Finding The Best Model Among Representative Compositional Models

Masayasu Muraoka[†] **Sonse Shimaoka**[‡] **Kazeto Yamamoto**[†]
Yotaro Watanabe[†] **Naoaki Okazaki**[†*] **Kentaro Inui**[†]

Tohoku University[†‡]
Japan Science and Technology Agency (JST)[*]

{muraoka,kazeto,yotaro-w,okazaki,inui}
@ecei.tohoku.ac.jp[†]
simaokasonse@yahoo.co.jp[‡]

Abstract

The field of distributional-compositional semantics has yielded a range of computational models for composing the vector of a phrase from those of constituent word vectors. Existing models have various ranges of their expressiveness, recursivity, and trainability. However, these models have not been examined closely for their compositionality. We implement and compare these models under the same conditions. The experimentally obtained results demonstrate that the model using different composition matrices for different dependency relations achieved state-of-the-art performance on a dataset for two-word compositions (Mitchell and Lapata, 2010).

1 Introduction

Computing the meaning of a text has posed a challenge in NLP for many years. Based on the distributional hypothesis (Firth, 1957), the meaning of a word is typically represented as a real-valued vector, with elements representing the frequencies of words that co-occur in the context of the word in a corpus. Numerous studies have demonstrated learned word vectors from a large text corpus (Bullinaria and Levy, 2007; Collobert and Weston, 2008; Turney and Pantel, 2010; Mnih and Kavukcuoglu, 2013; Mikolov et al., 2013).

In contrast, the same approach is not scalable to a complex linguistic unit (e.g., phrase or sentence) because of the data sparseness problem: the longer the length of a phrase, the fewer times the phrase occurs in a corpus. For this reason, we cannot acquire semantic information reliably from co-occurrence statistics of a phrase. Recently, numerous studies have explored compositional semantics, in which the meaning of a phrase, clause, or sentence is computed from those of its constituents (Mitchell and Lapata, 2008; Mitchell and Lapata, 2010; Guevara, 2010; Zanzotto et al., 2010; Socher et al., 2011; Baroni et al., 2012; Socher et al., 2012; Socher et al., 2013a; Socher et al., 2014). These studies mostly address theories and methods for computing a vector of a phrase from the vectors of its constituents; the simplest but effective approach is to take the average of the two input vectors.

A simple approach such as additive and multiplicative compositions has been a strong baseline over more complex models (Blacoe and Lapata, 2012; Socher et al., 2013b). However, Erk and Padó (2008) argued the importance of syntax relations: the simple additive/multiplicative approach yields the same vector for phrases *a horse draws* and *draw a horse*, ignoring the syntactic structure by which *horse* in the former phrase is a subject whereas *horse* in the latter is the object. They formulated a generalized composition function including such a composition. However, this generalized composition is too complex to learn. These models usually do not work well for now.

As described in this paper, through a human-correlation experiment, we explore the most useful model among the representative models that have been proposed to date in terms of the semantic composition. We cast the task of learning composition matrices, which are model parameters, to minimize the errors between phrase vectors composed

28th Pacific Asia Conference on Language, Information and Computation pages 65–74

by matrices and computed in a corpus. The experimentally obtained results demonstrate that the model using different composition matrices for different dependency relations achieved state-of-the-art performance for a dataset for two-word compositions (Mitchell and Lapata, 2010). Moreover, the results confirm the effectiveness of syntax-sensitive compositions.

The remainder of the paper is organized as follows. Section 2 presents a survey of the previous studies and their issues. Section 3 describes details of the methods and the training procedure. Section 4 reports and discusses the experimentally obtained results. We conclude this paper in Section 5.

2 Previous Work

In this section, we briefly overview representative methods for obtaining vector representations of word meanings. We then describe the previous work that composes the meaning of a phrase from its constituents, followed by the issues and limitations that arise in this work.

2.1 Obtaining word vectors

In distributional semantics, the meaning of a word is represented by a vector, i.e., a point in d-dimensional space. We can classify the previous studies for obtaining word vectors into two groups: approaches based on *co-occurrence statistics* and *language modeling*.

The former approach (Bullinaria and Levy, 2007; Mitchell and Lapata, 2010) counts the frequency of words co-occurring with a target word in a corpus, and refines the statistics using, for example, Pointwise Mutual Information (PMI). Vectors obtained using this method are high-dimensioned and sparse. Therefore, some methods compress vectors using a dimension reduction method such as Principal Component Analysis (PCA) and Non-negative Matrix Factorization (NMF).

The latter approach (Collobert and Weston, 2008; Mnih and Kavukcuoglu, 2013; Mikolov et al., 2013) formalizes the task of learning word vectors as a byproduct of a language model (Bengio et al., 2003), i.e., finding word vectors such that each word vector can be predicted from surrounding words. In these studies, word vectors are initialized by random values and are learned through back propagation on a neural network.

