博　　士　　論　　文

Joint Learning for Task-Oriented Representations in Natural Language Processing

(自然言語処理における
タスク指向表現の同時学習)

指導教員　　鶴岡 慶雅 准教授

東京大学大学院工学系研究科
電気系工学専攻

氏名　　**37-157077** 橋本 和真

提出日　　**2017**年**12**月**1**日

Abstract

Neural networks have been actively applied in the field of natural language processing in the recent years. Most of the neural network models in this field try to learn representations of meanings of words, phrases, sentences, and documents, using mathematical expressions. In existing neural network models in natural language processing, the overall systems are pipelined; that is, the overall systems consist of several sub-processes such as pre-processing (e.g., word segmentation and syntactic parsing). Moreover, each task is often handled separately, which leads to the absence of task-oriented information in lower-layer tasks, and to error propagations throughout the overall systems. In this dissertation, I propose joint learning methods to incorporate task-oriented information into representation learning for natural language processing. In experiments, I empirically show that learning task-oriented word embeddings, learning task-oriented semantic compositionality of phrases, and learning latent graph structures for sentences by a joint many-task model are all effective in improving accuracy on corresponding target tasks. I believe that the proposals in this dissertation have the promising potential to break the limitations in the existing pipelined natural language processing tasks.

Acknowledgements

This dissertation could not be completed without warm supports from surrounding researchers, friends, and family. First, I want to thank my supervisor, Yoshimasa Tsuruoka. He always helped me discuss new ideas, write papers, and many other things related to all of my research. I had been a member of his laboratory for six years; I started my first research here, and all of my research as a student were done with him. He let me think about my research by myself and do anything by myself as I want; if needed, he gave me insightful comments and suggestions. I really appreciate the relationship and learned many things from his way of thinking.

I also want to thank all of co-authors in my research papers. In particular, thanks to collaborative research and daily discussions with Akiko Eriguchi, my research areas have been widened, having led to working on machine translation and developing neural network libraries, and so on. Some of my work would not be possible without interactions with her.

In terms of research experiences, I would like to thank Richard Socher and Caiming Xiong for my intern at Salesforce Research, and Edouard Grave and Moustapha Cisse for my intern at Facebook AI Research. My research in NLP was motivated by Richard's talk in 2012, my research experiences at cutting-edge teams started from the interview with Caiming, and working on a new research area with Edouard and Moustapha was a great fun.

In the six years, having Hirotaka Kameko and Naoki Mizukami as my lab-mates was an important part of my laboratory life, although I regret not having had explicit opportunities to collaboratively work with them. It was a great fun to hear results about their research on Shogi commentary and Computer Mahjong player.

Finally, I would not be here without amazing supports from my family, allowing me to be a student for such a long time. I really appreciate the understandings about how challenging and interesting our research is.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Natural Language Processing (NLP) is a research area to make computers intelligently handle natural languages, such as Japanese and English. Recently statistical machine learning methods have been actively studied for tackling a variety of NLP tasks, according to the rapid increase of computational powers and the exponentially increasing text data in the Web. In particular, neural networks have become one of the most essential components in many NLP studies. Neural networks not only improve state-of-the-art accuracy on various kinds of NLP tasks, but also drastically change existing approaches. One typical example is Statistical Machine Translation (SMT); by neural networks, separately developed SMT components can be jointly modeled to train end-to-end single model which translates one language to another [Sutskever et al., 2014; Bahdanau et al., 2015].

Most of the neural network models in NLP assume that each word is represented with a real valued fixed-length vector, called *word embedding* [Bengio et al., 2003; Collobert et al., 2011; Mikolov et al., 2013b]. Word embeddings allow one to define similarity metrics for word meanings. Moreover, larger language units such as sentences and paragraphs can also be represented in a vector space. In particular, such sentence representations have been actively studied, including simple element-wise operations [Mitchell and Lapata, 2008], syntactic structure-based recursive neural networks [Socher et al., 2011a; Socher et al., 2012; Tai et al., 2015], and chain structure-based recurrent neural networks [Hochreiter and Schmidhuber, 1997]. The word embeddings and their composition functions to the compute sentence representations are parameterized in neural network models, and they are

learned in either supervised or unsupervised fashion. In supervised settings, human-annotated data is used in a variety of NLP tasks, such as machine translation and sentiment analysis. In unsupervised settings[1], large text corpora are used to model co-occurrence statistics of words [Firth, 1957; Mikolov et al., 2013b] and phrases [Hashimoto et al., 2014] for good feature extractors.

There are several assumptions in the above-mentioned representation learning methods. For example, co-occurrence-based vector representations capture general word and phrase similarities, and then they can be used in downstream tasks by fine-tuning the pre-trained parameters (or just by using the pre-trained parameters). Therefore, the pre-trained representations are not conditioned on any specific tasks. Another examples is the use of syntactic structures. Syntactic parsing is one of the most essential tasks in NLP and has been studied for a long time, and recently syntactic structures have proven to be effective in various kinds of neural NLP models [Eriguchi et al., 2016b; Eriguchi et al., 2017; Miwa and Bansal, 2016; Socher et al., 2011a; Socher et al., 2012; Tai et al., 2015]. The syntactic neural models rely on existing syntactic parsers either at training or test time. However, such syntactic parsers are trained by using human-annotated treebanks based on human-defined grammars, and thus it is not clear that relying on the existing parsers is the best option to improve accuracy on downstream NLP tasks. Moreover, the parsers would incorrectly parse input sentences in different domains because the parsers are usually trained with human-annotated data in one specific domain, such as news.

Such assumptions lead to constructing *pipelined* systems, where overall processes are split into several phases. So far we have commonly used the pipelined systems, but we can go beyond the pipelined systems by using recently developed sophisticated neural network models. For example, neural networks allow one to jointly handle multiple NLP tasks in a single model [Collobert et al., 2011] and moreover muti-modal data such as natural languages and images [Socher et al., 2014]. To improve upon the dominant pipelined approaches, I explore the ways to incorporate task-oriented information into representation learning methods in joint learning manners.

1.2 Proposals

In this dissertation, I propose several joint learning methods to incorporate task-oriented information into word and sentence representations. In particular, the applications are listed as follows:

[1] Human-written text is used as supervision, and in that sense, this can also be considered as supervised learning.

- learning task-oriented word embeddings,
- learning task-oriented semantic compositionality,
- learning a joint many-task model,
- learning task-oriented dependency graph structures.

Learning task-oriented word embeddings I propose a word embedding learning method which explicitly incorporates task-oriented features for relation extraction tasks. Standard word embeddings are learned with large text corpora by modeling word co-occurrence statistics. By contrast, my proposal incorporates classical feature representations for relation extraction in learning word embeddings to better capture task-oriented information from large text corpora.

Learning task-oriented semantic compositionality I propose a representation learning method for phrases by jointly learning task-oriented semantic compositionality detection model. There exist semantically compositional and non-compositional phrases, but it is challenging to manually define their compositionality levels. My proposal automatically detects the compositionality level for each phrase to better represent the meaning of each phrase by using both compositional and non-compositional representations. The overall model is optimized according to target tasks.

Learning a joint many-task model I propose a joint many-task model which handles five different NLP tasks in a single hierarchical neural network model. The hierarchical model starts from simple tasks and then gradually moves to more complex tasks, enabling interactions between different levels of NLP tasks. This hierarchical model leads to my next proposal to break the limitations of standard pipelined systems using syntactic structures.

Learning task-oriented dependency graph structures Finally, I propose a joint learning method to learn task-oriented dependency graph structures. In standard pipelined systems, syntactic structures are given by existing syntactic parsers. By contrast, my proposal allows one to learn dependency parsers which are optimized according to a specific task. Therefore, the proposed method can induce novel sentence structures which are different from human-defined grammars.

1.3 Contributions

The contributions of my proposals are summarized as follows:

Showing the importance of task-oriented pre-training Most of the existing approaches to pre-training word embeddings are solely based on language modeling (i.e., co-occurrence statistics). By showing the importance of incorporating task-specific information into the pre-training processes with large text corpora, my proposal encourages researchers to tackle the challenging but promising direction of more effectively using unannotated web-scale text data.

Showing the importance of semantic compositionality detection Existing approaches to representing meanings of words and phrases rely on pre-defined language units in pipelined manners. By showing the importance of learning compositionality of phrases in a task-oriented manner, my proposal encourages researchers to develop more effective methods than just compositionally treating sequences (or structures) of tokens.

Showing the importance of jointly modeling different tasks in a single model Many NLP tasks are related to each other, although most of the times they are separately handled in complex neural network models. By showing the effective hierarchical deep structures for efficiently handling multiple NLP tasks, my proposal encourages researchers to explore this research direction to better handle multiple NLP tasks by allowing them to interact with each other in a single model. Moreover, the proposed joint many-task model leads to my next proposal which has the potential to break the limitations in pipelined NLP systems.

Showing the possibility of breaking the limitations of pipelined systems The use of syntactic structures obtained by external syntactic parsers is a typical and crucial example in many pipelined NLP systems. By showing the effectiveness of learning latent structures inherent in natural language sentences in a task-oriented manner, my proposal encourages researchers to tackle the long-term challenge in NLP to break the limitations existing in many pipelined NLP systems.

1.4 Structure of this Dissertation

The remaining part of this dissertation is structured as follows:

Chapter 2 First, I describe how neural networks are used in NLP, which leads to the proposals in this dissertation.

Chapter 3 I present a method to learn task-oriented word embeddings specifically designed for relation extraction tasks.

Chapter 4 I present a method to automatically and softly determine which phrases are compositional or non-compositional, by jointly learning a compositionality detection model in a task-oriented manner.

Chapter 5 I present a method to learn task-oriented latent structures inherent in natural language sentences, for machine translation.

Chapter 6 Finally, I summarize this dissertation and discuss future directions.

Chapter 2

Neural Networks for Natural Language Processing

This chapter briefly introduces basics of "neural networks for NLP" and how the proposals in this dissertation are motivated by existing work. Several sample codes can be found at my public repositories: `https://github.com/hassyGo`. Basics of neural networks are well summarized in Goodfellow et al. (2016).

2.1 Word Embeddings

In the field of NLP, one of the most important units of language meanings is a word. Therefore researchers have worked on how to effectively represent word meanings in computers. In recent work, one dominant approach is embedding each word in a high-dimensional vector space [Bengio et al., 2003; Collobert et al., 2011; Mikolov et al., 2013b; Pennington et al., 2014]; therefore, such vectors representing the words are called *word embeddings*. For the word embeddings, a word embedding matrix is parameterized as

$$W_e \in \mathbb{R}^{d \times |V|}, \tag{2.1}$$

where d is the dimensionality of the embedding vectors, and V is a vocabulary including all the words used in a neural network model. For the i-th word w_i in the vocabulary, its word embedding

$v(w_i) \in \mathbb{R}^d$ is computed as follows:

$$v(w_i) = W_e[0, 0, \ldots, 0, 0, 1, 0, 0, \ldots, 0], \tag{2.2}$$

where the rightmost vector is so-called a one-hot vector for the i-th word. This computation is thus very sparse, and in practice we can just lookup the i-th column vector in W_e when implementing the word embeddings. The word embedding matrix is initialized with random values and then automatically tuned by error backpropagation with respect to a loss function L for specific tasks:

$$\frac{\partial L}{\partial W_e}, \tag{2.3}$$

or we can also write this equation for each word embedding used in a parameter update step:

$$\frac{\partial L}{\partial v(w_i)}. \tag{2.4}$$

For example, such tasks include word co-occurrence statistics modeling [Collobert et al., 2011; Mikolov et al., 2013b; Pennington et al., 2014], syntactic parsing [Socher et al., 2013; Stenetorp, 2013], machine translation [Sutskever et al., 2014], and many other NLP tasks.

2.1.1 Learning with Co-Occurrence Statistics

One major approach to learning the word embeddings is the use of large text corpora for modeling co-occurrence statistics of words, following the distributional hypothesis [Firth, 1957]. For example, text extracted from Wikipedia is widely used to learn the word embeddings. By using such text data, we can use millions or billions of sentences for training. There exist several ways to model the co-occurrence statistics, and one of the most widely-used methods is called Skipgram [Mikolov et al., 2013a; Mikolov et al., 2013b].[1] The idea of Skipgram is defining a word prediction task where context words are predicted for each word in a text corpus. In probabilistic language modeling, the task is predicting words given their context words, and previous methods also followed the language modeling ideas [Bengio et al., 2003; Collobert et al., 2011].

For each word w and its context word $\overline{w}$, the task is maximizing the probability $p(\overline{w}|w)$, and the loss function is defined as follows:

$$L(w, \overline{w}) = -\log p(\overline{w}|w), \tag{2.5}$$

[1] The Skipgram method is described just as an example here.

and the negative log-likelihood is one of the most used loss functions. Using the word embedding matrix W_e and another matrix $\tilde{W}_e$ of the same size, $p(\overline{w}|w)$ is computed as follows:

$$p(\overline{w}|w) = \frac{\exp\left(v(w) \cdot \tilde{v}(\overline{w})\right)}{\sum_{i=1}^{|V|} \exp\left(v(w) \cdot \tilde{v}(w_i)\right)}, \tag{2.6}$$

where the function is called *softmax*. That is, this is just a multi-class classification task with $|V|$ classes; however, obviously, the computational cost is very large when the vocabulary size is huge (e.g., $|V| = 100,000$). Assuming that what we need is not the precise probability value, but the learned embedding matrix, Mikolov et al. (2013b) proposed a very simple and fast method by avoiding the exact softmax computations by a *negative sampling* technique:

$$L(w, \overline{w}) = -\log \sigma(v(w) \cdot \tilde{v}(\overline{w})) - \sum_{i=1}^{N} \log \sigma(-v(w) \cdot \tilde{v}(w_i)), \tag{2.7}$$

where

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2.8}$$

is the logistic sigmoid function, and $N \ll |V|$ is the number of negative samples for each observed word. This loss function is similar to that of k-versus-rest multi-class classification.

The learned word embeddings capture general semantic or syntactic similarities between words, based on their usage in the training corpora. That is, not only synonym pairs, but also antonym pairs could be close to each other in the learned vector space. However, such word similarities should be downstream task-dependent; for example, in sentiment analysis, we do not want "good" and "bad" to be close to each other, but in the case of syntactic parsing, they can be close in terms of their syntactic roles. It should be thus useful if we can incorporate more task-oriented information for specific target tasks into the embedding learning process.

2.1.2 Learning with Human-Annotated Data

The co-occurrence-based word embeddings are widely used to initialize word embedding matrices in neural network models for many other tasks. For example, Ramachandran et al. (2017) have shown that pre-training model parameters by large scale language modeling is effective in improving sequence-to-sequence learning models. By the pre-training and fine-tuning techniques, we can incorporate task-oriented information into the word embeddings. However, for almost all of the NLP

tasks, the human-annotated data is limited and very expensive to construct. It is thus promising to explore the ways to effectively incorporate task-oriented information at the pre-training step with the large text data.

2.2 Phrase and Sentence Embeddings

Once we obtain the word representations, we also want to represent meanings of larger units, such as phrases and sentences. Modeling paragraph or document representations is also crucial in NLP, but here I focus on phrases and sentences.

2.2.1 Word Sequences

One recent dominant approach is to represent each phrase or sentence as a sequence of words, as typically and successfully used in sequence-to-sequence models [Sutskever et al., 2014]. Much smaller units, such as characters or sub-words, can also be used [Sennrich et al., 2016; Luong and Manning, 2016]. Another approach is to treat some phrases as single tokens (just like words) and to directly learn their embeddings [Mikolov et al., 2013b]. In any ways, the base units are pre-defined in pre-processing steps, and thus the overall processes are pipelined. In the pipelined systems, there is no reason why a specific pre-processing step is the best option for downstream tasks.

2.2.2 Syntactic Structures

Another line of learning phrase or sentence representations is using syntactic structures obtained by external syntactic parsers [Socher et al., 2011a; Socher et al., 2012; Socher et al., 2014]. In my previous work, it has been shown that phrase embeddings as well as word embeddings can be leaned by modeling "co-occurrence statistics of words and phrases" [Hashimoto et al., 2014]. In recently developed neural machine translation models which are mainly based on sequence-to-sequence learning, it has been shown that incorporating syntactic structures, such as constituency or dependency trees, is effective in improving translation accuracy [Eriguchi et al., 2016b; Eriguchi et al., 2017]. Like this, it is still worth investigating how to effectively use structures inherent in natural language sentences, although the sequence-to-sequence approaches are simpler and achieve state-of-the-art scores on many benchmark datasets.

However, again, the use of external parsers leads to constructing pipelined systems, where parsing errors are propagated throughout the systems. The overall accuracy would depend on the accuracy of the parsers, and it is not obvious that using the existing syntactic parsers trained with human-annotated treebanks is the best option for the downstream tasks. Therefore, it is a promising direction to explore the ways to learn task-oriented structures inherent in the sentences.

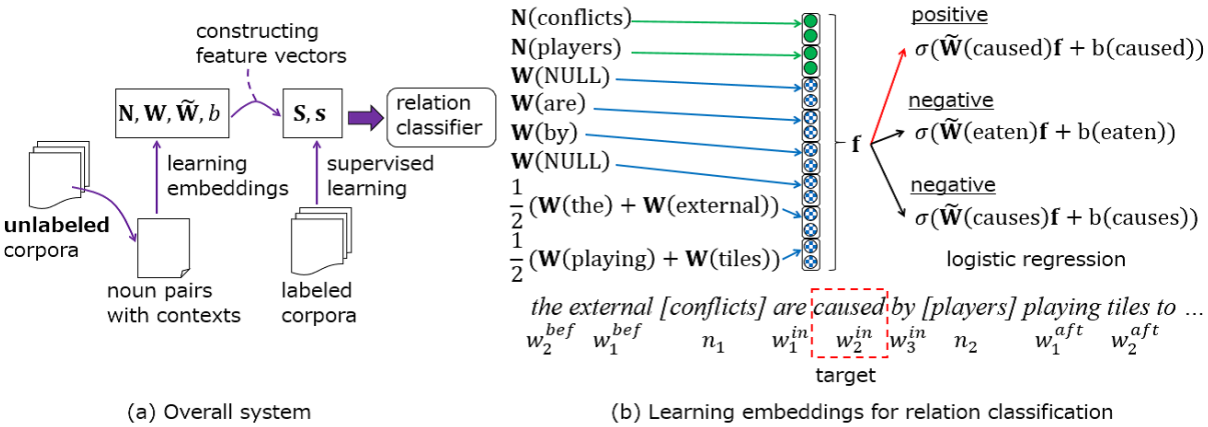
Chapter 3

Task-Oriented Learning of Word Embeddings

In this chapter, I describe a method for learning word embeddings where task-oriented feature representations for relation extraction are incorporated, by using large text corpora. Most of the existing word embedding learning methods use large text corpora to model co-occurrence statistics of words (or phrases). By contrast, this chapter presents how to incorporate task-specific information into the co-occurrence-based learning methods. This chapter corresponds to the following published paper:

Kazuma Hashimoto, Pontus Stenetorp, Makoto Miwa, and Yoshimasa Tsuruoka. 2015. Task-Oriented Learning of Word Embeddings for Semantic Relation Classification. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pp. 268–278.

Summary We present a novel learning method for word embeddings designed for relation classification. Our word embeddings are trained by predicting words between noun pairs using lexical relation-specific features on a large unlabeled corpus. This allows us to explicitly incorporate relation-specific information into the word embeddings. The learned word embeddings are then used to construct feature vectors for a relation classification model. On a well-established semantic relation classification task, our method significantly outperforms a baseline based on a previously introduced word embedding method, and compares favorably to previous state-of-the-art models that use syntactic information or manually constructed external resources.



(a) Overall system (b) Learning embeddings for relation classification

Figure 3.1: The overview of our system (a) and the embedding learning method (b). In the example sentence, each of *are*, *caused*, and *by* is treated as a target word to be predicted during training.

3.1 Introduction

Automatic classification of semantic relations has a variety of applications, such as information extraction and the construction of semantic networks [Girju et al., 2007; Hendrickx et al., 2010]. A traditional approach to relation classification is to train classifiers using various kinds of features with class labels annotated by humans. Carefully crafted features derived from lexical, syntactic, and semantic resources play a significant role in achieving high accuracy for semantic relation classification [Rink and Harabagiu, 2010].

In recent years there has been an increasing interest in using *word embeddings* as an alternative to traditional hand-crafted features. Word embeddings are represented as real-valued vectors and capture syntactic and semantic similarity between words. For example, *word2vec*[1] [Mikolov et al., 2013b] is a well-established tool for learning word embeddings. Although word2vec has successfully been used to learn word embeddings, these kinds of word embeddings capture only co-occurrence relationships between words [Levy and Goldberg, 2014a]. While simply adding word embeddings trained using window-based contexts as additional features to existing systems has proven valuable [Turian et al., 2010], more recent studies have focused on how to tune and enhance word embeddings for specific

[1] https://code.google.com/p/word2vec/.

tasks [Bansal et al., 2014; Boros et al., 2014; Chen et al., 2014; Guo et al., 2014; Nguyen and Grishman, 2014] and we continue this line of research for the task of relation classification.

In this work we present a learning method for word embeddings specifically designed to be useful for relation classification. The overview of our system and the embedding learning process are shown in Figure 3.1. First we train word embeddings by predicting each of the words between noun pairs using lexical relation-specific features on a large unlabeled corpus. We then use the word embeddings to construct lexical feature vectors for relation classification. Lastly, the feature vectors are used to train a relation classification model.

We evaluate our method on a well-established semantic relation classification task and compare it to a baseline based on word2vec embeddings and previous state-of-the-art models that rely on either manually crafted features, syntactic parses or external semantic resources. Our method significantly outperforms the word2vec-based baseline, and compares favorably with previous state-of-the-art models, despite relying only on lexical level features and no external annotated resources. Furthermore, our qualitative analysis of the learned embeddings shows that n-grams of our embeddings capture salient syntactic patterns similar to semantic relation types.

3.2 Related Work

A traditional approach to relation classification is to train classifiers in a supervised fashion using a variety of features. These features include lexical bag-of-words features and features based on syntactic parse trees. For syntactic parse trees, the paths between the target entities on constituency and dependency trees have been demonstrated to be useful [Bunescu and Mooney, 2005; Zhang et al., 2006]. On the shared task introduced by Hendrickx et al. (2010), Rink and Harabagiu (2010) achieved the best score using a variety of hand-crafted features which were then used to train a Support Vector Machine (SVM).

Recently, word embeddings have become popular as an alternative to hand-crafted features [Collobert et al., 2011]. However, one of the limitations is that word embeddings are usually learned by predicting a target word in its context, leading to only local co-occurrence information being captured [Levy and Goldberg, 2014a]. Thus, several recent studies have focused on overcoming this limitation. Le and Mikolov (2014) integrated paragraph information into a word2vec-based model, which allowed them to capture paragraph-level information. For dependency parsing, Bansal et al. (2014) and Chen et al. (2014) found ways to improve performance by integrating dependency-based

context information into their embeddings. Bansal et al. (2014) trained embeddings by defining parent and child nodes in dependency trees as contexts. Chen et al. (2014) introduced the concept of feature embeddings induced by parsing a large unannotated corpus and then learning embeddings for the manually crafted features. For information extraction, Boros et al. (2014) trained word embeddings relevant for event role extraction, and Nguyen and Grishman (2014) employed word embeddings for domain adaptation of relation extraction. Another kind of task-specific word embeddings was proposed by Tang et al. (2014), which used sentiment labels on tweets to adapt word embeddings for a sentiment analysis tasks. However, such an approach is only feasible when a large amount of labeled data is available.

3.3 Relation Classification Using Word Embedding-based Features

We propose a novel method for learning word embeddings designed for relation classification. The word embeddings are trained by *predicting each word between noun pairs*, given the corresponding low-level features for relation classification. In general, to classify relations between pairs of nouns the most important features come from the pairs themselves and the words between and around the pairs [Hendrickx et al., 2010]. For example, in the sentence in Figure 3.1 (b) there is a *cause-effect* relationship between the two nouns *conflicts* and *players*. To classify the relation, the most common features are the noun pair (*conflicts, players*), the words between the noun pair (*are, caused, by*), the words before the pair (*the, external*), and the words after the pair (*playing, tiles, to, ...*). As shown by Rink and Harabagiu (2010), the words between the noun pairs are the most effective among these features. Our main idea is to treat the most important features (the words between the noun pairs) as the targets to be predicted and other lexical features (noun pairs, words outside them) as their contexts. Due to this, we expect our embeddings to capture relevant features for relation classification better than previous models which only use window-based contexts.

In this section we first describe the learning process for the word embeddings, focusing on lexical features for relation classification (Figure 3.1 (b)). We then propose a simple and powerful technique to construct features which serve as input for a softmax classifier. The overview of our proposed system is shown in Figure 3.1 (a).

3.3.1 Learning Word Embeddings

Assume that there is a noun pair $\mathbf{n} = (n_1, n_2)$ in a sentence with M_{in} words between the pair and M_{out} words before and after the pair:

- $\mathbf{w}_{in} = (w_1^{in}, \ldots, w_{M_{in}}^{in})$,
- $\mathbf{w}_{bef} = (w_1^{bef}, \ldots, w_{M_{out}}^{bef})$, and
- $\mathbf{w}_{aft} = (w_1^{aft}, \ldots, w_{M_{out}}^{aft})$.

Our method predicts each target word $w_i^{in} \in \mathbf{w}_{in}$ using three kinds of information: $\mathbf{n}$, words around w_i^{in} in $\mathbf{w}_{in}$, and words in $\mathbf{w}_{bef}$ and $\mathbf{w}_{aft}$. Words are embedded in a d-dimensional vector space and we refer to these vectors as word embeddings. To discriminate between words in $\mathbf{n}$ from those in $\mathbf{w}_{in}$, $\mathbf{w}_{bef}$, and $\mathbf{w}_{aft}$, we have two sets of word embeddings: $\mathbf{N} \in \mathbb{R}^{d \times |\mathcal{N}|}$ and $\mathbf{W} \in \mathbb{R}^{d \times |\mathcal{W}|}$. $\mathcal{W}$ is a set of words and $\mathcal{N}$ is also a set of words but contains only nouns. Hence, the word *cause* has two embeddings: one in $\mathbf{N}$ and another in $\mathbf{W}$. In general *cause* is used as a noun and a verb, and thus we expect the noun embeddings to capture the meanings focusing on their noun usage. This is inspired by some recent work on word representations that explicitly assigns an independent representation for each word usage according to its part-of-speech tag [Baroni and Zamparelli, 2010; Grefenstette and Sadrzadeh, 2011; Hashimoto et al., 2013; Hashimoto et al., 2014; Kartsaklis and Sadrzadeh, 2013].

A feature vector $\mathbf{f} \in \mathbb{R}^{2d(2+c)\times 1}$ is constructed to predict w_i^{in} by concatenating word embeddings:

$$\begin{aligned} \mathbf{f} = [&\mathbf{N}(n_1); \mathbf{N}(n_2); \mathbf{W}(w_{i-1}^{in}); \ldots; \mathbf{W}(w_{i-c}^{in}); \\ &\mathbf{W}(w_{i+1}^{in}); \ldots; \mathbf{W}(w_{i+c}^{in}); \\ &\frac{1}{M_{out}} \sum_{j=1}^{M_{out}} \mathbf{W}(w_j^{bef}); \frac{1}{M_{out}} \sum_{j=1}^{M_{out}} \mathbf{W}(w_j^{aft})] \, . \end{aligned} \tag{3.1}$$

$\mathbf{N}(\cdot)$ and $\mathbf{W}(\cdot) \in \mathbb{R}^{d\times 1}$ corresponds to each word and c is the context size. A special *NULL* token is used if $i - j$ is smaller than 1 or $i + j$ is larger than M_{in} for each $j \in \{1, 2, \ldots, c\}$.

Our method then estimates a conditional probability $p(w|\mathbf{f})$ that the target word is a word w given the feature vector $\mathbf{f}$, using a logistic regression model:

$$p(w|\mathbf{f}) = \sigma(\tilde{\mathbf{W}}(w) \cdot \mathbf{f} + b(w)) \, , \tag{3.2}$$

where $\tilde{\mathbf{W}}(w) \in \mathbb{R}^{2d(2+c)\times 1}$ is a weight vector for w, $b(w) \in \mathbb{R}$ is a bias for w, and $\sigma(x) = \frac{1}{1+\mathrm{e}^{-x}}$ is the logistic function. Each column vector in $\tilde{\mathbf{W}} \in \mathbb{R}^{2d(c+1)\times|\mathcal{W}|}$ corresponds to a word. That is, we assign a logistic regression model for each word, and we can train the embeddings using the one-versus-rest approach to make $p(w_i^{in}|\mathbf{f})$ larger than $p(w'|\mathbf{f})$ for $w' \neq w_i^{in}$. However, naively optimizing the parameters of those logistic regression models would lead to prohibitive computational cost since it grows linearly with the size of the vocabulary.

When training we employ several procedures introduced by Mikolov et al. (2013b), namely, *negative sampling*, a modified unigram noise distribution and *subsampling*. For negative sampling the model parameters $\mathbf{N}$, $\mathbf{W}$, $\tilde{\mathbf{W}}$, and b are learned by maximizing the objective function $J_{unlabeled}$:

$$\sum_{\mathbf{n}} \sum_{i=1}^{M_{in}} \left(\log(p(w_i^{in}|\mathbf{f})) + \sum_{j=1}^{k} \log(1 - p(w_j'|\mathbf{f})) \right) , \tag{3.3}$$

where w_j' is a word randomly drawn from the unigram noise distribution weighted by an exponent of 0.75. Maximizing $J_{unlabeled}$ means that our method can discriminate between each target word and k noise words given the target word's context. This approach is much less computationally expensive than the one-versus-rest approach and has proven effective in learning word embeddings.

