Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

# Adaptive Multi-Compositionality for Recursive Neural Models with Applications to Sentiment Analysis

**Li Dong**[†*] **Furu Wei**[‡] **Ming Zhou**[‡] **Ke Xu**[†]
[†]State Key Lab of Software Development Environment, Beihang University, Beijing, China
[‡]Microsoft Research, Beijing, China
donglixp@gmail.com {fuwei,mingzhou}@microsoft.com kexu@nlsde.buaa.edu.cn

## Abstract

Recursive neural models have achieved promising results in many natural language processing tasks. The main difference among these models lies in the composition function, i.e., how to obtain the vector representation for a phrase or sentence using the representations of words it contains. This paper introduces a novel Adaptive Multi-Compositionality (AdaMC) layer to recursive neural models. The basic idea is to use more than one composition functions and adaptively select them depending on the input vectors. We present a general framework to model each semantic composition as a distribution over these composition functions. The composition functions and parameters used for adaptive selection are learned jointly from data. We integrate AdaMC into existing recursive neural models and conduct extensive experiments on the Stanford Sentiment Treebank. The results illustrate that AdaMC significantly outperforms state-of-the-art sentiment classification methods. It helps push the best accuracy of sentence-level negative/positive classification from 85.4% up to 88.5%.

## Introduction

Recursive Neural Models (RNMs), which utilize the recursive structure of the input (e.g., a sentence), are one family of popular deep learning models. They are particularly effective for many Natural Language Processing (NLP) tasks due to the compositional nature of natural language. Recently, many promising results have been reported on semantic relationship classification (Socher et al. 2012), syntactic parsing (Socher et al. 2013a), sentiment analysis (Socher et al. 2013b), and so on. The main difference among RNMs lies in the semantic composition method, i.e., how to obtain the vector representation for a phrase or sentence using the representations of words and phrases it contains. For instance, we can compute the word vector for the phrase "*not good*" with the vectors of the words "*not*" and "*good*". For many tasks, we even need to obtain the vector representations for sentences. The composition algorithm becomes the key to make the vector representations go beyond words to phrases and sentences.

There have been several attempts in literature to address the semantic composition for RNMs. Specifically, RNN (Socher et al. 2011) uses a global matrix to linearly combine the elements of vectors, while RNTN (Socher et al. 2013b) employs a global tensor to model the products of dimensions. Sometimes it is challenging to find a single powerful function to model the semantic composition. Intuitively, we can employ multiple composition functions, instead of only using a single global one. Instead of finding more complex composition functions, MV-RNN (Socher et al. 2012) assigns matrices for every words to make the compositions specific. However, the number of composition matrices is the same as vocabulary size, which makes the number of parameters quite large. It is easy to overfit the training data and difficult to be optimized. Moreover, MV-RNN needs another global matrix to linearly combine the composition matrices for phrases, which still makes these compositions not specific. In order to overcome these shortcomings and make the compositions specific, it is better to use a certain number of composition functions, and embed the role-sensitive (linguistic and semantic) information into word vectors to adaptively select these compositions rather than concrete words. The example "*not (so good)*" in sentiment analysis illustrates this point. To obtain the polarity of this phrase, we firstly combine the words "*so*" and "*good*", then combine the "*not*" and "*so good*". Specifically, the first combination is a strengthen composition which makes the sentiment polarity stronger, and the second step is a negation composition which negates the positive polarity to negative.

In this paper, we introduce a novel Adaptive Multi-Compositionality (AdaMC) method for RNMs. AdaMC consists of more than one composition functions, and adaptively selects them depending on the input vectors. The model learns to embed the semantic categories of words into their corresponding word vectors, and uses them to choose these composition functions adaptively. Specifically, we propose a parametrization method to compute the probability distribution for every function given the child vectors. We also introduce a hyper-parameter to model the adaptive preferences over the different composition functions and show three special cases of AdaMC. By adjusting this hyper-parameter, there is a continuous transition between these three special cases. Moreover, all these composition functions and how to select them are automatically learned from

[*]Contribution during internship at Microsoft Research.

supervisions, instead of choosing them heuristically or by manually defined rules. Hence, task-specific composition information can be embedded into representations and used in the task. We have conducted extensive experiments on the Stanford Sentiment Treebank. The experimental results illustrate that our approach significantly improves the baseline methods, and yields state-of-the-art accuracy on the sentiment classification task.

The main contributions of this paper are three-fold:

- We introduce an Adaptive Multi-Compositionality approach into recursive neural models to better perform semantic compositions;
- We propose a parametrization method to calculate the probabilities of different composition functions given two input vectors;
- We present empirical results on the public available Stanford Sentiment Treebank. AdaMC significantly outperforms state-of-the-art sentiment classification results.

