# Embedding WordNet Knowledge for Textual Entailment

**Yunshi Lan**
School of Information Systems
Singapore Management University
yslan.2015@phdis.smu.edu.sg

**Jing Jiang**
School of Information Systems
Singapore Management University
jingjiang@smu.edu.sg

## Abstract

In this paper, we study how we can improve a deep learning approach to textual entailment by incorporating lexical entailment relations from WordNet. Our idea is to embed the lexical entailment knowledge contained in WordNet in specially-learned word vectors, which we call "entailment vectors." We present a standard neural network model and a novel set-theoretic model to learn these entailment vectors from word pairs with known lexical entailment relations derived from WordNet. We further incorporate these entailment vectors into a decomposable attention model for textual entailment and evaluate the model on the SICK and the SNLI dataset. We find that using these special entailment word vectors, we can significantly improve the performance of textual entailment compared with a baseline that uses only standard word2vec vectors. The final performance of our model is close to or above the state of the art, but our method does not rely on any manually-crafted rules or extensive syntactic features.

## 1 Introduction

Recognizing textual entailment (RTE) is the task of determining whether a hypothesis sentence can be inferred from a given premise sentence. The task has been well studied since it was first introduced by Dagan et al. (2006). Recently, there has been much interest in applying deep learning models to RTE (Bowman et al., 2015; Rocktaschel et al., 2016; Wang and Jiang, 2016; Parikh et al., 2016; Sha et al., 2016). These models usually do not perform any linguistic analysis or require any feature engineering but have been shown to perform very well on this task.

Intuitively, lexical-level entailment relations should help sentence-level RTE. For example, Table 1 shows a premise and a hypothesis taken from the test data of the SICK dataset (Marelli et al., 2014). We can see that in this example the premise entails the hypothesis, and in order to correctly identify this relation, one has to know that the word *kettle* entails the word *pot*. However, if we train a neural network model on a set of labeled sentence pairs, and if the training dataset does not contain the word pair *kettle* and *pot* anywhere, it would be hard for the learned model to know that *kettle* entails *pot* and subsequently predict the relation between the premise and the hypothesis to be *entailment*. On the other hand, from WordNet we can easily find out that *pot* is a direct hypernym of *kettle* and therefore *kettle* should entail *pot*. If this kind of prior knowledge can be injected into a trained RTE model, then for sentence pairs with words that do not occur in the training data but can be found in WordNet, the RTE predictions could potentially be made easier.

| |
|---|
| Premise: *Someone is stirring chili in a* ***kettle****.* |
| Hypothesis: *Someone is stirring chili in a* ***pot****.* |

Table 1: An example of a pair of premise and hypothesis from SICK.

Indeed, WordNet (Miller, 1995) knowledge has been used to help RTE in a number of previous studies (Corley and Mihalcea, 2005; Beltagy et al., 2016; Martinez-Gomez et al., 2017). However, these





*Proceedings of the 27th International Conference on Computational Linguistics*, pages 270–281
Santa Fe, New Mexico, USA, August 20-26, 2018.

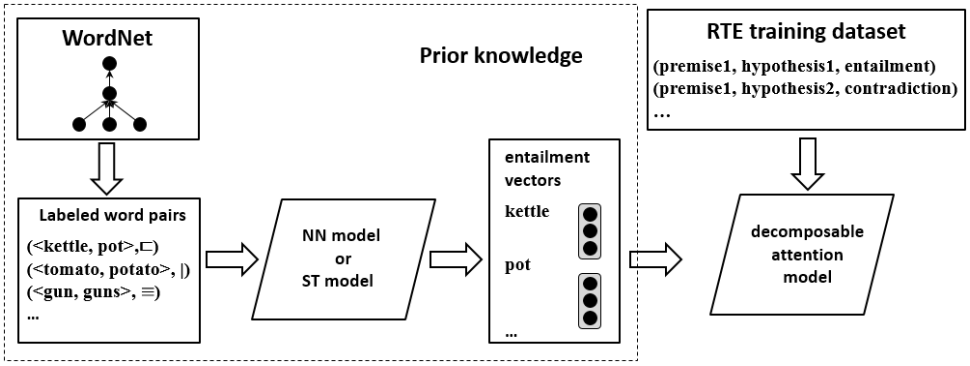



Figure 1: An overview of our approach.

previous studies were not based on neural network models, and thus their base models without using WordNet may not represent the state of the art or may require much human effort. For example, our baseline model based on a neural network model without using any WordNet knowledge can achieve an accuracy of 84.1% on the SICK test data, but the baseline model by Martinez-Gomez et al. (2017), which uses logical semantic representations and a theorem prover, can only achieve an accuracy of 76.7%. Given the advantages of deep learning approaches to RTE, it is therefore desirable to incorporate WordNet knowledge into deep learning based solutions to RTE. However, it is not immediately clear how WordNet knowledge could be easily brought into neural network models. To the best of our knowledge, there has not been any previous work in this direction.

In this paper, we propose to first embed lexical entailment knowledge contained in WordNet into word vectors. We call these special word vectors "entailment vectors." We then incorporate these entailment vectors into a recently proposed decomposable attention model for RTE. To learn these entailment vectors that encode lexical entailment relations, we can use a standard neural network. We also propose a set-theoretic model that has better theoretical justification. Our experiments show that using these entailment vectors learned from WordNet can indeed significantly improve the performance of RTE on the SICK dataset and the SNLI dataset. The performance of our method is also better compared with the state of the art on SICK.

## 2 Method

In this section, we present our method that learns the entailment vectors that encode prior knowledge of lexical entailment relations. We also present how we incorporate these entailment vectors into a neural network model for RTE. The overall framework of our method is illustrated in Figure 1.

### 2.1 Learning Entailment Vectors

We assume that our prior knowledge of lexical entailment relations is contained in word pairs and their entailment relations. Specifically, let $\mathcal{R}$ denote a set of lexical entailment relations. For example, $\mathcal{R}$ may contain *entailment* and *reverse-entailment*. Let $\mathcal{L} = \{(w_{i,1}, w_{i,2}, r_i)\}_{i=1}^{N}$ denote a set of $N$ word pairs together with their relation labels, where $w_{i,1}$ and $w_{i,2}$ are two words and $r_i \in \mathcal{R}$. For example, $\mathcal{L}$ may contain the triplet (<*pot*, *kettle*>, *reverse-entailment*). Let $\mathcal{V}$ denote the set of all unique words found in $\mathcal{L}$.