Table 1: Summary of the previous models. Vectors $\boldsymbol{u}$, $\boldsymbol{v} \in \mathbb{R}^d$ present input (word) vectors, σ is an activation function (e.g., sigmoid function and $tanh$). In general, the more parameters a model has, the greater the expressive power the model has during vector compositions.

Model	Function	Parameters
Add	$w_1\boldsymbol{u} + w_2\boldsymbol{v}$	$w_1, w_2 \in \mathbb{R}$
Fulladd	$W\begin{bmatrix}\boldsymbol{u}\\ \boldsymbol{v}\end{bmatrix}$	$W \in \mathbb{R}^{d\times 2d}$
RNN	$\sigma\left(W\begin{bmatrix}\boldsymbol{u}\\ \boldsymbol{v}\end{bmatrix}\right)$	$W \in \mathbb{R}^{d\times 2d}$
Lexfunc	$A_u\boldsymbol{v}$	$A_u \in \mathbb{R}^{d\times d}$
Relfunc	$\sigma\left(W_r\begin{bmatrix}\boldsymbol{u}\\ \boldsymbol{v}\end{bmatrix}\right)$	$W_r \in \mathbb{R}^{d\times 2d}$
Fulllex	$\sigma\left(W\begin{bmatrix}A_v\boldsymbol{u}\\ A_u\boldsymbol{v}\end{bmatrix}\right)$	$W \in \mathbb{R}^{d\times 2d}$, $A_u, A_v \in \mathbb{R}^{d\times d}$

2.2 Composing word vectors for phrases

The idea of computing a vector of a phrase from its constituents is based on the Principle of Compositionality (Frege, 1892), where the meaning of a complex unit (e.g., phrase or sentence) comprises the meanings of the constituents and the rule for combining the constituents. Equation 1 formulates this principle mathematically:

$$\boldsymbol{p} = f(\boldsymbol{u}, \boldsymbol{v}). \tag{1}$$

Here, given two input (e.g. word) vectors $\boldsymbol{u} \in \mathbb{R}^{d_1}$ and $\boldsymbol{v} \in \mathbb{R}^{d_1}$, the model f yields a phrase vector $\boldsymbol{p} \in \mathbb{R}^{d_2}$ as a composition of the input vectors. In other words, the model f is a function that computes a phrase vector $\boldsymbol{p}$ for the inputs $\boldsymbol{u}$ and $\boldsymbol{v}$. Setting $d = d_1 = d_2$ allows recursive compositions, i.e., generating phrase or sentence vectors consisting of three or more words.

Table 1 shows representative models from earlier works. The *Add* model (Mitchell and Lapata, 2008; Mitchell and Lapata, 2010) computes a linear combination of two input vectors $\boldsymbol{u}$, $\boldsymbol{v} \in \mathbb{R}^d$ with weights w_1, $w_2 \in \mathbb{R}$. This model works surprisingly well in practice despite its simplicity. The *Fulladd* model (Guevara, 2010; Zanzotto et al., 2010)

extends the *Add* model, applying a linear transformation to inputs with a weight matrix $W \in \mathbb{R}^{d \times 2d}$. This model can not only scale but also rotate input vectors, unlike the *Add* model.

Regarding linear transformation with a matrix W, *Recursive Neural Network (RNN)* model (Socher et al., 2011) achieves a nonlinear transformation through the use of an activation function (e.g., sigmoid function and $tanh$). *Lexfunc* model (Baroni et al., 2012) represents a dependent word u (e.g., adjective) as a matrix A_u and composes a phrase vector with a matrix-vector product $A_u \boldsymbol{v}$. The underlying idea of representing a dependent as a matrix is that a modifier (dependent) changes some properties of a governer and that it is achieved using a matrix transforming a vector of the governer[1].

Extending *RNN*, the *Relfunc* model (Socher et al., 2013a; Socher et al., 2014) incorporates syntactical relations in compositions, which composes phrase vectors with a different weight matrix for a syntactic relation between inputs. Generalizing *Lexfunc* and *RNN*, the *Fulllex* model Socher et al. (2012) defines the meaning of each word as a tuple of a vector and matrix, where a vector represents the meaning of the word itself and a matrix provides a function to other words for compositions. In addition to these models, the *Mult* model and the *Dil* model (Mitchell and Lapata, 2008; Mitchell and Lapata, 2010) have been proposed.

Table 2 presents the benefits and shortcomings of each model. It is easy to train the *Add* model because it has only two parameters. Apparently, the *Add* model has the least expressive power using very few parameters. However, this simple model has been a strong baseline in the literature (Blacoe and Lapata, 2012). Similarly to the *Add* model, *Fulladd* uses a linear composition function; we can find a global optimum for the convex training objective. In contrast, *RNN*, *Relfunc* and *Fulllex* are neural network models using nonlinear activation functions. The nonlinearity enriches the expressive power, but it makes training difficult because the training objectives are not convex.