## Related Work

Semantic composition has attracted extensive attention in vector based semantic models. Let *a* and *b* be two words, represented by the vectors $\mathbf{a}$ and $\mathbf{b}$. The goal of semantic composition is to compute vector $\mathbf{c}$ to represent the phrase *ab*. Most previous works focus on defining different composition functions to obtain vector $\mathbf{c}$. Landauer and Dutnais (1997) use the average of $\mathbf{a}$ and $\mathbf{b}$ to obtain the representation for *ab*. Mitchell and Lapata (2008; 2010) suggest using weighted addition ($\mathbf{c} = \alpha\mathbf{a} + \beta\mathbf{b}$) and element-wise multiplication ($\mathbf{c} = \mathbf{a} \odot \mathbf{b}$) to compute vector $\mathbf{c}$. Another operation is the tensor product (Smolensky 1990; Aerts and Czachor 2004; Clark, Coecke, and Sadrzadeh 2008; Widdows 2008), such as the outer product, for composition. The outer product of two vectors produces a matrix, which makes the representations become exponentially larger. An improvement of using the outer product is to use circular convolution (Plate 1991; Jones and Mewhort 2007) as composition functions. It compresses the result matrix of the outer product to a vector. Baroni and Zamparelli (2010) represent nouns as vectors, and adjectives as matrices. Then the authors apply matrix-by-vector multiplication for the adj-noun pair composition. Instead of using vector representations, Rudolph and Giesbrecht (2010) and Yessenalina and Cardie (2011) use matrices to represent the phrases and define the composition functions as matrix multiplication. However, most of previous settings are unsupervised instead of using the supervisions from the specific tasks, and evaluated by comparing the similarity between short phrases (e.g., adj-noun word pairs). For example, the "*very good*" and "*very bad*" are regarded as similar phrases, which sometimes is not feasible for specific tasks (e.g., sentiment analysis).

Recently, some works use the semantic composition in recursive neural networks to build deep models for NLP tasks, and have achieved some promising results. Socher et al. (2011) learn a matrix $W$ to combine the vectors $\mathbf{a}, \mathbf{b}$, and the composition result is $W\begin{bmatrix}\mathbf{a}\\ \mathbf{b}\end{bmatrix}$. Socher et al. (2012) assign matrix-vector pairs $(A, \mathbf{a})$ and $(B, \mathbf{b})$ for words *a* and *b*, where $A$,$B$ are matrices, and vectors $\mathbf{a}, \mathbf{b}$ are representations. When two phrases are combined, the composition result is $W\begin{bmatrix}B\mathbf{a}\\ A\mathbf{b}\end{bmatrix}$, where $W$ is a global linear mapping matrix. Yu, Deng, and Seide (2013) and Socher et al. (2013b) use a tensor layer to model the intersections between different dimensions of combined vectors.

Grefenstette and Sadrzadeh (2011; 2013) present a categorical compositional model, which employs formal semantics (grammatical structure) to guide the composition sequence. Hermann and Blunsom (2013) and Polajnar, Fagarasan, and Clark (2013) use the parsing results based on Combinatory Categorial Grammar (CCG) to guide the semantic composition. Socher et al. (2013a) propose to compute the composition vectors depending on the Part-Of-Speech (POS) tags of two phrases. Our work is significantly different from them. To begin with, we do not employ any syntactic categories (such as CCG combinators and types, and POS tags), which are assumed to have been obtained by existing parsers. In addition, sometimes the syntactic categories are not helpful to guide the compositions for some tasks. For instance, both "*not good*" and "*very good*" are adv-adj pairs. However, the former is a negation composition which negates the polarity strength, and the other one is a amplifier in the sentiment analysis task. Our proposed method learns to select the composition functions depending on the current vectors (adaptively). Moreover, our model can also leverage these syntactic knowledge in a unified way by regarding them as features, instead of using them heuristically or by manually crafted rules.

## Recursive Neural Models

The recursive neural models represent the phrases and words as $D$-dimensional vectors. The models perform compositions based on binary trees, and obtain the vector representations in a bottom-up way. Notably, the word vectors in the leaf nodes are regarded as parameters, and will be updated according to the supervisions. Specifically, the vectors of phrases are computed by the composition of their child vectors. The vector of node $i$ is calculated via:

$$\mathbf{v}^i = f\left(g\left(\mathbf{v}_l^i, \mathbf{v}_r^i\right)\right) \tag{1}$$

where $\mathbf{v}_l^i, \mathbf{v}_r^i$ are the vectors of its left child and right child, $g$ is the composition function, and $f$ is the nonlinearity function (such as tanh, sigmoid, softsign, etc.). As illustrated in Figure 1, the representation of "*so bad*" is calculated by the composition of "*so*" and "*bad*", and the representation of trigram "*not so bad*" is recursively obtained by the vectors of "*not*" and "*so bad*".