In order to encode the lexical entailment knowledge contained in $\mathcal{L}$, we propose to learn a dense vector for each word $w \in \mathcal{V}$ such that the entailment relation between two words in $\mathcal{V}$ can be easily detected from their word vectors. We refer to these dense word vectors as "entailment vectors." Note that these entailment vectors are different from commonly-used word embeddings such as GloVe and word2vec, because the entailment vectors are not learned from a large corpus and thus do not encode the distributional properties of words. They are learned from labeled word pairs in $\mathcal{L}$. We will show later that they can be used in combination with common word embeddings such as word2vec to help RTE.

In what follows, we first describe what lexical entailment relations we include in $\mathcal{R}$. We then present

| Name | Symbol | Set-theoretic definition | Example |
|---|---|---|---|
| (strict) entailment | $x_1 \sqsubset x_2$ | $x_1 \subset x_2$ | woman, person |
| (strict) reverse entailment | $x_1 \sqsupset x_2$ | $x_1 \supset x_2$ | person, woman |
| equivalence | $x_1 \equiv x_2$ | $x_1 = x_2$ | couch, sofa |
| alternation | $x_1 \mid x_2$ | $x_1 \cap x_2 = \varnothing \wedge x_1 \cup x_2 \neq \mathcal{D}$ | woman, man |
| negation | $x_1 \wedge x_2$ | $x_1 \cap x_2 = \varnothing \wedge x_1 \cup x_2 = \mathcal{D}$ | able, unable |
| cover | $x_1 \smile x_2$ | $x_1 \cap x_2 \neq \varnothing \wedge x_1 \cup x_2 = \mathcal{D}$ | person, non-woman |
| independence | $x_1 \# x_2$ | (else) | woman, doctor |

(a)

| | $c_{0,0}$ | $c_{0,1}$ | $c_{1,0}$ | $c_{1,1}$ |
|---|---|---|---|---|
| $x_1 \sqsubset x_2$ | – | $> 0$ | $= 0$ | – |
| $x_1 \sqsupset x_2$ | – | $= 0$ | $> 0$ | – |
| $x_1 \equiv x_2$ | – | $= 0$ | $= 0$ | – |
| $x_1 \mid x_2$ | $> 0$ | – | – | $= 0$ |
| $x_1 \wedge x_2$ | $= 0$ | – | – | $= 0$ |
| $x_1 \smile x_2$ | $= 0$ | – | – | $> 0$ |
| $x_1 \# x_2$ | $> 0$ | – | – | $> 0$ |

(b)

Table 2: (a) Seven basic semantic relations defined by MacCartney and Manning (2009). Here $\mathcal{D}$ is the universe that contains all entities. Each $x_1$ (or $x_2$) is a language unit that represents a subset of $\mathcal{D}$. (b) Criteria for different basic semantic relations based on the counters $c_{0,0}$, $c_{0,1}$, $c_{1,0}$ and $c_{1,1}$.

two different neural network models used to learn the entailment vectors from $\mathcal{L}$. We defer the description of how we derive labeled word pairs $\mathcal{L}$ from WordNet until Section 2.3.

**Lexical Entailment Relations**

Recall that eventually the entailment vectors will be used for textual entailment. In standard textual entailment datasets such as SICK and SNLI, there are three sentence-level entailment relations: *entailment*, *contradiction* and *neutral*. However, at word-level the entailment relations are different.

In a previous study, MacCartney and Manning (2009) developed a framework for natural language inference. As the basis of their framework, they defined seven basic semantic relations between language units, which we show in Table 2a. These relations cover all the possible relations between two language units $x_1$ and $x_2$. We can see that each relation has an equivalent set-theoretic definition, which dates back to Montague Semantics (Janssen, 2011). Each language unit $x_1$ or $x_2$ is modeled as a set, and is supposedly a subset of the universe $\mathcal{D}$. Therefore their relations can be determined by the relation between the two sets. For example, if the language units we consider are nouns, we can regard $\mathcal{D}$, the universe, as the set of all entities in the world. If $x_1$ is the word "person," we can regard $x_1$ as a set that contains all people. If $x_2$ is the word "woman," we can regard $x_2$ as a set that contains all people who are female. Since $x_1$ is a superset of $x_2$, based on the definition, $x_1$ reversely entails $x_2$, or $x_1 \sqsupset x_2$. This makes sense because "person" reversely entails "woman," or in other words, "woman" entails "person."

In this work, we only use a subset of these relations, namely, *entailment* ($\sqsubset$), *reverse-entailment* ($\sqsupset$), *alternation* ($|$) and *equivalence* ($\equiv$). In other words, $\mathcal{R} = \{\sqsubset, \sqsupset, |, \equiv\}$. We do not include *negation*, *cover* or *independence* because we find it hard to automatically derive word pairs with these relations from WordNet.

**Standard Neural Network Model**

To learn entailment vectors from $\mathcal{L}$, we first consider a standard neural network model. Let $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^d$ denote the entailment vectors of two words, which we are trying to learn, where $d$ is the number of dimensions of the vectors. We can combine the two vectors into a single vector as follows:

$$\mathbf{h} = \tanh(\mathbf{M}\begin{bmatrix}\mathbf{w}_1 \\ \mathbf{w}_2\end{bmatrix} + \mathbf{b}),$$

where $\mathbf{M} \in \mathbb{R}^{l \times 2d}$ is a weight matrix, $\mathbf{b} \in \mathbb{R}^l$ is a weight vector, $\tanh(\cdot)$ is applied element-wise to the vector, $\mathbf{h} \in \mathbb{R}^l$ is a hidden vector and $l$ is the number of dimensions of the hidden vector.