Regarding the performance of these models aside from *Relfunc* in the same condition, Dinu et al. (2013) concluded that *Lexfunc* performed the best among these models. According to their explanation, *Lexfunc* performs well because it considers linguistic relations between input words (e.g. modification, verb–object relation). However, *Lexfunc* cannot compose vectors recursively because of the different types of input–output representations (vector or matrix).

Table 2: Problems of representative models.

Model	Expressive	Recursivity	Training	Nonlinearity
Add	NA	✓	✓	NA
Fulladd	NA	✓	✓	NA
RNN	NA	✓	✓	✓
Lexfunc	✓	NA	Depends	NA
Relfunc	✓	✓	✓	✓
Fulllex	✓	✓	NA	✓

In contrast, *RNN*, *Relfunc*, and *Fulllex* can compose vectors recursively. The recursivity is an important property because it enables comparison of a phrase vector (e.g., *football player*) with a word vector (e.g., *footballer*). However, *Fulllex* has an enormous number of parameters, representing each word as a distinct tuple of a vector and a matrix. In *RNN*, on the other hand, all compositions are computed only by a single weight matrix. Consequently, it cannot distinguish different syntax relations in compositions. Located between *RNN* and *Fulllex*, *Relfunc* can compose various types of syntax relations more precisely than *RNN* with fewer parameters than *Fullex*.

These models have produced excellent results on many tasks such as syntax parsing or grounding between texts and images. However, no report in the literature describes an experiment examining semantic compositions directly under the same conditions. As described in this paper, we explore the best model that can perform semantic compositions well. Our experimentally obtained results show that the *Relfunc* model achieves state-of-the-art performance.

[1]For instance, we can regard *red* in the phrase *red car* as changing the property of *color* of the word *car*.

3 Details of Methods

3.1 Mathematical Expression of Models

The *Add* model in Table 1 composes two input vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ simply with two parameters w_1 and w_2 (and a bias term b):

$$\boldsymbol{p} = f(\boldsymbol{u}, \boldsymbol{v}) = w_1\boldsymbol{u} + w_2\boldsymbol{v} + b\mathbf{1}. \quad (2)$$

Here, $\boldsymbol{u}$, $\boldsymbol{v}$, and $\mathbf{1}$ are d-dimensional column vectors and all elements of $\mathbf{1}$ consist of 1.

Add can only scale whereas *Fulladd* can also rotate because of a weight matrix $W \in \mathbb{R}^{d\times(2d+1)}$:

$$\boldsymbol{p} = f(\boldsymbol{u}, \boldsymbol{v}) = W \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \\ b \end{bmatrix}. \quad (3)$$

Neural network models such as *RNN* in Table 1 compose a phrase vector $\boldsymbol{p}$ from two input vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ using a function $f : \mathbb{R}^{(2d+1)\times 1} \longrightarrow \mathbb{R}^{d\times 1}$,

$$\boldsymbol{p} = f(\boldsymbol{u}, \boldsymbol{v}) = \sigma\left(W \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \\ b \end{bmatrix}\right). \quad (4)$$

$\sigma(.)$ is an element-wise sigmoid function that yields a value for each element in the vector. In our work, we use $\tanh$ as a sigmoid function.

Socher et al. (2014) extends this model so that *Relfunc* can compose a vector depending on the relation r between two inputs. Equation 5 uses a composition matrix W_r and a bias term b_r for each relation r,

$$\boldsymbol{p} = f(\boldsymbol{u}, \boldsymbol{v}, r) = \sigma\left(W_r \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \\ b_r \end{bmatrix}\right). \quad (5)$$

Here, $W_r \in \mathbb{R}^{d\times(2d+1)}$ and $b_r \in \mathbb{R}$ are parameters trained for each relation r. Introducing relation-specific matrices, Equation 5 can compose a phrase vector more precisely than *RNN* given by Equation 4. In this work, we use syntactic dependencies as relations used for compositions. We also introduce two restricted variants of *Relfunc* here.

1. Relation-specific additive model (*Relfunc-add*) has two weight parameters $w_1, w_2 \in \mathbb{R}$ for each relation r:

$$\boldsymbol{p} = \sigma\left(\begin{bmatrix} w_{1,r}I & w_{2,r}I & \begin{matrix}1\\ \vdots \\ 1\end{matrix} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \\ b_r \end{bmatrix}\right) \quad (6)$$

2. Relation-specific component-wise additive model (*Relfunc-cadd*) is modeled by diagonal elements for $\boldsymbol{u}$ and $\boldsymbol{v}$:

$$\boldsymbol{p} = \sigma\left(\begin{bmatrix} w_{1,1} & 0 & w_{2,1} & 0 & 1 \\ & \ddots & & \ddots & \vdots \\ 0 & w_{1,d} & 0 & w_{2,d} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \\ b_r \end{bmatrix}\right) \quad (7)$$

These variants are used to verify the effect of non-diagonal elements of matrices W_r in the experiments.