The learned representations are then fed into a classifier to predict the labels for nodes. The softmax layer is used as a standard component, as shown in Figure 1. The $k$-th element of softmax$(\mathbf{z})$ is $\frac{\exp\{\mathbf{z}_k\}}{\sum_j \exp\{\mathbf{z}_j\}}$. It outputs the probability distribution over $K$ classes for a given input. To be more specific, the prediction distribution of node $i$ is calculated via:

$$\mathbf{y}^i = \text{softmax}\left(U\mathbf{v}^i\right) \tag{2}$$

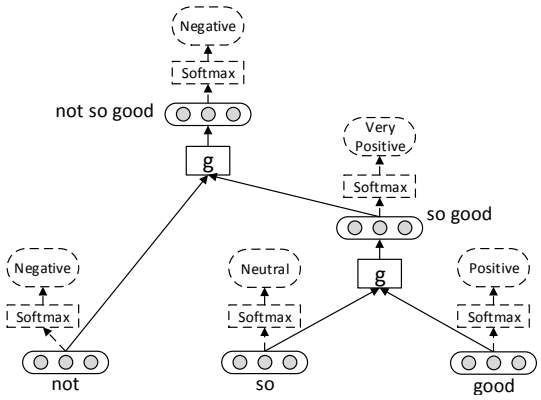



Figure 1: Composition process for "*not so good*". The $g$ is a global composition function in recursive neural models.

where $U \in \mathbb{R}^{K \times D}$ is the classification matrix, $\mathbf{v}^i$ is the vector representation of node $i$, and $\mathbf{y}^i$ is the prediction distribution for node $i$.

The main difference between recursive neural models lies in the design of composition functions. We describe two mainstream models and their composition functions.

### RNN: Recursive Neural Network

The RNN (Socher et al. 2011) is a standard member of recursive neural models. Every dimension of parent vector is calculated by weighted linear combination of the child vectors' dimensions. The vector representation of node $i$ is obtained via:

$$\mathbf{v}^i = f\left(W\begin{bmatrix}\mathbf{v}_l^i \\ \mathbf{v}_r^i\end{bmatrix} + \mathbf{b}\right) \tag{3}$$

where $W \in \mathbb{R}^{D \times 2D}$ is the composition matrix, $\mathbf{b}$ is the bias vector, $\mathbf{v}_l^i, \mathbf{v}_r^i$ are the left and right child vectors respectively, and $f$ is the nonlinearity function. The dimension of $\mathbf{v}_i$ is the same as its child vectors, and it is recursively used in the next composition.

### RNTN: Recursive Neural Tensor Network

The RNTN (Socher et al. 2013b) uses more parameters and a more powerful composition function than RNN. The main idea of RNTN is to employ tensors to model the intersection between every dimension of vectors. The vector of node $i$ is computed via:

$$\mathbf{v}^i = f\left(\begin{bmatrix}\mathbf{v}_l^i \\ \mathbf{v}_r^i\end{bmatrix}^T T^{[1:D]}\begin{bmatrix}\mathbf{v}_l^i \\ \mathbf{v}_r^i\end{bmatrix} + W\begin{bmatrix}\mathbf{v}_l^i \\ \mathbf{v}_r^i\end{bmatrix} + \mathbf{b}\right) \tag{4}$$

where $T^{[1:D]}$ is a tensor, $W \in \mathbb{R}^{D \times 2D}$ is a linear composition matrix, $\mathbf{b}$ is the bias vector, and $f$ is the nonlinearity function. The result of tensor product is

$$\begin{bmatrix}\mathbf{v}_l^i \\ \mathbf{v}_r^i\end{bmatrix}^T T^{[1:D]}\begin{bmatrix}\mathbf{v}_l^i \\ \mathbf{v}_r^i\end{bmatrix} = \begin{bmatrix}\begin{bmatrix}\mathbf{v}_l^i \\ \mathbf{v}_r^i\end{bmatrix}^T T^{[1]}\begin{bmatrix}\mathbf{v}_l^i \\ \mathbf{v}_r^i\end{bmatrix} \\ \vdots \\ \begin{bmatrix}\mathbf{v}_l^i \\ \mathbf{v}_r^i\end{bmatrix}^T T^{[D]}\begin{bmatrix}\mathbf{v}_l^i \\ \mathbf{v}_r^i\end{bmatrix}\end{bmatrix} \tag{5}$$

where $T^{[d]} \in \mathbb{R}^{2D \times 2D}$ is the $d$-th slice of $T^{[1:D]}$. It obtains a $D$-dimensional vector which defines multiple bilinear forms. When all the elements of $T^{[1:D]}$ are zero, the model is the same as RNN.

## Adaptive Multi-Compositionality for Recursive Neural Models

We illustrate the main idea of proposed Adaptive Multi-Compositionality method in Figure 2. We use a *composition pool* which consists of $C$ composition functions $\{g_1, \ldots, g_C\}$. To obtain the vector of "*so good*", we firstly feed its child vectors ($\mathbf{v}^{so}$ and $\mathbf{v}^{good}$) to a classifier to get the probability $P\left(g_h|\mathbf{v}^{so}, \mathbf{v}^{good}\right)$ for using each composition function $g_h$. Intuitively, we should choose composition functions which strengthen the polarity of "*good*", and predict the "*so good*" as very positive. Similarly, we compute the probability $P\left(g_h|\mathbf{v}^{not}, \mathbf{v}^{so\ good}\right)$ for every composition function. Ideally, we should select negate composition functions to obtain $\mathbf{v}^{not\ so\ good}$, and the polarity should be negative. As shown in this example, it is more reasonable to use multiple composition functions than finding a complex composition function for the recursive neural models.