The hidden vector $\mathbf{h}$ can then go through a linear transform followed by a softmax layer to be used to predict the relation label $r$ between the two words:

$$p(r \mid \mathbf{h}) = \text{softmax}(\mathbf{M}'\mathbf{h} + \mathbf{b}'), \quad (1)$$

where $\mathbf{M}' \in \mathbb{R}^{k \times l}$ and $\mathbf{b}' \in \mathbb{R}^k$ denote a weight matrix and a bias vector, and $k = |\mathcal{R}|$.

Given all the labeled word pairs in $\mathcal{L}$, we can use the cross entropy loss as the objective function to learn the entailment vector for each $w \in \mathcal{V}$ as well as the various parameters above.

**Set-Theoretic Model**

Although the standard neural network model above is straightforward, it is hard to explain how the learned word vectors can encode the lexical entailment relations. Inspired both by the set-theoretic definitions of the basic semantic relations shown in Table 2a and by some recent work on formal distributional semantics (Grefenstette, 2013; Rocktaschel et al., 2014), we hypothesize that a good entailment vector for a word essentially encodes which elements in $\mathcal{D}$ are members of the set representing this word. We now present a novel set-theoretic model to learn the entailment vectors.

To illustrate our idea, we first show that in the extreme case when word vectors we try to learn are binary vectors precisely representing set memberships, the semantic relation between two words can be determined by comparing the two vectors element-wise. Specifically, let $D$ denote the size of the universe $\mathcal{D}$. Since a word $x$ can be regarded as a subset of $\mathcal{D}$, we assume that the entailment vector of $x$ is a $D$-dimensional binary vector $\mathbf{x}$, where $\mathbf{x}_i$ is 1 if the $i^{\text{th}}$ element in $\mathcal{D}$ is inside the set $x$ and 0 otherwise. Given two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ representing two words, in order to determine the relationship between them, we need to check the relationship between the two sets. To do so, we define the following counters for $p, q \in \{0, 1\}$:

$$c_{p,q} \quad = \quad \sum_{i=1}^{D} \delta(x_{1,i}, p)\delta(x_{2,i}, q),$$

where $\delta(s, t)$ is 1 if $s$ is equal to $t$ and 0 otherwise. Essentially if we compare $\mathbf{x}_1$ and $\mathbf{x}_2$ element-wise, then $c_{0,0}$, $c_{0,1}$, $c_{1,0}$ and $c_{1,1}$ count the number of times we see (0, 0), (0, 1), (1, 0) and (1, 1), respectively. It is not hard to show that the seven basic semantic relations in Table 2a correspond to different values of these $c_{p,q}$. For example, if $c_{0,1}$ is 0 and $c_{1,0}$ is 0, then the two sets (words) are equivalent. If $c_{0,1}$ is positive and $c_{1,0}$ is 0, then $\mathbf{x}_1$ is a subset of $\mathbf{x}_2$, and therefore the first word entails the second word. Table 2b shows the criteria for each basic semantic relation based on the values of these counters $c_{0,0}$, $c_{0,1}$, $c_{1,0}$ and $c_{1,1}$.

We can use a simple example to illustrate the idea above. Suppose the universe contains four elements: $\mathcal{D} = \{dog, cat, tiger, computer\}$. The word *animal* is a set that contains *dog*, *cat* and *tiger*, and thus can be represented by the vector $[1, 1, 1, 0]$. The word *pet* should contain only *dog* and *cat*, so it can be represented by the vector $[1, 1, 0, 0]$. In this case, the values of the four counters are the following: $c_{0,0} = 1$, $c_{0,1} = 0$, $c_{1,0} = 1$, and $c_{1,1} = 2$. According to Table 2b, we can determine that *animal* reversely entails *pet* based on the values of these counters.

Generally speaking, however, the number of elements in the universe is huge and therefore it is not feasible to learn word vectors that precisely represent the memberships of all these elements. In the following model we propose, we do not strictly force the entailment vectors to be binary. We also introduce a hidden layer that is deterministically derived from the entailment vectors. This hidden layer essentially represents $c_{0,0}$, $c_{0,1}$, $c_{1,0}$ and $c_{1,1}$, and it is used for the prediction of the relation between two words.

Specifically, let $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^d$ represent the entailment vectors corresponding to words $w_1$ and $w_2$. Next, we introduce two complementary vectors:

$$\bar{\mathbf{w}}_1 = \mathbf{1} - \mathbf{w}_1, \qquad \bar{\mathbf{w}}_2 = \mathbf{1} - \mathbf{w}_2,$$

where $\mathbf{1}$ is a $d$-dimensional vector of 1s.

We then define a hidden vector $\mathbf{h}$ to be a 4-dimensional vector as follows:

$$\mathbf{h} = \frac{1}{d}\begin{bmatrix} \mathbf{w}_1^\top \mathbf{w}_2 & \mathbf{w}_1^\top \bar{\mathbf{w}}_2 & \bar{\mathbf{w}}_1^\top \mathbf{w}_2 & \bar{\mathbf{w}}_1^\top \bar{\mathbf{w}}_2 \end{bmatrix}^\top. \tag{2}$$

We can see that this hidden vector $\mathbf{h}$ roughly correspond to the four counters $c_{0,0}$, $c_{0,1}$, $c_{1,0}$ and $c_{1,1}$ defined above if the entailment vectors are binary vectors. But since we do not have the binary constraint, $\mathbf{h}$ actually contains real values rather than integer values. We also perform normalization by dividing the counts by $d$.

The same as shown in Eqn. (1), we can use $\mathbf{h}$ to predict the relation label between two words through a linear transform and a softmax layer. We can then again use the cross entropy loss as the objective function to learn the entailment vectors and the model parameters.

### 2.2 Using Entailment Vectors for RTE

Given the entailment vectors learned from $\mathcal{L}$ using either the standard neural network model or our proposed set-theoretic model, we can use them in a neural network model for textual entailment. In this paper we use a slightly modified version of the decomposable attention model proposed by Parikh et al. (2016) because of its relative simplicity and good performance on the SNLI (Bowman et al., 2015) dataset.