The *Fulllex* model, the most complicated model among those in Table 1, first multiplies each input vector by the other matrix, i.e., $\boldsymbol{u}$ is multiplied by $A_v \in \mathbb{R}^{d\times d}$ and $\boldsymbol{v}$ multiplied by $A_u \in \mathbb{R}^{d\times d}$. Subsequently, *Fulllex* composes the phrase vector in the same way for *RNN* and *Relfunc*,

$$\boldsymbol{p} = f(\boldsymbol{u}, \boldsymbol{v}) = \sigma\left(W \begin{bmatrix} A_v\boldsymbol{u} \\ A_u\boldsymbol{v} \\ b \end{bmatrix}\right). \quad (8)$$

3.2 Training

We train model-specific parameters θ (e.g., for *Add*, $\theta = \langle w_1, w_2, b\rangle$, and for *Relfunc*, $\theta = \langle W_r, b_r | r\rangle$) in a supervised setting where a gold phrase vector $\boldsymbol{q}$ is given for two input vectors of constituents $\boldsymbol{u}$ and $\boldsymbol{v}$. A training set consists of T training instances $\{((\boldsymbol{u}_t, \boldsymbol{v}_t), \boldsymbol{q}_t)\}_{t=1}^{T}$. The goal of training is to find optimal parameters θ such that the parameters can compose phrase vectors of good quality. We formalize this goal as a minimization problem of the objective function defined by the square errors between composed vectors and gold vectors,

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{2}\|\boldsymbol{p}_t - \boldsymbol{q}_t\|_2^2 + \lambda\|\theta\|_1. \quad (9)$$

Here, the vector $\boldsymbol{p}_t$ presents a phrase vector composed by Equations 2 - 8 from word vectors $(\boldsymbol{u}_t, \boldsymbol{v}_t)$. Vector $\boldsymbol{q}_t$ denotes a gold phrase vector. Therefore, the first term of Equation 9 represents a least-squares problem (York, 1966) defined for vectors $\boldsymbol{p}_t$ and $\boldsymbol{q}_t$. The second term of Equation 9 presents an L_1-regularization term with a hyper-parameter λ. We employ L_1-regularization instead of L_2-regularization to make the composition model compact.

We use stochastic gradient descent and backpropagation (Rumelhart et al., 1988) to minimize the objective.

In general, a weight matrix W is updated by the following equation,

$$W' = W - \alpha \frac{\partial J(\theta)}{\partial W}, \tag{10}$$

where α is a learning rate.

Using stochastic gradient descent, we update a weight matrix W every time one training instance is processed. The gradient of the objective is

$$\frac{\partial J(\theta)}{\partial W} = \frac{\partial J(\theta)}{\partial \boldsymbol{p}_t} \frac{\partial \boldsymbol{p}_t}{\partial W} = \boldsymbol{e}_t \begin{bmatrix} \boldsymbol{u}_t \\ \boldsymbol{v}_t \\ b \end{bmatrix}^T + \lambda \frac{\partial}{\partial W} \|\theta\|_1. \tag{11}$$

Here, $\boldsymbol{e}_t$ represents a d-dimensional column vector with k-th element of

$$e_{t,k} = (p_{t,k} - q_{t,k})(1 - p_{t,k}^2). \tag{12}$$

We used $\frac{d}{dx}\tanh(x) = 1 - \tanh(x)^2$ to derive this equation.

The second term of Equation 11 is not differentiable. Following the work of Langford et al. (2008) and Tsuruoka et al. (2009), we first update the weight matrix W without consideration of the L_1 penalty. Then, we use Equation 13 to apply the L_1 regularization,

$$w_{ij}' = \begin{cases} \max(0, w_{ij} - \alpha\lambda) & \text{if } w_{ij} > 0 \\ \min(0, w_{ij} + \alpha\lambda) & \text{if } w_{ij} < 0 \\ 0 & \text{otherwise} \end{cases}, \tag{13}$$

where w_{ij} denotes the (i, j) element of W.

A neural network model such as *RNN*, *Relfunc*, and *Fulllex* is nonlinear, which means that the naive training procedure might be trapped with a local minimum. To prevent local minima, we employ some technical methods. We update the learning rate α for every iteration epoch l using the temperature of the simulated annealing algorithm (Kirkpatrick et al., 1983).