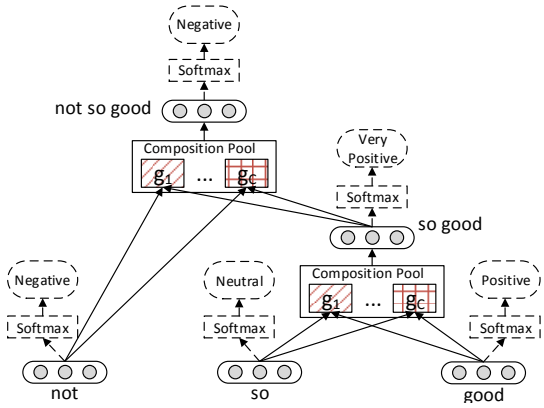



Figure 2: The composition pool consists of multiple composition functions. It selects the functions depending on the input child vectors, and produces the composition result using more than one composition functions.

Generally, we define the composition result $\mathbf{v}^i$ as:

$$\mathbf{v}^i = f\left(\sum_{h=1}^{C} P\left(g_h|\mathbf{v}_l^i, \mathbf{v}_r^i\right) g_h\left(\mathbf{v}_l^i, \mathbf{v}_r^i\right)\right) \tag{6}$$

where $f$ is the nonlinearity function, $g_1, \ldots, g_C$ are the composition functions, and $P\left(g_h|\mathbf{v}_l^i, \mathbf{v}_r^i\right)$ is the probability of employing $g_h$ given the child vectors $\mathbf{v}_l^i, \mathbf{v}_r^i$. For the composition functions, we use the same forms as in RNN (Equation (3)) and RNTN (Equation (4)). The key point is how to select them properly depending on the child vectors, i.e., how to define $P\left(g_h|\mathbf{v}_l^i, \mathbf{v}_r^i\right)$. We define a parametrization approach (named AdaMC), and show three special cases of it. By adjusting the parameter of AdaMC, there is a continuously transition between these special cases.

### AdaMC: Adaptive Multi-Compositionality

To start with, we define the $\beta$-softmax function as:

$$\beta\text{-softmax}\left(\mathbf{z}\right) = \frac{1}{\sum_i \exp\{\beta \mathbf{z}_i\}} \begin{bmatrix} \exp\{\beta \mathbf{z}_1\} \\ \vdots \\ \exp\{\beta \mathbf{z}_K\} \end{bmatrix} \tag{7}$$

where $\mathbf{z} = [\mathbf{z}_1 \dots \mathbf{z}_K]^T$ is a vector. This function is known as the Boltzmann distribution and Gibbs measure (Georgii 2011), which are widely used in statistical mechanics. When $\beta = 0$, the $\beta$-softmax function produces a uniform distribution; When $\beta = 1$, it is the same as softmax function; When $\beta \to \infty$, this function only activates the dimension with maximum weight, and sets its probability to 1. We then employ this function to compute the probability distribution for the composition functions via:

$$\begin{bmatrix} P\left(g_1|\mathbf{v}_l^i, \mathbf{v}_r^i\right) \\ \vdots \\ P\left(g_C|\mathbf{v}_l^i, \mathbf{v}_r^i\right) \end{bmatrix} = \beta\text{-softmax}\left(S \begin{bmatrix} \mathbf{v}_l^i \\ \mathbf{v}_r^i \end{bmatrix}\right) \tag{8}$$

where $S \in \mathbb{R}^{C \times 2D}$ is the matrix used to determine which composition function we use, and $\mathbf{v}_l^i, \mathbf{v}_r^i$ are the left and right child vectors.

### Avg-AdaMC: Average AdaMC

We average the composition results, and this is a special case of AdaMC ($\beta = 0$). The probability of using $g_h$ is:

$$P\left(g_h|\mathbf{v}_l^i, \mathbf{v}_r^i\right) = \frac{1}{C} \tag{9}$$

where $h = 1, \dots, C$, and $C$ is the number of composition functions.

### Weighted-AdaMC: Weighted Average AdaMC

One special case of AdaMC is setting $\beta = 1$. It uses the softmax probability to perform a weighted average.

$$\begin{bmatrix} P\left(g_1|\mathbf{v}_l^i, \mathbf{v}_r^i\right) \\ \vdots \\ P\left(g_C|\mathbf{v}_l^i, \mathbf{v}_r^i\right) \end{bmatrix} = \text{softmax}\left(S \begin{bmatrix} \mathbf{v}_l^i \\ \mathbf{v}_r^i \end{bmatrix}\right) \tag{10}$$

where $S \in \mathbb{R}^{C \times 2D}$ is the parameter matrix.