To reduce redundancy during training we use subsampling. A training sample, whose target word is w, is discarded with the probability $P_d(w) = 1 - \sqrt{\frac{t}{p(w)}}$, where t is a threshold which is set to 10^{-5} and $p(w)$ is a probability corresponding to the frequency of w in the training corpus. The more frequent a target word is, the more likely it is to be discarded. To further emphasize infrequent words, we apply the subsampling approach not only to target words, but also to noun pairs; concretely, by drawing two random numbers r_1 and r_2, a training sample whose noun pair is (n_1, n_2) is discarded if $P_d(n_1)$ is larger than r_1 or $P_d(n_2)$ is larger than r_2.

Since the feature vector $\mathbf{f}$ is constructed as defined in Eq. (3.1), at each training step, $\tilde{\mathbf{W}}(w)$ is updated based on information about what pair of nouns surrounds w, what word n-grams appear in a small window around w, and what words appear outside the noun pair. Hence, the weight vector $\tilde{\mathbf{W}}(w)$ captures rich information regarding the target word w.

3.3.2 Constructing Feature Vectors

Once the word embeddings are trained, we can use them for relation classification. Given a noun pair $\mathbf{n} = (n_1, n_2)$ with its context words $\mathbf{w}_{in}$, $\mathbf{w}_{bef}$, and $\mathbf{w}_{aft}$, we construct a feature vector to classify the

relation between n_1 and n_2 by concatenating three kinds of feature vectors:

$\mathbf{g}_{\mathrm{n}}$ the word embeddings of the noun pair,

$\mathbf{g}_{\mathrm{in}}$ the averaged n-gram embeddings between the pair, and

$\mathbf{g}_{\mathrm{out}}$ the concatenation of the averaged word embeddings in $\mathbf{w}_{bef}$ and $\mathbf{w}_{aft}$.

The feature vector $\mathbf{g}_{\mathrm{n}} \in \mathbb{R}^{2d \times 1}$ is the concatenation of $\mathbf{N}(n_1)$ and $\mathbf{N}(n_2)$:

$$\mathbf{g}_{\mathrm{n}} = [\mathbf{N}(n_1); \mathbf{N}(n_2)] \,. \tag{3.4}$$

Words between the noun pair contribute to classifying the relation, and one of the most common ways to incorporate an arbitrary number of words is treating them as a bag of words. However, word order information is lost for bag-of-words features such as averaged word embeddings. To incorporate the word order information, we first define n-gram embeddings $\mathbf{h}_i \in \mathbb{R}^{4d(1+c) \times 1}$ between the noun pair:

$$\begin{aligned} \mathbf{h}_i = [&\mathbf{W}(w_{i-1}^{in}); \ldots; \mathbf{W}(w_{i-c}^{in}); \\ &\mathbf{W}(w_{i+1}^{in}); \ldots; \mathbf{W}(w_{i+c}^{in}); \tilde{\mathbf{W}}(w_i^{in})] \,. \end{aligned} \tag{3.5}$$

Note that $\tilde{\mathbf{W}}$ can also be used and that the value used for n is $(2c+1)$. As described in Section 3.3.1, $\tilde{\mathbf{W}}$ captures meaningful information about each word and after the first embedding learning step we can treat the embeddings in $\tilde{\mathbf{W}}$ as features for the words. Mnih and Kavukcuoglu (2013) have demonstrated that using embeddings like those in $\tilde{\mathbf{W}}$ is useful in representing the words. We then compute the feature vector $\mathbf{g}_{\mathrm{in}}$ by averaging $\mathbf{h}_i$:

$$\mathbf{g}_{\mathrm{in}} = \frac{1}{M_{in}} \sum_{i=1}^{M_{in}} \mathbf{h}_i \,. \tag{3.6}$$

We use the averaging approach since M_{in} depends on each instance. The feature vector $\mathbf{g}_{\mathrm{in}}$ allows us to represent word sequences of arbitrary lengths as fixed-length feature vectors using the simple operations: concatenation and averaging.

The words before and after the noun pair are sometimes important in classifying the relation. For example, in the phrase "pour n_1 into n_2", the word *pour* should be helpful in classifying the relation.

As with Eq. (3.1), we use the concatenation of the averaged word embeddings of words before and after the noun pair to compute the feature vector $\mathbf{g}_{\text{out}} \in \mathbb{R}^{2d\times 1}$:

$$\mathbf{g}_{\text{out}} = \frac{1}{M_{out}}[\sum_{j=1}^{M_{out}} \mathbf{W}(w_j^{bef}); \sum_{j=1}^{M_{out}} \mathbf{W}(w_j^{aft})] \,. \tag{3.7}$$

As described above, the overall feature vector $\mathbf{e} \in \mathbb{R}^{4d(2+c)\times 1}$ is constructed by concatenating $\mathbf{g}_{\text{n}}$, $\mathbf{g}_{\text{in}}$, and $\mathbf{g}_{\text{out}}$. We would like to emphasize that we only use simple operations: averaging and concatenating the learned word embeddings. The feature vector $\mathbf{e}$ is then used as input for a softmax classifier, without any complex transformation such as matrix multiplication with non-linear functions.

3.3.3 Supervised Learning

Given a relation classification task we train a softmax classifier using the feature vector $\mathbf{e}$ described in Section 3.3.2. For each k-th training sample with a corresponding label l_k among L predefined labels, we compute a conditional probability given its feature vector $\mathbf{e}_k$:

$$p(l_k|\mathbf{e}_k) = \frac{\exp(\mathbf{o}(l_k))}{\sum_{i=1}^{L}\exp(\mathbf{o}(i))}, \tag{3.8}$$

where $\mathbf{o} \in \mathbb{R}^{L\times 1}$ is defined as $\mathbf{o} = \mathbf{S}\mathbf{e}_k + \mathbf{s}$, and $\mathbf{S} \in \mathbb{R}^{L\times 4d(2+c)}$ and $\mathbf{s} \in \mathbb{R}^{L\times 1}$ are the softmax parameters. $\mathbf{o}(i)$ is the i-th element of $\mathbf{o}$. We then define the objective function as:

$$J_{labeled} = \sum_{k=1}^{K} \log(p(l_k|\mathbf{e}_k)) - \frac{\lambda}{2}\|\theta\|^2 \,. \tag{3.9}$$

K is the number of training samples and λ controls the L-2 regularization. $\theta = (\mathbf{N}, \mathbf{W}, \tilde{\mathbf{W}}, \mathbf{S}, \mathbf{s})$ is the set of parameters and $J_{labeled}$ is maximized using AdaGrad [Duchi et al., 2011]. We have found that *dropout* [Hinton et al., 2012a] is helpful in preventing our model from overfitting. Concretely, elements in $\mathbf{e}$ are randomly omitted with a probability of 0.5 at each training step. Recently dropout has been applied to deep neural network models for natural language processing tasks and proven effective [Irsoy and Cardie, 2014; Paulus et al., 2014].

In what follows, we refer to the above method as **RelEmb**. While RelEmb uses only low-level features, a variety of useful features have been proposed for relation classification. Among them, we use dependency path features [Bunescu and Mooney, 2005] based on the untyped binary dependencies of the Stanford parser to find the shortest path between target nouns. The dependency path

features are computed by averaging word embeddings from $\mathbf{W}$ on the shortest path, and are then concatenated to the feature vector $\mathbf{e}$. Furthermore, we directly incorporate semantic information using word-level semantic features from Named Entity (NE) tags and WordNet hypernyms, as used in previous work [Rink and Harabagiu, 2010; Socher et al., 2012; Yu et al., 2014]. We refer to this extended method as **RelEmb**$_{\mathrm{FULL}}$. Concretely, RelEmb$_{\mathrm{FULL}}$ uses the same binary features as in Socher et al. (2012). The features come from NE tags and WordNet hypernym tags of target nouns provided by a sense tagger [Ciaramita and Altun, 2006].

3.4 Experimental Settings

3.4.1 Training Data

For pre-training we used a snapshot of the English Wikipedia[2] from November 2013. First, we extracted 80 million sentences from the original Wikipedia file, and then used *Enju*[3] [Miyao and Tsujii, 2008] to automatically assign part-of-speech (POS) tags. From the POS tags we used *NN*, *NNS*, *NNP*, or *NNPS* to locate noun pairs in the corpus. We then collected training data by listing pairs of nouns and the words between, before, and after the noun pairs. A noun pair was omitted if the number of words between the pair was larger than 10 and we consequently collected 1.4 billion pairs of nouns and their contexts [4]. We used the 300,000 most frequent words and the 300,000 most frequent nouns and treated out-of-vocabulary words as a special *UNK* token.

3.4.2 Initialization and Optimization

We initialized the embedding matrices $\mathbf{N}$ and $\mathbf{W}$ with zero-mean gaussian noise with a variance of $\frac{1}{d}$. $\tilde{\mathbf{W}}$ and b were zero-initialized. The model parameters were optimized by maximizing the objective function in Eq. (3.3) using stochastic gradient ascent. The learning rate was set to α and linearly decreased to 0 during training, as described in Mikolov et al. (2013a). The hyperparameters are the embedding dimensionality d, the context size c, the number of negative samples k, the initial learning

[2] `http://dumps.wikimedia.org/enwiki/`.

[3] Despite Enju being a syntactic parser we only use the POS tagger component. The accuracy of the POS tagger is about 97.2% on the WSJ corpus.

[4] The training data, the training code, and the learned model parameters used in this paper are publicly available at `http://www.logos.t.u-tokyo.ac.jp/~hassy/publications/conll2015/`

rate α, and M_{out}, the number of words outside the noun pairs. For hyperparameter tuning, we first fixed α to 0.025 and M_{out} to 5, and then set d to $\{50,\ 100,\ 300\}$, c to $\{1,\ 2,\ 3\}$, and k to $\{5,\ 15,\ 25\}$.

At the supervised learning step, we initialized $\mathbf{S}$ and $\mathbf{s}$ with zeros. The hyperparameters, the learning rate for AdaGrad, λ, M_{out}, and the number of iterations, were determined via 10-fold cross validation on the training set for each setting. Note that M_{out} can be tuned at the supervised learning step, adapting to a specific dataset.

3.5 Evaluation

3.5.1 Evaluation Dataset

We evaluated our method on the SemEval 2010 Task 8 data set[5] [Hendrickx et al., 2010], which involves predicting the semantic relations between noun pairs in their contexts. The dataset, containing 8,000 training and 2,717 test samples, defines nine classes (*Cause-Effect, Entity-Origin, etc.*) for ordered relations and one class (*Other*) for other relations. Thus, the task can be treated as a 19-class classification task. Two examples from the training set are shown below.

(a) Financial [stress]$_{\mathrm{E}_1}$ is one of the main causes of [divorce]$_{\mathrm{E}_2}$

(b) The [burst]$_{\mathrm{E}_1}$ has been caused by water hammer [pressure]$_{\mathrm{E}_2}$

Training example (a) is classified as *Cause-Effect(E_1, E_2)* which denotes that E_2 is an effect caused by E_1, while training example (b) is classified as *Cause-Effect(E_2, E_1)* which is the inverse of *Cause-Effect(E_1, E_2)*. We report the official macro-averaged F1 scores and accuracy.

3.5.2 Models

To empirically investigate the performance of our proposed method we compared it to several baselines and previously proposed models.

3.5.2.1 Random and word2vec Initialization

Rand-Init. The first baseline is RelEmb itself, but without applying the learning method on the unlabeled corpus. In other words, we train the softmax classifier from Section 3.3.3 on the labeled

[5] http://docs.google.com/View?docid=dfvxd49s_36c28v9pmw.

training data with randomly initialized model parameters.

W2V-Init. The second baseline is RelEmb using word embeddings learned by word2vec. More specifically, we initialize the embedding matrices $\mathbf{N}$ and $\mathbf{W}$ with the word2vec embeddings. Related to our method, word2vec has a set of weight vectors similar to $\tilde{\mathbf{W}}$ when trained with negative sampling and we use these weight vectors as a replacement for $\tilde{\mathbf{W}}$. We trained the word2vec embeddings using the *CBOW* model with subsampling on the full Wikipedia corpus. As with our experimental settings, we fix the learning rate to 0.025, and investigate several hyperparameter settings. For hyperparameter tuning we set the embedding dimensionality d to $\{50,\ 100,\ 300\}$, the context size c to $\{1,\ 3,\ 9\}$, and the number of negative samples k to $\{5,\ 15,\ 25\}$.

3.5.2.2 SVM-Based Systems

A simple approach to the relation classification task is to use SVMs with standard binary bag-of-words features. The bag-of-words features included the noun pairs and words between, before, and after the pairs, and we used LIBLINEAR[6] as our classifier.

3.5.2.3 Neural Network Models

Socher et al. (2012) used Recursive Neural Network (RNN) models to classify the relations. Subsequently, Ebrahimi and Dou (2015) and Hashimoto et al. (2013) proposed RNN models to better handle the relations. These methods rely on syntactic parse trees.

Yu et al. (2014) introduced their novel Factor-based Compositional Model (FCM) and presented results from several model variants, the best performing being FCM_{EMB} and FCM_{FULL}. The former only uses word embedding information and the latter relies on dependency paths and NE features, in addition to word embeddings.

Zeng et al. (2014) used a Convolutional Neural Network (CNN) with WordNet hypernyms. Noteworthy in relation to the RNN-based methods, the CNN model does not rely on parse trees. More recently, dos Santos et al. (2015) have introduced CR-CNN by extending the CNN model and achieved the best result to date. The key point of CR-CNN is that it improves the classification score by omitting the noisy class "Other" in the dataset described in Section 3.5.1. We call CR-CNN using the "Other" class $\text{CR-CNN}_{\text{Other}}$ and CR-CNN omitting the class $\text{CR-CNN}_{\text{Best}}$.

[6] http://www.csie.ntu.edu.tw/~cjlin/liblinear/.

	Features for classifiers	F1 / ACC (%)
$\text{RelEmb}_{\text{FULL}}$	embeddings, dependency paths, WordNet, NE	83.5 / 79.9
RelEmb	embeddings	82.8 / 78.9
RelEmb (W2V-Init)	embeddings	81.8 / 77.7
RelEmb (Rand-Init)	embeddings	78.2 / 73.5
SVM	bag of words	76.5 / 72.0
SVM [Rink and Harabagiu, 2010]	bag of words, POS, dependency paths, WordNet, paraphrases, TextRunner, Google n-grams, etc.	82.2 / 77.9
$\text{CR-CNN}_{\text{Best}}$ [dos Santos et al., 2015]	embeddings, word position embeddings	84.1 / n/a
FCM_{FULL} [Yu et al., 2014]	embeddings, dependency paths, NE	83.0 / n/a
$\text{CR-CNN}_{\text{Other}}$ [dos Santos et al., 2015]	embeddings, word position embeddings	82.7 / n/a
CRNN [Ebrahimi and Dou, 2015]	embeddings, parse trees, WordNet, NE, POS	82.7 / n/a
CNN [Zeng et al., 2014]	embeddings, WordNet	82.7 / n/a
MVRNN [Socher et al., 2012]	embeddings, parse trees, WordNet, NE, POS	82.4 / n/a
FCM_{EMB} [Yu et al., 2014]	embeddings	80.6 / n/a
RNN [Hashimoto et al., 2013]	embeddings, parse trees, phrase categories, etc.	79.4 / n/a

Table 3.1: Scores on the test set for SemEval 2010 Task 8.

3.5.3 Results and Discussion

The scores on the test set for SemEval 2010 Task 8 are shown in Table 3.1. RelEmb achieves 82.8% of F1 which is better than those of almost all models compared and comparable to that of the previous state of the art, except for $\text{CR-CNN}_{\text{Best}}$. Note that RelEmb does not rely on external semantic features and syntactic parse features[7]. Furthermore, $\text{RelEmb}_{\text{FULL}}$ achieves 83.5% of F1. We calculated a confidence interval $(82.0,\ 84.9)$ ($p < 0.05$) using bootstrap resampling [Noreen, 1989].

3.5.3.1 Comparison with the Baselines

RelEmb significantly outperforms not only the Rand-Init baseline, but also the W2V-Init baseline. These results show that our task-specific word embeddings are more useful than those trained using window-based contexts. A point that we would like to emphasize is that the baselines are unexpectedly

[7]While we use a POS tagger to locate noun pairs, RelEmb does not explicitly use POS features at the supervised learning step.

strong. As was noted by Wang and Manning (2012), we should carefully implement strong baselines and see whether complex models can outperform these baselines.

3.5.3.2 Comparison with SVM-Based Systems

RelEmb performs much better than the bag-of-words-based SVM. This is not surprising given that we use a large unannotated corpus and embeddings with a large number of parameters. RelEmb also outperforms the SVM system of Rink and Harabagiu (2010), which demonstrates the effectiveness of our task-specific word embeddings, despite our only requirement being a large unannotated corpus and a POS tagger.

3.5.3.3 Comparison with Neural Network Models

RelEmb outperforms the RNN models. In our preliminary experiments, we have found some undesirable parse trees when computing vector representations using RNN-based models and such parsing errors might hamper the performance of the RNN models.

FCM_{FULL}, which relies on dependency paths and NE features, achieves a better score than that of RElEmb. Without such features, RelEmb outperforms FCM_{EMB} by a large margin. By incorporating external resources, $\text{RelEmb}_{\text{FULL}}$ outperforms FCM_{FULL}.

RelEmb compares favorably to $\text{CR-CNN}_{\text{Other}}$, despite our method being less computationally expensive than $\text{CR-CNN}_{\text{Other}}$. When classifying an instance, the number of the floating number multiplications is $4d(2+c)L$ in our method since our method requires only one matrix-vector product for the softmax classifier as described in Section 3.3.3. c is the window size, d is the word embedding dimensionality, and L is the number of the classes. In $\text{CR-CNN}_{\text{Other}}$, the number is $(Dc(d+2d')N + DL)$, where D is the dimensionality of the convolution layer, d' is the position embedding dimensionality, and N is the average length of the input sentences. Here, we omit the cost of the hyperbolic tangent function in $\text{CR-CNN}_{\text{Other}}$ for simplicity. Using the best hyperparameter settings, the number is roughly 3.8×10^4 in our method, and 1.6×10^7 in $\text{CR-CNN}_{\text{Other}}$ assuming N is 10. dos Santos et al. (2015) also boosted the score of $\text{CR-CNN}_{\text{Other}}$ by omitting the noisy class "Other" by a ranking-based classifier, and achieved the best score ($\text{CR-CNN}_{\text{Best}}$). Our results may also be improved by using the same technique, but the technique is dataset-dependent, so we did not incorporate the technique.

c	d	$k = 5$	$k = 15$	$k = 25$
1	50	80.5	81.0	80.9
	100	80.9	81.3	81.2
2	50	80.9	81.3	81.3
	100	81.3	81.6	81.7
3	50	81.0	81.0	81.5
	100	81.3	81.9	**82.2**
	300	-	-	82.0

Table 3.2: Cross-validation results for RelEmb.

c	d	$k = 5$	$k = 15$	$k = 25$
1	50	80.5	80.7	80.9
	100	81.1	81.2	81.0
	300	81.2	**81.3**	81.2
3	50	80.4	80.7	80.8
	100	81.0	81.0	80.9
9	50	80.0	79.8	80.2
	100	80.3	80.4	80.1

Table 3.3: Cross-validation results for the W2V-Init.

3.5.4 Analysis on Training Settings

We perform analysis of the training procedure focusing on RelEmb.

3.5.4.1 Effects of Tuning Hyperparameters

In Tables 3.2 and 3.3, we show how tuning the hyperparameters of our method and word2vec affects the classification results using 10-fold cross validation on the training set. The same split is used for each setting, so all results are comparable to each other. The best settings for the cross validation are used to produce the results reported in Table 3.1.

Table 3.2 shows F1 scores obtained by RelEmb. The results for $d = 50,\ 100$ show that RelEmb

$\mathbf{g}_{\mathrm{n}}$	$\mathbf{g}_{\mathrm{in}}$	$\mathbf{g}'_{\mathrm{in}}$	$\mathbf{g}_{\mathrm{n}}, \mathbf{g}_{\mathrm{in}}$	$\mathbf{g}_{\mathrm{n}}, \mathbf{g}_{\mathrm{in}}, \mathbf{g}_{\mathrm{out}}$
61.8	70.2	68.2	81.1	82.2

Table 3.4: Cross-validation results for ablation tests.

Method	Score
RelEmb $\mathbf{N}$	0.690
RelEmb $\mathbf{W}$	0.599
W2V-Init	0.687

Table 3.5: Evaluation on the WordSim-353 dataset.

benefits from relatively large context sizes. The n-gram embeddings in RelEmb capture richer information by setting c to 3 compared to setting c to 1. Relatively large numbers of negative samples also slightly boost the scores. As opposed to these trends, the score does not improve using $d = 300$. We use the best setting ($c = 3$, $d = 100$, $k = 25$) for the remaining analysis. We note that RelEmb$_{\mathrm{FULL}}$ achieves an F1-score of 82.5.

We also performed similar experiments for the W2V-Init baseline, and the results are shown in Table 3.3. In this case, the number of negative samples does not affect the scores, and the best score is achieved by $c = 1$. As discussed in Bansal et al. (2014), the small context size captures the syntactic similarity between words rather than the topical similarity. This result indicates that syntactic similarity is more important than topical similarity for this task. Compared to the word2vec embeddings, our embeddings capture not only local context information using word order, but also long-range co-occurrence information by being tailored for the specific task.

3.5.4.2 Ablation Tests

As described in Section 3.3.2, we concatenate three kinds of feature vectors, $\mathbf{g}_{\mathrm{n}}$, $\mathbf{g}_{\mathrm{in}}$, and $\mathbf{g}_{\mathrm{out}}$, for supervised learning. Table 3.4 shows classification scores for ablation tests using 10-fold cross validation. We also provide a score using a simplified version of $\mathbf{g}_{\mathrm{in}}$, where the feature vector $\mathbf{g}'_{\mathrm{in}}$ is computed by averaging the word embeddings $[\mathbf{W}(w_i^{in}); \tilde{\mathbf{W}}(w_i^{in})]$ of the words between the noun pairs. This feature vector $\mathbf{g}'_{\mathrm{in}}$ then serves as a bag-of-words feature.

Cause-Effect(E_1,E_2)				**Content-Container(E_1,E_2)**				**Message-Topic(E_1,E_2)**			
resulted	poverty	caused	the	inside	was	inside	a	discuss	magazines	relating	to
caused	stability	caused	the	in	was	in	a	explaining	to	discuss	aspects
generated	coast	resulted	in	hidden	hidden	in	a	discussing	concerned	about	NULL
cause	fire	caused	due	was	was	inside	the	relating	interview	relates	to
causes	that	resulted	in	stored	was	hidden	in	describing	to	discuss	the
Cause-Effect(E_2,E_1)				**Content-Container(E_2,E_1)**				**Message-Topic(E_2,E_1)**			
after	caused	by	radiation	full	NULL	full	of	subject	were	related	in
from	caused	by	infection	included	was	full	of	related	was	related	in
caused	stomach	caused	by	contains	a	full	NULL	discussed	been	discussed	in
triggered	caused	by	genetic	contained	a	full	and	documented	is	related	through
due	anger	caused	by	stored	a	full	forty	received	the	subject	of

Table 3.6: Top five unigrams and trigrams with the highest scores for six classes.

Table 3.4 clearly shows that the averaged n-gram embeddings contribute the most to the semantic relation classification performance. The difference between the scores of $\mathbf{g}_{\text{in}}$ and $\mathbf{g}'_{\text{in}}$ shows the effectiveness of our averaged n-gram embeddings.

3.5.4.3 Effects of Dropout

At the supervised learning step we use dropout to regularize our model. Without dropout, our performance drops from 82.2% to 81.3% of F1 on the training set using 10-fold cross validation.

3.5.4.4 Performance on a Word Similarity Task

As described in Section 3.3.1, we have the noun-specific embeddings $\mathbf{N}$ as well as the standard word embeddings $\mathbf{W}$. We evaluated the learned embeddings using a word-level semantic evaluation task called *WordSim-353* [Finkelstein et al., 2001]. This dataset consists of 353 pairs of nouns and each pair has an averaged human rating which corresponds to a semantic similarity score. Evaluation is performed by measuring Spearman's rank correlation between the human ratings and the cosine similarity scores of the embeddings. Table 3.5 shows the evaluation results. We used the best settings reported in Table 3.2 and 3.3 since our method is designed for relation classification and it is not clear how to tune the hyperparameters for the word similarity task. As shown in the result table, the noun-specific embeddings perform better than the standard embeddings in our method, which indicates

the noun-specific embeddings capture more useful information in measuring the semantic similarity between nouns. The performance of the noun-specific embeddings is roughly the same as that of the word2vec embeddings.

3.5.5 Qualitative Analysis on the Embeddings

Using the n-gram embeddings $\mathbf{h}_i$ in Eq. (3.5), we inspect which n-grams are relevant to each relation class after the supervised learning step of RelEmb. When the context size c is 3, we can use at most 7-grams. The learned weight matrix $\mathbf{S}$ in Section 3.3.3 is used to detect the most relevant n-grams for each class. More specifically, for each n-gram embedding $(n = 1, 3)$ in the training set, we compute the dot product between the n-gram embedding and the corresponding components in $\mathbf{S}$. We then select the pairs of n-grams and class labels with the highest scores. In Table 3.6 we show the top five n-grams for six classes. These results clearly show that the n-gram embeddings capture salient syntactic patterns which are useful for the relation classification task.

3.6 Conclusions and Future Work

We have presented a method for learning word embeddings specifically designed for relation classification. The word embeddings are trained using large unlabeled corpora to capture lexical features for relation classification. On a well-established semantic relation classification task our method significantly outperforms the baseline based on word2vec. Our method also compares favorably to previous state-of-the-art models that rely on syntactic parsers and external semantic resources, despite our method requiring only access to an unannotated corpus and a POS tagger. For future work, we will investigate how well our method performs on other domains and datasets and how relation labels can help when learning embeddings in a semi-supervised learning setting.

Chapter 4

Task-Oriented Learning of Semantic Compositionality of Phrases

In this chapter, I describe a method for jointly learning phrase embeddings and their compositionality in a task-oriented manner. Here I first present a phrase embedding learning method for transitive verbs in a co-occurrence modeling task. Then I present how to automatically learn compositionality levels of the phrases. Section 4.1 and Section 4.2 in this chapter correspond to the following published papers, respectively.

Section 4.1 Kazuma Hashimoto and Yoshimasa Tsuruoka. 2015. Learning Embeddings for Transitive Verb Disambiguation by Implicit Tensor Factorization. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pp. 1–11.

Section 4.2 Kazuma Hashimoto and Yoshimasa Tsuruoka. 2016. Adaptive Joint Learning of Compositional and Non-Compositional Phrase Embeddings. In *Proceedings of the 54th Annual Meeting of Association for Computational Linguistics*, pp. 205–215.

4.1 Learning Meanings of Transitive Verb Phrases

Summary We present an implicit tensor factorization method for learning the embeddings of transitive verb phrases. Unlike the implicit matrix factorization methods recently proposed for learning

word embeddings, our method directly models the interaction between predicates and their two arguments, and learns verb phrase embeddings. By representing transitive verbs as matrices, our method captures multiple meanings of transitive verbs and disambiguates them taking their arguments into account. We evaluate our method on a widely-used verb disambiguation task and three phrase similarity tasks. On the disambiguation task, our method outperforms previous state-of-the-art methods. Our experimental results also show that adjuncts provide useful information in learning the meanings of verb phrases.

4.1.1 Introduction

There is a growing interest in learning vector-space representations of words and phrases using large training corpora in the field of Natural Language Processing (NLP) [Mikolov et al., 2013b; Mitchell and Lapata, 2010]. The phrase representations are usually computed by composition models that combine the meanings of words into the meanings of phrases. While some studies focus on representing entire phrases or sentences using syntactic structures [Hermann and Blunsom, 2013; Socher et al., 2011a], others focus on representing the meaning of transitive verb phrases [Grefenstette and Sadrzadeh, 2011; Grefenstette et al., 2013; Kartsaklis et al., 2012].

In this paper, we investigate vector-space representations of transitive verb phrases. The meaning of a transitive verb is often ambiguous and disambiguated by its arguments, i.e., subjects and objects. Investigation of transitive verb phrases should therefore provide insights into how composition models can capture such semantic interactions between words. Moreover, in practice, capturing the meanings of transitive verb phrases should be useful in many real-world NLP applications such as semantic retrieval [Miyao et al., 2006] and question answering (*Who did What to Whom?*) [Srihari and Li, 2000].

There are several approaches to representing transitive verb phrases in a vector space using large unannotated corpora. One is based on tensor calculus [Grefenstette and Sadrzadeh, 2011; Kartsaklis et al., 2012; Van de Cruys et al., 2013] and another is based on neural networks [Hashimoto et al., 2014; Muraoka et al., 2014; Tsubaki et al., 2013]. In the tensor-based methods, transitive verbs are represented as matrices, and they are constructed by using the pre-trained word embeddings of their subjects and objects. One limitation of this approach is that the embeddings of subject-verb-object phrases are computed statically, i.e., the composition process and the embedding (or matrix) construction process are conducted separately. In the neural network-based methods, the embeddings

of words and phrases can be learned jointly [Hashimoto et al., 2014]. However, the strong interaction between verbs and their arguments is not fully captured in their method because it relies on shallow neural networks using diagonal weight matrices which are designed to work on large training corpora.