In addition, to date, the learning rate α is constant to all matrices W_r in *Relfunc*. However, the distribution of relations in the training data is highly skewed. Because the number of updates for a relation is directly proportional to the number of instances of the relation in the dataset, some matrixes are updated frequently, and some are rarely updated. Therefore, we use the diagonal variant of AdaGrad (Duchi et al., 2011; Socher et al., 2013a). This enables the learning rate to vary each matrix W_r.

4 Experiment

In this section, we explain the method for constructing vectors for words and phrases for the supervision data, followed by an explanation of some details of the training procedure. We then report experimentally obtained results.

4.1 Obtaining vectors for words and phrases as supervision data

Following the work of Dinu et al. (2013), we constructed word and phrase vectors as follows. We used a concatenation of three large corpora: PukWaC[2] (Baroni et al., 2009) (2 billion tokens), WaCkypedia_EN(Wikipedia 2009 dump) (Baroni et al., 2009) (about 800 million tokens), and ClueWeb09[3] (5 billion pages in English). The distribution of PukWaC and WaCkypedia_EN includes parse results from TreeTagger and MaltParser. We used Stanford CoreNLP[4] to parse ClueWeb09. Counting frequencies of occurrences of lemmas of content words (nouns, adjectives, verbs, and adverbs), we identified the top 10,000 most frequent words; we represent the set of these lemmas (except adverbs) as vocabulary V.

We then find the frequencies of phrases consisting only of two words in V (adjective–noun, noun–noun, verb–noun). For words in V and phrases appearing more than 1,000 times in the corpora, we build a co-occurrence matrix: each row is a vector of a target word or phrase; an element in a row represents the frequency of co-occurrences of the target word/phrase with a context word (content lemma). We regard content lemmas appearing in the same sentence within a distance of 50 words from a target word as contexts. Then we transform each element of the co-occurrence matrix into Pointwise Mutual

[2] http://wacky.sslmit.unibo.it/

[3] http://lemurproject.org/clueweb09/

[4] http://nlp.stanford.edu/software/corenlp.shtml

Information (PMI) (Evert, 2005). Finally, we compress the matrix into d dimension using Principal Component Analysis (PCA) (Roweis, 1998) with EM algorithm[5]. In this way, we obtained 10,000 word vectors and 17,433 phrase vectors.

4.2 Gold-standard data

We conducted a human-correlation experiment using the dataset[6] created in Mitchell and Lapata (2010). Each instance in the dataset is a triplet ⟨phrase1, phrase2, similarity⟩: a similarity is a semantic similarity between the phrases annotated by humans, with a value ranging from 1 (least similar) to 7 (most similar). We designate this as human-similarity. For example, the similarity between *vast amount* and *large quantity* is 7 (most similar) whereas the similarity between *hear word* and *remember name* is 1 (least similar).

For each POS pair (adjective–noun, noun–noun, verb–noun), the dataset includes 108 instances annotated by 18 human subjects (1,944 in total). We measure Spearman's ρ between the human similarity and the cosine similarity between each input pair of two phrase vectors composed using a model. Because one POS pair can include dependency relations of several types , *Relfunc* composes phrase vectors in a POS pair with several matrices. A high correlation indicates that the model can compose a phrase vector that reflects its semantic meaning.

4.3 Training

Excluding the phrases in the evaluation dataset, our training set includes 16,845 phrase types for building a training set. For each phrase p type, we include $0.001 \times \mathrm{freq}(p)$ duplicates in the training data, where $\mathrm{freq}(p)$ is the frequency of occurrences of the phrase p in the corpora. In this way, we obtained a training set consisting of $T = 175,899$ instances of phrases.

We set other hyper-parameters as described below:

- Dimension $d \in \{50, 100, 200\}$.
- Learning rate $\alpha = 1/1.1^{l-1}$. l is an epoch count.
- L_1-regularization coefficient $\lambda \in \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$.
- Convergence condition: $|J^{l-1} - J^l| < 10^{-6}$
- Maximum number of epochs: 100

Because models are sensitive to d and λ, we find d and λ with the highest performance with respect to each model. We observed that all models converged in 50 to 100 epochs. We prepared 31 weight matrices W_r corresponding to all types of dependency relations. A weight matrix and a weight of a bias term are initialized as ($\mathcal{N}(\mu, \sigma^2)$ denotes a normal distribution with mean μ and variance σ^2),

$$\begin{aligned} W &= 0.01[\boldsymbol{I}_{d\times d}, \boldsymbol{I}_{d\times d}, \mathbf{0}_{d\times 1}] \\ &\quad + \mathcal{N}(\mathbf{0}_{(2d+1)\times 1}, 0.001\boldsymbol{I}_{(2d+1)\times d}), \\ b &= \mathcal{N}(0.0, 0.001). \end{aligned} \tag{14}$$

We use a server running on four processors (12-core, 2.2 GHz, AMD Opteron 6174) with 256 GB main memory. Using 10 threads, approximately 7 hours were needed to train a model[7]. We use the idea of Iterative Parameter Mixture (McDonald et al., 2010) to parallelize the training process. Each thread receives a subset of the training data, and estimates parameters individually on the subset. After all threads finish an epoch for the subsets, we take the average of the parameters from all threads, and distribute it to the threads for the next epoch.