### Max-AdaMC: Max Output AdaMC

If we set $\beta \to \infty$ in AdaMC, it is a greedy selection algorithm and only outputs the composition result with maximum softmax probability.

### Model Training

We use the softmax classifier to predict the probabilities for classes, and compare the distribution with ground truth. We define the target vector $\mathbf{t}^i$ for node $i$, which is a binary vector. If the correct label is $k$, we set $\mathbf{t}_k^i$ to 1 and the others to 0. Our goal is to minimize the cross-entropy error between the predicted distribution $\mathbf{y}^i$ and target distribution $\mathbf{t}^i$. For each sentence, the objective function is defined as:

$$\min_{\Theta} \; E(\Theta) = -\sum_i \sum_j \mathbf{t}_j^i \log \mathbf{y}_j^i + \sum_{\theta \in \Theta} \lambda_\theta \|\theta\|_2^2 \tag{11}$$

where $\Theta$ represents the parameters, and the second term is a $L_2$-regularization penalty.

We employ back-propagation algorithm (Rumelhart, Hinton, and Williams 1986) to propagate the error from the top node to the leaf nodes. The derivatives are computed and gathered to update the parameters. The details can be found in the **supplemental material** due to space limitations. The AdaGrad (Duchi, Hazan, and Singer 2011) is used to solve this non-convex optimization problem.

## Experiments

### Dataset Description

We evaluate the models on Stanford Sentiment Treebank[1]. This corpus contains the labels of syntactically plausible phrases, which allows us to train the compositional models based on the parsing trees. The treebank is built upon 10,662 critic reviews in Rotten Tomatoes[2], which is originally used for sentence-level sentiment classification (Pang and Lee 2005). The Stanford Parser (Klein and Manning 2003) is used to parse all these reviews to parsing trees, and extract 215,154 phrases. Next, the workers in Amazon Mechanical Turk annotate polarity levels for all these phrases. Most of the shorter phrases are annotated as neutral, and longer phrases tend to be with stronger polarity. All the sentiment scales are merged to five categories (*very negative*, *negative*, *neutral*, *positive*, *very positive*).

### Experiment Settings

We use the standard dataset splits (train: 8,544, dev: 1,101, test: 2,210) in all the experiments. For all these models, we tune the parameters on the dev dataset. We use the mini-batch version AdaGrad in our experiments with the batch size between 20 and 30. We employ $f = \tanh$ as the non-linearity function as it is significantly better than the models without using nonlinearity (Socher et al. 2011). Each word in the vocabulary is assigned with a vector representation. To initialize the parameters, we randomly sample values from a uniform distribution $\mathcal{U}\left(-\epsilon, +\epsilon\right)$, where $\epsilon$ is a small value. It should be noted that the word vectors are regarded as parameters, and will be updated in the training process.

### Evaluation

We compare different methods on the Sentiment Treebank in this section to evaluate the effectiveness of our methods.

**SVM**. Support Vector Machine (SVM) achieves good performance in the sentiment classification task (Pang and Lee 2005). We use the bag-of-words features in our experiments.

**MNB/bi-MNB**. As indicated in the work of Wang and Manning (2012), Multinomial Naïve Bayes (MNB) often outperforms SVM for sentence-level sentiment classification. The MNB uses uni-gram features, and bi-MNB also uses bi-gram features.

**VecAvg**. This model (Landauer and Dutnais 1997) averages child vectors to obtain the parent vector. It ignores the word order when performing compositions.

[1] http://nlp.stanford.edu/sentiment/treebank.html

[2] http://www.rottentomatoes.com

| Method | Fine-grained | | Pos./Neg. | |
|---|---|---|---|---|
| | All | Root | All | Root |
| SVM | 64.3 | 40.7 | 84.6 | 79.4 |
| MNB | 67.2 | 41.0 | 82.6 | 81.8 |
| bi-MNB | 71.0 | 41.9 | 82.7 | 83.1 |
| VecAvg | 73.3 | 32.7 | 85.1 | 80.1 |
| MV-RNN | 78.7 | 44.4 | 86.8 | 82.9 |
| RNN | 79.0 | 43.2 | 86.1 | 82.4 |
| Avg-AdaMC-RNN | 80.1 | 43.4 | 89.1 | 84.9 |
| Max-AdaMC-RNN | 80.3 | 43.8 | 91.0 | 85.6 |
| Weighted-AdaMC-RNN | 80.7 | 45.4 | **93.6** | 86.5 |
| AdaMC-RNN | **80.8** | **45.8** | 93.4 | **87.1** |
| RNTN | 80.7 | 45.7 | 87.6 | 85.4 |
| Avg-AdaMC-RNTN | 80.6 | 45.7 | 89.7 | 86.3 |
| Max-AdaMC-RNTN | 80.3 | 45.6 | 91.3 | 86.6 |
| Weighted-AdaMC-RNTN | **81.0** | **46.3** | **93.8** | **88.4** |
| AdaMC-RNTN | **81.1** | **46.7** | **94.1** | **88.5** |

Table 1: Results of evaluation on the Sentiment Treebank. The top three methods are in **bold** and the best is also **underlined**. Our methods (AdaMC-RNN, AdaMC-RNTN) achieve best performances when $\beta$ is set to 2.