To bridge the gap between the two approaches, we present an implicit tensor factorization method for learning the embeddings of transitive verb phrases. We assume a three-mode tensor in which the value of each element represents the level of plausibility of a tuple of a predicate and its two arguments [Van de Cruys et al., 2013]. We then implicitly factorize the tensor into three latent factors, namely one predicate tensor and two argument matrices. This is motivated by the recently proposed implicit matrix factorization methods for learning word embeddings [Levy and Goldberg, 2014b; Mikolov et al., 2013b]. Our method trains matrices representing predicates and embeddings of their arguments so that they maximize the accuracy of predicting the plausibility of the predicate-argument tuples in the training corpus. The transitive verb matrices and the embeddings of their subject and object are thus jointly learned. Furthermore, this method allows us to exploit the role of prepositional adjuncts when learning the meaning of verb phrases by modeling the relationship between prepositions and verb phrases.

Our experimental results show that our method enables predicates and their arguments to strongly interact with each other and that adjuncts are useful in learning the meaning of verb phrases. We evaluate our method using a widely-used verb disambiguation task and three phrase similarity tasks. On the disambiguation dataset provided by Grefenstette and Sadrzadeh (2011), we have achieved a Spearman's rank correlation score of 0.614, which is significantly higher than the state of the art (0.456). This result demonstrates that the direct interaction between verbs and their arguments is important in tackling verb disambiguation tasks. Qualitative evaluation further shows that the meanings of ambiguous verbs can be disambiguated according to their arguments and the learned verb matrices capture multiple meanings of transitive verbs.

4.1.2 Method

To learn the embeddings of transitive verb phrases, we focus on the role of adjuncts, which optionally complement the meaning of the verb phrases. For example, in the following sentence, the prepositional phrase starting from the preposition "in" is an adjunct of the verb "make":

> An importer might be able to make payment in his own domestic currency.

Predicate	Argument 1	Argument 2
make	an importer	payment
in	make payment	his own domestic currency

Table 4.1: Example output from the Enju parser.

Predicate	Argument 1	Argument 2
make	importer	payment
in	importer make payment	currency

Table 4.2: Modified examples.

The transitive verb "make" is inherently ambiguous, but this sentence tells us that the action expressed by the verb phrase "make payment" is carried out by means of a currency. If we further observe the verb phrase "pay money" with a similar adjunct in another sentence, those two sentences tell us that the two phrases "make payment" and "pay money" are semantically similar to each other. We therefore expect such prepositional adjuncts to be useful in learning the meaning of verb phrases. In the disambiguation processes, strong interactions between transitive verbs and their arguments are desirable as with the method in Tsubaki et al. (2013). More specifically, the meaning of "make" changes according to its object "payment" and the meaning of "pay" changes according to its object "money".

We use the probabilistic HPSG parser *Enju*[1] [Miyao and Tsujii, 2008] to identify transitive verbs with their subjects and objects, and as adjuncts, we also extract prepositional phrases with transitive verbs. In the grammar of the Enju parser, each word in a sentence is a predicate with zero or more arguments, i.e., prepositions, too, are treated as predicates.

In the example sentence shown above, the transitive verb "make" and the preposition "in" are predicates which take two arguments. Table 4.1 shows the output from the Enju parser. In this example, the transitive verb "make" takes two arguments: the first argument (the subject) is the noun phrase "an importer" and the second argument (the object) is the noun "payment". The preposition "in" also takes two arguments: the first argument is the verb phrase "make payment" and the second argument

[1] `http://kmcs.nii.ac.jp/enju/`.

is the noun phrase "his own domestic currency". For simplicity we use only the head word of each noun phrase, so the subjects and objects of the transitive verbs are nouns and the second arguments of the prepositions are also nouns. We further modify the output by incorporating the subject for each verb phrase which is the first argument of prepositions when the subject exists [2]. Therefore, the words used in this paper are verbs, nouns, and prepositions. Table 4.2 shows the modified output of the examples in Table 4.1.

To model the co-occurrence statistics of predicate-argument structures, we follow Van de Cruys et al. (2013) and assume a three-mode tensor, which is just a three-dimensional array, $\mathcal{T} \in \mathbb{R}^{|\mathbb{P}|\times|\mathbb{A}_1|\times|\mathbb{A}_2|}$ in which plausibility scores are stored as real values. $\mathbb{P}$ is the set of predicates of a particular category in the training corpus, $\mathbb{A}_1$ is the set of the first argument of the predicates in $\mathbb{P}$, and $\mathbb{A}_2$ is the set of the second argument. When treating transitive verbs as predicates, $\mathbb{A}_1$ is the set of their subjects and $\mathbb{A}_2$ is the set of their objects. Table 4.1 shows an example, where "make", "an importer", and "payment" are a member of $\mathbb{P}$, $\mathbb{A}_1$, and $\mathbb{A}_2$, respectively. The plausibility score $\mathcal{T}(i,j,k)$ corresponds to the tuple of the i-th ($1 \leq i \leq |\mathbb{P}|$) predicate having the j-th ($1 \leq j \leq |\mathbb{A}_1|$) first-argument and the k-th ($1 \leq k \leq |\mathbb{A}_2|$) second-argument. The larger the value of $\mathcal{T}(i,j,k)$ is, the more plausible the tuple (i,j,k) is. In the above example, if the tuple (i,j,k) corresponds to "make", "an importer", and "payment" and i' corresponds to "eat", the value of $\mathcal{T}(i,j,k)$ is expected to be larger than that of $\mathcal{T}(i',j,k)$.

As with Van de Cruys et al. (2013), we factorize the large three-mode tensor $\mathcal{T}$ into three factors:

- three-mode tensor $\mathcal{P} \in \mathbb{R}^{|\mathbb{P}|\times d\times d}$,
- matrix $\mathbf{A}_1 \in \mathbb{R}^{d\times|\mathbb{A}_1|}$, and
- matrix $\mathbf{A}_2 \in \mathbb{R}^{d\times|\mathbb{A}_2|}$.

The dimensionality d is a hyperparameter that determines the size of the latent factors. Using these factors, we can compute a plausibility score:

$$\mathcal{T}(i,j,k) = \mathbf{a}_1(j)^{\mathrm{T}}\mathbf{P}(i)\mathbf{a}_2(k) \tag{4.1}$$

where $\mathbf{a}_1(j) \in \mathbb{R}^{d\times 1}$ and $\mathbf{a}_2(k) \in \mathbb{R}^{d\times 1}$ are the j-th and k-th column vectors of $\mathbf{A}_1$ and $\mathbf{A}_2$, respectively, and $\mathbf{P}(i) \in \mathbb{R}^{d\times d}$ is the i-th slice, which is just a matrix, of $\mathcal{P}$. $\mathbf{a}_1(j)^{\mathrm{T}}$ is the transpose of $\mathbf{a}_1(j)$.

[2] Subjects can be absent. For example, in the sentence "Learning word embedings is interesting" the subject of the transitive verb "learn" is absent.

By this tensor factorization, each predicate in $\mathbb{P}$ is represented with a matrix, which we call a predicate matrix, and each argument in $\mathbb{A}_1$ and $\mathbb{A}_2$ is represented with a vector, which we call an argument embedding. As with Hashimoto et al. (2014), arguments are not restricted to words in our method, and thus we compute argument embeddings using a composition function when the arguments consist of more than two words.

To learn the predicate matrices and argument embeddings, we define a plausibility judgment task by using a cost function for each predicate-argument tuple observed in the training corpus. For each predicate-argument tuple (i, j, k), the cost function $E(i, j, k)$ is defined as follows:

$$\begin{aligned} -\log \sigma(\mathcal{T}(i,j,k)) &- \log(1-\sigma(\mathcal{T}(i',j,k))) \\ &- \log(1-\sigma(\mathcal{T}(i,j',k))) \\ &- \log(1-\sigma(\mathcal{T}(i,j,k'))) \end{aligned} \tag{4.2}$$

where $i' \in \mathbb{P}$ is a randomly drawn predicate, and $j' \in \mathbb{A}_1$ and $k' \in \mathbb{A}_2$ are randomly drawn arguments. $\sigma(x)$ is the logistic function, so the cost function $E(i, j, k)$ in Eq. (4.2) measures whether we can discriminate between the plausible tuple and other three implausible tuples by means of logistic regressions. We follow Mikolov et al. (2013b) to draw the random predicates and arguments according to their frequencies weighted by an exponent of 0.75 and ensure that each of the randomly generated tuples is not observed in the corpus. The overall objective function is defined as the sum of the cost functions for all observed predicate-argument tuples and minimized by AdaGrad [Duchi et al., 2011] in a mini-batch setting.

The partial derivative $\frac{\partial E(i,j,k)}{\partial \mathbf{P}(i)}$ for updating the model parameters is computed as follows:

$$\begin{aligned} \frac{\partial E(i,j,k)}{\partial \mathbf{P}(i)} = (\sigma(\mathcal{T}(i,j,k)) - 1)\mathbf{a}_1(j) \otimes \mathbf{a}_2(k) + \\ \sigma(\mathcal{T}(i,j',k))\mathbf{a}_1(j') \otimes \mathbf{a}_2(k) + \\ \sigma(\mathcal{T}(i,j,k'))\mathbf{a}_1(j) \otimes \mathbf{a}_2(k') \end{aligned} \tag{4.3}$$

where $\otimes$ denotes the outer-product of two vectors. Similarly, the partial derivatives $\frac{\partial E(i,j,k)}{\partial \mathbf{a}_1(j)}$ and

$\frac{\partial E(i,j,k)}{\partial \mathbf{a}_2(k)}$ are computed as follows:

$$\begin{aligned}\frac{\partial E(i,j,k)}{\partial \mathbf{a}_1(j)} = (\sigma(\mathcal{T}(i,j,k)) - 1)\mathbf{P}(i)\mathbf{a}_2(k) + \\ \sigma(\mathcal{T}(i',j,k))\mathbf{P}(i')\mathbf{a}_2(k) + \\ \sigma(\mathcal{T}(i,j,k'))\mathbf{P}(i)\mathbf{a}_2(k')\end{aligned} \tag{4.4}$$

$$\begin{aligned}\frac{\partial E(i,j,k)}{\partial \mathbf{a}_2(k)} = (\sigma(\mathcal{T}(i,j,k)) - 1)\mathbf{P}(i)^{\mathrm{T}}\mathbf{a}_1(j) + \\ \sigma(\mathcal{T}(i',j,k))\mathbf{P}(i')^{\mathrm{T}}\mathbf{a}_1(j) + \\ \sigma(\mathcal{T}(i,j',k))\mathbf{P}(i)^{\mathrm{T}}\mathbf{a}_1(j')\end{aligned} \tag{4.5}$$

which can be used to learn the composition function using the backpropagation algorithm if the arguments are not words. When the arguments are words, we then use the partial derivatives to directly update the argument embeddings. $\mathbf{P}(i')$, $\mathbf{a}_1(j')$, and $\mathbf{a}_2(k')$ are also updated but for the sake of brevity the partial derivatives for them are not shown here. Equation (4.3) shows that a predicate matrix is updated to capture the information about which argument pairs are or are not relevant to the predicate. Argument embeddings are learned to capture similar information.

4.1.2.1 Transitive Verb Phrases with Adjuncts

While our method is applicable to any categories of predicates which take two arguments, in this paper, we focus on learning the embeddings of transitive verb phrases by treating transitive verbs and prepositions as predicates. Thus, we factorize two tensors $\mathcal{T}_v$ and $\mathcal{T}_p$ for transitive verbs and prepositions, respectively. $\mathcal{T}_v$ is factorized into a verb tensor $\mathcal{V}$ (corresponding to $\mathcal{P}$), a subject matrix $\mathbf{S}$ (corresponding to $\mathbf{A}_1$), and an object matrix $\mathbf{O}$ (corresponding to $\mathbf{A}_2$). To compute argument embeddings composed by subject-verb-object tuples, we use the *copy-subject* function in Kartsaklis et al. (2012):

$$\mathbf{s}(m) \odot (\mathbf{V}(l)\mathbf{o}(n)) \tag{4.6}$$

where $\mathbf{V}(l)$ is a verb matrix, $\mathbf{s}(m)$ is a subject embedding, and $\mathbf{o}(n)$ is an object embedding. $\odot$ denotes the element-wise multiplication of two vectors. The composed verb phrase embeddings are

taken as the first arguments of the prepositions. The copy-subject function is also used to compute verb-object phrase embeddings by omitting the subject embedding in Eq. (4.6):

$$\mathbf{V}(l)\mathbf{o}(n) \tag{4.7}$$

Compared with other composition functions defined in Kartsaklis et al. (2012), such as the copy-object function, the copy-subject function allows us to compute embeddings for both of subject-verb-object and verb-object phrases.

In the case of the copy-subject function, assuming that Eq. (4.4) is defined as $\boldsymbol{\delta}_1$, the subject embedding in Eq. (4.6) is updated using the following partial derivative:

$$\frac{\partial E(i,j,k)}{\partial \mathbf{s}(m)} = \boldsymbol{\delta}_1 \odot (\mathbf{V}(l)\mathbf{o}(n)) \tag{4.8}$$

We then define $\boldsymbol{\delta}_2$ as follows:

$$\boldsymbol{\delta}_2 = \boldsymbol{\delta}_1 \odot \mathbf{s}(m) \tag{4.9}$$

and update the verb matrix and object embedding using $\boldsymbol{\delta}_2$:

$$\frac{\partial E(i,j,k)}{\partial \mathbf{V}(l)} = \boldsymbol{\delta}_2 \mathbf{o}(n)^{\mathrm{T}} \tag{4.10}$$

$$\frac{\partial E(i,j,k)}{\partial \mathbf{o}(n)} = \mathbf{V}(l)^{\mathrm{T}} \boldsymbol{\delta}_2 \tag{4.11}$$

The model parameters used in the composition function are shared across the overall proposed method. That is, the verb matrices and subject/object embeddings are used for computing the composed embeddings and the plausibility scores in Eq. (4.1).

4.1.2.2 Relationship to Previous Work

Representing transitive verbs with matrices and computing transitive verb phrase embeddings have been proposed by Grefenstette and Sadrzadeh (2011) and others [Kartsaklis et al., 2012; Milajevs et al., 2014; Polajnar et al., 2014]. All of them first construct word embeddings by using existing methods and then compute or learn transitive verb matrices. This kind of approach requires one to figure out which word embeddings are suitable for each method or task [Milajevs et al., 2014]. By contrast, our method does not require any other word embedding methods and instead jointly learns

word embeddings and matrices from scratch, which saves us from the time-consuming process to test which word representations learned by existing methods are suitable for which composition models. Moreover, our method *learns* the embeddings of transitive verb phrases by using adjuncts rather than statically computing them using learned word embeddings and matrices as done in the previous work.

The information stored in the verb matrices learned by Eq. (4.3) is similar to that in Grefenstette and Sadrzadeh (2011). In Grefenstette and Sadrzadeh (2011), a verb matrix is computed by the sum of the outer-products of the embeddings of its subject-object pairs observed in the corpus. By using such matrices, Kartsaklis et al. (2012) proposed the copy-subject function which has proven effective in representing transitive verb phrases. Using the copy-subject function is therefore a reasonable choice for our composition function.

The use of adjuncts constructed by prepositional phrases for learning verb phrase embeddings has been presented in Hashimoto et al. (2014). However, they used a variety of categories of predicates simultaneously, and thus it is not clear how adjuncts are useful in improving the embeddings of transitive verb phrases. In this paper, we use only transitive verbs and prepositions and clarify the effects of adjuncts. Moreover, the interactions between predicates and their arguments are weak in their method because their method relies on shallow neural networks using diagonal weight matrices. In contrast, our method allows the predicates to directly interact with their arguments.

The way of factorizing the three-mode tensors is based on Van de Cruys et al. (2013). The main difference between our method and theirs is that our method can treat phrases as the arguments. Their method is based on co-occurrence count statistics, and thus it is not straightforward to modify their method to treat phrases as well as words.

Our implicit tensor factorization method is motivated by Levy and Goldberg (2014b). They introduced a way to interpret the recently developed word embedding learning method [Mikolov et al., 2013b] by using matrix factorization. While their method only produces embeddings of single tokens, our method jointly learns word and phrase embeddings by focusing on the relationship between predicates and their two arguments.

4.1.3 Experimental Settings

4.1.3.1 Training Corpora

We separately used two corpora as our training corpora. The first one is the British National Corpus (BNC), from which we extracted 6 million sentences. The second one is a snapshot of the English

Wikipedia[3] (enWiki) from November 2013. We extracted 80 million sentences from the original Wikipedia file. We then used the Enju parser [Miyao and Tsujii, 2008] to parse all the extracted sentences.

Using the parsing results, we constructed the vocabulary for each training corpus. To be more specific, we used the 100,000 most frequent base-form words paired with their corresponding part-of-speech tags in each corpus. Using verbs, nouns, and prepositions in the vocabulary, we extracted predicate-argument tuples whose predicate categories are *verb_arg12* or *prep_arg12* defined in the Enju parser. We then pre-processed the output as shown in Table 4.2. Consequently, BNC consists of about 1.38 million instances (1.23 million types) for the verb data and about 0.93 million instances (0.88 million types) for the preposition data, and enWiki consists of about 23.6 million instances (15.8 million types) for the verb data and about 17.3 million instances (13.5 million types) for the preposition data. We call the verb data **SVO** and the combination of the two data **SVOPN** [4].

For each corpus, we randomly split the data into the training data (80%), the development data (10%), and the test data (10%). We used the development data for tuning hyperparameters to be used in downstream NLP tasks. When splitting the data, we ensured that each type of predicate-argument tuples appeared in only one of the three parts. Hence, for example, instances in the test data do not appear in either of the training or the development data.

To evaluate our method on the plausibility judgment task, for each predicate-argument tuple type in the development and the test data, we randomly sampled implausible tuples N times in the same way as defining the cost function in Eq. (4.2). That is, we prepared N sets of the development and the test data. For each set of the development and the test data, we calculated the accuracy of the plausibility judgment task; concretely, for each type of predicate-argument tuples (i, j, k), we evaluated whether $\mathcal{T}(i, j, k)$ is larger than all of $\mathcal{T}(i', j, k)$, $\mathcal{T}(i, j', k)$, and $\mathcal{T}(i, j, k')$ and then calculated the ratio of the number of types counted as correct to the total number of the types in the development or the test data. Finally, we calculated the average accuracy of the N set. For BNC, we set N to 50 and for enWiki we set N to 10.

[3] http://dumps.wikimedia.org/enwiki/

[4] The training data, the training code, and the learned model parameters used in this paper are publicly available at http://www.logos.t.u-tokyo.ac.jp/~hassy/publications/cvsc2015/

4.1.3.2 Initialization and Hyperparameters

We initialized the noun embeddings, the verb matrices, and the preposition matrices with zero-mean gaussian noise with a variance of $\frac{1}{d}$, $\frac{1}{d^2}$, and $\frac{1}{d^2}$, respectively. The hyperparameters for training the embeddings and the matrices are the embedding dimensionality d, the learning rate α for AdaGrad [Duchi et al., 2011], the mini-batch size, and the number of iterations n over the training data. In our preliminary experiments, we have found that varying the mini-batch size is not so influential in our experimental results. We thus fixed the mini-batch size to 100. For other hyperparameters, we set d to $\{25, 50, 100\}$, α to $\{0.01, 0.02, 0.04, 0.06, 0.08, 0.1\}$, and the maximum number of n to 20. We selected the values of the hyperparameters so that the accuracy of the plausibility task was maximized on the development data described in Section 4.1.3.1.

4.1.3.3 Baseline Method

We mainly compared our method with the method called *PAS-CLBLM* in Hashimoto et al. (2014) since PAS-CLBLM is designed to learn composed representations as well as word embeddings using a variety of predicate-argument structures. PAS-CLBLM is modeled as a word predication model using predicate-argument structures, which means that, as with our method, the training relies on the co-occurrence statistics of predicate-argument structures. PAS-CLBLM achieved state-of-the-art results on transitive verb phrase similarity tasks. To train PAS-CLBLM, we used the same data described in Section 4.1.3.1. We selected the Wadd_{nl} function in PAS-CLBLM to compute the embedding of each subject-verb-object tuple (i, j, k):

$$\tanh(\mathbf{w}_{\text{s}} \odot \mathbf{s}(j) + \mathbf{w}_{\text{v}} \odot \mathbf{v}(i) + \mathbf{w}_{\text{o}} \odot \mathbf{o}(k)) \tag{4.12}$$

where $\mathbf{w}_{\text{s}}, \mathbf{w}_{\text{v}}, \mathbf{w}_{\text{o}} \in \mathbb{R}^{d \times 1}$ are the weight vectors (or the diagonal weight matrices) for composition and $\mathbf{s}(j), \mathbf{v}(i), \mathbf{o}(k) \in \mathbb{R}^{d \times 1}$ are the embeddings of the subjects, verbs, and objects, respectively. PAS-CLBLM has the same hyperparameters as our method described in Section 4.1.3.2. We used the development data for tuning the hyperparameters and added $d = 200$ to the candidate values for d since PAS-CLBLM is computationally less expensive than our method. We thus evaluated PAS-CLBLM also on the plausibility judgment task. Concretely, for each type of predicate-argument tuples (i, j, k) in the development data, the tuple is counted as correct when the predication scores for i, j, and k are larger than those for i', j', and k', respectively.

	d	BNC Acc. (%)	enWiki Acc. (%)
Our method	25	57.44 (0.11)	64.77 (0.03)
	50	**57.80** (0.11)	66.98 (0.03)
	100	57.48 (0.10)	**68.18** (0.03)
PAS-CLBLM	25	54.44 (0.14)	60.40 (0.03)
	50	**55.69** (0.13)	63.42 (0.02)
	100	55.66 (0.12)	64.81 (0.02)
	200	55.48 (0.15)	**65.20** (0.03)

Table 4.3: Evaluation results on the plausibility judgment task on the SVO development data.

4.1.4 Results and Discussion

We first tuned the hyperparameters in both our method and the baseline method using the plausibility judgment task. Table 4.3 shows the average accuracy with the standard deviation for each dimensionality on the SVO development data[5]. As shown in the table, our method outperforms PAS-CLBLM on both BNC and enWiki. The number of the model parameters in PAS-CLBLM ($d = 200$) is larger than that of the model parameters in our method ($d = 50$). This result demonstrates that the model architecture itself is more important than the number of the model parameters. The results on the SVO test data were 57.76% (our method, $d = 50$) and 55.66% (PAS-CLBLM, $d = 50$) for BNC. For enWiki, the results were 68.18% (our method, $d = 100$) and 65.19% (PAS-CLBLM, $d = 200$). We observed a similar trend on the SVOPN data and in the next section, for each embedding dimensionality, we used the model parameters which performed best on the plausibility task.

4.1.4.1 Evaluation on Transitive Verb Tasks

[5]Van de Cruys (2014) reported much higher accuracy in a similar evaluation setting with a neural network model, but as discussed in Chambers and Jurafsky (2010), this is because using the uniform distribution over words for producing implausible tuples leads to optimistic results.

[6]We replicated the results reported in their paper using the model parameters publicly provided at `http://www.logos.t.u-tokyo.ac.jp/~hassy/publications/emnlp2014/`.

	Data	d	Dis. GS'11	Phrase similarity ML'10	KS'13	KS'14
Our method	SVO	25	0.410	0.511	0.392	0.440
		50	0.374	0.550	0.164	0.290
		100	0.373	0.474	0.312	0.418
	SVOPN	25	**0.574**	0.543	0.439	0.432
		50	0.535	**0.586**	0.403	0.397
		100	0.508	0.545	**0.487**	**0.517**
PAS-CLBLM	SVO	25	0.270	0.601	0.592	0.722
		50	**0.412**	0.581	0.523	0.721
		100	0.390	0.463	0.465	0.699
		200	0.369	0.458	0.434	0.602
	SVOPN	25	0.241	0.562	0.550	0.715
		50	0.281	**0.605**	**0.590**	**0.760**
		100	0.337	0.593	0.585	0.758
		200	0.342	0.561	0.549	0.744
Milajevs et al. (2014)			0.456	n/a	n/a	0.732
Hashimoto et al. (2014)[6]			0.422	0.669	0.612	0.770
Polajnar et al. (2014)			0.35	n/a	0.58	n/a

Table 4.4: Results for the transitive verb tasks using the BNC data.

We evaluated the learned embeddings of transitive verbs using a transitive verb disambiguation task and three tasks for measuring the semantic similarity between transitive verb phrases. Each phrase pair in the four datasets is paired with multiple human ratings: the higher the rating is, the more semantically similar the phrases are. To evaluate the learned verb phrase embeddings on each dataset, we used the Spearman's rank correlation between the human ratings and the cosine similarity between the phrase embeddings. We calculated the correlation scores using averaged human ratings. Each phrase pair in the datasets was annotated by more than two annotators and we took the average of the multiple human ratings for each phrase pair.

	Data	d	Dis. GS'11	Phrase similarity ML'10	KS'13	KS'14
Our method	SVO	25	0.438	0.403	0.255	0.406
		50	0.480	0.416	0.359	0.481
		100	0.433	0.392	0.239	0.409
	SVOPN	25	0.576	0.435	0.372	0.555
		50	**0.614**	0.495	**0.422**	**0.566**
		100	0.576	**0.558**	0.420	0.548
PAS-CLBLM	SVO	25	0.342	0.500	0.407	0.624
		50	0.313	0.527	0.502	0.710
		100	0.358	0.534	0.470	0.655
		200	0.361	0.535	0.459	0.653
	SVOPN	25	0.171	0.571	**0.583**	0.697
		50	0.320	0.501	0.518	0.729
		100	0.321	**0.606**	0.540	0.742
		200	**0.374**	0.588	0.515	**0.744**
Milajevs et al. (2014)			0.456	n/a	n/a	0.732
Hashimoto et al. (2014)			0.422	0.669	0.612	0.770
Polajnar et al. (2014)			0.35	n/a	0.58	n/a

Table 4.5: Results for the transitive verb tasks using the enWiki data.

Transitive verb disambiguation. The first dataset **GS'11** is provided by Grefenstette and Sadrzadeh (2011). GS'11 consists of pairs of transitive verbs and each verb pair takes the same subject and object. As discussed in previous work [Kartsaklis and Sadrzadeh, 2013; Milajevs et al., 2014; Polajnar et al., 2014], GS'11 has an aspect of a verb sense disambiguation task. For example, the transitive verb "run" is known as a polysemous word and this task requires one to identify the meanings of "run" and "operate" are similar to each other when taking "people" as their subject and "company" as their object. In the same setting, however, the meanings of "run" and "move" are not similar to each other. The task is suitable for evaluating our method since our method allows verbs and their subjects and objects to multiplicatively interact with each other.

Transitive verb phrase similarity. The other datasets are **ML'10** provided by Mitchell and Lapata (2010), **KS'13** provided by Kartsaklis and Sadrzadeh (2013), and **KS'14** provided by Kartsaklis and Sadrzadeh (2014). ML'10 consists of pairs of verb-object phrases and KS'13 complements ML'10 by incorporating an appropriate subject for each verb-object phrase. KS'14 is the re-annotated version of KS'13 using a cloud sourcing service. Unlike GS'11, these three datasets require one to capture the topical similarity rather than the disambiguation aspect [Polajnar et al., 2014].

4.1.4.2 Result Overview

Table 4.4 and 4.5 show the evaluation results using BNC and enWiki, respectively. The results are shown for each method, data type, and embedding dimensionality. These tables also show the results from other work [Hashimoto et al., 2014; Milajevs et al., 2014; Polajnar et al., 2014] on the same tasks while the training settings, such as the corpus and information used in the training, are different from those in this work. However, the evaluation settings are the same with those in the previous work. That is, in the previous work, averaged human ratings were used to evaluate the Spearman's rank correlation scores, similarity scores between subject-verb-object phrases were used for GS'11, KS'13, and KS'14, and similarity scores between verb-object phrases were used for ML'10.

Effects of using adjuncts. Except for the results for GS'11 using PAS-CLBLM, the correlation scores consistently improve when using the SVOPN data compared with using the SVO data, which shows using adjuncts is helpful in learning the meanings of verb phrases. Using the SVO data alone, verb phrase embeddings themselves are not directly learned but computed separately. By contrast, the

	Our method				PAS-CLBLM	
	SVO		SVOPN		SVOPN	
	make dollar	make saving	make cash	earn billion	make cash	make penny
make	make pay	use money	make dollar	earn million	make dollar	make baht
money	make profit	make cent	make profit	make gamble	make yen	make salary
	make cash	do business	earn baht	make pound	make pay	make profit
	earn profit	sell coin	earn pound	earn earning	make fund	make rupee
	make repayment	make expenditure	make loan	pay reimbursement	make loan	make cost
make	make loan	pay subsidy	make repayment	pay remuneration	make repayment	make receipt
payment	pay amount	pay deposit	pay fine	make raise	make compensation	make guarantee
	make offer	make transaction	pay amount	pay cost	make expense	make rebate
	pay compensation	pay donor	pay surcharge	pay fee	make debt	make purchase
	use material	use approach	use number	use one	make usage	make sort
make	use type	use method	use concept	use element	make placement	make size
use	use concept	use technique	use approach	use set	make kind	make utilization
	use form	use instrument	use method	use system	make quality	make redundancy
	use one	use system	use model	use type	make alternative	make handling

Table 4.6: Nearest neighbor verb-object phrases.

SVOPN data provides the opportunity for learning verb phrase embeddings.

Effects of the training corpora. In previous work on learning and evaluating word embeddings, it is generally observed that increasing the training data results in better results. However, as opposed to our expectation, Table 4.4 and 4.5 show that using enWiki does not necessarily lead to better results. A possible explanation is that the nature of the training corpus matters the most. The usage of each word depends on the training corpora, and at least for these verb sense tasks, the size of BNC is sufficient and the nature of BNC fits these tasks.