The setup of the textual entailment task is as follows. We are given two input sentences: a premise $X = ((\mathbf{x}_1, \mathbf{x}'_1), (\mathbf{x}_2, \mathbf{x}'_2), \ldots, (\mathbf{x}_m, \mathbf{x}'_m))$ and a hypothesis $Z = ((\mathbf{z}_1, \mathbf{z}'_1), (\mathbf{z}_2, \mathbf{z}'_2), \ldots, (\mathbf{z}_n, \mathbf{z}'_n))$, where each $\mathbf{x}_i$ (or $\mathbf{z}_j$) is a standard word embedding such as the word2vec embedding of the $i^{\text{th}}$ (or $j^{\text{th}}$) word in the premise (or hypothesis), and $\mathbf{x}'_i$ (or $\mathbf{z}'_j$) is the newly-learned entailment vector of the $i^{\text{th}}$ (or $j^{\text{th}}$) word in the premise (or hypothesis), as described in Section 2.1.[1] The goal is to predict a label $y \in \{\textit{entailment}, \textit{contradiction}, \textit{neutral}\}$ from $X$ and $Z$.

**The Modified Decomposable Attention Model**

The decomposable attention model consists of the following three steps.

**Attend:** At this step, we derive attention weights from the word embeddings from $X$ and $Z$. We first define

$$e_{ij} = F(\mathbf{x}_i)^\top F(\mathbf{z}_j),$$
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'=1}^{n} \exp(e_{ij'})}, \qquad \beta_{ij} = \frac{\exp(e_{ij})}{\sum_{i'=1}^{m} \exp(e_{i'j})},$$

where $F(\cdot)$ is a standard single-layer feed-forward neural network with ReLU activations (Glorot et al., 2011). We then define

$$\tilde{\mathbf{x}}_i = \sum_{j=1}^{n} \alpha_{ij} \cdot \mathbf{z}_j, \qquad \tilde{\mathbf{x}}'_i = \sum_{j=1}^{n} \alpha_{ij} \cdot \mathbf{z}'_j,$$
$$\tilde{\mathbf{z}}_j = \sum_{i=1}^{m} \beta_{ij} \cdot \mathbf{x}_i, \qquad \tilde{\mathbf{z}}'_j = \sum_{i=1}^{m} \beta_{ij} \cdot \mathbf{x}'_i.$$

$\tilde{\mathbf{x}}_i$ is a weighted version of the word embeddings from $Z$ to match $\mathbf{x}_i$, and similarly $\tilde{\mathbf{x}}'_i$ is a weighted version of the entailment vectors from $Z$ to match $\mathbf{x}'_i$. The same idea applies to $\tilde{\mathbf{z}}_j$ and $\tilde{\mathbf{z}}'_j$.

**Compare:** At this step, another single-layer feed-forward neural network $G$ with ReLU activations is used to compare the aligned words. Normally, without considering the entailment vectors, the comparison is done as follows:

$$\mathbf{v}_{1,i} = G(\mathbf{x}_i \oplus \tilde{\mathbf{x}}_i), \qquad \mathbf{v}_{2,j} = G(\mathbf{z}_j \oplus \tilde{\mathbf{z}}_j),$$

where $\oplus$ is concatenation, and $\mathbf{v}_{1,i}$ and $\mathbf{v}_{2,j}$ represent the comparison results for the $i$-th word in $X$ and the $j$-th word in $Z$, respectively.

When we do consider the entailment vectors $\mathbf{x}'_i$ and $\mathbf{z}'_j$, as well as their counterparts $\tilde{\mathbf{x}}'_i$ and $\tilde{\mathbf{z}}'_j$ from the **Attend** step, we need to perform the comparison slightly differently. Here we use two different ways to perform the comparison, depending on whether the entailment vectors are learned using the standard neural network model or the set-theoretic model. If the standard neural network model is used to learn the entailment vectors, we simply concatenate all the vectors to perform comparison as follows:

$$\mathbf{v}_{1,i} = G(\mathbf{x}_i \oplus \tilde{\mathbf{x}}_i \oplus \mathbf{x}'_i \oplus \tilde{\mathbf{x}}'_i), \qquad \mathbf{v}_{2,j} = G(\mathbf{z}_j \oplus \tilde{\mathbf{z}}_j \oplus \mathbf{z}'_j \oplus \tilde{\mathbf{z}}'_j).$$

[1] For those words which do not appear in $\mathcal{L}$ (the set of unique words in the training data), we randomly generate their entailment vectors in the range $(-0.1, 0.1)$.

However, if the entailment vectors are learned using the set-theoretic model, because of the special properties of the entailment vectors as explained in Section 2.1, we perform comparison as follows:

$$\mathbf{v}_{1,i} = G(\mathbf{x}_i \oplus \tilde{\mathbf{x}}_i \oplus (\mathbf{x}'_i \odot \tilde{\mathbf{x}}'_i) \oplus (\mathbf{x}'_i \odot (1 - \tilde{\mathbf{x}}'_i)) \oplus ((1 - \mathbf{x}'_i) \odot \tilde{\mathbf{x}}'_i) \oplus ((1 - \mathbf{x}'_i) \odot (1 - \tilde{\mathbf{x}}'_i))),$$
$$\mathbf{v}_{2,j} = G(\mathbf{z}_j \oplus \tilde{\mathbf{z}}_j \oplus (\mathbf{z}'_j \odot \tilde{\mathbf{z}}'_j) \oplus (\mathbf{z}'_j \odot (1 - \tilde{\mathbf{z}}'_j)) \oplus ((1 - \mathbf{z}'_j) \odot \tilde{\mathbf{z}}'_j) \oplus ((1 - \mathbf{z}'_j) \odot (1 - \tilde{\mathbf{z}}'_j))),$$

where $\odot$ is element-wise multiplication of two vectors.