**RNN/RNTN**. Recursive Neural Network (Socher et al. 2011) computes the parent vector by weighted linear combination of the child vectors' dimensions. Recursive Neural Tensor Network (Socher et al. 2013b) employs tensors to model intersections between different dimension of child vectors. We use the same settings as in the original papers.

**MV-RNN**. This model (Socher et al. 2012) assigns a composition matrix for each word. Besides performing compositions to obtain the vector representations, the model uses another global matrix to combine these composition matrices for phrases. The above baseline results are reported in (Socher et al. 2013b).

**AdaMC**. Compared with the original models, AdaMC-RNN and AdaMC-RNTN employ multiple composition functions, and determine composition functions depending on the child vectors in every step. We use 15 composition functions, and set the size of word vectors as 25 for AdaMC-RNN and 15 for AdaMC-RNTN in the experiments.

**Avg/Max/Weighted-AdaMC**. Special cases of AdaMC are used in RNN and RNTN. The number of composition functions and dimension of word vectors are the same as in AdaMC-RNN and AdaMC-RNTN.

Table 1 shows the evaluation results of different models. SVM and MNB are two effective baselines for sentiment classification (Wang and Manning 2012). We notice that VecAvg performs better than bag-of-words methods on evaluations for all nodes, while the performances of root nodes are worse than them. It indicates that VecAvg achieves better results on short phrases, but it loses some sentiment information in the composition process for long phrases. We also obtain the conclusion that it is more difficult to get good composition results for long phrases than for short ones. So comparing the evaluation results for long fragments is more meaningful to us. Compared with VecAvg, the performances of RNN improve on both short and long phrases. MV-RNN uses more composition matrices and improves the accuracies than RNN. Moreover, the RNTN, which employs tensors to model the intersections between different semantic dimensions, achieves better results than MV-RNN, RNN and bag-of-words models. This illustrates that more powerful composition functions help capture complex semantic compositions especially for the longer phrases and the effectiveness of recursive neural models.

We then compare our Adaptive Multi-Compositionality (AdaMC) method with baselines. First of all, we apply our approaches in RNN, and there are significant gains of the evaluation metrics. Specifically, the all nodes and root nodes fine-grained accuracies of AdaMC-RNN increase by 1.8% and 2.6% respectively than RNN which only employs one global composition functions. For polarity (positive/negative) classification, the all nodes and root nodes accuracies of our method rise by 7.3% and 4.7% respectively.

Moreover, we find that AdaMC-RNN surpasses MV-RNN. The MV-RNN assigns specific composition matrix for every word. It is easy to overfit training data, because the number of composition matrices is the same as vocabulary size. MV-RNN needs another global matrix to compute the composition matrices for long phrases, which makes the composition non-specific. However, AdaMC-RNN employs a certain number of composition functions and adaptively selects them depending on the combined vectors. The experimental results illustrate that AdaMC-RNN is a better way to achieve specific compositions than MV-RNN.

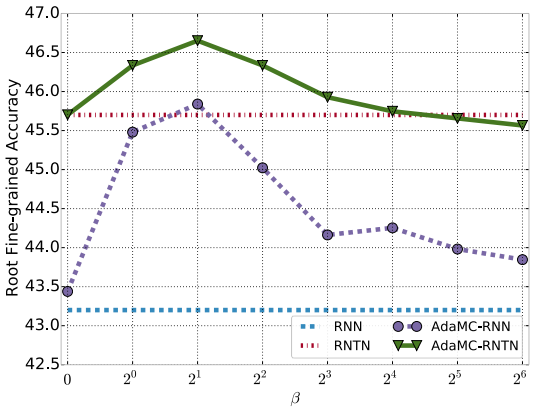
Furthermore, the fine-grained accuracies of AdaMC-RNN are comparable with RNTN, and the positive/negative accuracies are better than RNTN. Notably, although AdaMC-RNN employs fewer parameters than RNTN, it achieves better performances without employing tensors. The results indicate using multiple composition functions is another good approach to improve the semantic composition besides finding more powerful composition functions (such as tensors).

We also apply our method in RNTN, and obtain state-of-the-art performances. Compared with RNTN, all nodes and root nodes fine-grained accuracies of AdaMC-RNTN rise by 0.4% and 1.0% respectively. Considering positive/negative classification, the all nodes and root nodes accuracies of our method increase by 6.5% and 3.1% respectively than RNTN. This demonstrates our method can boost the performances even if we have used powerful composition functions.