4.1.4.3 Disambiguation Task

Our method outperforms both the baseline and the previous state of the art for GS'11, which demonstrates that our method better handles the disambiguation of transitive verbs. This result is somewhat expected since our method provides stronger interaction between predicates and their arguments than the baseline method.

Table 4.6 shows some examples[7] of verb-object phrases with their nearest neighbor ones in the

[7]The verb-object phrase "make use" is the part of the idiomatic expression "make use of".

embedding space according to the cosine similarity. For our method, we show the results of using the SVO and SVOPN data, and for PAS-CLBLM, we show the results of using the SVOPN data. In each setting, we used the enWiki data with $d = 50$.

Table 4.6 clearly shows the difference between our method and the baseline method. In our method, the meaning of "make" becomes close to those of "earn", "pay", and "use" when taking "money", "payment", and "use", respectively, as its object. By contrast, PAS-CLBLM simply emphasizes the head word "make". In previous work, it is also reported that the weighed addition composition functions put more weight on head words [Hashimoto et al., 2014; Muraoka et al., 2014; Socher et al., 2013]. As opposed to these previous methods, our method has the ability of selecting the meaning of transitive verbs according to their objects.

Table 4.6 also shows that the phrase embeddings in our method are influenced by using the adjunct data (i.e., the SVOPN data). For example, in the example of "make money", the results for using the SVO data include "use money" as the nearest neighbors. When using the SVOPN data, the focus seems to shift to the true meaning of "make money".

4.1.4.4 Phrase Similarity Task

In the phrase similarity tasks, our method compares favorably to PAS-CLBLM for ML'10, but PAS-CLBLM outperforms our method for KS'13 and KS'14. These results are consistent with those in previous work. In Milajevs et al. (2014) and Polajnar et al. (2014), using the simplest composition function (the element-wise vector addition) achieves much better correlation scores than other tensor-based complex composition functions. These results indicate that our method is suitable for capturing the disambiguation rather than capturing the topical similarity between phrases.

4.1.4.5 Qualitative Evaluation on Verb Matrices

Finally, we inspect the learned verb matrices using the SVO data of enWiki with $d = 50$. Compared with the word embeddings, the verb matrices have two-dimensional structure. According to Eq. (4.3), each row vector and each column vector in a verb matrix are updated to capture the information about what subject-object pairs are relevant (or irrelevant) to the verb.

Table 4.7 shows the nearest neighbor verbs using the cosine similarity between row (or column) vectors in the verb matrices. For reference, we also show the results using the vectorized representation of the verb matrices (denoted as "all" in the table). While the entire matrices capture the general

Verb		Nearest neighbors
run	27th col.	operate, execute, insert, hold, grid, produce, add, assume, manage, render
	34th row	release, operate, create, override, govern, oversee, distribute, host, organize
	all	operate, start, manage, own, launch, continue, establish, open, maintain
encode	28th row	denature, transfect, phosphorylate, polymerize, subtend, acid
	39th row	format, store, decode, embed, concatenate, encrypt, memorize
	all	concatenate, permute, phosphorylate, quantize, composite, transfect, transduce

Table 4.7: Nearest neighbor verbs.

similarity between verbs as with word embeddings, some specific rows (or columns) capture the multiple meanings of usages of the verbs.

4.1.5 Related Work

Based on the distributional hypothesis [Firth, 1957], various methods for word embeddings have been actively studied [Levy and Goldberg, 2014b; Mikolov et al., 2013b]. Recent studies also investigate how to learn phrase and/or sentence embeddings using syntactic structures and word embeddings [Socher et al., 2011b]. Along the same line of research, there is a growing body of work on representing transitive verb phrases using word embeddings [Grefenstette and Sadrzadeh, 2011; Hashimoto et al., 2014; Kartsaklis et al., 2012; Tsubaki et al., 2013]. Those studies can be split into two approaches: one is based on tensor calculus and the other is based on neural networks.

In contrast to the recent studies on word embeddings, the tensor-based methods represent words with tensors which are not limited to vectors. That is, higher order tensors such as matrices and three-mode tensors are also used. In the case of representing transitive verb phrases, for example, each transitive verb is represented as a matrix and each noun is represented as a vector in Grefenstette

and Sadrzadeh (2011). Based on Coecke et al. (2010), Grefenstette and Sadrzadeh (2011) presented a method for calculating a verb matrix using word embeddings of its observed subjects and objects. The word embeddings were constructed by the method in Mitchell and Lapata (2008). Grefenstette and Sadrzadeh (2011) then introduced composition functions using the verb matrices and the noun embeddings. Their approach has been followed by some recent studies [Kartsaklis et al., 2012; Milajevs et al., 2014; Polajnar et al., 2014; Van de Cruys et al., 2013].

In the neural network-based methods each word is usually represented with a vector. Tsubaki et al. (2013) presented a neural network language model focusing on the binary relationship between verbs and their objects. Their co-compositionality method enables verb embeddings to be multiplicatively influenced by the objects, and vice versa. Subsequently, Hashimoto et al. (2014) introduced a method which jointly learns word and phrase embeddings by using a variety of predicate-argument structures. While their method achieves state-of-the-art results on phrase similarity tasks, the interaction between predicates and their arguments is weak.

4.1.6 Conclusion and Future Work

We have presented an implicit matrix factorization method for learning the embeddings of transitive verb phrases. The verb matrices learned by our method capture the multiple meanings of transitive verbs and we have shown that adjuncts play an important role in learning the meanings of transitive verb phrases. In our experiments, our method outperforms the previous state of the art on a transitive verb disambiguation task. In future work, we will investigate how the learned phrase embeddings improve real-world NLP applications.

4.2 Learning Compositionality of Phrases

Summary We present a novel method for jointly learning compositional and non-compositional phrase embeddings by adaptively weighting both types of embeddings using a compositionality scoring function. The scoring function is used to quantify the level of compositionality of each phrase, and the parameters of the function are jointly optimized with the objective for learning phrase embeddings. In experiments, we apply the adaptive joint learning method to the task of learning embeddings of transitive verb phrases, and show that the compositionality scores have strong correlation with human ratings for verb-object compositionality, substantially outperforming the previous state of the art. Moreover, our embeddings improve upon the previous best model on a transitive verb disambiguation task. We also show that a simple ensemble technique further improves the results for both tasks.

4.2.1 Introduction

Representing words and phrases in a vector space has proven effective in a variety of language processing tasks [Pham et al., 2015; Sutskever et al., 2014]. In most of the previous work, phrase embeddings are computed from word embeddings by using various kinds of composition functions. Such composed embeddings are called *compositional embeddings*. An alternative way of computing phrase embeddings is to treat phrases as single units and assigning a unique embedding to each candidate phrase [Mikolov et al., 2013b; Yazdani et al., 2015]. Such embeddings are called *non-compositional embeddings*.

Relying solely on non-compositional embeddings has the obvious problem of data sparsity (i.e. rare or unknown phrase problems). At the same time, however, using compositional embeddings is not always the best option since some phrases are inherently non-compositional. For example, the phrase "bear fruits" means "to yield results"[8] but it is hard to infer its meaning by composing the meanings of "bear" and "fruit". Treating all phrases as compositional also has a negative effect in learning the composition function because the words in those idiomatic phrases are not just uninformative but can serve as noisy samples in the training. These problems have motivated us to adaptively combine both types of embeddings.

Most of the existing methods for learning phrase embeddings can be divided into two approaches. One approach is to learn compositional embeddings by regarding all phrases as compositional [Pham

[8]The definition is found at `http://idioms.thefreedictionary.com/bear+fruit`.

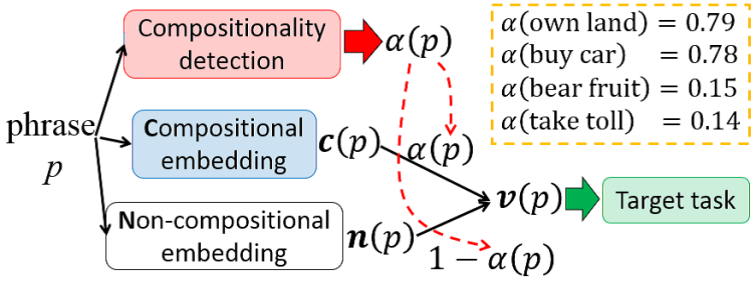


Figure 4.1: The overview of our method and examples of the compositionality scores. Given a phrase p, our method first computes the compositionality score $\alpha(p)$ (Eq. (4.15)), and then computes the phrase embedding $\boldsymbol{v}(p)$ using the compositional and non-compositional embeddings, $\boldsymbol{c}(p)$ and $\boldsymbol{n}(p)$, respectively (Eq. (4.14)).

et al., 2015; Socher et al., 2012]. The other approach is to learn both types of embeddings separately and use the better ones [Kartsaklis and Sadrzadeh, 2014; Muraoka et al., 2014]. Kartsaklis and Sadrzadeh (2014) show that non-compositional embeddings are better suited for a phrase similarity task, whereas Muraoka et al. (2014) report the opposite results on other tasks. These results suggest that we should not stick to either of the two types of embeddings unconditionally and could learn better phrase embeddings by considering the compositionality levels of the individual phrases in a more flexible fashion.

In this paper, we propose a method that jointly learns compositional and non-compositional embeddings by adaptively weighting both types of phrase embeddings using a compositionality scoring function. The scoring function is used to quantify the level of compositionality of each phrase and learned in conjunction with the target task for learning phrase embeddings. In experiments, we apply our method to the task of learning transitive verb phrase embeddings and demonstrate that it allows us to achieve state-of-the-art performance on standard datasets for compositionality detection and verb disambiguation.

4.2.2 Method

In this section, we describe our approach in the most general form, without specifying the function to compute the compositional embeddings or the target task for optimizing the embeddings.

Figure 5.2 shows the overview of our proposed method. At each iteration of the training (i.e. gradient calculation) of a certain target task (e.g. language modeling or sentiment analysis), our method first computes a compositionality score for each phrase. Then the score is used to weight the compositional and non-compositional embeddings of the phrase in order to compute the expected embedding of the phrase which is to be used in the target task. Some examples of the compositionality scores are also shown in the figure.

4.2.2.1 Compositional Phrase Embeddings

The compositional embedding $\boldsymbol{c}(p) \in \mathbb{R}^{d \times 1}$ of a phrase $p = (w_1, \cdots, w_L)$ is formulated as

$$\boldsymbol{c}(p) = f(\boldsymbol{v}(w_1), \cdots, \boldsymbol{v}(w_L)), \tag{4.13}$$

where d is the dimensionality, L is the phrase length, $\boldsymbol{v}(\cdot) \in \mathbb{R}^{d \times 1}$ is a word embedding, and $f(\cdot)$ is a composition function. The function can be simple ones such as element-wise addition or multiplication [Mitchell and Lapata, 2008]. More complex ones such as recurrent neural networks [Sutskever et al., 2014] are also commonly used. The word embeddings and the composition function are jointly learned on a certain target task. Since compositional embeddings are built on word-level (i.e. unigram) information, they are less prone to the data sparseness problem.

4.2.2.2 Non-Compositional Phrase Embeddings

In contrast to the compositional embedding, the non-compositional embedding of a phrase $\boldsymbol{n}(p) \in \mathbb{R}^{d \times 1}$ is independently parameterized, i.e., the phrase p is treated just like a single word. Mikolov et al. (2013b) show that non-compositional embeddings are preferable when dealing with idiomatic phrases. Some recent studies [Kartsaklis and Sadrzadeh, 2014; Muraoka et al., 2014] have discussed the (dis)advantages of using compositional or non-compositional embeddings. However, in most cases, a phrase is neither completely compositional nor completely non-compositional. To the best of our knowledge, there is no method that allows us to jointly learn both types of phrase embeddings by incorporating the levels of compositionality of the phrases as real-valued scores.

4.2.2.3 Adaptive Joint Learning

To simultaneously consider both compositional and non-compositional aspects of each phrase, we compute a phrase embedding $\boldsymbol{v}(p)$ by adaptively weighting $\boldsymbol{c}(p)$ and $\boldsymbol{n}(p)$ as follows:

$$\boldsymbol{v}(p) = \alpha(p)\boldsymbol{c}(p) + (1 - \alpha(p))\boldsymbol{n}(p), \tag{4.14}$$

where $\alpha(\cdot)$ is a scoring function that quantifies the compositionality levels, and outputs a real value ranging from 0 to 1. What we expect from the scoring function is that large scores indicate high levels of compositionality. In other words, when $\alpha(p)$ is close to 1, the compositional embedding is mainly considered, and vice versa. For example, we expect $\alpha(\text{buy car})$ to be large and $\alpha(\text{bear fruit})$ to be small as shown in Figure 5.2.

We parameterize the scoring function $\alpha(p)$ as logistic regression:

$$\alpha(p) = \sigma(\boldsymbol{W} \cdot \boldsymbol{\phi}(p)), \tag{4.15}$$

where $\boldsymbol{\phi}(p) \in \mathbb{R}^{N\times 1}$ is a feature vector of the phrase p, $\boldsymbol{W} \in \mathbb{R}^{N\times 1}$ is a weight vector, N is the number of features, and $\sigma(\cdot)$ is the logistic function. The weight vector $\boldsymbol{W}$ is jointly optimized in conjunction with the objective J for the target task of learning phrase embeddings $\boldsymbol{v}(p)$.

Updating the model parameters Given the partial derivative $\boldsymbol{\delta}_p = \frac{\partial J}{\partial \boldsymbol{v}(p)} \in \mathbb{R}^{d\times 1}$ for the target task, we can compute the partial derivative for updating $\boldsymbol{W}$ as follows:

$$\delta_\alpha = \alpha(p)(1 - \alpha(p))\{\boldsymbol{\delta}_p \cdot (\boldsymbol{c}(p) - \boldsymbol{n}(p))\} \tag{4.16}$$

$$\frac{\partial J}{\partial \boldsymbol{W}} = \delta_\alpha \boldsymbol{\phi}(p). \tag{4.17}$$

If $\boldsymbol{\phi}(p)$ is not constructed by static features but is computed by a feature learning model such as neural networks, we can propagate the error term δ_α into the feature learning model by the following equation:

$$\frac{\partial J}{\partial \boldsymbol{\phi}(p)} = \delta_\alpha \boldsymbol{W}. \tag{4.18}$$

When we use only static features, as in this work, we can simply compute the partial derivatives of J with respect to $\boldsymbol{c}(p)$ and $\boldsymbol{n}(p)$ as follows:

$$\frac{\partial J}{\partial \boldsymbol{c}(p)} = \alpha(p)\boldsymbol{\delta}_p \tag{4.19}$$

$$\frac{\partial J}{\partial \boldsymbol{n}(p)} = (1 - \alpha(p))\boldsymbol{\delta}_p. \tag{4.20}$$

As mentioned above, Eq. (4.19) and (4.20) show that the non-compositional embeddings are mainly updated when $\alpha(p)$ is close to 0, and vice versa. The partial derivative $\frac{\partial J}{\partial \boldsymbol{c}(p)}$ is used to update the model parameters in the composition function via the backpropagation algorithm. Any differentiable composition functions can be used in our method.

Expected behavior of our method The training of our method depends on the target task; that is, the model parameters are updated so as to minimize the cost function as described above. More concretely, $\alpha(p)$ for each phrase p is adaptively adjusted so that the corresponding parameter updates contribute to minimizing the cost function. As a result, different phrases will have different $\alpha(p)$ values depending on their compositionality. If the size of the training data were almost infinitely large, $\alpha(p)$ for all phrases would become nearly zero, and the non-compositional embeddings $\boldsymbol{n}(p)$ are dominantly used (since that would allow the model to better fit the data). In reality, however, the amount of the training data is limited, and thus the compositional embeddings $\boldsymbol{c}(p)$ are effectively used to overcome the data sparseness problem.

4.2.3 Learning Verb Phrase Embeddings

This section describes a particular instantiation of our approach presented in the previous section, focusing on the task of learning the embeddings of transitive verb phrases.

4.2.3.1 Word and Phrase Prediction in Predicate-Argument Relations

Acquisition of selectional preference using embeddings has been widely studied, where word and/or phrase embeddings are learned based on syntactic links [Bansal et al., 2014; Hashimoto and Tsuruoka, 2015; Levy and Goldberg, 2014b; Van de Cruys, 2014]. As with language modeling, these methods perform word (or phrase) prediction using (syntactic) contexts.

In this work, we focus on verb-object relationships and employ a phrase embedding learning method presented in Hashimoto and Tsuruoka (2015). The task is a plausibility judgment task for predicate-argument tuples. They extracted Subject-Verb-Object (*SVO*) and SVO-Preposition-Noun (*SVOPN*) tuples using a probabilistic HPSG parser, *Enju* [Miyao and Tsujii, 2008], from the training corpora. Transitive verbs and prepositions are extracted as predicates with two arguments. For example, the extracted tuples include (S, V, O) = ("importer", "make", "payment") and (SVO, P, N) = ("importer make payment", "in", "currency"). The task is to discriminate between observed and unobserved tuples, such as the (S, V, O) tuple mentioned above and (S, V', O) = ("importer", "*eat*", "payment"), which is generated by replacing "make" with "eat". The (S, V', O) tuple is unlikely to be observed.

For each tuple (p, a_1, a_2) observed in the training data, a cost function is defined as follows:

$$\begin{aligned} &-\log\sigma(s(p,a_1,a_2)) - \log\sigma(-s(p',a_1,a_2)) \\ &\qquad - \log\sigma(-s(p,a_1',a_2)) \\ &\qquad - \log\sigma(-s(p,a_1,a_2')), \end{aligned} \tag{4.21}$$

where $s(\cdot)$ is a plausibility scoring function, and p, a_1 and a_2 are a predicate and its arguments, respectively. Each of the three unobserved tuples (p', a_1, a_2), (p, a_1', a_2), and (p, a_1, a_2') is generated by replacing one of the entries with a random sample.

In their method, each predicate p is represented with a matrix $\boldsymbol{M}(p) \in \mathbb{R}^{d\times d}$ and each argument a with an embedding $\boldsymbol{v}(a) \in \mathbb{R}^{d\times 1}$. The matrices and embeddings are learned by minimizing the cost function using *AdaGrad* [Duchi et al., 2011]. The scoring function is parameterized as

$$s(p, a_1, a_2) = \boldsymbol{v}(a_1) \cdot (\boldsymbol{M}(p)\boldsymbol{v}(a_2)), \tag{4.22}$$

and the VO and SVO embeddings are computed as

$$\boldsymbol{v}(VO) = \boldsymbol{M}(V)\boldsymbol{v}(O) \tag{4.23}$$

$$\boldsymbol{v}(SVO) = \boldsymbol{v}(S) \odot \boldsymbol{v}(VO), \tag{4.24}$$

as proposed by Kartsaklis et al. (2012). The operator $\odot$ denotes element-wise multiplication. In

summary, the scores are computed as

$$s(V, S, O) = \boldsymbol{v}(S) \cdot \boldsymbol{v}(VO) \tag{4.25}$$

$$s(P, SVO, N) = \boldsymbol{v}(SVO) \cdot (\boldsymbol{M}(P)\boldsymbol{v}(N)). \tag{4.26}$$

With this method, the word and composed phrase embeddings are jointly learned based on co-occurrence statistics of predicate-argument structures. Using the learned embeddings, they achieved state-of-the-art accuracy on a transitive verb disambiguation task [Grefenstette and Sadrzadeh, 2011].

4.2.3.2 Applying the Adaptive Joint Learning

In this section, we apply our adaptive joint learning method to the task described in Section 4.2.3.1. We here redefine the computation of $\boldsymbol{v}(VO)$ by first replacing $\boldsymbol{v}(VO)$ in Eq. (4.23) with $\boldsymbol{c}(VO)$ as,

$$\boldsymbol{c}(VO) = \boldsymbol{M}(V)\boldsymbol{v}(O), \tag{4.27}$$

and then assigning VO to p in Eq. (4.14) and (4.15):

$$\boldsymbol{v}(VO) = \alpha(VO)\boldsymbol{c}(VO) + (1 - \alpha(VO))\boldsymbol{n}(VO), \tag{4.28}$$

$$\alpha(VO) = \sigma(\boldsymbol{W} \cdot \boldsymbol{\phi}(VO)). \tag{4.29}$$

The $\boldsymbol{v}(VO)$ in Eq. (4.28) is used in Eq. (4.24) and (4.25). We assume that the candidates of the phrases are given in advance. For the phrases not included in the candidates, we set $\boldsymbol{v}(VO) = \boldsymbol{c}(VO)$. This is analogous to the way a human guesses the meaning of an idiomatic phrase she does not know. We should note that $\boldsymbol{\phi}(VO)$ can be computed for phrases not included in the candidates, using partial features among the features described below. If any features do not fire, $\boldsymbol{\phi}(VO)$ becomes 0.5 according to the logistic function.

For the feature vector $\boldsymbol{\phi}(VO)$, we use the following simple binary and real-valued features:

- indices of V, O, and VO
- frequency and Pointwise Mutual Information (PMI) values of VO.

More concretely, the first set of the features (indices of V, O, and VO) is the concatenation of traditional one-hot vectors. The second set of features, frequency and PMI [Church and Hanks, 1990] features,

have proven effective in detecting the compositionality of transitive verbs in McCarthy et al. (2007) and Venkatapathy and Joshi (2005). Given the training corpus, the frequency feature for a VO pair is computed as

$$freq(VO) = \log(count(VO)), \tag{4.30}$$

where $count(VO)$ counts how many times the VO pair appears in the training corpus, and the PMI feature is computed as

$$\mathrm{PMI}(VO) = \log \frac{count(VO)count(*)}{count(V)count(O)}, \tag{4.31}$$

where $count(V)$, $count(O)$, and $count(*)$ are the counts of the verb V, the object O, and all VO pairs in the training corpus, respectively. We normalize the frequency and PMI features so that their maximum absolute value becomes 1.

4.2.4 Experimental Settings

4.2.4.1 Training Data

As the training data, we used two datasets, one small and one large: the British National Corpus (BNC) [Leech, 1992] and the English Wikipedia. More concretely, we used the publicly available data[9] preprocessed by Hashimoto and Tsuruoka (2015). The BNC data consists of 1.38 million SVO tuples and 0.93 million SVOPN tuples. The Wikipedia data consists of 23.6 million SVO tuples and 17.3 million SVOPN tuples. Following the provided code[10], we used exactly the same train/development/test split (0.8/0.1/0.1) for training the overall model. As the third training data, we also used the concatenation of the two data, which is hereafter referred to as *BNC-Wikipedia*.

We applied our adaptive joint learning method to verb-object phrases observed more than K times in each corpus. K was set to 10 for the BNC data and 100 for the Wikipedia and BNC-Wikipedia data. Consequently, the non-compositional embeddings were assigned to 17,817, 28,933, and 30,682 verb-object phrase types in the BNC, Wikipedia, and BNC-Wikipedia data, respectively.

4.2.4.2 Training Details

The model parameters consist of d-dimensional word embeddings for nouns, non-compositional phrase embeddings, $d \times d$-dimensional matrices for verbs and prepositions, and a weight vector $\boldsymbol{W}$ for $\alpha(VO)$.

[9] http://www.logos.t.u-tokyo.ac.jp/~hassy/publications/cvsc2015/
[10] https://github.com/hassyGo/SVOembedding

All the model parameters are jointly optimized. We initialized the embeddings and matrices with zero-mean gaussian random values with a variance of $\frac{1}{d}$ and $\frac{1}{d^2}$, respectively, and $\boldsymbol{W}$ with zeros. Initializing $\boldsymbol{W}$ with zeros forces the initial value of each $\alpha(VO)$ to be 0.5 since we use the logistic function to compute $\alpha(VO)$.

The optimization was performed via mini-batch AdaGrad [Duchi et al., 2011]. We fixed d to 25 and the mini-batch size to 100. We set candidate values for the learning rate ε to $\{0.01, 0.02, 0.03, 0.04, 0.05\}$. For the weight vector $\boldsymbol{W}$, we employed L2-norm regularization and set the coefficient λ to $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 0\}$. For selecting the hyperparameters, each training process was stopped when the evaluation score on the development split decreased. Then the best performing hyperparameters were selected for each training dataset. Consequently, ε was set to 0.05 for all training datasets, and λ was set to 10^{-6}, 10^{-3}, and 10^{-5} for the BNC, Wikipedia, and BNC-Wikipedia data, respectively. Once the training is finished, we can use the learned embeddings and the scoring function in downstream target tasks.

4.2.5 Evaluation on the Compositionality Detection Function

4.2.5.1 Evaluation Settings

Datasets First, we evaluated the learned compositionality detection function on two datasets, **VJ'05**[11] and **MC'07**[12], provided by Venkatapathy and Joshi (2005) and McCarthy et al. (2007), respectively. VJ'05 consists of 765 verb-object pairs with human ratings for the compositionality. MC'07 is a subset of VJ'05 and consists of 638 verb-object pairs. For example, the rating of "buy car" is 6, which is the highest score, indicating the phrase is highly compositional. The rating of "bear fruit " is 1, which is the lowest score, indicating the phrase is highly non-compositional.

Evaluation metric The evaluation was performed by calculating Spearman's rank correlation scores[13] between the averaged human ratings and the learned compositionality scores $\alpha(VO)$.

Ensemble technique We also produced the result by employing an *ensemble* technique. More concretely, we used the averaged compositionality scores from the results of the BNC and Wikipedia data for the ensemble result.

[11] `http://www.dianamccarthy.co.uk/downloads/SVAJ2005compositionality_rating.txt`
[12] `http://www.dianamccarthy.co.uk/downloads/emnlp2007data.txt`
[13] We used the Scipy 0.12.0 implementation in Python.

Method	MC'07	VJ'05
Proposed method (Wikipedia)	0.508	0.514
Proposed method (BNC)	0.507	0.507
Proposed method (BNC-Wikipedia)	0.518	0.527
Proposed method (Ensemble)	0.550	0.552
Kiela and Clark (2013) w/ WordNet	n/a	0.461
Kiela and Clark (2013)	n/a	0.420
DSPROTO [McCarthy et al., 2007]	0.398	n/a
PMI [McCarthy et al., 2007]	0.274	n/a
Frequency [McCarthy et al., 2007]	0.141	n/a
DSPROTO+ [McCarthy et al., 2007]	0.454	n/a
Human agreement	0.702	0.716

Table 4.8: Compositionality detection task.

4.2.5.2 Result Overview

Table 4.8 shows our results and the state of the art. Our method outperforms the previous state of the art in all settings. The result denoted as *Ensemble* is the one that employs the ensemble technique, and achieves the strongest correlation with the human-annotated datasets. Even without the ensemble technique, our method performs better than all of the previous methods.

Kiela and Clark (2013) used window-based co-occurrence vectors and improved their score using WordNet hypernyms. By contrast, our method does not rely on such external resources, and only needs parsed corpora. We should note that Kiela and Clark (2013) reported that their score did not improve when using parsed corpora. Our method also outperforms DSPROTO+, which used a small amount of the labeled data, while our method is fully unsupervised.

We calculated confidence intervals ($P < 0.05$) using bootstrap resampling [Noreen, 1989]. For example, for the results using the BNC-Wikipedia data, the intervals on MC'07 and VJ'05 are (0.455, 0.574) and (0.475, 0.579), respectively. These results show that our method significantly outperforms the previous state-of-the-art results.

Phrase		Gold standard	(a) BNC	(b) Wikipedia	BNC-Wikipedia	Ensemble ((a)+(b))×0.5
(A)	buy car	6	0.78	0.71	0.80	0.74
	own land	6	0.79	0.73	0.76	0.76
	take toll	1.5	0.14	0.11	0.06	0.13
	shed light	1	0.21	0.07	0.07	0.14
	bear fruit	1	0.15	0.19	0.17	0.17
(B)	make noise	6	0.37	0.33	0.30	0.35
	have reason	5	0.26	0.39	0.33	0.33
(C)	smoke cigarette	6	0.56	0.90	0.78	0.73
	catch eye	1	0.48	0.14	0.17	0.31

Table 4.9: Examples of the compositionality scores.

4.2.5.3 Analysis of Compositionality Scores

Figure 4.2 shows how $\alpha(VO)$ changes for the seven phrases during the training on the BNC data. As shown in the figure, starting from 0.5, $\alpha(VO)$ for each phrase converges to its corresponding value. The differences in the trends indicate that our method can adaptively learn compositionality levels for the phrases. Table 4.9 shows the learned compositionality scores for the three groups of the examples along with the gold-standard scores given by the annotators. The group (A) is considered to be consistent with the gold-standard scores, the group (B) is not, and the group (C) shows examples for which the difference between the compositionality scores of our results is large.

Characteristics of light verbs The verbs "take", "make", and "have" are known as *light verbs* [14], and the scoring function tends to assign low scores to light verbs. In other words, our method can recognize that the light verbs are frequently used to form idiomatic (i.e. non-compositional) phrases. To verify the assumption, we calculated the average compositionality score for each verb by averaging the compositionality scores paired with its candidate objects. Here we used 135 verbs which take more than 30 types of objects in the BNC data. Table 4.10 shows the 10 highest and lowest average scores with the corresponding verbs. We see that relatively low scores are assigned to the light verbs as well

[14] In Section 5.2.2 in Newton (2006), the term *light verb* is used to refer to verbs which can be used in combination with some other element where their contribution to the meaning of the whole construction is reduced in some way.