We trained models in Table 1 with the same experimental setting (the same objective, the same training set, and the same hyper-parameters) except for *Lexfunc*. This enables performance comparisons between different models. The reason for the absence of *Lexfunc* is that it requires a vector and a matrix for composition of a phrase. Two constituents for a phrase are given as vectors in our experiments. Therefore, we cannot conduct an experiment with *Lexfunc* on the same setting.

4.4 Results

Table 3 reports the correlation of similarity values with the gold-standard data. *Upper-bound* presents the mean of inter-subject correlations (between a subject and the others). *Corpus* obtains a phrase

[5]To handle a large amount of data, we implemented an online variant of PCA.

[6]http://homepages.inf.ed.ac.uk/s0453356/share

[7]We used Python modules numpy and multiprocessing for implementation of the training algorithm.

Table 3: Spearman's ρ of each POS pair, where * denotes statistical significance ($p < 0.01$) between *Relfunc* and the most competitive model among the other models: *Add-smp*.

	JJ-NN	**NN-NN**	**VB-NN**
Corpus	0.380	0.449	0.215
Add-smp	0.457	0.460	0.406
Add	0.335	0.304	0.338
Fulladd	0.359	0.359	0.366
RNN	0.364	0.360	0.367
Relfunc-add	0.440	0.455	0.388
Relfunc-cadd	0.419	0.445	0.413
Relfunc	**0.469***	**0.481***	**0.430***
Fulllex	0.322	0.160	0.222
Upper-bound	0.539	0.490	0.505

vector simply from the co-occurrence statistics in the corpora (similarly to the supervision instances). This setting corresponds to the distributional hypothesis applied to phrases without considering semantic composition. The reason for the low performance of this approach is that some phrase vectors are unavailable[8] or unreliable in the corpora because of the data sparseness problem.

Add-smp is the model in Table 1 with the weight parameter fixed: $w_1 = w_2 = 1.0$. This approach is equivalent to the simple additive baseline that adds two word vectors without training. As Table 3 shows, *Add-smp* model is a strong competitive model, beating *RNN* and *Fulladd* models. However, the *Relfunc* model outperformed all the tested models including *Add-smp* in all relations. The differences between *Relfunc* and *Add-smp* are significant ($p < 0.01$) in all relations.

Furthermore, *Relfunc* outperforms *Relfunc-add* and *Relfunc-cadd*, which are the variants of *Relfunc*. This result underscores the importance of non-diagonal elements of weight matrices.

Although we cannot compare these results directly with those reported from other studies (Dinu et al., 2013; Blacoe and Lapata, 2012) because of the different computations of Spearman's ρ[9], our results are comparable. These results demonstrate the effectiveness of using a different weight matrix for each relation of compositions.

[8]When a phrase vector is not available from the corpus, we define the similarity as zero.

[9]Reports of those studies did not describe explicitly how they computed the correlation coefficient.