**Aggregate:** Given $\mathbf{v}_{1,i}$ $(1 \le i \le m)$ and $\mathbf{v}_{2,j}$ $(1 \le j \le n)$ as defined above, we first use a standard LSTM model to process the two sequences separately, and then we use MaxPooling to aggregate the derived hidden vectors in order to obtain a single vector for each sequence:

$$\mathbf{h}_{1,i} = \text{LSTM}(\mathbf{v}_{1,i}, \mathbf{h}_{1,i-1}), \qquad \mathbf{h}_{2,j} = \text{LSTM}(\mathbf{v}_{2,j}, \mathbf{h}_{2,j-1}),$$
$$\mathbf{v}_1 = \text{MaxPooling}(\mathbf{h}_{1,1}, \mathbf{h}_{1,2}, \ldots, \mathbf{h}_{1,m}), \qquad \mathbf{v}_2 = \text{MaxPooling}(\mathbf{h}_{2,1}, \mathbf{h}_{2,2}, \ldots, \mathbf{h}_{2,n}).$$

We then use $\mathbf{v}_1$ and $\mathbf{v}_2$ to make the final prediction:

$$p(y|\mathbf{v}_1, \mathbf{v}_2) = \text{softmax}(\mathbf{W}(\mathbf{v}_1 \oplus \mathbf{v}_2) + \mathbf{c}),$$

where $\mathbf{W}$ and $\mathbf{c}$ are parameters to be learned.

### 2.3 Implementation Details

In this section, we describe how we derive labeled word pairs from WordNet to construct $\mathcal{L}$. Because eventually we will test the learned entailment vectors on the SICK and SNLI textual entailment datasets, we focus on extracting word pairs with at least one word appearing in SICK or SNLI.

Specifically, we follow the following steps to construct $\mathcal{L}$. In all the steps, $w_1$ and $w_2$ refer to nouns found in WordNet.

- If the synset of $w_1$ is a hypernym of the synset of $w_2$, we add $(w_1, w_2, \sqsupset)$ and $(w_2, w_1, \sqsubset)$ to $\mathcal{L}$.
- If $w_1$ and $w_2$ are in the same synset, or $w_1$ $(w_2)$ is the plural form of $w_2$ $(w_1)$, we add $(w_1, w_2, \equiv)$ and $(w_2, w_1, \equiv)$ to $\mathcal{L}$.
- If the synsets of $w_1$ and $w_2$ share the same parent synset, we add $(w_1, w_2, |)$ and $(w_2, w_1, |)$ to $\mathcal{L}$.
- We remove word pairs from $\mathcal{L}$ where neither word occurs in the SICK or the SNLI dataset.

## 3 Experiments

### 3.1 Direct Evaluation of Entailment Vectors

As a first step to evaluate our method, we first test whether our entailment vectors learned from labeled word pairs can indeed encode lexical entailment relations. To do this, we report the prediction accuracy on $\mathcal{L}$. Recall that $\mathcal{L}$ is derived from nouns in WordNet. In total we have 13,029 unique words and 67,935 labeled word pairs in $\mathcal{L}$. The four entailment relations are quite evenly distributed except for alternation, which has roughly twice as many word pairs as the other relations.

We compare the entailment vectors learned from the standard neural network model and from the set-theoretic model. We refer to these two as **NN** and **ST**. In addition, we also consider one baseline method, which train an SVM with RBF kernel on the word2vec embeddings of two words to predict their semantic relation.[2] We refer to this baseline as **word2vec**.

For all the methods, we take two thirds of $\mathcal{L}$ for training/validation and the remaining one third for testing. We repeat this three times and report the average accuracies. We tune the hyperparameters as follows: The dimensionality of the entailment vectors is chosen from $\{25, 50, 100, 200\}$ and the learning rate from $\{0.05, 0.1, 0.15\}$. We applied stochastic gradient descent with a mini-batch size of 5 for optimization.

[2]We have also tried to use a standard neural network model instead of SVM on the word2vec embeddings, and the performance is 68.5%, which is similar to using SVM.

| | Acc±SD | F1±SD | $d$ | Acc(CS) | F1(CS) |
|---|---|---|---|---|---|
| NN | 90.19±0.005 | 89.24±0.458 | 50 | 13.35 | 23.56 |
| ST | **94.51**±0.002 | **93.96**±0.106 | 200 | **93.32** | **96.55** |
| word2vec | 69.01±0.002 | 67.70±0.354 | 300 | 53.00 | 69.28 |

(a)

| Dataset | SICK(ss) | SICK(ns) | SNLI(ss) | SNLI(ns) |
|---|---|---|---|---|
| train | 4439 | 7163 | 549367 | 460451 |
| dev | 485 | 795 | 9842 | 51695 |
| test | 4906 | 1882 | 9824 | 61054 |

(b)

| Model | SICK (ss) | | SICK (ns) | | SNLI (ss) | | SNLI (ns) | |
|---|---|---|---|---|---|---|---|---|
| | Dev | Test | Dev | Test | Dev | Test | Dev | Test |
| w2v | 84.2 | 84.1 | 85.4 | 81.8 | 86.5 | 86.2 | 84.7 | 82.8 |
| w2v+NN | 84.4 | 85.1* | **86.3** | 84.9** | 86.6 | 86.3 | 85.0 | 83.3** |
| w2v+ST | **84.6** | **85.4**** | 85.8 | **85.1**** | **86.7** | 86.5 | **85.3** | **83.6**** |
| MLN-eclassif | - | 85.1 | - | - | - | - | - | - |
| DAM | - | - | - | - | - | 86.3 | - | - |
| CAFE | - | - | - | - | - | **88.5** | - | - |

(c)

Table 3: (a) Results of direct evaluation on $\mathcal{L}$. $d$ is the optimal dimensionality of the entailment vectors (for NN and ST) or of the word embeddings (for word2vec) based on validation. SD stands for standard deviation. (b) Numbers of sentence pairs in the SICK and the SNLI datasets. (c) Textual entailment results on the two datasets by different methods. MLN-eclassif (Beltagy et al., 2016) and CAFE (Tay et al., 2018) represent the state of the art on the SICK and the SNLI datasets, respectively. DAM (Beltagy et al., 2016) is the original decomposable attention model. The numbers shown in the table are what was reported in these papers. ** and * indicate statistical significance ($p \leq 0.01$) and ($p \leq 0.05$) compared with w2v by McNemars test, respectively.