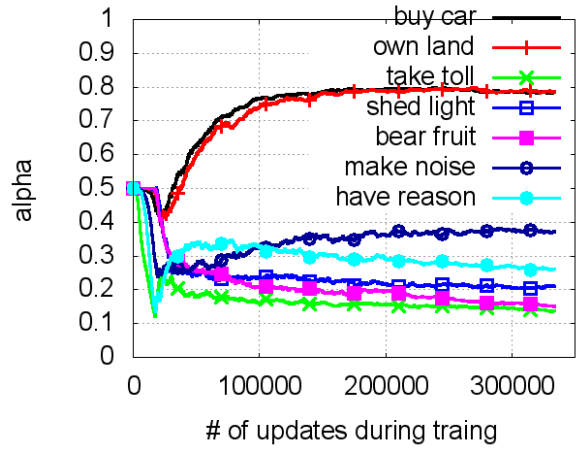


Figure 4.2: Trends of $\alpha(VO)$ during the training on the BNC data.

as other verbs which often form idiomatic phrases. As shown in the group (B) in Table 4.9, however, light verb phrases are not always non-compositional. Despite this, the learned function assigns low scores to compositional phrases formed by the light verbs. These results suggest that using a more flexible scoring function may further strengthen our method.

Context dependence Both our method and the two datasets, VJ'05 and MC'07, assume that the compositionality score can be computed for each phrase with no contextual information. However, in general, the compositionality level of a phrase depends on its contextual information. For example, the meaning of the idiomatic phrase "bear fruit" can be compositionaly interpreted as "to yield fruit" for a plant or tree. We manually inspected the BNC data to check whether the phrase "bear fruit" is used as the compositional meaning or the idiomatic meaning ("to yield results"). As a result, we have found that most of the usage was its idiomatic meaning. In the model training, our method is affected by the majority usage and fits the evaluation datasets where the phrase "bear fruit" is regarded as highly non-compositional. Incorporating contextual information into the compositionality scoring function is a promising direction of future work.

Highest average scores		Lowest average scores	
approve	0.83	bear	0.37
reject	0.72	play	0.38
discuss	0.71	have	0.38
visit	0.70	make	0.39
want	0.70	break	0.40
describe	0.70	take	0.40
involve	0.69	raise	0.41
own	0.68	reach	0.41
attend	0.68	gain	0.42
reflect	0.67	draw	0.42

Table 4.10: The 10 highest and lowest average compositionality scores with the corresponding verbs on the BNC data.

4.2.5.4 Effects of Ensemble

We used the two different corpora for constructing the training data, and our method achieves the state-of-the-art results in all settings. To inspect the results on VJ'05, we calculated the correlation score between the outputs from our results of the BNC and Wikipedia data. The correlation score is 0.674 and that is, the two different corpora lead to reasonably consistent results, which indicates the robustness of our method. However, the correlation score is still much lower than perfect correlation; in other words, there are disagreements between the outputs learned with the corpora. The group (C) in Table 4.9 shows such two examples. In these cases, the ensemble technique is helpful in improving the results as shown in the examples.

Another interesting observation in our results is that the result of the ensemble technique outperforms that of the BNC-Wikipedia data as shown in Table 4.8. This shows that separately using the training corpora of different nature and then performing the ensemble technique can yield better results. By contrast, many of the previous studies on embedding-based methods combine different corpora into a single dataset, or use multiple corpora just separately and compare them [Hashimoto and Tsuruoka, 2015; Muraoka et al., 2014; Pennington et al., 2014]. It would be worth investigating whether the results in the previous work can be improved by ensemble techniques.

4.2.6 Evaluation on the Phrase Embeddings

4.2.6.1 Evaluation Settings

Dataset Next, we evaluated the learned embeddings on the transitive verb disambiguation dataset **GS'11**[15] provided by Grefenstette and Sadrzadeh (2011). GS'11 consists of 200 pairs of transitive verbs and each verb pair takes the same subject and object. For example, the transitive verb "run" is known as a polysemous word and this task requires one to identify the meanings of "run" and "operate" as similar to each other when taking "people" as their subject and "company" as their object. In the same setting, however, the meanings of "run" and "move" are not similar to each other. Each pair has multiple human ratings indicating how similar the phrases of the pair are.

Evaluation metric The evaluation was performed by calculating Spearman's rank correlation scores between the human ratings and the cosine similarity scores of $\boldsymbol{v}(SVO)$ in Eq. (4.24). Following the previous studies, we used the gold-standard ratings in two ways: averaging the human ratings for each SVO tuple (GS'11a) and treating each human rating separately (GS'11b).

Ensemble technique We used the same ensemble technique described in Section 4.2.5.1. In this task we produced two ensemble results: *Ensemble A* and *Ensemble B*. The former used the averaged cosine similarity from the results of the BNC and Wikipedia data, and the latter further incorporated the result of the BNC-Wikipedia data.

Baselines We compared our adaptive joint learning method with two baseline methods. One is the method in Hashimoto and Tsuruoka (2015) and it is equivalent to fixing $\alpha(VO)$ to 1 in our method. The other is fixing $\alpha(VO)$ to 0.5 in our method, which serves as a baseline to evaluate how effective the proposed adaptive weighting method is.

4.2.6.2 Result Overview

Table 4.11 shows our results and the state of the art, and our method outperforms almost all of the previous methods in both datasets. Again, the ensemble technique further improves the results, and overall, Ensemble B yields the best results.

[15] `http://www.cs.ox.ac.uk/activities/compdistmeaning/GS2011data.txt`

The scores in Hashimoto and Tsuruoka (2015), the baseline results with $\alpha(VO) = 1$ in our method, have been the best to date. As shown in Table 4.11, our method outperforms the baseline results with $\alpha(VO) = 0.5$ as well as those with $\alpha(VO) = 1$. We see that our method improves the baseline scores by adaptively combining compositional and non-compositional embeddings. Along with the results in Table 4.8, these results show that our method allows us to improve the composition function by jointly learning non-compositional embeddings and the scoring function for compositionality detection.

4.2.6.3 Analysis of the Learned Embeddings

We inspected the effects of adaptively weighting the compositional and non-compositional embeddings. Table 4.12 shows the five closest neighbor phrases in terms of the cosine similarity for the three idiomatic phrases "take toll", "catch eye", and "bear fruit" as well as the two non-idiomatic phrases "make noise" and "buy car". The examples trained with the Wikipedia data are shown for our method and the two baselines, i.e., $\alpha(VO) = 1$ and $\alpha(VO) = 0.5$. As shown in Table 4.9, the compositionality levels of the first three phrases are low and their non-compositional embeddings are dominantly used to represent their meaning.

One observation with $\alpha(VO) = 1$ is that head words (i.e. verbs) are emphasized in the shown examples except "take toll" and "make noise". As with other embedding-based methods, the compositional embeddings are highly affected by their component words. As a result, the phrases consisting of the same verb and the similar objects are often listed as the closest neighbors. By contrast, our method flexibly allows us to adaptively omit the information about the component words. Therefore, our method puts more weight on capturing the idiomatic aspects of the example phrases by adaptively using the non-compositional embeddings.

The results of $\alpha(VO) = 0.5$ are similar to those with our proposed method, but we can see some differences. For example, the phrase list for "make noise" of our proposed method captures offensive meanings, whereas that of $\alpha(VO) = 0.5$ is somewhat ambiguous. As another example, the phrase lists for "buy car" show that our method better captures the semantic similarity between the objects than $\alpha(VO) = 0.5$. This is achieved by adaptively assigning a relatively large compositionality score (0.71) to the phrase to use the information about the object "car".

We should note that "make noise" is highly compositional but our method outputs $\alpha(\text{make noise}) = 0.33$, and the phrase list of $\alpha(VO) = 1$ is the most appropriate in this case. Improving the composi-

tionality detection function should thus further improve the learned embeddings.

4.2.7 Related Work

Learning embeddings of words and phrases has been widely studied, and the phrase embeddings have proven effective in many language processing tasks, such as machine translation [Cho et al., 2014b; Sutskever et al., 2014], sentiment analysis and semantic textual similarity [Tai et al., 2015]. Most of the phrase embeddings are constructed by word-level information via various kinds of composition functions like long short-term memory [Hochreiter and Schmidhuber, 1997] recurrent neural networks. Such composition functions should be powerful enough to efficiently encode information about all the words into the phrase embeddings. By simultaneously considering the compositionality of the phrases, our method would be helpful in saving the composition models from having to be powerful enough to perfectly encode the non-compositional phrases. As a first step towards this purpose, in this paper we have shown the effectiveness of our method on the task of learning verb phrase embeddings.

Many studies have focused on detecting the compositionality of a variety of phrases [Lin, 1999], including the ones on verb phrases [Diab and Bhutada, 2009; McCarthy et al., 2003] and compound nouns [Farahmand et al., 2015; Reddy et al., 2011]. Compared to statistical feature-based methods [McCarthy et al., 2007; Venkatapathy and Joshi, 2005], recent methods use word and phrase embeddings [Kiela and Clark, 2013; Yazdani et al., 2015]. The embedding-based methods assume that word embeddings are given in advance and as a post-processing step, learn or simply employ composition functions to compute phrase embeddings. In other words, there is no distinction between compositional and non-compositional phrases. Yazdani et al. (2015) further proposed to incorporate latent annotations (binary labels) for the compositionality of the phrases. However, binary judgments cannot consider numerical scores of the compositionality. By contrast, our method adaptively weights the compositional and non-compositional embeddings using the compositionality scoring function.

4.2.8 Conclusion and Future Work

We have presented a method for adaptively learning compositional and non-compositional phrase embeddings by jointly detecting compositionality levels of phrases. Our method achieves the state of the art on a compositionality detection task of verb-object pairs, and also improves upon the previous state-of-the-art method on a transitive verb disambiguation task. In future work, we will apply our

method to other kinds of phrases and tasks.

Method	GS'11a	GS'11b
Proposed method (Wikipedia)	0.598	0.461
Proposed method (BNC)	0.595	0.463
Proposed method (BNC-Wikipedia)	0.623	0.483
Proposed method (Ensemble A)	0.661	0.511
Proposed method (Ensemble B)	0.680	0.524
$\alpha(VO) = 0.5$ (Wikipedia)	0.491	0.386
$\alpha(VO) = 0.5$ (BNC)	0.599	0.462
$\alpha(VO) = 0.5$ (BNC-Wikipedia)	0.610	0.477
$\alpha(VO) = 0.5$ (Ensemble A)	0.612	0.474
$\alpha(VO) = 0.5$ (Ensemble B)	0.638	0.495
$\alpha(VO) = 1$ (Wikipedia)	0.576	n/a
$\alpha(VO) = 1$ (BNC)	0.574	n/a
Milajevs et al. (2014)	0.456	n/a
Polajnar et al. (2014)	n/a	0.370
Hashimoto et al. (2014)	0.420	0.340
Polajnar et al. (2015)	n/a	0.330
Grefenstette and Sadrzadeh (2011)	n/a	0.210
Human agreement	0.750	0.620

Table 4.11: Transitive verb disambiguation task. The results for $\alpha(VO) = 1$ are reported in Hashimoto and Tsuruoka (2015).

	Proposed method		$\alpha(VO) = 1$	$\alpha(VO) = 0.5$
take toll	$\alpha(\text{take toll}) = 0.11$	put strain	deplete division	put strain
		place strain	necessitate monitoring	cause lack
		cause strain	deplete pool	befall army
		have affect	create pollution	exacerbate weakness
		exacerbate injury	deplete field	cause strain
catch eye	$\alpha(\text{catch eye}) = 0.14$	catch attention	catch ear	grab attention
		grab attention	catch heart	make impression
		make impression	catch e-mail	catch attention
		lift spirit	catch imagination	become legend
		become favorite	catch attention	inspire playing
bear fruit	$\alpha(\text{bear fruit}) = 0.19$	accentuate effect	bear herb	increase richness
		enhance beauty	bear grain	reduce biodiversity
		enhance atmosphere	bear spore	fuel boom
		rejuvenate earth	bear variety	enhance atmosphere
		enhance habitat	bear seed	worsen violence
make noise	$\alpha(\text{make noise}) = 0.33$	attack intruder	make sound	burn can
		attack trespasser	do beating	kill monster
		avoid predator	get bounce	wash machine
		attack diver	get pulse	lightn flash
		attack pedestrian	lose bit	cook raman
buy car	$\alpha(\text{buy car}) = 0.71$	buy bike	buy truck	buy bike
		buy machine	buy bike	buy instrument
		buy motorcycle	buy automobile	buy chip
		buy automobile	buy motorcycle	buy scooter
		purchase coins	buy vehicle	buy motorcycle

Table 4.12: Examples of the closest neighbors in the learned embedding space. All of the results were obtained by using the Wikipedia data, and the values of $\alpha(VO)$ are the same as those in Table 4.9.

Chapter 5

Task-Oriented Learning of Dependency Structures by a Joint Many-Task Model

In this chapter, I propose a method for learning latent (dependency) graph structures for input sentences in a task-oriented manner. First I present a joint many-task model which hierarchically handles multiple NLP tasks to jointly learn pipelined NLP tasks. Then I propose how to learn a task-oriented graph-based dependency parser in the multi-task model, by taking machine translation as an example task. Section 5.1 and Section 5.2 in this chapter correspond to the following published papers, respectively.

Section 5.1 Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. 2017. A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 446–456.

Section 5.2 Kazuma Hashimoto and Yoshimasa Tsuruoka. 2017. Neural Machine Translation with Source-Side Latent Graph Parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 125–135.

5.1 A Joint Many-Task Model for NLP

Summary Transfer and multi-task learning have traditionally focused on either a single source-target pair or very few, similar tasks. Ideally, the linguistic levels of morphology, syntax and semantics

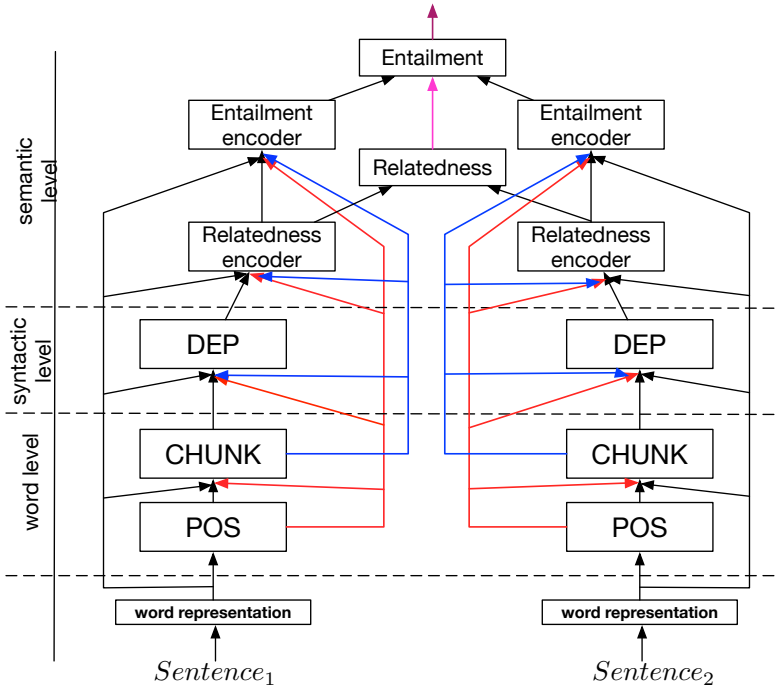


Figure 5.1: Overview of the joint many-task model predicting different linguistic outputs at successively deeper layers.

would benefit each other by being trained in a single model. We introduce a joint many-task model together with a strategy for successively growing its depth to solve increasingly complex tasks. Higher layers include shortcut connections to lower-level task predictions to reflect linguistic hierarchies. We use a simple regularization term to allow for optimizing all model weights to improve one task's loss without exhibiting catastrophic interference of the other tasks. Our single end-to-end model obtains state-of-the-art or competitive results on five different tasks from tagging, parsing, relatedness, and entailment tasks.

5.1.1 Introduction

The potential for leveraging multiple levels of representation has been demonstrated in various ways in the field of Natural Language Processing (NLP). For example, Part-Of-Speech (POS) tags are used for syntactic parsers. The parsers are used to improve higher-level tasks, such as natural language inference [Chen et al., 2016] and machine translation [Eriguchi et al., 2016b]. These systems are often pipelines and not trained end-to-end.

Deep NLP models have yet shown benefits from predicting many increasingly complex tasks each at a successively deeper layer. Existing models often ignore linguistic hierarchies by predicting different tasks either entirely separately or at the same depth [Collobert et al., 2011].

We introduce a Joint Many-Task (JMT) model, outlined in Figure 5.1.1, which predicts increasingly complex NLP tasks at successively deeper layers. Unlike traditional pipeline systems, our single JMT model can be trained end-to-end for POS tagging, chunking, dependency parsing, semantic relatedness, and textual entailment, by considering linguistic hierarchies. We propose an adaptive training and regularization strategy to grow this model in its depth. With the help of this strategy we avoid catastrophic interference between the tasks. Our model is motivated by Søgaard and Goldberg (2016) who showed that predicting two different tasks is more accurate when performed in different layers than in the same layer [Collobert et al., 2011]. Experimental results show that our single model achieves competitive results for all of the five different tasks, demonstrating that using linguistic hierarchies is more important than handling different tasks in the same layer.

5.1.2 The Joint Many-Task Model

This section describes the inference procedure of our model, beginning at the lowest level and working our way to higher layers and more complex tasks; our model handles the five different tasks in the order of POS tagging, chunking, dependency parsing, semantic relatedness, and textual entailment, by considering linguistic hierarchies. The POS tags are used for chunking, and the chunking tags are used for dependency parsing [Attardi and Dell ' Orletta, 2008]. Tai et al. (2015) have shown that dependencies improve the relatedness task. The relatedness and entailment tasks are closely related to each other. If the semantic relatedness between two sentences is very low, they are unlikely to entail each other. Based on this observation, we make use of the information from the relatedness task for improving the entailment task.

5.1.2.1 Word Representations

For each word w_t in the input sentence $\boldsymbol{s}$ of length L, we use two types of embeddings.
Word embeddings: We use Skip-gram [Mikolov et al., 2013b] to train word embeddings.
Character embeddings: Character n-gram embeddings are trained by the same Skip-gram objective. We construct the character n-gram vocabulary in the training data and assign an embedding for each entry. The final character embedding is the average of the *unique* character n-gram embeddings of w_t. For example, the character n-grams ($n = 1, 2, 3$) of the word "Cat" are {C, a, t, #B#C, Ca, at, t#E#, #B#Ca, Cat, at#E#}, where "#B#" and "#E#" represent the beginning and the end of each word, respectively. Using the character embeddings efficiently provides morphological features. Each word is subsequently represented as x_t, the concatenation of its corresponding word and character embeddings shared across the tasks.[1]

5.1.2.2 Word-Level Task: POS Tagging

The first layer of the model is a bi-directional LSTM [Graves and Schmidhuber, 2005; Hochreiter and Schmidhuber, 1997] whose hidden states are used to predict POS tags. We use the following Long Short-Term Memory (LSTM) units for the forward direction:

$$\begin{aligned} i_t &= \sigma\left(W_i g_t + b_i\right),\ f_t = \sigma\left(W_f g_t + b_f\right), \\ u_t &= \tanh\left(W_u g_t + b_u\right), \\ c_t &= i_t \odot u_t + f_t \odot c_{t-1}, \\ o_t &= \sigma\left(W_o g_t + b_o\right),\ h_t = o_t \odot \tanh\left(c_t\right), \end{aligned} \tag{5.1}$$

where we define the input g_t as $g_t = [\overrightarrow{h}_{t-1}; x_t]$, i.e. the concatenation of the previous hidden state and the word representation of w_t. The backward pass is expanded in the same way, but a different set of weights are used.

For predicting the POS tag of w_t, we use the concatenation of the forward and backward states in a one-layer bi-LSTM layer corresponding to the t-th word: $h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t]$. Then each h_t ($1 \leq t \leq L$) is fed into a standard $\mathrm{softmax}$ classifier with a single ReLU layer which outputs the probability vector $y^{(1)}$ for each of the POS tags.

[1] Bojanowski et al. (2017) previously proposed to train the character n-gram embeddings by the Skip-gram objective.

5.1.2.3 Word-Level Task: Chunking

Chunking is also a word-level classification task which assigns a chunking tag (`B-NP`, `I-VP`, etc.) for each word. The tag specifies the region of major phrases (e.g., noun phrases) in the sentence.

Chunking is performed in the second bi-LSTM layer on top of the POS layer. When stacking the bi-LSTM layers, we use Eq. (5.1) with input $g_t^{(2)} = [h_{t-1}^{(2)}; h_t^{(1)}; x_t; y_t^{(pos)}]$, where $h_t^{(1)}$ is the hidden state of the first (POS) layer. We define the weighted label embedding $y_t^{(pos)}$ as follows:

$$y_t^{(pos)} = \sum_{j=1}^{C} p(y_t^{(1)} = j | h_t^{(1)}) \ell(j), \tag{5.2}$$

where C is the number of the POS tags, $p(y_t^{(1)} = j | h_t^{(1)})$ is the probability value that the j-th POS tag is assigned to w_t, and $\ell(j)$ is the corresponding label embedding. The probability values are predicted by the POS layer, and thus no gold POS tags are needed. This output embedding is similar to the K-best POS tag feature which has been shown to be effective in syntactic tasks [Andor et al., 2016; Alberti et al., 2015]. For predicting the chunking tags, we employ the same strategy as POS tagging by using the concatenated bi-directional hidden states $h_t^{(2)} = [\overrightarrow{h}_t^{(2)}; \overleftarrow{h}_t^{(2)}]$ in the chunking layer. We also use a single ReLU hidden layer before the softmax classifier.

5.1.2.4 Syntactic Task: Dependency Parsing

Dependency parsing identifies syntactic relations (such as an adjective modifying a noun) between word pairs in a sentence. We use the third bi-LSTM layer to classify relations between all pairs of words. The input vector for the LSTM includes hidden states, word representations, and the label embeddings for the two previous tasks: $g_t^{(3)} = [h_{t-1}^{(3)}; h_t^{(2)}; x_t; (y_t^{(pos)} + y_t^{(chk)})]$, where we computed the chunking vector in a similar fashion as the POS vector in Eq. (5.2).

We predict the parent node (*head*) for each word. Then a dependency label is predicted for each child-parent pair. This approach is related to Dozat and Manning (2017) and Zhang et al. (2017), where the main difference is that our model works on a multi-task framework. To predict the parent node of w_t, we define a matching function between w_t and the candidates of the parent node as $m(t, j) = h_t^{(3)} \cdot (W_d h_j^{(3)})$, where W_d is a parameter matrix. For the root, we define $h_{L+1}^{(3)} = r$ as a parameterized vector. To compute the probability that w_j (or the root node) is the parent of w_t, the

scores are normalized:

$$p(j|h_t^{(3)}) = \frac{\exp\left(m\left(t,j\right)\right)}{\sum_{k=1,k\neq t}^{L+1}\exp\left(m\left(t,k\right)\right)}. \tag{5.3}$$

The dependency labels are predicted using $[h_t^{(3)}; h_j^{(3)}]$ as input to a $\mathrm{softmax}$ classifier with a single ReLU layer. We greedily select the parent node and the dependency label for each word. When the parsing result is not a well-formed tree, we apply the first-order Eisner's algorithm [Eisner, 1996] to obtain a well-formed tree from it.

5.1.2.5 Semantic Task: Semantic relatedness

The next two tasks model the semantic relationships between two input sentences. The first task measures the semantic relatedness between two sentences. The output is a real-valued relatedness score for the input sentence pair. The second task is textual entailment, which requires one to determine whether a premise sentence entails a hypothesis sentence. There are typically three classes: entailment, contradiction, and neutral. We use the fourth and fifth bi-LSTM layer for the relatedness and entailment task, respectively.

Now it is required to obtain the sentence-level representation rather than the word-level representation $h_t^{(4)}$ used in the first three tasks. We compute the sentence-level representation $h_{\mathbf{s}}^{(4)}$ as the element-wise maximum values across all of the word-level representations in the fourth layer:

$$h_{\mathbf{s}}^{(4)} = \max\left(h_1^{(4)}, h_2^{(4)}, \ldots, h_L^{(4)}\right). \tag{5.4}$$

This max-pooling technique has proven effective in text classification tasks [Lai et al., 2015].

To model the semantic relatedness between s and s', we follow Tai et al. (2015). The feature vector for representing the semantic relatedness is computed as follows:

$$d_1(s,s') = \left[\left|h_{\mathbf{s}}^{(4)} - h_{\mathbf{s}'}^{(4)}\right|; h_{\mathbf{s}}^{(4)} \odot h_{\mathbf{s}'}^{(4)}\right], \tag{5.5}$$

where $\left|h_{\mathbf{s}}^{(4)} - h_{\mathbf{s}'}^{(4)}\right|$ is the absolute values of the element-wise subtraction, and $h_{\mathbf{s}}^{(4)} \odot h_{\mathbf{s}'}^{(4)}$ is the element-wise multiplication. Then $d_1(s,s')$ is fed into a $\mathrm{softmax}$ classifier with a single Maxout hidden layer [Goodfellow et al., 2013] to output a relatedness score (from 1 to 5 in our case).

5.1.2.6 Semantic Task: Textual entailment

For entailment classification, we also use the max-pooling technique as in the semantic relatedness task. To classify the premise-hypothesis pair (s, s') into one of the three classes, we compute the feature vector $d_2(s, s')$ as in Eq. (5.5) except that we do not use the absolute values of the element-wise subtraction, because we need to identify which is the premise (or hypothesis). Then $d_2(s, s')$ is fed into a $\mathrm{softmax}$ classifier.

To use the output from the relatedness layer directly, we use the label embeddings for the relatedness task. More concretely, we compute the class label embeddings for the semantic relatedness task similar to Eq. (5.2). The final feature vectors that are concatenated and fed into the entailment classifier are the weighted relatedness label embedding and the feature vector $d_2(s, s')$. We use three Maxout hidden layers before the classifier.

5.1.3 Training the JMT Model

The model is trained jointly over all datasets. During each epoch, the optimization iterates over each full training dataset in the same order as the corresponding tasks described in the modeling section.

5.1.3.1 Pre-Training Word Representations

We pre-train word embeddings using the Skip-gram model with negative sampling [Mikolov et al., 2013b]. We also pre-train the character n-gram embeddings using Skip-gram.[2] The only difference is that each input word embedding is replaced with its corresponding average character n-gram embedding. These embeddings are fine-tuned during the model training. We denote the embedding parameters as θ_e. More details about this pre-training method are described as follows:

Pre-training details We pre-train the character n-gram embeddings using the objective function of the Skip-gram model with negative sampling [Mikolov et al., 2013b]. We build the vocabulary of the character n-grams based on the training corpus, the case-sensitive English Wikipedia text. This is because such case-sensitive information is important in handling some types of words like named entities. Assuming that a word w has its corresponding K character n-grams $\{cn_1, cn_2, \ldots, cn_K\}$, where

[2]The training code and the pre-trained embeddings are available at `https://github.com/hassyGo/charNgram2vec`.

any overlaps and unknown ones are removed. Then the word w is represented with an embedding $v_c(w)$ computed as follows:

$$v_c(w) = \frac{1}{K}\sum_{i=1}^{K} v(cn_i), \tag{5.6}$$

where $v(cn_i)$ is the parameterized embedding of the character n-gram cn_i, and the computation of $v_c(w)$ is exactly the same as the one used in our JMT model explained in Section 2.1.

The remaining part of the pre-training process is the same as the original Skip-gram model. For each word-context pair $(w, \overline{w})$ in the training corpus, N negative context words are sampled, and the objective function is defined as follows:

$$\sum_{(w,\overline{w})} \left(-\log \sigma(v_c(w) \cdot \tilde{v}(\overline{w})) - \sum_{i=1}^{N} \log \sigma(-v_c(w) \cdot \tilde{v}(\overline{w}_i)) \right), \tag{5.7}$$

where $\sigma(\cdot)$ is the logistic sigmoid function, $\tilde{v}(\overline{w})$ is the weight vector for the context word $\overline{w}$, and $\overline{w}_i$ is a negative sample. It should be noted that the weight vectors for the context words are parameterized for the words without any character information.

5.1.3.2 Training the POS Layer

Let $\theta_{\mathrm{POS}} = (W_{\mathrm{POS}}, b_{\mathrm{POS}}, \theta_e)$ denote the set of model parameters associated with the POS layer, where W_{POS} is the set of the weight matrices in the first bi-LSTM and the classifier, and b_{POS} is the set of the bias vectors. The objective function to optimize θ_{POS} is defined as follows:

$$\begin{aligned} J_1(\theta_{\mathrm{POS}}) = &- \sum_s \sum_t \log p(y_t^{(1)} = \alpha | h_t^{(1)}) \\ &+ \lambda \|W_{\mathrm{POS}}\|^2 + \delta \|\theta_e - \theta'_e\|^2, \end{aligned} \tag{5.8}$$

where $p(y_t^{(1)} = \alpha_{w_t} | h_t^{(1)})$ is the probability value that the correct label α is assigned to w_t in the sentence s, $\lambda \|W_{\mathrm{POS}}\|^2$ is the L2-norm regularization term, and λ is a hyperparameter.

We call the second regularization term $\delta \|\theta_e - \theta'_e\|^2$ a *successive* regularization term. The successive regularization is based on the idea that we do not want the model to forget the information learned for the other tasks. In the case of POS tagging, the regularization is applied to θ_e, and θ'_e is the embedding parameter after training the final task in the top-most layer at the previous training epoch. δ is a hyperparameter.

5.1.3.3 Training the Chunking Layer

The objective function is defined as follows:

$$J_2(\theta_{\text{chk}}) = -\sum_s \sum_t \log p(y_t^{(2)} = \alpha | h_t^{(2)}) + \lambda \|W_{\text{chk}}\|^2 + \delta \|\theta_{\text{POS}} - \theta'_{\text{POS}}\|^2, \tag{5.9}$$

which is similar to that of POS tagging, and θ_{chk} is $(W_{\text{chk}}, b_{\text{chk}}, E_{\text{POS}}, \theta_e)$, where W_{chk} and b_{chk} are the weight and bias parameters including those in θ_{POS}, and E_{POS} is the set of the POS label embeddings. θ'_{POS} is the one after training the POS layer at the current training epoch.