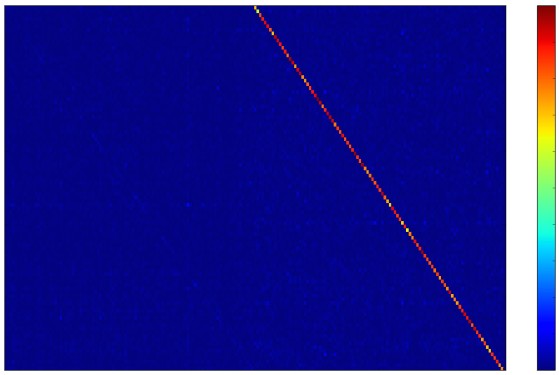


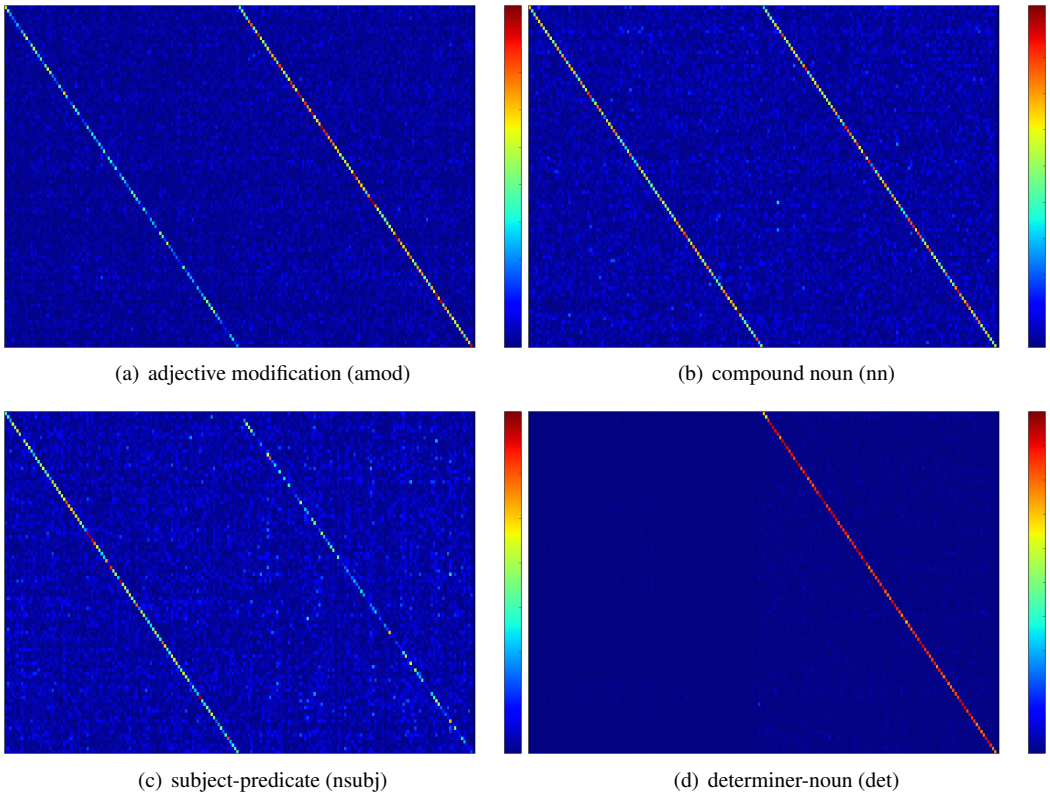
Figure 1: Weight matrix of *RNN*.

4.5 What the Learned Weight Matrices Look Like

To explore why *Relfunc* outperforms *RNN*, we visualize the weight matrices learned by the two models in Figures 1 and 2. In the figures, the left side (split by the center) presents the weights for the left word. The right side presents weights for the right word. The smaller a weight value in the matrix is, the dimmer the element is visualized; the larger a weight value is, and the brighter the element is visualized.

Figure 1 visualizes the weight matrix trained by *RNN*. The diagonal elements in the left and right sides tend to be larger than the non-diagonal elements. This fact indicates that the i-th elements of input word vectors most strongly influence the i-th element of a phrase vector. The diagonal elements of the right side are brighter than those of the left side, which implies that *RNN* treats a right word as more important than a left word in semantic compositions. That implication is reasonable because the right word is usually the head of the phrase and is therefore more important. However, such is not always the case. For example, in subject–predicate constructions, the subject should be regarded as being as important as the predicate. The *RNN* model cannot manage such cases.

In contrast, *Relfunc* learns the relative importance of phrase components depending on the types of syntactic constructions. Figure 2 demonstrates how



(a) adjective modification (amod)

(b) compound noun (nn)

(c) subject-predicate (nsubj)

(d) determiner-noun (det)

Figure 2: Weight matrices of *Relfunc*.

the weight matrices are learned differently depending on syntactic dependency relations. In the matrix for adjective modification, for example, the elements of the right diagonal tend to be larger than those of the left, which reflects a tendency by which a modified word (right word) is more important than a modifier (left word); yet, the left diagonal is assigned reasonably large weight compared with that of the *RNN* weight matrix. Different types of constructions require different weight biases. Subject–predicate constructions, for example, assign more weight on the left diagonal.

Next we examine the effects of this difference using examples. Table 4 presents examples of similarity scores assigned by human judgment (averaged human-similarity scores) and those given by three models: *Relfunc*, *Add-smp*, and *RNN*. For the first two examples, the three models estimate the similarity almost equally well. For the third example, *important part* and *significant role*, *RNN* fails to express that they are quite similar. This might be true because *RNN* assigns too much weight to head words, *part* and *role*, and loses the information given by their modifiers.

The fourth examples, *previous day* and *long period*, show the importance of learning the proper balance of weights between the left and right words. *Add-smp* overestimates the similarity between the two phrases whereas *Relfunc* and *RNN* appropriately and specifically examines the difference between the head words, *day* and *period*.

We have specifically addressed only the weights of the diagonal elements. However, it should also be noted that the non-diagonal elements play non-negligible roles as demonstrated by the performance gain between *Relfunc* and *Relfunc-cadd* (see Table 3). For further exploration of the model's behavior, more sophisticated methods of analyzing the weight matrices and word vectors must be used. That goal is left as a subject of future work.