We show the results in terms of accuracy and F1 in Table 3a. We can observe the following from the table. (1) The entailment vectors give better accuracies for predicting the entailment relations compared with using word2vec embeddings. (2) Between the standard neural network model and our set-theoretic model, the set-theoretic model achieves better performance.

Levy et al. (2015) previously showed that for some models using word embedding vectors to identify hypernyms, the model is essentially memorizing "prototypical" hypernyms. In order to check whether this happens to our model, we create a special set of labeled word pairs in which we introduce some "confusing" word pairs. We use the following rule to generate these word pairs: If we know ($\langle w_2, w_1 \rangle, \sqsubset$ ) and ($\langle w_1, w_3 \rangle, |$), then we can infer that ($\langle w_2, w_3 \rangle, |$). We obtain 719 labeled word pairs in this way. For example, ($\langle$*staple gun, musical instrument*$\rangle, |$) is one in this set. We denote this dataset as confusion set (CS). The accuracy and F1 scores are shown in the last two columns of Table 3a, respectively. We can see that the entailment vectors learned using the standard neural network models perform very poorly on this special set, suggesting that the learned entailment vectors may be memorizing the prototypical hypernyms. Using word2vec embeddings performs poorly, too. However, entailment vectors learned from the set-theoretic model perform very well, suggesting that the set-theoretic model we proposed does not simply learn "prototypical" hypernyms.

### 3.2 Evaluation on Textual Entailment

In this section, we evaluate whether the learned entailment vectors can help RTE. Here the entailment vectors are trained using the entire set of labeled word pairs derived from WordNet as we have described in Section 2.3. We use the SICK dataset and SNLI dataset for this experiment.

Recall that we have earlier hypothesized that when there are many word pairs in the test data that are not found in the training data, external lexical entailment knowledge about these word pairs is especially important. In order to verify this hypothesis, we also construct a different split of the data. We use the following steps to build this split:

- For every sentence pair, find all possible matching word pairs that occur in the WordNet data. E.g., we extract ⟨someone, someone⟩, ⟨stirring, stirring⟩, ⟨chili, chili⟩ and ⟨kettle, pot⟩ from the sentence pair in Table 1.
- For every word pair, randomly assign all sentence pairs containing such word pair into either the

training set or the test set. This is to ensure that the test set contains word pairs that have not appeared in the training data at all.
- Randomly split the training set into the final training set and a development set with a ratio of 9:1.

We denote the original split of the data as the "standard split" (ss) and the new split of the data as the "non-overlap split" (ns). This results in two additional specific-split datasets. Table 3b shows the numbers of sentence pairs of all the four datasets.

We compare the following settings of using entailment vectors in the decomposable attention model for RTE:

- w2v: This is the original decomposable attention model for RTE without using the entailment vectors. The word embeddings used are word2vec.
- w2v+NN: This setting uses the modified decomposable attention model with both word2vec word embeddings and the entailment vectors learned from the standard neural network model.
- w2v+ST: This setting uses the modified decomposable attention model with both word2vec word embeddings and the entailment vectors learned from the set-theoretic model.

The SICK dataset has a vocabulary of 2,331 words, 1,560 of which can be found in the WordNet dataset. For the SNLI dataset, the vocabulary contains 32,497 words, of which only 3,599 appear in the WordNet. During training, we experiment with 2-layer feed-forward neural networks with 200 neurons in all sets. Learning rate is set to be 0.03 and batch size is 30. In order to better compare different models, we use the McNemar test to test the statistical significance of the performance differences between different models.

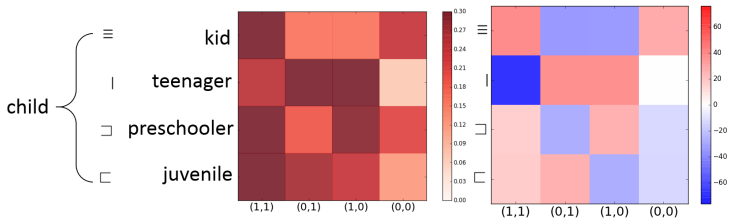
We show the experiment results in Table 3c. We have the following observations. (1) The result of our baseline w2v on the standard split of the SNLI dataset is almost the same as the one reported by Parikh et al. (2016). This shows that our baseline implement is as strong as the DAM model. (2) For the standard split of the datsets, both w2v+NN and w2v+ST can outperform w2v, and w2v+ST's accuracy is significantly better than w2v. In particular, our model outperforms the state-of-the-art performance on the SICK dataset. Even though the performance on SNLI is not better than the best result reported (Tay et al., 2018), where extensive syntactic features were used, we still can outperform the basic model (Parikh et al., 2016) by injecting WordNet knowledge into the model. (3) For the non-overlap split, especially on the SICK dataset, using only word2vec, the RTE performance on the development set is quite high, but when it comes to the test set, the performance drops drastically. This demonstrates that without observing the necessary word pairs from the training set, it is hard to make the right predictions on the test set. In contrast, using entailment vectors performs better in the non-overlap split setting. For the SNLI dataset, the improvement is not so obvious. This may be because SNLI has a low percentage of words that can be found in WordNet. (4) Comparing w2v+NN and w2v+ST, we can see that w2v+ST performs slightly better than w2v+NN most of the time. This shows the advantage of our set-theoretic model for learning the entailment vectors.

### 3.3 Further Analyses

We also conduct some further analyses on the results to better understand our method.

**Analysis of the Set-Theoretic Model**

Recall that for the set-theoretic model, the hidden vectors $\mathbf{h}$ as defined in Eqn. (2) roughly correspond to the four counters $c_{p,q}$, and these counters are essential to how the set-theoretic model works. To see whether the set-theoretic model indeed works as we have expected, we plot the values of $\mathbf{h}$ for some example word pairs. Figure 2a shows the $\mathbf{h}$ vectors for four word pairs from WordNet with different lexical entailment relations. Figure 2b shows the weights applied to $\mathbf{h}$ as defined in Eqn. (1) to make the final predictions of lexical entailment relations. We can see in Figure 2a that for the word pair $(\langle \text{child, juvenile} \rangle, \sqsubset)$, it has a large value for the counter $c_{0,1}$, and this counter has a positive weight for relation $\sqsubset$ in Figure 2b. On the other hand, for $\sqsupset$, the counter $c_{1,0}$ has a positive weight. Also, $|$ needs positive weights for $c_{1,0}$ and $c_{0,1}$ while $\equiv$ needs negative weights for them. All these match our expectations from Table 2b.