5.1.3.4 Training the Dependency Layer

The objective function is defined as follows:

$$J_3(\theta_{\text{dep}}) = -\sum_s \sum_t \log p(\alpha | h_t^{(3)}) p(\beta | h_t^{(3)}, h_\alpha^{(3)}) + \lambda(\|W_{\text{dep}}\|^2 + \|W_d\|^2) + \delta \|\theta_{\text{chk}} - \theta'_{\text{chk}}\|^2, \tag{5.10}$$

where $p(\alpha | h_t^{(3)})$ is the probability value assigned to the correct parent node α for w_t, and $p(\beta | h_t^{(3)}, h_\alpha^{(3)})$ is the probability value assigned to the correct dependency label β for the child-parent pair (w_t, α). θ_{dep} is defined as $(W_{\text{dep}}, b_{\text{dep}}, W_d, r, E_{\text{POS}}, E_{\text{chk}}, \theta_e)$, where W_{dep} and b_{dep} are the weight and bias parameters including those in θ_{chk}, and E_{chk} is the set of the chunking label embeddings.

5.1.3.5 Training the Relatedness Layer

Following Tai et al. (2015), the objective function is defined as follows:

$$J_4(\theta_{\text{rel}}) = \sum_{(s,s')} \text{KL}\left(\hat{p}(s,s') \middle\| p(h_s^{(4)}, h_{s'}^{(4)})\right) + \lambda \|W_{\text{rel}}\|^2 + \delta \|\theta_{\text{dep}} - \theta'_{\text{dep}}\|^2, \tag{5.11}$$

where $\hat{p}(s,s')$ is the gold distribution over the defined relatedness scores, $p(h_s^{(4)}, h_{s'}^{(4)})$ is the predicted distribution given the the sentence representations, and $\text{KL}\left(\hat{p}(s,s') \middle\| p(h_s^{(4)}, h_{s'}^{(4)})\right)$ is the KL-divergence between the two distributions. θ_{rel} is defined as $(W_{\text{rel}}, b_{\text{rel}}, E_{\text{POS}}, E_{\text{chk}}, \theta_e)$.

5.1.3.6 Training the Entailment Layer

The objective function is defined as follows:

$$J_5(\theta_{\text{ent}}) = -\sum_{(s,s')} \log p(y^{(5)}_{(s,s')} = \alpha | h^{(5)}_s, h^{(5)}_{s'}) + \lambda \|W_{\text{ent}}\|^2 + \delta \|\theta_{\text{rel}} - \theta'_{\text{rel}}\|^2, \tag{5.12}$$

where $p(y^{(5)}_{(s,s')} = \alpha | h^{(5)}_s, h^{(5)}_{s'})$ is the probability value that the correct label α is assigned to the premise-hypothesis pair (s, s'). θ_{ent} is defined as $(W_{\text{ent}}, b_{\text{ent}}, E_{\text{POS}}, E_{\text{chk}}, E_{\text{rel}}, \theta_e)$, where E_{rel} is the set of the relatedness label embeddings.

5.1.4 Related Work

Many deep learning approaches have proven to be effective in a variety of NLP tasks and are becoming more and more complex. They are typically designed to handle single tasks, or some of them are designed as general-purpose models [Kumar et al., 2016; Sutskever et al., 2014] but applied to different tasks independently.

For handling multiple NLP tasks, multi-task learning models with deep neural networks have been proposed [Collobert et al., 2011; Luong et al., 2016], and more recently Søgaard and Goldberg (2016) have suggested that using different layers for different tasks is more effective than using the same layer in jointly learning closely-related tasks, such as POS tagging and chunking. However, the number of tasks was limited or they have very similar task settings like word-level tagging, and it was not clear how lower-level tasks could be also improved by combining higher-level tasks.

More related to our work, Godwin et al. (2016) also followed Søgaard and Goldberg (2016) to jointly learn POS tagging, chunking, and language modeling, and Zhang and Weiss (2016) have shown that it is effective to jointly learn POS tagging and dependency parsing by sharing internal representations. In the field of relation extraction, Miwa and Bansal (2016) proposed a joint learning model for entity detection and relation extraction. All of them suggest the importance of multi-task learning, and we investigate the potential of handling different types of NLP tasks rather than closely-related ones in a single hierarchical deep model.

In the field of computer vision, some transfer and multi-task learning approaches have also been proposed [Li and Hoiem, 2016; Misra et al., 2016]. For example, Misra et al. (2016) proposed a

multi-task learning model to handle different tasks. However, they assume that each data sample has annotations for the different tasks, and do not explicitly consider task hierarchies.

Recently, Rusu et al. (2016) have proposed a progressive neural network model to handle multiple reinforcement learning tasks, such as Atari games. Like our JMT model, their model is also successively trained according to different tasks using different layers called columns in their paper. In their model, once the first task is completed, the model parameters for the first task are fixed, and then the second task is handled with new model parameters. Therefore, accuracy of the previously trained tasks is never improved. In NLP tasks, multi-task learning has the potential to improve not only higher-level tasks, but also lower-level tasks. Rather than fixing the pre-trained model parameters, our successive regularization allows our model to continuously train the lower-level tasks without significant accuracy drops.

5.1.5 Experimental Settings

5.1.5.1 Datasets

POS tagging: To train the POS tagging layer, we used the Wall Street Journal (WSJ) portion of Penn Treebank, and followed the standard split for the training (Section 0-18), development (Section 19-21), and test (Section 22-24) sets. The evaluation metric is the word-level accuracy.

Chunking: For chunking, we also used the WSJ corpus, and followed the standard split for the training (Section 15-18) and test (Section 20) sets as in the CoNLL 2000 shared task. We used Section 19 as the development set and employed the IOBES tagging scheme. The evaluation metric is the F1 score defined in the shared task.

Dependency parsing: We also used the WSJ corpus for dependency parsing, and followed the standard split for the training (Section 2-21), development (Section 22), and test (Section 23) sets. We obtained Stanford style dependencies using the version 3.3.0 of the Stanford converter. The evaluation metrics are the Unlabeled Attachment Score (UAS) and the Labeled Attachment Score (LAS), and punctuations are excluded for the evaluation.

Semantic relatedness: For the semantic relatedness task, we used the SICK dataset [Marelli et al., 2014], and followed the standard split for the training, development, and test sets. The evaluation metric is the Mean Squared Error (MSE) between the gold and predicted scores.

Textual entailment: For textual entailment, we also used the SICK dataset and exactly the same data split as the semantic relatedness dataset. The evaluation metric is the accuracy.

5.1.5.2 Training Details

We set the dimensionality of the embeddings and the hidden states in the bi-LSTMs to 100. At each training epoch, we trained our model in the order of POS tagging, chunking, dependency parsing, semantic relatedness, and textual entailment. We used mini-batch stochastic gradient decent and empirically found it effective to use a gradient clipping method with growing clipping values for the different tasks; concretely, we employed the simple function: $\min(3.0, depth)$, where $depth$ is the number of bi-LSTM layers involved in each task, and 3.0 is the maximum value. We applied our successive regularization to our model, along with L2-norm regularization and dropout [Srivastava et al., 2014]. More details are summarized as follows:

Pre-training embeddings We used the `word2vec` toolkit to pre-train the word embeddings. We created our training corpus by selecting lowercased English Wikipedia text and obtained 100-dimensional Skip-gram word embeddings trained with the context window size 1, the negative sampling method (15 negative samples), and the sub-sampling method (10^{-5} of the sub-sampling coefficient). We also pre-trained the character n-gram embeddings using the same parameter settings with the case-sensitive Wikipedia text. We trained the character n-gram embeddings for $n = 1, 2, 3, 4$ in the pre-training step.

Embedding initialization We used the pre-trained word embeddings to initialize the word embeddings, and the word vocabulary was built based on the training data of the five tasks. All words in the training data were included in the word vocabulary, and we employed the *word-dropout* method [Kiperwasser and Goldberg, 2016] to train the word embedding for the unknown words. We also built the character n-gram vocabulary for $n = 2, 3, 4$, following Wieting et al. (2016), and the character n-gram embeddings were initialized with the pre-trained embeddings. All of the label embeddings were initialized with uniform random values in $[-\sqrt{6/(dim + C)}, \sqrt{6/(dim + C)}]$, where $dim = 100$ is the dimensionality of the label embeddings and C is the number of labels.

Weight initialization The dimensionality of the hidden layers in the bi-LSTMs was set to 100. We initialized all of the softmax parameters and bias vectors, except for the forget biases in the LSTMs, with zeros, and the weight matrix W_d and the root node vector r for dependency parsing were also initialized with zeros. All of the forget biases were initialized with ones. The other weight matrices were initialized with uniform random values in $[-\sqrt{6/(row + col)}, \sqrt{6/(row + col)}]$, where row

and col are the number of rows and columns of the matrices, respectively.

Optimization At each epoch, we trained our model in the order of POS tagging, chunking, dependency parsing, semantic relatedness, and textual entailment. We used mini-batch stochastic gradient decent to train our model. The mini-batch size was set to 25 for POS tagging, chunking, and the SICK tasks, and 15 for dependency parsing. We used a gradient clipping strategy with growing clipping values for the different tasks; concretely, we employed the simple function: $\min(3.0, depth)$, where $depth$ is the number of bi-LSTM layers involved in each task, and 3.0 is the maximum value. The learning rate at the k-th epoch was set to $\frac{\varepsilon}{1.0+\rho(k-1)}$, where ε is the initial learning rate, and ρ is the hyperparameter to decrease the learning rate. We set ε to 1.0 and ρ to 0.3. At each epoch, the same learning rate was shared across all of the tasks.

Regularization We set the regularization coefficient to 10^{-6} for the LSTM weight matrices, 10^{-5} for the weight matrices in the classifiers, and 10^{-3} for the successive regularization term excluding the classifier parameters of the lower-level tasks, respectively. The successive regularization coefficient for the classifier parameters was set to 10^{-2}. We also used *dropout* [Hinton et al., 2012b]. The dropout rate was set to 0.2 for the vertical connections in the multi-layer bi-LSTMs [Pham et al., 2014], the word representations and the label embeddings of the entailment layer, and the classifier of the POS tagging, chunking, dependency parsing, and entailment. A different dropout rate of 0.4 was used for the word representations and the label embeddings of the POS, chunking, and dependency layers, and the classifier of the relatedness layer.

5.1.6 Results and Discussion

Table 5.1 shows our results on the test sets of the five tasks.[3] The column "Single" shows the results of handling each task separately using single-layer bi-LSTMs, and the column "JMT_{all}" shows the results of our JMT model. The single task settings only use the annotations of their own tasks. For example, when handling dependency parsing as a single task, the POS and chunking tags are *not* used. We can see that all results of the five tasks are improved in our JMT model, which shows that our JMT model can handle the five different tasks in a single model. Our JMT model allows us to

[3] In chunking evaluation, we only show the results of "Single" and "JMT_{AB}" because the sentences for chunking evaluation overlap the training data for dependency parsing.

		Single	JMT_{all}	JMT_{AB}	JMT_{ABC}	JMT_{DE}	JMT_{CD}	JMT_{CE}
A ↑	POS	97.45	97.55	97.52	97.54	n/a	n/a	n/a
B ↑	Chunking	95.02	n/a	95.77	n/a	n/a	n/a	n/a
C ↑	Dependency UAS	93.35	94.67	n/a	94.71	n/a	93.53	93.57
	Dependency LAS	91.42	92.90	n/a	92.92	n/a	91.62	91.69
D ↓	Relatedness	0.247	0.233	n/a	n/a	0.238	0.251	n/a
E ↑	Entailment	81.8	86.2	n/a	n/a	86.8	n/a	82.4

Table 5.1: Test set results for the five tasks. In the relatedness task, the lower scores are better.

access arbitrary information learned from the different tasks. If we want to use the model just as a POS tagger, we can use only first bi-LSTM layer.

Table 5.1 also shows the results of five subsets of the different tasks. For example, in the case of "JMT_{ABC}", only the first three layers of the bi-LSTMs are used to handle the three tasks. In the case of "JMT_{DE}", only the top two layers are used as a two-layer bi-LSTM by omitting all information from the first three layers. The results of the closely-related tasks ("AB", "ABC", and "DE") show that our JMT model improves both of the high-level and low-level tasks. The results of "JMT_{CD}" and "JMT_{CE}" show that the parsing task can be improved by the semantic tasks.

It should be noted that in our analysis on the greedy parsing results of the "JMT_{ABC}" setting, we have found that more than 95% are well-formed dependency trees on the development set. In the 1,700 sentences of the development data, 11 results have multiple root notes, 11 results have no root nodes, and 61 results have cycles. These 83 parsing results are converted into well-formed trees by Eisner's algorithm, and the accuracy does not significantly change (UAS: 94.52%→94.53%, LAS: 92.61%→92.62%).

5.1.6.1 Comparison with Published Results

POS tagging Table 5.2 shows the results of POS tagging, and our JMT model achieves the score close to the state-of-the-art results. The best result to date has been achieved by Ling et al. (2015), which uses character-based LSTMs. Incorporating the character-based encoders into our JMT model would be an interesting direction, but we have shown that the simple pre-trained character n-gram embeddings lead to the promising result.

Method	Acc. ↑
JMT_{all}	97.55
Ling et al. (2015)	**97.78**
Kumar et al. (2016)	97.56
Ma and Hovy (2016)	97.55
Søgaard (2011)	97.50
Collobert et al. (2011)	97.29
Tsuruoka et al. (2011)	97.28
Toutanova et al. (2003)	97.27

Table 5.2: POS tagging results.

Method	F1 ↑
JMT_{AB}	**95.77**
Single	95.02
Søgaard and Goldberg (2016)	95.56
Suzuki and Isozaki (2008)	95.15
Collobert et al. (2011)	94.32
Kudo and Matsumoto (2001)	93.91
Tsuruoka et al. (2011)	93.81

Table 5.3: Chunking results.

Method	UAS ↑	LAS ↑
JMT_{all}	94.67	92.90
Single	93.35	91.42
Dozat and Manning (2017)	**95.74**	**94.08**
Andor et al. (2016)	94.61	92.79
Alberti et al. (2015)	94.23	92.36
Zhang et al. (2017)	94.10	91.90
Weiss et al. (2015)	93.99	92.05
Dyer et al. (2015)	93.10	90.90
Bohnet (2010)	92.88	90.71

Table 5.4: Dependency results.

Method	MSE ↓
JMT_{all}	**0.233**
JMT_{DE}	0.238
Zhou et al. (2016)	0.243
Tai et al. (2015)	0.253

Table 5.5: Semantic relatedness results.

Method	Acc. ↑
JMT_{all}	86.2
JMT_{DE}	**86.8**
Yin et al. (2016)	86.2
Lai and Hockenmaier (2014)	84.6

Table 5.6: Textual entailment results.

Chunking Table 5.3 shows the results of chunking, and our JMT model achieves the state-of-the-art result. Søgaard and Goldberg (2016) proposed to jointly learn POS tagging and chunking in different layers, but they only showed improvement for chunking. By contrast, our results show that the low-level tasks are also improved.

Dependency parsing Table 5.4 shows the results of dependency parsing by using only the WSJ corpus in terms of the dependency annotations.[4] It is notable that our simple greedy dependency parser outperforms the model in Andor et al. (2016) which is based on beam search with global information. The result suggests that the bi-LSTMs efficiently capture global information necessary for dependency parsing. Moreover, our single task result already achieves high accuracy without the POS and

[4]Choe and Charniak (2016) employed a tri-training method to expand the training data with 400,000 trees in addition to the WSJ data, and they reported 95.9 UAS and 94.1 LAS by converting their constituency trees into dependency trees. Kuncoro et al. (2017) also reported high accuracy (95.8 UAS and 94.6 LAS) by using a converter.

chunking information. The best result to date has been achieved by the model propsoed in Dozat and Manning (2017), which uses higher dimensional representations than ours and proposes a more sophisticated attention mechanism called *biaffine attention*. It should be promising to incorporate their attention mechanism into our parsing component.

Semantic relatedness Table 5.5 shows the results of the semantic relatedness task, and our JMT model achieves the state-of-the-art result. The result of "JMT_{DE}" is already better than the previous state-of-the-art results. Both of Zhou et al. (2016) and Tai et al. (2015) explicitly used syntactic trees, and Zhou et al. (2016) relied on attention mechanisms. However, our method uses the simple max-pooling strategy, which suggests that it is worth investigating such simple methods before developing complex methods for simple tasks. Currently, our JMT model does not explicitly use the learned dependency structures, and thus the explicit use of the output from the dependency layer should be an interesting direction of future work.

Textual entailment Table 5.6 shows the results of textual entailment, and our JMT model achieves the state-of-the-art result. The previous state-of-the-art result in Yin et al. (2016) relied on attention mechanisms and dataset-specific data pre-processing and features. Again, our simple max-pooling strategy achieves the state-of-the-art result boosted by the joint training. These results show the importance of jointly handling related tasks.

5.1.6.2 Analysis on the Model Architectures

We investigate the effectiveness of our model in detail. All of the results shown in this section are the development set results.

Shortcut connections Our JMT model feeds the word representations into all of the bi-LSTM layers, which is called the shortcut connection. Table 5.7 shows the results of "JMT_{all}" with and without the shortcut connections. The results without the shortcut connections are shown in the column of "w/o SC". These results clearly show that the importance of the shortcut connections, and in particular, the semantic tasks in the higher layers strongly rely on the shortcut connections. That is, simply stacking the LSTM layers is not sufficient to handle a variety of NLP tasks in a single model. In the supplementary material, it is qualitatively shown how the shortcut connections work in our model.

	JMT_{all}	w/o SC	w/o LE	w/o SC&LE
POS	97.88	97.79	97.85	97.87
Chunking	97.59	97.08	97.40	97.33
Dependency UAS	94.51	94.52	94.09	94.04
Dependency LAS	92.60	92.62	92.14	92.03
Relatedness	0.236	0.698	0.261	0.765
Entailment	84.6	75.0	81.6	71.2

Table 5.7: Effectiveness of the Shortcut Connections (SC) and the Label Embeddings (LE).

	JMT_{ABC}	w/o SC&LE	All-3
POS	97.90	97.87	97.62
Chunking	97.80	97.41	96.52
Dependency UAS	94.52	94.13	93.59
Dependency LAS	92.61	92.16	91.47

Table 5.8: Effectiveness of using different layers for different tasks.

Output label embeddings Table 5.7 also shows the results without using the output labels of the POS, chunking, and relatedness layers, in the column of "w/o LE". These results show that the explicit use of the output information from the classifiers of the lower layers is important in our JMT model. The results in the column of "w/o SC&LE" are the ones without both of the shortcut connections and the label embeddings.

Different layers for different tasks Table 5.8 shows the results of our "JMT_{ABC}" setting and that of not using the shortcut connections and the label embeddings ("w/o SC&LE") as in Table 5.7. In addition, in the column of "All-3", we show the results of using the highest (i.e., the third) layer for all of the three tasks without any shortcut connections and label embeddings, and thus the two settings "w/o SC&LE" and "All-3" require exactly the same number of the model parameters. The "All-3" setting is similar to the multi-task model of Collobert et al. (2011) in that task-specific output layers are used but most of the model parameters are shared. The results show that using the same layers for

	JMT_{all}	w/o SR	w/o VC
POS	97.88	97.85	97.82
Chunking	97.59	97.13	97.45
Dependency UAS	94.51	94.46	94.38
Dependency LAS	92.60	92.57	92.48
Relatedness	0.236	0.239	0.241
Entailment	84.6	84.2	84.8

Table 5.9: Effectiveness of the Successive Regularization (SR) and the Vertical Connections (VC).

	JMT_{all}	Random
POS	97.88	97.83
Chunking	97.59	97.71
Dependency UAS	94.51	94.66
Dependency LAS	92.60	92.80
Relatedness	0.236	0.298
Entailment	84.6	83.2

Table 5.10: Effects of the order of training.

the three different tasks hampers the effectiveness of our JMT model, and the design of the model is much more important than the number of the model parameters.

Successive regularization In Table 5.9, the column of "w/o SR" shows the results of omitting the successive regularization terms described in Section 5.1.3. We can see that the accuracy of chunking is improved by the successive regularization, while other results are not affected so much. The chunking dataset used here is relatively small compared with other low-level tasks, POS tagging and dependency parsing. Thus, these results suggest that the successive regularization is effective when dataset sizes are imbalanced.

Vertical connections We investigated our JMT results without using the vertical connections in the five-layer bi-LSTMs. More concretely, when constructing the input vectors g_t, we do not use the bi-

	Single	Single+
POS	97.52	
Chunking	95.65	96.08
Dependency UAS	93.38	93.88
Dependency LAS	91.37	91.83
Relatedness	0.239	0.665
Entailment	83.8	66.4

Table 5.11: Effects of depth for the *single* tasks.

Single	W&C	Only W
POS	97.52	96.26
Chunking	95.65	94.92
Dependency UAS	93.38	92.90
Dependency LAS	91.37	90.44

Table 5.12: Effects of the character embeddings.

LSTM hidden states of the previous layers. Table 5.9 also shows the JMT_{all} results with and without the vertical connections. As shown in the column of "w/o VC", we observed the competitive results. Therefore, in the target tasks used in our model, sharing the word representations and the output label embeddings is more effective than just stacking the bi-LSTM layers.

Order of training Our JMT model iterates the training process in the order described in Section 5.1.3. Our hypothesis is that it is important to start from the lower-level tasks and gradually move to the higher-level tasks. Table 5.10 shows the results of training our model by randomly shuffling the order of the tasks for each epoch in the column of "Random". We see that the scores of the semantic tasks drop by the random strategy. In our preliminary experiments, we have found that constructing the mini-batch samples from different tasks also hampers the effectiveness of our model, which also supports our hypothesis.

Single (POS)	Overall Acc.	Acc. for unknown words
W&C	97.52	90.68 (3,502/3,862)
Only W	96.26	71.44 (2,759/3,862)

Table 5.13: POS tagging scores on the development set with and without the character n-gram embeddings, focusing on accuracy for unknown words. The overall accuracy scores are taken from Table 12. There are 3,862 unknown words in the sentences of the development set.

Depth The single task settings shown in Table 5.1 are obtained by using single layer bi-LSTMs, but in our JMT model, the higher-level tasks use successively deeper layers. To investigate the gap between the different number of the layers for each task, we also show the results of using multi-layer bi-LSTMs for the single task settings, in the column of "Single+" in Table 5.11. More concretely, we use the same number of the layers with our JMT model; for example, three layers are used for dependency parsing, and five layers are used for textual entailment. As shown in these results, deeper layers do not always lead to better results, and the joint learning is more important than making the models complex only for single tasks.

Character n-gram embeddings Finally, Table 5.12 shows the results for the three single tasks with and without the pre-trained character n-gram embeddings. The column of "W&C" corresponds to using both of the word and character n-gram embeddings, and that of "Only W" corresponds to using only the word embeddings. These results clearly show that jointly using the pre-trained word and character n-gram embeddings is helpful in improving the results. The pre-training of the character n-gram embeddings is also effective; for example, without the pre-training, the POS accuracy drops from 97.52% to 97.38% and the chunking accuracy drops from 95.65% to 95.14%.

5.1.6.3 Better Handling Unknown Words with Character N-gram Embeddings

One expectation from the use of the character n-gram embeddings is to better handle unknown words. We verified this assumption in the single task setting for POS tagging, based on the results reported in Table 5.12. Table 5.13 shows that the joint use of the word and character n-gram embeddings improves the score by about 19% in terms of the accuracy for unknown words.

We also show the results of the single task setting for dependency parsing in Table 5.14. Again,

	Overall scores		Scores for unknown words	
Single (Dependency)	UAS	LAS	UAS	LAS
W&C	93.38	91.37	92.21 (900/976)	87.81 (857/976)
Only W	92.90	90.44	91.39 (892/976)	81.05 (791/976)

Table 5.14: Dependency parsing scores on the development set with and without the character n-gram embeddings, focusing on UAS and LAS for unknown words. The overall scores are taken from Table 12. There are 976 unknown words in the sentences of the development set.

we can see that using the character-level information is effective, and in particular, the improvement of the LAS score is large. These results suggest that it is better to use not only the word embeddings, but also the character n-gram embeddings by default. Recently, the joint use of word and character information has proven to be effective in language modeling [Miyamoto and Cho, 2016], but just using the simple character n-gram embeddings is fast and also effective.

5.1.6.4 Analysis on Dependency Parsing

Our dependency parser is based on the idea of predicting a head (or parent) for each word, and thus the parsing results do not always lead to correct trees. To inspect this aspect, we checked the parsing results on the development set (1,700 sentences), using the "$\mathrm{JMT}_{\mathrm{ABC}}$" setting.

In the dependency annotations used in this work, each sentence has only one root node, and we have found 11 sentences with multiple root nodes and 11 sentences with no root nodes in our parsing results. We show two examples below:

(a) Underneath the headline " Diversification , " it **counsels** , " Based on the events of the past week , all investors **need** to know their portfolios are balanced to help protect them against the market 's volatility . "

(b) Mr. Eskandarian , who resigned his Della Femina post in September , becomes chairman and chief executive of Arnold .

In the example (a), the two boldfaced words "counsels" and "need" are predicted as child nodes of the root node, and the underlined word "counsels" is the correct one based on the gold annotations. This

example sentence (a) consists of multiple internal sentences, and our parser misunderstood that both of the two verbs are the heads of the sentence.

In the example (b), none of the words is connected to the root node, and the correct child node of the root is the underlined word "chairman". Without the internal phrase "who resigned... in September", the example sentence (b) is very simple, but we have found that such a simplified sentence is still not parsed correctly. In many cases, verbs are linked to the root nodes, but sometimes other types of words like nouns can be the candidates. In our model, the single parameterized vector r is used to represent the root node for each sentence. Therefore, the results of the examples (a) and (b) suggest that it would be needed to capture various types of root nodes, and using sentence-dependent root representations would lead to better results in future work.

5.1.6.5 Analysis on Semantic Tasks

We inspected the development set results on the semantic tasks using the "JMT_{all}" setting. In our model, the highest-level task is the textual entailment task. We show an example premise-hypothesis pair which is misclassified in our results:

> Premise: "A surfer is riding a *big* wave across dark green water", and
>
> Hypothesis: "The surfer is riding a *small* wave".

The predicted label is `entailment`, but the gold label is `contradiction`. This example is very easy by focusing on the difference between the two words "big" and "small". However, our model fails to correctly classify this example because there are few opportunities to learn the difference. Our model relies on the pre-trained word embeddings based on word co-occurrence statistics [Mikolov et al., 2013b], and it is widely known that such co-occurrence-based embeddings can rarely discriminate between antonyms and synonyms [Ono et al., 2015]. Moreover, the other four tasks in our JMT model do not explicitly provide the opportunities to learn such semantic aspects. Even in the training data of the textual entailment task, we can find only one example to learn the difference between the two words, which is not enough to obtain generalization capacities. Therefore, it is worth investigating the explicit use of external dictionaries or the use of pre-trained word embeddings learned with such dictionaries [Ono et al., 2015], to see whether our JMT model is further improved not only for the semantic tasks, but also for the low-level tasks.

5.1.6.6 How Do Shared Embeddings Change

In our JMT model, the word and character n-gram embedding matrices are shared across all of the five different tasks. To better qualitatively explain the importance of the shortcut connections shown in Table 7, we inspected how the shared embeddings change when fed into the different bi-LSTM layers. More concretely, we checked closest neighbors in terms of the cosine similarity for the word representations before and after fed into the forward LSTM layers. In particular, we used the corresponding part of W_u in Eq. (1) to perform linear transformation of the input embeddings, because u_t directly affects the hidden states of the LSTMs. Thus, this is a context-independent analysis.

Table 5.15 shows the examples of the word "standing". The row of "Embedding" shows the cases of using the shared embeddings, and the others show the results of using the linear-transformed embeddings. In the column of "Only word", the results of using only the word embeddings are shown. The closest neighbors in the case of "Embedding" capture the semantic similarity, but after fed into the POS layer, the semantic similarity is almost washed out. This is not surprising because it is sufficient to cluster the words of the same POS tags: here, `NN`, `VBG`, etc. In the chunking layer, the similarity in terms of verbs is captured, and this is because it is sufficient to identify the coarse chunking tags: here, `VP`. In the dependency layer, the closest neighbors are adverbs, gerunds of verbs, and nouns, and all of them can be child nodes of verbs in dependency trees. However, this information is not sufficient in further classifying the dependency labels. Then we can see that in the column of "Word and char", jointly using the character n-gram embeddings adds the morphological information, and as shown in Table 12, the LAS score is substantially improved.

In the case of semantic tasks, the projected embeddings capture not only syntactic, but also semantic similarities. These results show that different tasks need different aspects of the word similarities, and our JMT model efficiently transforms the shared embeddings for the different tasks by the simple linear transformation. Therefore, without the shortcut connections, the information about the word representations are fed into the semantic tasks after transformed in the lower layers where the semantic similarities are not always important. Indeed, the results of the semantic tasks are very poor without the shortcut connections.

5.1.6.7 Further Discussions

Training strategies In our JMT model, it is not obvious when to stop the training while trying to maximize the scores of all the five tasks. We focused on maximizing the accuracy of dependency

parsing on the development data in our experiments. However, the sizes of the training data are different across the different tasks; for example, the semantic tasks include only 4,500 sentence pairs, and the dependency parsing dataset includes 39,832 sentences with word-level annotations. Thus, in general, dependency parsing requires more training epochs than the semantic tasks, but currently, our model trains all of the tasks for the same training epochs. The same strategy for decreasing the learning rate is also shared across all the different tasks, although our growing gradient clipping method described in Section 5.1.5.2 helps improve the results. Indeed, we observed that better scores of the semantic tasks can be achieved before the accuracy of dependency parsing reaches the best score. Developing a method for achieving the best scores for all of the tasks at the same time is important future work.