Table 4: Examples of human-similarity and co-similarity of three models.

instance	**GOLD**	**Relfunc**	**Add-smp**	**RNN**
certain circumstance particular case	6.1	0.76	0.74	0.64
national government cold air	1.0	-0.06	-0.07	-0.02
important part significant role	6.3	0.62	0.64	0.31
previous day long period	1.8	0.36	0.52	0.32

5 Conclusion and Future Work

As presented in this paper, we described the properties of the previous methods: the expressive power, the recursivity, and the difficulty of training. To investigate the impact on these properties, we reimplemented these models and conducted a human-correlation experiment, which demonstrated the state-of-the-art performance of *Relfunc* and the usefulness of the syntactic information in composition. Moreover, learned weight matrices suggest that compositions require different calculations based on their linguistic properties. In future studies, we will extend this work to examine the goodness of models when they compose phrases consisting of three or more words. We will address this problem for tasks of paraphrase detection or entailment recognition.

Acknowledgments

This study was partly supported by Japan Society for the Promotion of Science (JSPS) KAKENHI Grant No. 23240018 and Japan Science and Technology Agency (JST).

References

Marco Baroni, Silvia Bernardini, Adriano Ferraresi, and Eros Zanchetta. 2009. The wacky wide web: a collection of very large linguistically processed web-crawled corpora. *Language Resources and Evaluation*, 43(3):209–226.

Marco Baroni, Raffaella Bernardi, Ngoc-Quynh Do, and Chung-chieh Shan. 2012. Entailment above the word level in distributional semantics. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL '12)*, pages 23–32.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

William Blacoe and Mirella Lapata. 2012. A comparison of vector-based representations for semantic composition. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL '12')*, pages 546–556.

John A. Bullinaria and Joseph P. Levy. 2007. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*, 39(3):510–526.

Ronan Collobert and Jason Weston. 2008. A unified architecture for Natural Language Processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML '08)*, pages 160–167.

Georgiana Dinu, Nghia The Pham, and Marco Baroni. 2013. General estimation and evaluation of compositional distributional semantics models. In *Proceedings of the ACL 2013 Workshop on Continuous Vector Space Models and their Compositionality (CVSC 2013)*, pages 50–58.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July.

Katrin Erk and Sebastian Padó. 2008. A structured vector space model for word meaning in context. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP 2008)*, pages 897–906.

Stefan Evert. 2005. *The statistics of word cooccurrences: word pairs and collocations*. Ph.D. thesis, Universitt Stuttgart.

John R. Firth. 1957. *Papers in Linguistics 1934-51*. Oxford University Press.

Gottlob Frege. 1892. On sense and reference. In *Ludlow (1997)*, pages 563–584.

Emiliano Guevara. 2010. A regression model of adjective-noun compositionality in distributional semantics. In *Proceedings of the 2010 Workshop on GEometrical Models of Natural Language Semantics (GEMS '10)*, pages 33–37.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by simulated annealing. *Science*, 220(4598):671–680.

John Langford, Lihong Li, and Tong Zhang. 2008. Sparse online learning via truncated gradient. *CoRR*, abs/0806.4686.

Ryan McDonald, Keith Hall, and Gideon Mann. 2010. Distributed training strategies for the structured perceptron. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 456–464.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pages 3111–3119.

Jeff Mitchell and Mirella Lapata. 2008. Vector-based models of semantic composition. In *Proceedings of ACL-08: HLT*, pages 236–244.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive Science*, 34(8):1388–1429.

Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pages 2265–2273.

Sam Roweis. 1998. EM algorithms for PCA and SPCA. In *Neural Information Systems 10 (NIPS'97)*, pages 626–632.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1988. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA.

Richard Socher, Cliff C. Lin, Andrew Y. Ng, and Christopher D. Manning. 2011. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1201–1211.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013a. Parsing With Compositional Vector Grammars. In *ACL*.

Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Ng. 2013b. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems 26*, pages 926–934.

Richard Socher, Andrej Karpathy, Quoc V. Le, Christopher D. Manning, and Andrew Y. Ng. 2014. Grounded compositional semantics for finding and describing images with sentences. *TACL*, 2:207–218.

Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. 2009. Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1*, ACL '09, pages 477–485. Association for Computational Linguistics.

Peter D. Turney and Patrick Pantel. 2010. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, 37(1):141–188.

Derek York. 1966. Least-squares fitting of a straight line. *Canadian Journal of Physics*, 44(5):1079–1086.

Fabio Massimo Zanzotto, Ioannis Korkontzelos, Francesca Fallucchi, and Suresh Manandhar. 2010. Estimating linear models for compositional distributional semantics. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 1263–1271.