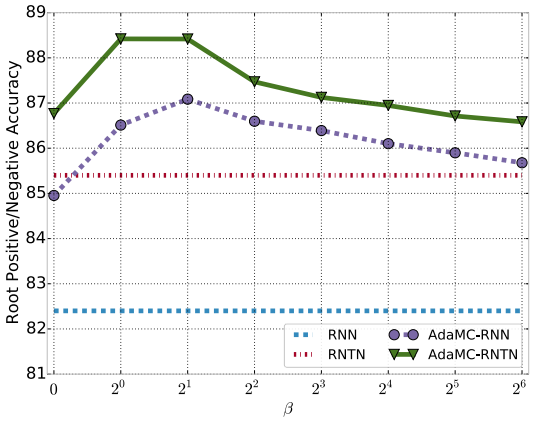
Finally, we evaluate the special cases ($\beta = 0, 1$, and $\beta \to \infty$) of AdaMC models, and they are all of help to improve the results. The performances of Weighted-AdaMC ($\beta = 1$) are most similar to AdaMC, and are better than Avg-AdaMC ($\beta = 0$) and Max-AdaMC ($\beta \to \infty$) in both RNN and RNTN. To be specific, the results of Max-AdaMC-RNN surpass Avg-AdaMC-RNN. However, the fine-grained accuracies of Avg-AdaMC-RNTN are slightly better than Max-AdaMC-RNTN, and the positive/negative accuracies are just the opposite. We will further explore the differences between these special cases in next section.

## Effects of $\beta$

We compare different $\beta$ for AdaMC defined in Equation (8). Different parameter $\beta$ leads to different composition selection schemes. When $\beta = 1$, the model directly employs the




(a) Root Fine-grained Accuracy




(b) Root Pos/Neg Accuracy

Figure 3: The curve shows the accuracy for root nodes as $\beta = 0, 2^0, 2^1, \ldots, 2^6$ increases. AdaMC-RNN and AdaMC-RNTN achieve the best results at $\beta = 2^1$.

| | |
|---|---|
| really bad | very bad / only dull / much bad / extremely bad / (all that) bad |
| (is n't) (necessarily bad) | (is n't) (painfully bad) / not mean-spirited / not (too slow) / not well-acted / (have otherwise) (been bland) |
| great (Broadway play) | great (cinematic innovation) / great subject / great performance / energetic entertainment / great (comedy filmmaker) |
| (arty and) jazzy | (Smart and) fun / (verve and) fun / (unique and) entertaining / (gentle and) engrossing / (warmth and) humor |

Table 2: We use cosine similarity of composition selection vectors (Equation (8)) for phrases to query the nearest compositions.

probabilities outputted by the softmax classifier as weights to combine the composition results. The $\beta = 0$ makes the probabilities obey a uniform distribution, while $\beta \to \infty$ results in a maximum probability selection algorithm.

As demonstrated in Figure 3, the overall conclusion is that the optimal $\beta$ tends to be between the setting of Weighted-AdaMC and Max-AdaMC. Both the AdaMC-RNN and AdaMC-RNTN achieve the best root fine-grained and positive/negative accuracies at $\beta = 2$, and they have a similar trend. Specifically, the Weighted-AdaMC ($\beta = 1$) performs better than Avg-AdaMC ($\beta = 0$) and Max-AdaMC ($\beta \to \infty$). It indicates that adaptive (role-sensitive) compositionality selection is useful to model the compositions.

The Avg-AdaMC does not consider role-sensitive information, while Max-AdaMC does not use multiple composition functions to get the smoothed results. Weighted-AdaMC employs the probabilities obtained by softmax classifier to make trade-offs between them, hence it obtains better performances. Based on this observation, we introduce the parameter $\beta$ and employ the Boltzmann distribution in AdaMC to adjust the effects of these two perspectives.

## Adaptive Compositionality Examples

To analyze the adaptive compositionality selection, we compute the probability distribution (as in Equation (8)) for every composition, i.e., $[P(g_1|\mathbf{a},\mathbf{b}) \ldots P(g_C|\mathbf{a},\mathbf{b})]^T$ where $\mathbf{a}, \mathbf{b}$ are the child vectors. As demonstrated in Table 2, we query some composition examples and employ cosine similarity as our similarity metric. To be specific, "*really bad*" is a strengthen composition which makes the sentiment polarity stronger, and the most similar compositions are of the same type. Notably, we notice that the bi-gram "*all that*" is detected as an intensification indicator. The second example is a negation composition. "*(have otherwise) (been bland)*" is regarded as a similar composition, which illustrates the combination of "*have*" and "*otherwise*" keeps the negation semantics of "*otherwise*". The third case and the similar compositions are the combinations of a sentiment adjective word and noun phrase. This demonstrates our model embeds some word-category information in word vectors to select the composition functions. The last example is two sentiment words connected by "*and*". The results illustrate our model recognize this kind of conjunction which joins two non-contrasting items. From all these cases, we find the compositions which are of similar types are closer, and our method learns to distinguish these different composition types according to the supervisions of specific tasks.