(a) (b)

Figure 2: (a) Visualization of the 4-dimensional $\mathbf{h}$ for four types of relations with the corresponding word pairs labeled on the left and the meaning of the 4 dimensions labeled at the bottom. (b) Visualization of the weights applied to $\mathbf{h}$ with the corresponding relations labeled on the left and the meanings of the 4 dimensions labeled at the bottom.

| | w2v | w2v+ST(concrete) | w2v+ST(abstract) | w2v+ST |
|---|---|---|---|---|
| SICK(ns) | 81.8% | 84.4% | 84.0% | 85.1% |

Table 4: Evaluations of concrete concepts and abstract concepts on SICK(ns) data.

**Analysis of Concrete and Abstract concepts**

In Section 2.1, we show that our entailment vectors are mainly inspired by the set-theoretic definitions of basic semantic relations. Intuitively they work mostly for nouns presenting concrete concepts such as "animal" and "dog." It would be interesting to see how well they work for abstract nouns such as "idea" and "proposal." In order to check this, we conduct the following analysis.

We make use of an existing dataset[3] that contains concreteness ratings for 40,000 common English lemmas, collected using Amazon Mechanical Turk (Brysbaert et al., 2014). The ratings are from 0 to 5. We select words with scores 4.5 and higher to form our concrete concept list and words with scores 2.5 and lower to form another abstract concept list. This results in 596 concrete words and 250 abstract words. We then re-run the textual entailment experiments on the SICK data with the "non-overlap split" (ns). But instead of incorporating the entailment vectors of all words found in $\mathcal{L}$, we try two new settings, where we incorporate only the entailment vectors of the words found in the concrete word list or the abstract word list. We use entailment vectors learned from the set-theoretic model. We refer to these two new settings as w2v+ST(concrete) and w2v+ST(abstract). We show the performance in terms of accuracy in Table 4. For comparison, we also show w2v, which is our baseline that does not use any entailment vectors, and w2v+ST, which uses entailment vectors of all words. As we can see from the table, incorporating entailment vectors of concrete words works slightly better than of abstract words, but both settings perform better than the baseline w2v. This shows that even for abstract concepts, our model is still able to capture lexical-level entailment knowledge and incorporate such knowledge into the textual entailment model. Overall, incorporating entailment vectors of all words still works the best.

**Case Studies**

In Table 5 we show some successful as well as erroneous RTE predictions made by our method using the entailment vectors trained by the set-theoretic model. In the first example, since the words *kettle* and *pot* never co-occur in the training set, using only word2vec embeddings makes a wrong prediction. But injecting the entailment vectors trained using the set-theoretic model from the WordNet labeled word pairs, the knowledge that *kettle* entails *pot* is brought into the model, and therefore the w2v+ST method can correctly make a prediction. We can see that the w2v+NN method could not inject such knowledge either. The same thing happens in the next example, where WordNet provides the knowledge $(\langle \textit{inside, outside} \rangle, |)$, which is never learned from the training set.

We also provide two negative examples where w2v+ST makes wrong predictions. In the third example, the knowledge $(\langle \textit{lawn, field} \rangle, \sqsubset)$ provided by WordNet is not applicable, because here *lawn* and *field* are

[3]Available at `http://crr.ugent.be/archives/1330`

| ID | sentence pair | ground truth | w2v | w2v+NN | w2v+ST | WordNet triplet |
|---|---|---|---|---|---|---|
| 2727 | Someone is stirring chili in a kettle<br>Someone is stirring chili in a pot | *entailment* | *neutral* | *neutral* | *entailment* | $(\langle$*kettle, pot*$\rangle, \sqsubset)$ |
| 4633 | The black and white dog is running inside<br>The black and white dog is running outside | *contradiction* | *neutral* | *neutral* | *contradiction* | $(\langle$*inside, outside*$\rangle, \mid)$ |
| 4149 | A shirtless man is playing football on a lawn<br>A shirtless man is playing football on a field | *neutral* | *entailment* | *entailment* | *entailment* | $(\langle$*lawn, field*$\rangle, \sqsubset)$ |
| 686 | A man is hanging up the phone<br>A man is making a call on a cell phone | *contradiction* | *neutral* | *neutral* | *neutral* | |

Table 5: Examples of successful and erroneous predictions.

considered to be different. The last example shows that for paraphrases not at the lexical level, our entailment vectors cannot help.

## 4 Related Work

**Lexical entailment:** Our work is related to recognizing lexical entailment. Many existing methods for lexical entailment are based on the distributional inclusion hypothesis and leverage co-occurrence information of words from a large corpus (Weeds et al., 2004; Geffet and Dagan, 2005; Kotlerman et al., 2010; Bernardi et al., 2012). In contrast, our work uses labeled word pairs derived from WordNet to learn entailment vectors to encode the lexical entailment relations. The semantic relations we consider are also designed specifically for natural language inference.

**Word embeddings:** Recently, neural-network-based approaches have been developed to learn vector representations of words (Mikolov et al., 2013; Pennington et al., 2014). These word embeddings are trained from an unlabeled large corpus and for general usage. In contrast, our entailment word vectors are not meant to be used as general-purpose word embeddings. They are designed specifically for incorporating external knowledge from WordNet into neural network models for RTE. Although there has been some work attempting to inject external lexical knowledge into word embeddings (Faruqui et al., 2015; Chen et al., 2015), they have not investigated the usage of these enhanced word embeddings for RTE.

