, Shai Shalev-Shwartz, Amnon Shashua, and Yoav Shoham. 2020. SenseBERT: Driving some sense into BERT. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4656–4667, Online. Association for Computational Linguistics.

H. Liu and P. Singh. 2004. Conceptnet — a practical commonsense reasoning tool-kit. *BT Technology Journal*, 22(4):211–226.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Ro{bert}a: A robustly optimized {bert} pretraining approach.

Bill MacCartney and Christopher D. Manning. 2009. An extended model of natural logic. In *Proceedings of the Eighth International Conference on Computational Semantics (IWCS-8)*, Tilburg, Netherlands.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 216–223, Reykjavik, Iceland. European Language Resources Association (ELRA).

Pascual Martínez-Gómez, Koji Mineshima, Yusuke Miyao, and Daisuke Bekki. 2016. ccg2lambda: A compositional semantics system. In *Proceedings* *of ACL 2016 System Demonstrations*, pages 85–90, Berlin, Germany. Association for Computational Linguistics.

Pascual Martínez-Gómez, Koji Mineshima, Yusuke Miyao, and Daisuke Bekki. 2017. On-demand injection of lexical knowledge for recognising textual entailment. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 710–720, Valencia, Spain. Association for Computational Linguistics.

Tom McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3428–3448, Florence, Italy. Association for Computational Linguistics.

George A. Miller. 1995. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41.

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

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 101–108, Online. Association for Computational Linguistics.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. `http://is.muni.cz/publication/884893/en`.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Hitomi Yanaka, Koji Mineshima, Daisuke Bekki, Kentaro Inui, Satoshi Sekine, Lasha Abzianidze, and Johan Bos. 2019a. Can neural networks understand monotonicity reasoning? In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 31–40, Florence, Italy. Association for Computational Linguistics.

Hitomi Yanaka, Koji Mineshima, Daisuke Bekki, Kentaro Inui, Satoshi Sekine, Lasha Abzianidze, and Johan Bos. 2019b. HELP: A dataset for identifying shortcomings of neural models in monotonicity reasoning. In *Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (*SEM 2019)*, pages 250–255, Minneapolis, Minnesota. Association for Computational Linguistics.

Hitomi Yanaka, Koji Mineshima, Pascual Martínez-Gómez, and Daisuke Bekki. 2018. Acquisition of phrase correspondences using natural deduction proofs. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 756–766, New Orleans, Louisiana. Association for Computational Linguistics.

Wenpeng Yin and Hinrich Schütze. 2017. Task-specific attentive pooling of phrase alignments contributes to sentence matching. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 699–709, Valencia, Spain. Association for Computational Linguistics.

Amir Zeldes. 2017. The GUM corpus: Creating multilayer resources in the classroom. *Language Resources and Evaluation*, 51(3):581–612.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. CoNLL 2018 shared task: Multilingual parsing from raw text to Universal Dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium. Association for Computational Linguistics.