## Conclusion and Future Work

We propose an Adaptive Multi-Compositionality (AdaMC) method for recursive neural models to achieve better semantic compositions in this paper. AdaMC uses more than one composition functions and adaptively selects them depending on the input vectors. We present a general framework to model the composition as a distribution over the composition functions. We integrate AdaMC into existing popular recursive neural models (such as RNN and RNTN) and conduct experiments for sentence-level sentiment analysis

tasks. Experimental results on the Stanford Sentiment Treebank show that AdaMC significantly improves the baselines with fewer parameters. We further compare the distribution similarities of composition functions for phrase pairs, and the results verify the effectiveness of AdaMC on modeling and leveraging the semantic categories of words and phrases in the process of composition. There are several interesting directions for further research studies. For instance, we can evaluate our method in other NLP tasks. Moreover, external information (such as part-of-speech tags) can be used as features to select the composition functions. In addition, we can mix different types of composition functions (such as the linear combination approach in RNN and the tensor based approach in RNTN) to achieve more flexible choices in the adaptive composition methods.

## Acknowledgments

We gratefully acknowledge helpful discussions with Richard Socher. This research was partly supported by the National 863 Program of China (No. 2012AA011005), the fund of SKLSDE (Grant No. SKLSDE-2013ZX-06), and Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20111102110019).

## References

Aerts, D., and Czachor, M. 2004. Quantum aspects of semantic analysis and symbolic artificial intelligence. *Journal of Physics A: Mathematical and General* 37(12):L123.

Baroni, M., and Zamparelli, R. 2010. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *EMNLP*, EMNLP '10, 1183–1193.

Clark, S.; Coecke, B.; and Sadrzadeh, M. 2008. A compositional distributional model of meaning. In *Proceedings of the Second Quantum Interaction Symposium*, 133–140.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR* 12:2121–2159.

Georgii, H. 2011. *Gibbs Measures and Phase Transitions*. De Gruyter studies in mathematics. De Gruyter.

Grefenstette, E., and Sadrzadeh, M. 2011. Experimental support for a categorical compositional distributional model of meaning. In *EMNLP*, 1394–1404.

Grefenstette, E.; Dinu, G.; Zhang, Y.-Z.; Sadrzadeh, M.; and Baroni, M. 2013. Multi-step regression learning for compositional distributional semantics. In *Proceedings of the 10th International Conference on Computational Semantics*.

Hermann, K. M., and Blunsom, P. 2013. The role of syntax in vector space models of compositional semantics. In *ACL*, 894–904.

Jones, M. N., and Mewhort, D. J. 2007. Representing word meaning and order information in a composite holographic lexicon. *Psychological review* 114(1):1.

Klein, D., and Manning, C. D. 2003. Accurate unlexicalized parsing. In *ACL*, 423–430.

Landauer, T. K., and Dutnais, S. T. 1997. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review* 211–240.

Mitchell, J., and Lapata, M. 2008. Vector-based models of semantic composition. In *ACL*, 236–244.

Mitchell, J., and Lapata, M. 2010. Composition in distributional models of semantics. *Cognitive Science* 34(8):1388–1439.

Pang, B., and Lee, L. 2005. Seeing stars: exploiting class relationships for sentiment categorization with respect to rating scales. In *ACL*, 115–124.

Plate, T. 1991. Holographic reduced representations: Convolution algebra for compositional distributed representations. In *IJCAI*, 30–35. Citeseer.

Polajnar, T.; Fagarasan, L.; and Clark, S. 2013. Learning type-driven tensor-based meaning representations. *CoRR* abs/1312.5985.

Rudolph, S., and Giesbrecht, E. 2010. Compositional matrix-space models of language. In *ACL*, 907–916.

Rumelhart, D.; Hinton, G.; and Williams, R. 1986. Learning representations by back-propagating errors. *Nature* 323(6088):533–536.

Smolensky, P. 1990. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence* 46(12):159 – 216.

Socher, R.; Lin, C. C.; Ng, A. Y.; and Manning, C. D. 2011. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *ICML*.

Socher, R.; Huval, B.; Manning, C. D.; and Ng, A. Y. 2012. Semantic compositionality through recursive matrix-vector spaces. In *EMNLP-CoNLL*, 1201–1211.

Socher, R.; Bauer, J.; Manning, C. D.; and Ng, A. Y. 2013a. Parsing With Compositional Vector Grammars. In *ACL*.

Socher, R.; Perelygin, A.; Wu, J. Y.; Chuang, J.; Manning, C. D.; Ng, A. Y.; and Potts, C. 2013b. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *EMNLP*, 1631–1642.

Wang, S., and Manning, C. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *ACL*, 90–94.

Widdows, D. 2008. Semantic vector products: Some initial investigations. In *Second AAAI Symposium on Quantum Interaction*, volume 26, 28th. Citeseer.

Yessenalina, A., and Cardie, C. 2011. Compositional matrix-space models for sentiment analysis. In *EMNLP*, 172–182.

Yu, D.; Deng, L.; and Seide, F. 2013. The deep tensor neural network with applications to large vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on* 21(2):388–396.