**Textual entailment with lexical knowledge:** Many studies have attempted to inject lexical knowledge into the RTE task. Balkir et al. (2015) proposed compositional distributional models to extend representations from words to sentences. By mapping from a premise to a hypothesis by dependency trees or proposition extraction and then comparing lexical entailment between nodes or words, Herrera et al. (2005) and Ofoghi and Yearwood (2009) tried to solve RTE via WordNet. Beltagy et al. (2013), Beltagy et al. (2016) and Martinez-Gomez et al. (2017) first changed sentences to logical forms and then leveraged logic inference engines combined with lexical entailment axioms to make entailment judgments. But we are not aware of any work on how to apply lexical knowledge to textual entailment based on neural networks.

## 5 Conclusions

In this paper, we propose to learn entailment vectors that encode lexical entailment knowledge from WordNet. We incorporate such entailment vectors into a decomposable attention model for RTE. Our empirical evaluation shows that the entailment vectors can indeed improve the performance of RTE when evaluated on the SICK and the SNLI datasets. In the future, we plan to study how to automatically obtain more training data for learning entailment vectors and how to encode phrase-level entailment relations in such entailment vectors. Our data and code have been made publicly available.[4]

## Acknowledgment

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its International Research Centres in Singapore Funding Initiative.

[4] https://github.com/lanyunshi/embedding-for-textual-entailment

## References


Esma Balkir, Dimitri Kartsaklis, and Mehrnoosh Sadrzadeh. 2015. Sentence entailment in compositional distributional semantics. In *Proceedings of Conference on International Symposium on Artificial Intelligence and Mathematics*.

Islam Beltagy, Cuong Chau, Gemma Boleda, Dan Garrette, Katrin Erk, and Raymond Mooney. 2013. Montague meets Markov: Deep semantics with probabilistic logical form. In *Proceedings of the Joint Conference on Lexical and Computational Semantics*.

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

Macro Baroni Raffaella Bernardi, Ngoc-Quynh Do, and Chung chieh Shan. 2012. Entailment above the word level in distributional semantics. In *Proceedings of Conference of the European Chapter of the ACL*.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*.

Marc Brysbaert, Amy Beth Warriner, and Victor Kuperman. 2014. Concreteness ratings for 40 thousand generally known english word lemmas. *Behavior Research Methods*, 46:904–911.

Zhigang Chen, Wei Lin, Qian Chen, Xiaoping Chen, Si Wei, Hui Jiang, and Xiaodan Zhu. 2015. Revisiting word embedding for contrasting meaning. In *Proceedings of Annual Conference of the Association for Computational Linguistics*.

Courtney Corley and Rada Mihalcea. 2005. Measuring the semantic similarity of texts. In *Proceedings of ACL Workshop on Empirical Modeling of Semantic Equivalence and Entailment*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The PASCAL recognising textual entailment challenge. *Machine Learning Challenges*, 3944:177–190.

Manaal Faruqui, Jesse Dodge, Sujay K.Jauhar, Chris Dyer Eduard Hovy, and Noah A.Smith. 2015. Retrofitting word vectors to semantic lexicons. In *Proceedings of Conference of the North American Chapter of the ACL*.

Maayan Geffet and Ido Dagan. 2005. The distributional inclusion hypotheses and lexical entailment. In *Proceedings of Annual Conference of the Association for Computational Linguistics*.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of International Conference on Artificial Intelligence and Statistics*.

Edward Grefenstette. 2013. Towards a formal distributional semantics: Simulating logical calculi with tensors. In *Proceedings of Joint Conference on Lexical and Computational Semantics*.

Jesus Herrera, Anselmo Penas, and Felisa Verdejo. 2005. Textual entailment recognition based on dependency analysis and WordNet. In *Proceedings of the 1st PASCAL Recognising Textual Entailment*.

Theo M. V. Janssen. 2011. Montague semantics. The Stanford Encyclopedia of Philosophy. `http://plato.stanford.edu/archives/win2011/entries/montague-semantics/`.

Lili Kotlerman, Ido Dagan, Idan Szpektor, and Maayan Zhitomirsky-Geffet. 2010. Directional distributional similarity for lexcial inference. *Natural Language Engineering*, 16:359–389.

Omer Levy, Steffen Remus, Chris Biemann, and Ido Dagan. 2015. Do supervised distributional methods really learn lexical inference relations? In *Proceedings of Conference of the North American Chapter of the ACL*.

Bill MacCartney and Christopher D. Manning. 2009. An extended model of natural logic. In *Proceedings of International Conference on Computational Semantics*.

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 International Conference on Language Resources and Evaluation*.

Pascual Martinez-Gomez, Koji Mineshima, Yusuke Miyao, and Daisuke Bekki. 2017. On-demand injection of lexical knowledge for recognising textual entailment. In *Proceedings of Conference of the European Chapter of the ACL*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Proceedings of International Conference on Learning Representations*.

George A. Miller. 1995. WordNet: A lexical database for english. *Communications of the ACM*, 38:39–41.

Bahadorreza Ofoghi and John Yearwood. 2009. From lexical entailment to recognizing textual entailment using linguistic resources. In *Proceedings of the Australasian Language Technology Association Workshop*.

Ankur P. Parikh, Osar Taskstrom, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*.

Tim Rocktaschel, Matko Bosnjak, Sameer Singh, and Sebastian Riedel. 2014. Low-dimensional embeddings of logic. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*.

Tim Rocktaschel, Edward Grefenstette, Karl Moritz Hermann, Tomas Kocisky, and Phil Blunsom. 2016. Reasoning about entailment with neural attention. In *Proceedings of International Conference on Learning Representations*.

Lei Sha, Baobao Chang, Zhifang Sui, and Sujian Li. 2016. Reading and thinking: Re-read LSTM unit for textual entailment recognition. In *Proceedings of the 26th International Conference on Computational Linguistics*.

Y. Tay, L. A. Tuan, and S. Cheung Hui. 2018. A Compare-Propagate Architecture with Alignment Factorization for Natural Language Inference. *ArXiv e-prints: 1801.00102*.

Shuohang Wang and Jing Jiang. 2016. Learning natural language inference with LSTM. In *Proceedings of Conference of the North American Chapter of the ACL.*

Julie Weeds, David Weir, and Diana McCarthy. 2004. Characterising measures of lexical distributional similarity. In *Proceedings of International Conference on Computational Linguistics*.