More tasks Our JMT model has the potential of handling more tasks than the five tasks used in our experiments; examples include entity detection and relation extraction as in Miwa and Bansal (2016) as well as language modeling [Godwin et al., 2016]. It is also a promising direction to train each task for multiple domains by focusing on domain adaptation [Søgaard and Goldberg, 2016]. In particular, incorporating language modeling tasks provides an opportunity to use large text data. Such large text data was used in our experiments to pre-train the word and character n-gram embeddings. However, it would be preferable to efficiently use it for improving the entire model.

Task-oriented learning of low-level tasks Each task in our JMT model is supervised by its corresponding dataset. However, it would be possible to learn low-level tasks by optimizing high-level tasks, because the model parameters of the low-level tasks can be directly modified by learning the high-level tasks. One example has already been presented in Hashimoto and Tsuruoka (2017), where our JMT model is extended to learning task-oriented latent graph structures of sentences by training our dependency parsing component according to a neural machine translation objective.

5.1.7 Conclusion

We presented a joint many-task model to handle multiple NLP tasks with growing depth in a single end-to-end model. Our model is successively trained by considering linguistic hierarchies, directly feeding word representations into all layers, explicitly using low-level predictions, and applying successive regularization. In experiments on five NLP tasks, our single model achieves the state-of-the-art

or competitive results on chunking, dependency parsing, semantic relatedness, and textual entailment.

	Word and char	Only word
Embedding	leaning	stood
	kneeling	stands
	saluting	sit
	clinging	pillar
	railing	cross-legged
POS	warning	ladder
	waxing	rc6280
	dunking	bethle
	proving	warning
	tipping	f-a-18
Chunking	applauding	fight
	disdaining	favor
	pickin	pick
	readjusting	rejoin
	reclaiming	answer
Dependency	guaranteeing	patiently
	resting	hugging
	grounding	anxiously
	hanging	resting
	hugging	disappointment
Relatedness	stood	stood
	stands	unchallenged
	unchallenged	stands
	notwithstanding	beside
	judging	exists
Entailment	nudging	beside
	skirting	stands
	straddling	pillar
	contesting	swung
	footing	ovation

Table 5.15: Closest neighbors of the word “standing” in the embedding space and the projected space in each forward LSTM.

5.2 Latent Graph Parsing for Machine Translation

Summary This paper presents a novel neural machine translation model which jointly learns translation and source-side latent graph representations of sentences. Unlike existing pipelined approaches using syntactic parsers, our end-to-end model learns a latent graph parser as part of the encoder of an attention-based neural machine translation model, and thus the parser is optimized according to the translation objective. In experiments, we first show that our model compares favorably with state-of-the-art sequential and pipelined syntax-based NMT models. We also show that the performance of our model can be further improved by pre-training it with a small amount of treebank annotations. Our final ensemble model significantly outperforms the previous best models on the standard English-to-Japanese translation dataset.

5.2.1 Introduction

Neural Machine Translation (NMT) is an active area of research due to its outstanding empirical results [Bahdanau et al., 2015; Luong et al., 2015; Sutskever et al., 2014]. Most of the existing NMT models treat each sentence as a sequence of tokens, but recent studies suggest that syntactic information can help improve translation accuracy [Eriguchi et al., 2016b; Eriguchi et al., 2017; Sennrich and Haddow, 2016; Stahlberg et al., 2016]. The existing syntax-based NMT models employ a syntactic parser trained by supervised learning in advance, and hence the parser is not adapted to the translation tasks. An alternative approach for leveraging syntactic structure in a language processing task is to jointly learn syntactic trees of the sentences along with the target task [Socher et al., 2011b; Yogatama et al., 2017].

Motivated by the promising results of recent joint learning approaches, we present a novel NMT model that can learn a task-specific latent graph structure for each source-side sentence. The graph structure is similar to the dependency structure of the sentence, but it can have cycles and is learned specifically for the translation task. Unlike the aforementioned approach of learning single syntactic trees, our latent graphs are composed of "soft" connections, i.e., the edges have real-valued weights (Figure 5.2). Our model consists of two parts: one is a task-independent parsing component, which we call a *latent graph parser*, and the other is an attention-based NMT model. The latent parser can be independently pre-trained with human-annotated treebanks and is then adapted to the translation task.

In experiments, we demonstrate that our model can be effectively pre-trained by the treebank an-

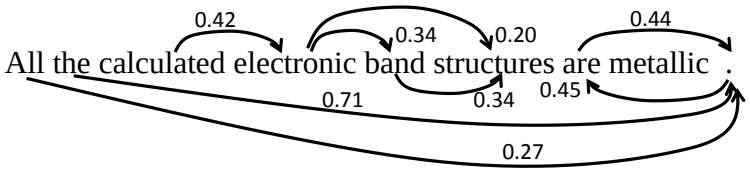


Figure 5.2: An example of the learned latent graphs. Edges with a small weight are omitted.

notations, outperforming a state-of-the-art sequential counterpart and a pipelined syntax-based model. Our final ensemble model outperforms the previous best results by a large margin on the WAT English-to-Japanese dataset.

5.2.2 Latent Graph Parser

We model the latent graph parser based on dependency parsing. In dependency parsing, a sentence is represented as a tree structure where each node corresponds to a word in the sentence and a unique *root* node (ROOT) is added. Given a sentence of length N, the parent node $H_{w_i} \in \{w_1, \ldots, w_N, \mathrm{ROOT}\}$ ($H_{w_i} \neq w_i$) of each word w_i ($1 \leq i \leq N$) is called its *head*. The sentence is thus represented as a set of tuples $(w_i, H_{w_i}, \ell_{w_i})$, where ℓ_{w_i} is a dependency label.

In this paper, we remove the constraint of using the tree structure and represent a sentence as a set of tuples $(w_i, p(H_{w_i}|w_i), p(\ell_{w_i}|w_i))$, where $p(H_{w_i}|w_i)$ is the probability distribution of w_i's parent nodes, and $p(\ell_{w_i}|w_i)$ is the probability distribution of the dependency labels. For example, $p(H_{w_i} = w_j|w_i)$ is the probability that w_j is the parent node of w_i. Here, we assume that a special token $\langle\mathrm{EOS}\rangle$ is appended to the end of the sentence, and we treat the $\langle\mathrm{EOS}\rangle$ token as ROOT. This approach is similar to that of graph-based dependency parsing [McDonald et al., 2005] in that a sentence is represented with a set of weighted arcs between the words. To obtain the *latent graph* representation of the sentence, we use a dependency parsing model based on multi-task learning proposed by Hashimoto et al. (2017).

5.2.2.1 Word Representation

The i-th input word w_i is represented with the concatenation of its d_1-dimensional word embedding $v_{dp}(w_i) \in \mathbb{R}^{d_1}$ and its character n-gram embedding $c(w_i) \in \mathbb{R}^{d_1}$: $x(w_i) = [v_{dp}(w_i); c(w_i)]$. $c(w_i)$ is computed as the average of the embeddings of the character n-grams in w_i.

5.2.2.2 POS Tagging Layer

Our latent graph parser builds upon multi-layer bi-directional Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) units [Hochreiter and Schmidhuber, 1997]. In the first layer, POS tagging is handled by computing a hidden state $h_i^{(1)} = [\overrightarrow{h}_i^{(1)}; \overleftarrow{h}_i^{(1)}] \in \mathbb{R}^{2d_1}$ for w_i, where $\overrightarrow{h}_i^{(1)} = \mathrm{LSTM}(\overrightarrow{h}_{i-1}^{(1)}, x(w_i)) \in \mathbb{R}^{d_1}$ and $\overleftarrow{h}_i^{(1)} = \mathrm{LSTM}(\overleftarrow{h}_{i+1}^{(1)}, x(w_i)) \in \mathbb{R}^{d_1}$ are hidden states of the forward and backward LSTMs, respectively. $h_i^{(1)}$ is then fed into a softmax classifier to predict a probability distribution $p_i^{(1)} \in \mathbb{R}^{C^{(1)}}$ for word-level tags, where $C^{(1)}$ is the number of POS classes. The model parameters of this layer can be learned not only by human-annotated data, but also by backpropagation from higher layers, which are described in the next section.

5.2.2.3 Dependency Parsing Layer

Dependency parsing is performed in the second layer. A hidden state $h_i^{(2)} \in \mathbb{R}^{2d_1}$ is computed by $\overrightarrow{h}_i^{(2)} = \mathrm{LSTM}(\overrightarrow{h}_{i-1}^{(2)}, [x(w_i); y(w_i); \overrightarrow{h}_i^{(1)}])$ and $\overleftarrow{h}_i^{(2)} = \mathrm{LSTM}(\overleftarrow{h}_{i+1}^{(2)}, [x(w_i); y(w_i); \overleftarrow{h}_i^{(1)}])$, where $y(w_i) = W_\ell^{(1)} p_i^{(1)} \in \mathbb{R}^{d_2}$ is the POS information output from the first layer, and $W_\ell^{(1)} \in \mathbb{R}^{d_2 \times C^{(1)}}$ is a weight matrix.

Then, (soft) edges of our latent graph representation are obtained by computing the probabilities

$$p(H_{w_i} = w_j | w_i) = \frac{\exp\left(m(i,j)\right)}{\sum_{k \neq i} \exp\left(m(i,k)\right)}, \tag{5.13}$$

where $m(i,k) = h_k^{(2)\mathrm{T}} W_{dp} h_i^{(2)}$ $(1 \leq k \leq N+1, k \neq i)$ is a scoring function with a weight matrix $W_{dp} \in \mathbb{R}^{2d_1 \times 2d_1}$. While the models of Hashimoto et al. (2017), Zhang et al. (2017), and Dozat and Manning (2017) learn the model parameters of their parsing models only by human-annotated data, we allow the model parameters to be learned by the translation task.

Next, $[h_i^{(2)}; z(H_{w_i})]$ is fed into a softmax classifier to predict the probability distribution $p(\ell_{w_i}|w_i)$, where $z(H_{w_i}) \in \mathbb{R}^{2d_1}$ is the weighted average of the hidden states of the parent nodes: $\sum_{j \neq i} p(H_{w_i} = w_j | w_i) h_j^{(2)}$. This results in the latent graph representation $(w_i, p(H_{w_i}|w_i), p(\ell_{w_i}|w_i))$ of the input sentence.

5.2.3 NMT with Latent Graph Parser

The latent graph representation described in Section 5.2.2 can be used for any sentence-level tasks, and here we apply it to an Attention-based NMT (ANMT) model [Luong et al., 2015]. We modify the

encoder and the decoder in the ANMT model to learn the latent graph representation.

5.2.3.1 Encoder with Dependency Composition

The ANMT model first encodes the information about the input sentence and then generates a sentence in another language. The encoder represents the word w_i with a word embedding $v_{enc}(w_i) \in \mathbb{R}^{d_3}$. It should be noted that $v_{enc}(w_i)$ is different from $v_{dp}(w_i)$ because each component is separately modeled. The encoder then takes the word embedding $v_{enc}(w_i)$ and the hidden state $h_i^{(2)}$ as the input to a uni-directional LSMT:

$$h_i^{(enc)} = \mathrm{LSTM}(h_{i-1}^{(enc)}, [v_{enc}(w_i); h_i^{(2)}]), \tag{5.14}$$

where $h_i^{(enc)} \in \mathbb{R}^{d_3}$ is the hidden state corresponding to w_i. That is, the encoder of our model is a three-layer LSTM network, where the first two layers are bi-directional.

In the sequential LSTMs, relationships between words in distant positions are not *explicitly* considered. In our model, we explicitly incorporate such relationships into the encoder by defining a dependency composition function:

$$dep(w_i) = \tanh(W_{dep}[h_i^{enc}; \overline{h}(H_{w_i}); p(\ell_{w_i}|w_i)]), \tag{5.15}$$

where $\overline{h}(H_{w_i}) = \sum_{j \neq i} p(H_{w_i} = w_j|w_i) h_j^{(enc)}$ is the weighted average of the hidden states of the parent nodes.

Note on character n-gram embeddings In NMT models, sub-word units are widely used to address rare or unknown word problems [Sennrich et al., 2016]. In our model, the character n-gram embeddings are fed through the latent graph parsing component. To the best of our knowledge, the character n-gram embeddings have never been used in NMT models. Wieting et al. (2016), Bojanowski et al. (2017), and Hashimoto et al. (2017) have reported that the character n-gram embeddings are useful in improving several NLP tasks by better handling unknown words.

5.2.3.2 Decoder with Attention Mechanism

The decoder of our model is a single-layer LSTM network, and the initial state is set with $h_{N+1}^{(enc)}$ and its corresponding memory cell. Given the t-th hidden state $h_t^{(dec)} \in \mathbb{R}^{d_3}$, the decoder predicts the t-th

word in the target language using an attention mechanism. The attention mechanism in Luong et al. (2015) computes the weighted average of the hidden states $h_i^{(enc)}$ of the encoder:

$$s(i,t) = \frac{\exp\left(h_t^{(dec)} \cdot h_i^{(enc)}\right)}{\sum_{j=1}^{N+1} \exp\left(h_t^{(dec)} \cdot h_j^{(enc)}\right)}, \tag{5.16}$$

$$a_t = \sum_{i=1}^{N+1} s(i,t) h_i^{(enc)}, \tag{5.17}$$

where $s(i,t)$ is a scoring function which specifies how much each source-side hidden state contributes to the word prediction.

In addition, like the attention mechanism over constituency tree nodes [Eriguchi et al., 2016b], our model uses attention to the dependency composition vectors:

$$s'(i,t) = \frac{\exp\left(h_t^{(dec)} \cdot dep(w_i)\right)}{\sum_{j=1}^{N} \exp\left(h_t^{(dec)} \cdot dep(w_j)\right)}, \tag{5.18}$$

$$a'_t = \sum_{i=1}^{N} s'(i,t) dep(w_i), \tag{5.19}$$

To predict the target word, a hidden state $\tilde{h}_t^{(dec)} \in \mathbb{R}^{d_3}$ is then computed as follows:

$$\tilde{h}_t^{(dec)} = \tanh(\tilde{W}[h_t^{(dec)}; a_t; a'_t]), \tag{5.20}$$

where $\tilde{W} \in \mathbb{R}^{d_3 \times 3d_3}$ is a weight matrix. $\tilde{h}_t^{(dec)}$ is fed into a softmax classifier to predict a target word distribution. $\tilde{h}_t^{(dec)}$ is also used in the transition of the decoder LSTMs along with a word embedding $v_{dec}(w_t) \in \mathbb{R}^{d_3}$ of the target word w_t:

$$h_{t+1}^{(dec)} = \mathrm{LSTM}(h_t^{(dec)}, [v_{dec}(w_t); \tilde{h}_t^{(dec)}]), \tag{5.21}$$

where the use of $\tilde{h}_t^{(dec)}$ is called *input feeding* proposed by Luong et al. (2015).

The overall model parameters, including those of the latent graph parser, are jointly learned by minimizing the negative log-likelihood of the prediction probabilities of the target words in the training data. To speed up the training, we use BlackOut sampling [Ji et al., 2016]. By this joint learning using Equation (5.15) and (5.19), the latent graph representations are automatically learned according to the target task.

Implementation Tips Inspired by Zoph et al. (2016), we further speed up BlackOut sampling by sharing noise samples across words in the same sentences. This technique has proven to be effective

in RNN language modeling, and we have found that it is also effective in the NMT model. We have also found it effective to share the model parameters of the target word embeddings and the softmax weight matrix for word prediction [Inan et al., 2016; Press and Wolf, 2017]. Also, we have found that a parameter averaging technique [Hashimoto et al., 2013] is helpful in improving translation accuracy.

Translation At test time, we use a novel beam search algorithm which combines statistics of sentence lengths [Eriguchi et al., 2016b] and length normalization [Cho et al., 2014a]. During the beam search step, we use the following scoring function for a generated word sequence $y = (y_1, y_2, \ldots, y_{L_y})$ given a source word sequence $x = (x_1, x_2, \ldots, x_{L_x})$:

$$\frac{1}{L_y}\left(\sum_{i=1}^{L_y} \log p(y_i|x, y_{1:i-1}) + \log p(L_y|L_x)\right), \tag{5.22}$$

where $p(L_y|L_x)$ is the probability that sentences of length L_y are generated given source-side sentences of length L_x. The statistics are taken by using the training data in advance. In our experiments, we have empirically found that this beam search algorithm helps the NMT models to avoid generating translation sentences that are too short.

5.2.4 Experimental Settings

5.2.4.1 Data

We used an English-to-Japanese translation task of the Asian Scientific Paper Excerpt Corpus (ASPEC) [Nakazawa et al., 2016b] used in the Workshop on Asian Translation (WAT), since it has been shown that syntactic information is useful in English-to-Japanese translation [Eriguchi et al., 2016b; Neubig et al., 2015]. We followed the data preprocessing instruction for the English-to-Japanese task in Eriguchi et al. (2016b). The English sentences were tokenized by the tokenizer in the Enju parser [Miyao and Tsujii, 2008], and the Japanese sentences were segmented by the KyTea tool[5]. Among the first 1,500,000 translation pairs in the training data, we selected 1,346,946 pairs where the maximum sentence length is 50. In what follows, we call this dataset the *large* training dataset. We further selected the first 20,000 and 100,000 pairs to construct the *small* and *medium* training datasets, respectively. The development data include 1,790 pairs, and the test data 1,812 pairs.

[5] `http://www.phontron.com/kytea/`.

For the small and medium datasets, we built the vocabulary with words whose minimum frequency is two, and for the large dataset, we used words whose minimum frequency is three for English and five for Japanese. As a result, the vocabulary of the target language was 8,593 for the small dataset, 23,532 for the medium dataset, and 65,680 for the large dataset. A special token $\langle \text{UNK} \rangle$ was used to replace words which were not included in the vocabularies. The character n-grams ($n = 2, 3, 4$) were also constructed from each training dataset with the same frequency settings.

5.2.4.2 Parameter Optimization and Translation

We turned hyper-parameters of the model using development data. We set $(d_1, d_2) = (100, 50)$ for the latent graph parser. The word and character n-gram embeddings of the latent graph parser were initialized with the pre-trained embeddings in Hashimoto et al. (2017).[6] The weight matrices in the latent graph parser were initialized with uniform random values in $[-\frac{\sqrt{6}}{\sqrt{row+col}}, +\frac{\sqrt{6}}{\sqrt{row+col}}]$, where row and col are the number of rows and columns of the matrices, respectively. All the bias vectors and the weight matrices in the softmax layers were initialized with zeros, and the bias vectors of the forget gates in the LSTMs were initialized by ones [Jozefowicz et al., 2015].

We set $d_3 = 128$ for the small training dataset, $d_3 = 256$ for the medium training dataset, and $d_3 = 512$ for the large training dataset. The word embeddings and the weight matrices of the NMT model were initialized with uniform random values in $[-0.1, +0.1]$. The training was performed by mini-batch stochastic gradient descent with momentum. For the BlackOut objective [Ji et al., 2016], the number of the negative samples was set to 2,000 for the small and medium training datasets, and 2,500 for the large training dataset. The mini-batch size was set to 128, and the momentum rate was set to 0.75 for the small and medium training datasets and 0.70 for the large training dataset. A gradient clipping technique was used with a clipping value of 1.0. The initial learning rate was set to 1.0, and the learning rate was halved when translation accuracy decreased. We used the BLEU scores obtained by greedy translation as the translation accuracy and checked it at every half epoch of the model training. We saved the model parameters at every half epoch and used the saved model parameters for the parameter averaging technique. For regularization, we used L2-norm regularization with a coefficient of 10^{-6} and applied dropout [Srivastava et al., 2014] to Equation (5.20) with a dropout rate of 0.2.

The beam size for the beam search algorithm was 12 for the small and medium training datasets,

[6] The pre-trained embeddings can be found at `https://github.com/hassyGo/charNgram2vec`.

and 50 for the large training dataset. We used BLEU [Papineni et al., 2002], RIBES [Isozaki et al., 2010], and perplexity scores as our evaluation metrics. Note that lower perplexity scores indicate better accuracy.

5.2.4.3 Pre-Training of Latent Graph Parser

The latent graph parser in our model can be optionally pre-trained by using human annotations for dependency parsing. In this paper we used the widely-used Wall Street Journal (WSJ) training data to jointly train the POS tagging and dependency parsing components. We used the standard training split (Section 0-18) for POS tagging. We followed Chen and Manning (2014) to generate the training data (Section 2-21) for dependency parsing. From each training dataset, we selected the first K sentences to pre-train our model. The training dataset for POS tagging includes 38,219 sentences, and that for dependency parsing includes 39,832 sentences.

The parser including the POS tagger was first trained for 10 epochs in advance according to the multi-task learning procedure of Hashimoto et al. (2017), and then the overall NMT model was trained. When pre-training the POS tagging and dependency parsing components, we did not apply dropout to the model and did not fine-tune the word and character n-gram embeddings to avoid strong overfitting.

5.2.4.4 Model Configurations

LGP-NMT is our proposed model that learns the Latent Graph Parsing for NMT.

LGP-NMT+ is constructed by pre-training the latent parser in LGP-NMT as described in Section 5.2.4.3.

SEQ is constructed by removing the dependency composition in Equation (5.15), forming a sequential NMT model with the multi-layer encoder.

DEP is constructed by using pre-trained dependency relations rather than learning them. That is, $p(H_{w_i} = w_j|w_i)$ is fixed to 1.0 such that w_j is the head of w_i. The dependency labels are also given by the parser which was trained by using all the training samples for parsing and tagging.

	BLEU	RIBES	Perplexity
LGP-NMT	14.31±1.49	65.96±1.86	41.13±2.66
LGP-NMT+	16.81±0.31	69.03±0.28	38.33±1.18
SEQ	15.37±1.18	67.01±1.55	38.12±2.52
UNI	15.13±1.67	66.95±1.94	39.25±2.98
DEP	13.34±0.67	64.95±0.75	43.89±1.52

Table 5.16: Evaluation on the development data using the small training dataset (20,000 pairs).

K	BLEU	RIBES	Perplexity
0	14.31±1.49	65.96±1.86	41.13±2.66
5,000	16.99±1.00	69.03±0.93	37.14±1.96
10,000	16.81±0.31	69.03±0.28	38.33±1.18
All	16.09±0.56	68.19±0.59	39.24±1.88

Table 5.17: Effects of the size K of the training datasets for POS tagging and dependency parsing.

UNI is constructed by fixing $p(H_{w_i} = w_j|w_i)$ to $\frac{1}{N}$ for all the words in the same sentence. That is, the uniform probability distributions are used for equally connecting all the words.

5.2.5 Results on Small and Medium Datasets

We first show our translation results using the small and medium training datasets. We report averaged scores with standard deviations across five different runs of the model training.

5.2.5.1 Small Training Dataset

Table 5.16 shows the results of using the small training dataset. LGP-NMT performs worse than SEQ and UNI, which shows that the small training dataset is not enough to learn useful latent graph structures from scratch. However, LGP-NMT+ (K = 10,000) outperforms SEQ and UNI, and the standard deviations are the smallest. Therefore, the results suggest that pre-training the parsing and tagging components can improve the translation accuracy of our proposed model. We can also see that DEP performs the worst. This is not surprising because previous studies, e.g., Li et al. (2015),

	BLEU	RIBES	Perplexity
LGP-NMT	28.70±0.27	77.51±0.13	12.10±0.16
LGP-NMT+	29.06±0.25	77.57±0.24	12.09±0.27
SEQ	28.60±0.24	77.39±0.15	12.15±0.12
UNI	28.25±0.35	77.13±0.20	12.37±0.08
DEP	26.83±0.38	76.05±0.22	13.33±0.23

Table 5.18: Evaluation on the development data using the medium training dataset (100,000 pairs).

have reported that using syntactic structures do not always outperform competitive sequential models in several NLP tasks.

Now that we have observed the effectiveness of pre-training our model, one question arises naturally:

> how many training samples for parsing and tagging are necessary for improving the translation accuracy?

Table 5.17 shows the results of using different numbers of training samples for parsing and tagging. The results of $K = 0$ and $K = 10{,}000$ correspond to those of LGP-NMT and LGP-NMT+ in Table 5.16, respectively. We can see that using the small amount of the training samples performs better than using all the training samples.[7] One possible reason is that the domains of the translation dataset and the parsing (tagging) dataset are considerably different. The parsing and tagging datasets come from WSJ, whereas the translation dataset comes from abstract text of scientific papers in a wide range of domains, such as biomedicine and computer science. These results suggest that our model can be improved by a small amount of parsing and tagging datasets in different domains. Considering the recent universal dependency project[8] which covers more than 50 languages, our model has the potential of being applied to a variety of language pairs.

5.2.5.2 Medium Training Dataset

Table 5.18 shows the results of using the medium training dataset. In contrast with using the small training dataset, LGP-NMT is slightly better than SEQ. LGP-NMT significantly outperforms UNI, which shows that our adaptive learning is more effective than using the uniform graph weights. By pre-training our model, LGP-NMT+ significantly outperforms SEQ in terms of the BLEU score. Again, DEP performs the worst among all the models.

By using our beam search strategy, the Brevity Penalty (BP) values of our translation results are equal to or close to 1.0, which is important when evaluating the translation results using the BLEU scores. A BP value ranges from 0.0 to 1.0, and larger values mean that the translated sentences have relevant lengths compared with the reference translations. As a result, our BLEU evaluation results are affected only by the word n-gram precision scores. BLEU scores are sensitive to the BP values, and thus our beam search strategy leads to more solid evaluation for NMT models.

5.2.6 Results on Large Dataset

Table 5.19 shows the BLEU and RIBES scores on the development data achieved with the large training dataset. Here we focus on our models and SEQ because UNI and DEP consistently perform worse than the other models as shown in Table 5.16 and 5.18. The averaging technique and attention-based unknown word replacement [Jean et al., 2015; Hashimoto et al., 2016] improve the scores. Again, we see that the translation scores of our model can be further improved by pre-training the model.

Table 5.20 shows our results on the test data, and the previous best results summarized in Nakazawa et al. (2016a) and the WAT website[9] are also shown. Our proposed models, LGP-NMT and LGP-NMT+, outperform not only SEQ but also all of the previous best results. Notice also that our implementation of the sequential model (SEQ) provides a very strong baseline, the performance of which is already comparable to the previous state of the art, even without using ensemble techniques. The confidence interval ($p \leq 0.05$) of the RIBES score of LGP-NMT+ estimated by bootstrap resampling [Noreen, 1989] is $(82.27, 83.37)$, and thus the RIBES score of LGP-NMT+ is significantly

[7] We did not observe such significant difference when using the larger datasets, and we used all the training samples in the remaining part of this paper.

[8] `http://universaldependencies.org/`.

[9] `http://lotus.kuee.kyoto-u.ac.jp/WAT/evaluation/list.php?t=1&o=1`.

B./R.	Single	+Averaging	+UnkRep
LGP-NMT	38.05/81.98	38.44/82.23	38.77/82.29
LGP-NMT+	38.75/82.13	39.01/82.40	39.37/82.48
SEQ	38.24/81.84	38.26/82.14	38.61/82.18

Table 5.19: BLEU (B.) and RIBES (R.) scores on the development data using the large training dataset.

	BLEU	RIBES
LGP-NMT	39.19	82.66
LGP-NMT+	39.42	82.83
SEQ	38.96	82.18
Ensemble of the above three models	41.18	83.40
Cromieres et al. (2016)	38.20	82.39
Neubig et al. (2015)	38.17	81.38
Eriguchi et al. (2016a)	36.95	82.45
Neubig and Duh (2014)	36.58	79.65
Zhu (2015)	36.21	80.91
Lee et al. (2015)	35.75	81.15

Table 5.20: BLEU and RIBES scores on the test data.

better than that of SEQ, which shows that our latent parser can be effectively pre-trained with the human-annotated treebank.

The sequential NMT model in Cromieres et al. (2016) and the tree-to-sequence NMT model in Eriguchi et al. (2016b) rely on ensemble techniques while our results mentioned above are obtained using single models. Moreover, our model is more compact[10] than the previous best NMT model in Cromieres et al. (2016). By applying the ensemble technique to LGP-NMT, LGP-NMT+, and SEQ, the BLEU and RIBES scores are further improved, and both of the scores are significantly better than the previous best scores.

[10]Our training time is within five days on a `c4.8xlarge` machine of Amazon Web Service by our CPU-based C++ code, while it is reported that the training time is more than two weeks in Cromieres et al. (2016) by their GPU code.

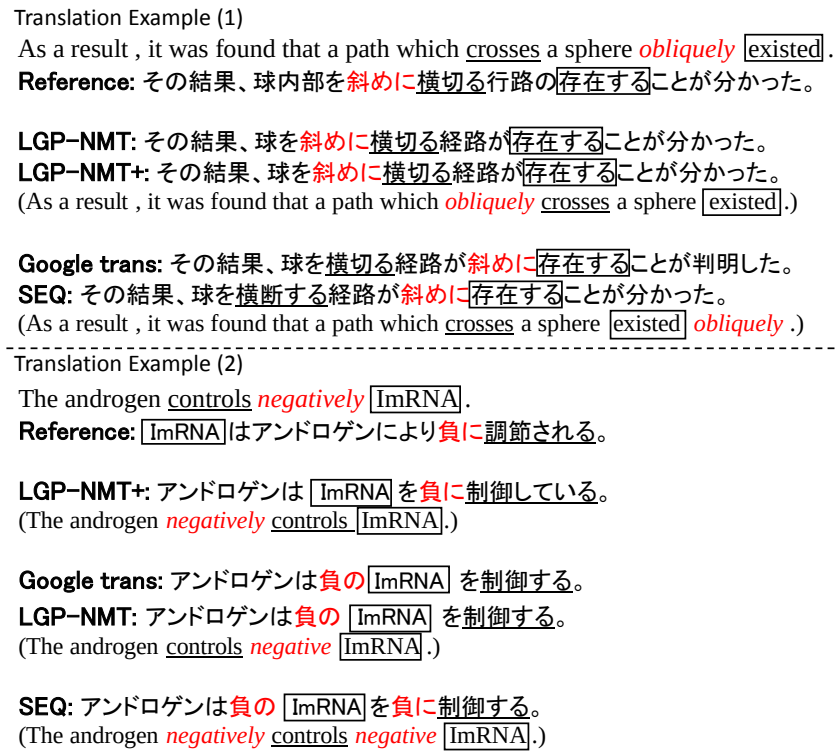


Figure 5.3: English-to-Japanese translation examples for focusing on the usage of adverbs.

5.2.6.1 Analysis on Translation Examples

Figure 5.3 shows two translation examples[11] to see how the proposed model works and what is missing in the state-of-the-art sequential NMT model, SEQ. Besides the reference translation, the outputs of our models with and without pre-training, SEQ, and Google Translation[12] are shown.

Selectional Preference In the translation example (1) in Figure 5.3, we see that the adverb "obliquely" is interpreted differently across the systems. As in the reference translation, "obliquely" is a modifier of the verb "crosses". Our models correctly capture the relationship between the two words, whereas Google Translation and SEQ treat "obliquely" as a modifier of the verb "existed". This error is not a

[11] These English sentences were created by manual simplification of sentences in the development data.
[12] The translations were obtained at `https://translate.google.com` in Feb. and Mar. 2017.

surprise since the verb "existed" is located closer to "obliquely" than the verb "crosses". A possible reason for the correct interpretation by our models is that they can better capture long-distance dependencies and are less susceptible to surface word distances. This is an indication of our models' ability of capturing domain-specific selectional preference that cannot be captured by purely sequential models. It should be noted that simply using standard treebank-based parsers does not necessarily address this error, because our pre-trained dependency parser interprets that "obliquely" is a modifier of the verb "existed".

Adverb or Adjective The translation example (2) in Figure 5.3 shows another example where the adverb "negatively" is interpreted as an adverb or an adjective. As in the reference translation, "negatively" is a modifier of the verb "controls". Only LGP-NMT+ correctly captures the adverb-verb relationship, whereas "negatively" is interpreted as the adjective "negative" to modify the noun "Im-RNA" in the translation results from Google Translation and LGP-NMT. SEQ interprets "negatively" as both an adverb and an adjective, which leads to the repeated translations. This error suggests that the state-of-the-art NMT models are strongly affected by the word order. By contrast, the pre-training strategy effectively embeds the information about the POS tags and the dependency relations into our model.

5.2.6.2 Analysis on Learned Latent Graphs

Without Pre-Training We inspected the latent graphs learned by LGP-NMT. Figure 5.2 shows an example of the learned latent graph obtained for a sentence taken from the development data of the translation task. It has long-range dependencies and cycles as well as ordinary left-to-right dependencies. We have observed that the punctuation mark "." is often pointed to by other words with large weights. This is primarily because the hidden state corresponding to the mark in each sentence has rich information about the sentence.

To measure the correlation between the latent graphs and human-defined dependencies, we parsed the sentences on the development data of the WSJ corpus and converted the graphs into dependency trees by Eisner's algorithm [Eisner, 1996]. For evaluation, we followed Chen and Manning (2014) and measured Unlabeled Attachment Score (UAS). The UAS is 24.52%, which shows that the implicitly-learned latent graphs are partially consistent with the human-defined syntactic structures. Similar trends have been reported by Yogatama et al. (2017) in the case of binary constituency parsing. We

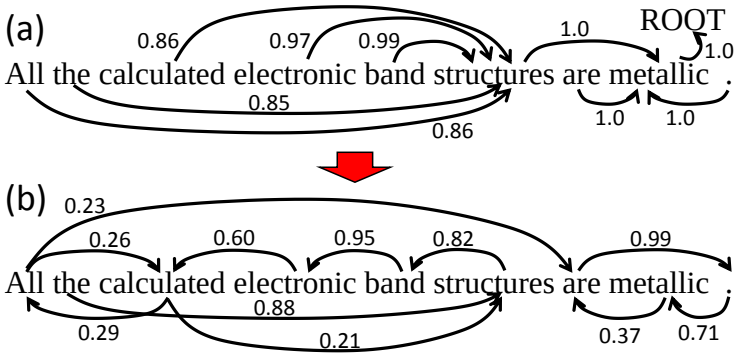


Figure 5.4: An example of the pre-trained dependency structures (a) and its corresponding latent graph adapted by our model (b).

checked the most dominant gold dependency labels which were assigned for the dependencies detected by LGP-NMT. The labels whose ratio is more than 3% are `nn`, `amod`, `prep`, `pobj`, `dobj`, `nsubj`, `num`, `det`, `advmod`, and `poss`. We see that dependencies between words in distant positions, such as subject-verb-object relations, can be captured.

With Pre-Training We also inspected the pre-trained latent graphs. Figure 5.4-(a) shows the dependency structure output by the pre-trained latent parser for the same sentence in Figure 5.2. This is an ordinary dependency tree, and the head selection is almost deterministic; that is, for each word, the largest weight of the head selection is close to 1.0. By contrast, the weight values are more evenly distributed in the case of LGP-NMT as shown in Figure 5.2. After the overall NMT model training, the latent parser is adapted to the translation task, and Figure 5.4-(b) shows the adapted latent graph. Again, we can see that the adapted weight values are also distributed and different from the original pre-trained weight values, which suggests that human-defined syntax is not always optimal for the target task.

The UAS of the pre-trained dependency trees is 92.52%[13], and that of the adapted latent graphs is 18.94%. Surprisingly, the resulting UAS (18.94%) is lower than the UAS of our model without pre-training (24.52%). However, in terms of the translation accuracy, our model with pre-training is better than that without pre-training. These results suggest that human-annotated treebanks can

[13]The UAS is significantly lower than the reported score in Hashimoto et al. (2017). The reason is described in Section 5.2.4.3.

provide useful prior knowledge to guide the overall model training by pre-training, but the resulting sentence structures adapted to the target task do not need to highly correlate with the treebanks.

5.2.7 Related Work

While initial studies on NMT treat each sentence as a sequence of words [Bahdanau et al., 2015; Luong et al., 2015; Sutskever et al., 2014], researchers have recently started investigating into the use of syntactic structures in NMT models [Bastings et al., 2017; Chen et al., 2017; Eriguchi et al., 2016a; Eriguchi et al., 2016b; Eriguchi et al., 2017; Li et al., 2017; Sennrich and Haddow, 2016; Stahlberg et al., 2016; Yang et al., 2017]. In particular, Eriguchi et al. (2016b) introduced a tree-to-sequence NMT model by building a tree-structured encoder on top of a standard sequential encoder, which motivated the use of the dependency composition vectors in our proposed model. Prior to the advent of NMT, the syntactic structures had been successfully used in statistical machine translation systems [Neubig and Duh, 2014; Yamada and Knight, 2001]. These syntax-based approaches are pipelined; a syntactic parser is first trained by supervised learning using a treebank such as the WSJ dataset, and then the parser is used to automatically extract syntactic information for machine translation. They rely on the output from the parser, and therefore parsing errors are propagated through the whole systems. By contrast, our model allows the parser to be adapted to the translation task, thereby providing a first step towards addressing ambiguous syntactic and semantic problems, such as domain-specific selectional preference and PP attachments, in a task-oriented fashion.

Our model learns latent graph structures in a source-side language. Eriguchi et al. (2017) have proposed a model which learns to parse and translate by using automatically-parsed data. Thus, it is also an interesting direction to learn latent structures in a target-side language.

As for the learning of latent syntactic structure, there are several studies on learning task-oriented syntactic structures. Yogatama et al. (2017) used a reinforcement learning method on shift-reduce action sequences to learn task-oriented binary constituency trees. They have shown that the learned trees do not necessarily highly correlate with the human-annotated treebanks, which is consistent with our experimental results. Socher et al. (2011b) used a recursive autoencoder model to greedily construct a binary constituency tree for each sentence. The autoencoder objective works as a regularization term for sentiment classification tasks. Prior to these deep learning approaches, Wu (1997) presented a method for *bilingual parsing*. One of the characteristics of our model is directly using the soft connections of the graph edges with the real-valued weights, whereas all of the above-mentioned methods

use one best structure for each sentence. Our model is based on dependency structures, and it is a promising future direction to jointly learn dependency and constituency structures in a task-oriented fashion.

Finally, more related to our model, Kim et al. (2017) applied their *structured attention networks* to a Natural Language Inference (NLI) task for learning dependency-like structures. They showed that pre-training their model by a parsing dataset did not improve accuracy on the NLI task. By contrast, our experiments show that such a parsing dataset can be effectively used to improve translation accuracy by varying the size of the dataset and by avoiding strong overfitting. Moreover, our translation examples show the concrete benefit of learning task-oriented latent graph structures.

5.2.8 Conclusion and Future Work

We have presented an end-to-end NMT model by jointly learning translation and source-side latent graph representations. By pre-training our model using treebank annotations, our model significantly outperforms both a pipelined syntax-based model and a state-of-the-art sequential model. On English-to-Japanese translation, our model outperforms the previous best models by a large margin. In future work, we investigate the effectiveness of our approach in different types of target tasks.

Chapter 6

Conclusions

In this dissertation, I presented joint learning methods for task-oriented representations in NLP. I have empirically shown that the proposed methods are effective in incorporating task-oriented information into word embedding learning, phrase embedding learning, and latent structure learning. There are many future directions in this challenging research field, NLP. Some of them are listed as follows:

- Developing task-oriented pre-training methods for more complex tasks (e.g., machine translation) than text classification. Explicitly incorporating real-world knowledge is an interesting future direction.
- Like the compositionality learning method, it would be a promising direction to learn task-oriented word segmentation for unsegmented languages, such Japanese and Chinese.
- Not only latent structures in sentences, but also those in documents are worth investigating, because relationships between sentences are crucial in understanding stories.

References

[Alberti et al.2015] Chris Alberti, David Weiss, Greg Coppola, and Slav Petrov. 2015. Improved Transition-Based Parsing and Tagging with Neural Networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1354–1359.

[Andor et al.2016] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally Normalized Transition-Based Neural Networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452.

[Attardi and Dell ' Orletta2008] Giuseppe Attardi and Felice Dell ' Orletta. 2008. Chunking and Dependency Parsing. In *Proceedings of LREC 2008 Workshop on Partial Parsing*.

[Bahdanau et al.2015] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of the 3rd International Conference on Learning Representations*.

[Bansal et al.2014] Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2014. Tailoring Continuous Word Representations for Dependency Parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 809–815.

[Baroni and Zamparelli2010] Marco Baroni and Roberto Zamparelli. 2010. Nouns are Vectors, Adjectives are Matrices: Representing Adjective-Noun Constructions in Semantic Space. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1193.

[Bastings et al.2017] Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Simaan. 2017. Graph Convolutional Encoders for Syntax-aware Neural Machine Translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1947–1957.

[Bengio et al.2003] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3:1137–1155.

[Bohnet2010] Bernd Bohnet. 2010. Top Accuracy and Fast Dependency Parsing is not a Contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 89–97.

[Bojanowski et al.2017] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

[Boros et al.2014] Emanuela Boros, Romaric Besançon, Olivier Ferret, and Brigitte Grau. 2014. Event Role Extraction using Domain-Relevant Word Representations. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1852–1857.

[Bunescu and Mooney2005] Razvan Bunescu and Raymond Mooney. 2005. A Shortest Path Dependency Kernel for Relation Extraction. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 724–731.

[Chambers and Jurafsky2010] Nathanael Chambers and Daniel Jurafsky. 2010. Improving the Use of Pseudo-Words for Evaluating Selectional Preferences. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 445–453.

[Chen and Manning2014] Danqi Chen and Christopher Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 740–750.

[Chen et al.2014] Wenliang Chen, Yue Zhang, and Min Zhang. 2014. Feature Embedding for Dependency Parsing. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 816–826.

[Chen et al.2016] Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, and Hui Jiang. 2016. Enhancing and Combining Sequential and Tree LSTM for Natural Language Inference. *arXiv*, cs.CL 1609.06038.

[Chen et al.2017] Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. 2017. Improved Neural Machine Translation with a Syntax-Aware Encoder and Decoder. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1936–1945.

[Cho et al.2014a] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014a. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111.

[Cho et al.2014b] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014b. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

[Choe and Charniak2016] Do Kook Choe and Eugene Charniak. 2016. Parsing as Language Modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336.

[Church and Hanks1990] Kenneth Church and Patrick Hanks. 1990. Word Association Norms, Mutual Information and Lexicography. *Computational Linguistics*, 19(2):263–312.

[Ciaramita and Altun2006] Massimiliano Ciaramita and Yasemin Altun. 2006. Broad-Coverage Sense Disambiguation and Information Extraction with a Supersense Sequence Tagger. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 594–602.

[Coecke et al.2010] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. 2010. Mathematical Foundations for a Compositional Distributional Model of Meaning. *CoRR*, abs/1003.4394.

[Collobert et al.2011] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12:2493–2537.

[Cromieres et al.2016] Fabien Cromieres, Chenhui Chu, Toshiaki Nakazawa, and Sadao Kurohashi. 2016. Kyoto University Participation to WAT 2016. In *Proceedings of the 3rd Workshop on Asian Translation*, pages 166–174.

[Diab and Bhutada2009] Mona Diab and Pravin Bhutada. 2009. Verb Noun Construction MWE Token Classification. In *Proceedings of the Workshop on Multiword Expressions: Identification, Interpretation, Disambiguation and Applications*, pages 17–22.

[dos Santos et al.2015] Cicero Nogueira dos Santos, Bing Xiang, and Bowen Zhou. 2015. Classifying Relations by Ranking with Convolutional Neural Networks. In *Proceedings of the Joint Conference of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, pages 626–634.

[Dozat and Manning2017] Timothy Dozat and Christopher D. Manning. 2017. Deep Biaffine Attention for Neural Dependency Parsing. In *Proceedings of the 5th International Conference on Learning Representations*.

[Duchi et al.2011] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159.

[Dyer et al.2015] Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-Based Dependency Parsing with Stack Long Short-Term Memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343.

[Ebrahimi and Dou2015] Javid Ebrahimi and Dejing Dou. 2015. Chain Based RNN for Relation Classification. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1244–1249.

[Eisner1996] Jason Eisner. 1996. Efficient Normal-Form Parsing for Combinatory Categorial Grammar. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 79–86.

[Eriguchi et al.2016a] Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016a. Character-based Decoding in Tree-to-Sequence Attention-based Neural Machine Translation. In *Proceedings of the 3rd Workshop on Asian Translation*, pages 175–183.

[Eriguchi et al.2016b] Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016b. Tree-to-Sequence Attentional Neural Machine Translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 823–833.

[Eriguchi et al.2017] Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. 2017. Learning to Parse and Translate Improves Neural Machine Translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 72–78.

[Farahmand et al.2015] Meghdad Farahmand, Aaron Smith, and Joakim Nivre. 2015. A Multiword Expression Data Set: Annotating Non-Compositionality and Conventionalization for English Noun Compounds. In *Proceedings of the 11th Workshop on Multiword Expressions*, pages 29–33.

[Finkelstein et al.2001] Lev Finkelstein, Gabrilovich Evgenly, Matias Yossi, Rivlin Ehud, Solan Zach, Wolfman Gadi, and Ruppin Eytan. 2001. Placing Search in Context: The Concept Revisited. In *Proceedings of the Tenth International World Wide Web Conference*.

[Firth1957] John Rupert Firth. 1957. A synopsis of linguistic theory 1930-55. In *Studies in Linguistic Analysis*, pages 1–32.

[Girju et al.2007] Roxana Girju, Preslav Nakov, Vivi Nastase, Stan Szpakowicz, Peter Turney, and Deniz Yuret. 2007. SemEval-2007 Task 04: Classification of Semantic Relations between Nominals. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, pages 13–18.

[Godwin et al.2016] Jonathan Godwin, Pontus Stenetorp, and Sebastian Riedel. 2016. Deep Semi-Supervised Learning with Linguistically Motivated Sequence Labeling Task Hierarchies. *arXiv*, cs.CL 1612.09113.

[Goodfellow et al.2013] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. 2013. Maxout Networks. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1319–1327.

[Goodfellow et al.2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Graves and Schmidhuber2005] Alex Graves and Jurgen Schmidhuber. 2005. Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. *Neural Networks*, 18(5):602–610.

[Grefenstette and Sadrzadeh2011] Edward Grefenstette and Mehrnoosh Sadrzadeh. 2011. Experimental Support for a Categorical Compositional Distributional Model of Meaning. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1394–1404.

[Grefenstette et al.2013] Edward Grefenstette, Georgiana Dinu, Yao-Zhong Zhang, Mehrnoosh Sadrzadeh, and Marco Baroni. 2013. Multi-Step Regression Learning for Compositional Distributional Semantics. In *Proceedings of the 10th International Conference on Computational Semantics*, pages 131–142.

[Guo et al.2014] Jiang Guo, Wanxiang Che, Haifeng Wang, and Ting Liu. 2014. Revisiting Embedding Features for Simple Semi-supervised Learning. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 110–120.

[Hashimoto and Tsuruoka2015] Kazuma Hashimoto and Yoshimasa Tsuruoka. 2015. Learning Embeddings for Transitive Verb Disambiguation by Implicit Tensor Factorization. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 1–11.

[Hashimoto and Tsuruoka2017] Kazuma Hashimoto and Yoshimasa Tsuruoka. 2017. Neural Machine Translation with Source-Side Latent Graph Parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 125–135.

[Hashimoto et al.2013] Kazuma Hashimoto, Makoto Miwa, Yoshimasa Tsuruoka, and Takashi Chikayama. 2013. Simple Customization of Recursive Neural Networks for Semantic Relation Classification. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1372–1376.

[Hashimoto et al.2014] Kazuma Hashimoto, Pontus Stenetorp, Makoto Miwa, and Yoshimasa Tsuruoka. 2014. Jointly Learning Word Representations and Composition Functions Using Predicate-Argument Structures. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1544–1555.

[Hashimoto et al.2016] Kazuma Hashimoto, Akiko Eriguchi, and Yoshimasa Tsuruoka. 2016. Domain Adaptation and Attention-Based Unknown Word Replacement in Chinese-to-Japanese Neural Machine Translation. In *Proceedings of the 3rd Workshop on Asian Translation*, pages 75–83.

[Hashimoto et al.2017] Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. 2017. A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 446–456.

[Hendrickx et al.2010] Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. 2010. SemEval-2010 Task 8: Multi-Way Classification of Semantic Relations between Pairs of Nominals. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 33–38.

[Hermann and Blunsom2013] Karl Moritz Hermann and Phil Blunsom. 2013. The Role of Syntax in Vector Space Models of Compositional Semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 894–904.

[Hinton et al.2012a] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2012a. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

[Hinton et al.2012b] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2012b. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

[Hochreiter and Schmidhuber1997] Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[Inan et al.2016] Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling. *arXiv*, cs.CL 1611.01462, November.

[Irsoy and Cardie2014] Ozan Irsoy and Claire Cardie. 2014. Deep Recursive Neural Networks for Compositionality in Language. In *Advances in Neural Information Processing Systems 27*, pages 2096–2104.

[Isozaki et al.2010] Hideki Isozaki, Tsutomu Hirao, Kevin Duh, Katsuhito Sudoh, and Hajime Tsukada. 2010. Automatic Evaluation of Translation Quality for Distant Language Pairs. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 944–952.

[Jean et al.2015] Sébastien Jean, Orhan Firat, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. Montreal Neural Machine Translation Systems for WMT ' 15. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 134–140.

[Ji et al.2016] Shihao Ji, S. V. N. Vishwanathan, Nadathur Satish, Michael J. Anderson, and Pradeep Dubey. 2016. BlackOut: Speeding up Recurrent Neural Network Language Models With Very Large Vocabularies. In *Proceedings of the 4th International Conference on Learning Representations*.

[Jozefowicz et al.2015] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An Empirical Exploration of Recurrent Network Architectures. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 2342–2350.

[Kartsaklis and Sadrzadeh2013] Dimitri Kartsaklis and Mehrnoosh Sadrzadeh. 2013. Prior Disambiguation of Word Tensors for Constructing Sentence Vectors. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1590–1601.

[Kartsaklis and Sadrzadeh2014] Dimitri Kartsaklis and Mehrnoosh Sadrzadeh. 2014. A Study of Entanglement in a Categorical Framework of Natural Language. In *Proceedings of the 11th Workshop on Quantum Physics and Logic (QPL)*, Kyoto, Japan, June.

[Kartsaklis et al.2012] Dimitri Kartsaklis, Mehrnoosh Sadrzadeh, and Stephen Pulman. 2012. A Unified Sentence Space for Categorical Distributional-Compositional Semantics: Theory and Ex-

periments. In *Proceedings of the 24th International Conference on Computational Linguistics*, pages 549–558.

[Kiela and Clark2013] Douwe Kiela and Stephen Clark. 2013. Detecting Compositionality of Multi-Word Expressions using Nearest Neighbours in Vector Space Models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1427–1432.

[Kim et al.2017] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. 2017. Deep Biaffine Attention for Neural Dependency Parsing. In *Proceedings of the 5th International Conference on Learning Representations*.

[Kiperwasser and Goldberg2016] Eliyahu Kiperwasser and Yoav Goldberg. 2016. Easy-First Dependency Parsing with Hierarchical Tree LSTMs. *Transactions of the Association for Computational Linguistics*, 4:445–461.

[Kudo and Matsumoto2001] Taku Kudo and Yuji Matsumoto. 2001. Chunking with Support Vector Machines. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics*.

[Kumar et al.2016] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask Me Anything: Dynamic Memory Networks for Natural Language Processing. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1378–1387.

[Kuncoro et al.2017] Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What Do Recurrent Neural Network Grammars Learn About Syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 1249–1258.

[Lai and Hockenmaier2014] Alice Lai and Julia Hockenmaier. 2014. Illinois-LH: A Denotational and Distributional Approach to Semantics. In *Proceedings of the 8th International Workshop on Semantic Evaluation*, pages 329–334.

[Lai et al.2015] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent Convolutional Neural Networks for Text Classification. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 2267–2273.

[Le and Mikolov2014] Quoc Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, ICML '14, pages 1188–1196.

[Lee et al.2015] Hyoung-Gyu Lee, JaeSong Lee, Jun-Seok Kim, and Chang-Ki Lee. 2015. NAVER Machine Translation System for WAT 2015. In *Proceedings of the 2nd Workshop on Asian Translation*, pages 69–73.

[Leech1992] Geoffrey Leech. 1992. 100 Million Words of English: the British National Corpus. *Language Research*, 28(1):1–13.

[Levy and Goldberg2014a] Omer Levy and Yoav Goldberg. 2014a. Neural Word Embedding as Implicit Matrix Factorization. In *Advances in Neural Information Processing Systems 27*, pages 2177–2185.

[Levy and Goldberg2014b] Omer Levy and Yoav Goldberg. 2014b. Neural Word Embedding as Implicit Matrix Factorization. In *Advances in Neural Information Processing Systems 27*, pages 2177–2185.

[Li and Hoiem2016] Zhizhong Li and Derek Hoiem. 2016. Learning without Forgetting. *CoRR*, abs/1606.09282.

[Li et al.2015] Jiwei Li, Thang Luong, Dan Jurafsky, and Eduard Hovy. 2015. When Are Tree Structures Necessary for Deep Learning of Representations? In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2304–2314.

[Li et al.2017] Junhui Li, Deyi Xiong, Zhaopeng Tu, Muhua Zhu, Min Zhang, and Guodong Zhou. 2017. Modeling Source Syntax for Neural Machine Translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 688–697.

[Lin1999] Dekang Lin. 1999. Automatic Identification of Non-compositional Phrases. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 317–324.

[Ling et al.2015] Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. 2015. Finding Function in Form: Compositional Character

Models for Open Vocabulary Word Representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530.

[Luong and Manning2016] Minh-Thang Luong and Christopher D. Manning. 2016. Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1054–1063.

[Luong et al.2015] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421.

[Luong et al.2016] Minh-Thang Luong, Ilya Sutskever, Quoc V. Le, Oriol Vinyals, and Lukasz Kaiser. 2016. Multi-task Sequence to Sequence Learning. In *Proceedings of the 4th International Conference on Learning Representations*.

[Ma and Hovy2016] Xuezhe Ma and Eduard Hovy. 2016. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074.

[Marelli et al.2014] Marco Marelli, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. 2014. SemEval-2014 Task 1: Evaluation of Compositional Distributional Semantic Models on Full Sentences through Semantic Relatedness and Textual Entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 1–8.

[McCarthy et al.2003] Diana McCarthy, Bill Keller, and John Carroll. 2003. Detecting a Continuum of Compositionality in Phrasal Verbs. In *Proceedings of the ACL 2003 Workshop on Multiword Expressions: Analysis, Acquisition and Treatment*, pages 73–80.

[McCarthy et al.2007] Diana McCarthy, Sriram Venkatapathy, and Aravind Joshi. 2007. Detecting Compositionality of Verb-Object Combinations using Selectional Preferences. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 369–379.

[McDonald et al.2005] Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online Large-Margin Training of Dependency Parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 91–98.

[Mikolov et al.2013a] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of Workshop at the International Conference on Learning Representations*.

[Mikolov et al.2013b] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119.

[Milajevs et al.2014] Dmitrijs Milajevs, Dimitri Kartsaklis, Mehrnoosh Sadrzadeh, and Matthew Purver. 2014. Evaluating Neural Word Representations in Tensor-Based Compositional Settings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 708–719.

[Misra et al.2016] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. 2016. Cross-stitch Networks for Multi-task Learning. *CoRR*, abs/1604.03539.

[Mitchell and Lapata2008] Jeff Mitchell and Mirella Lapata. 2008. Vector-based models of semantic composition. In *Proceedings of 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 236–244.

[Mitchell and Lapata2010] Jeff Mitchell and Mirella Lapata. 2010. Composition in Distributional Models of Semantics. *Cognitive Science*, 34(8):1388–1439.

[Miwa and Bansal2016] Makoto Miwa and Mohit Bansal. 2016. End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1105–1116.

[Miyamoto and Cho2016] Yasumasa Miyamoto and Kyunghyun Cho. 2016. Gated Word-Character Recurrent Language Model. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1992–1997.

[Miyao and Tsujii2008] Yusuke Miyao and Jun'ichi Tsujii. 2008. Feature Forest Models for Probabilistic HPSG Parsing. *Computational Linguistics*, 34(1):35–80, March.

[Miyao et al.2006] Yusuke Miyao, Tomoko Ohta, Katsuya Masuda, Yoshimasa Tsuruoka, Kazuhiro Yoshida, Takashi Ninomiya, and Jun'ichi Tsujii. 2006. Semantic Retrieval for the Accurate Identification of Relational Concepts in Massive Textbases. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 1017–1024.

[Mnih and Kavukcuoglu2013] Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems 26*, pages 2265–2273.

[Muraoka et al.2014] Masayasu Muraoka, Sonse Shimaoka, Kazeto Yamamoto, Yotaro Watanabe, Naoaki Okazaki, and Kentaro Inui. 2014. Finding The Best Model Among Representative Compositional Models. In *Proceedings of the 28th Pacific Asia Conference on Language, Information, and Computation*, pages 65–74.

[Nakazawa et al.2016a] Toshiaki Nakazawa, Hideya Mino, Chenchen Ding, Isao Goto, Graham Neubig, Sadao Kurohashi, and Eiichiro Sumita. 2016a. Overview of the 3rd Workshop on Asian Translation. In *Proceedings of the 3rd Workshop on Asian Translation (WAT2016)*.

[Nakazawa et al.2016b] Toshiaki Nakazawa, Manabu Yaguchi, Kiyotaka Uchimoto, Masao Utiyama, Eiichiro Sumita, Sadao Kurohashi, and Hitoshi Isahara. 2016b. ASPEC: Asian Scientific Paper Excerpt Corpus. In *Proceedings of the 10th Conference on International Language Resources and Evaluation*.

[Neubig and Duh2014] Graham Neubig and Kevin Duh. 2014. On the Elements of an Accurate Tree-to-String Machine Translation System. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 143–149.

[Neubig et al.2015] Graham Neubig, Makoto Morishita, and Satoshi Nakamura. 2015. Neural Reranking Improves Subjective Quality of Machine Translation: NAIST at WAT2015. In *Proceedings of the 2nd Workshop on Asian Translation (WAT2015)*, pages 35–41.

[Newton2006] Mark Newton. 2006. *Basic English Syntax with Exercises*. Bölcsész Konzorcium.

[Nguyen and Grishman2014] Thien Huu Nguyen and Ralph Grishman. 2014. Employing Word Representations and Regularization for Domain Adaptation of Relation Extraction. In *Proceedings*

of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 68–74.

[Noreen1989] Eric W. Noreen. 1989. *Computer-Intensive Methods for Testing Hypotheses: An Introduction*. Wiley-Interscience.

[Ono et al.2015] Masataka Ono, Makoto Miwa, and Yutaka Sasaki. 2015. Word Embedding-based Antonym Detection using Thesauri and Distributional Information. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 984–989.

[Papineni et al.2002] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 311–318.

[Paulus et al.2014] Romain Paulus, Richard Socher, and Christopher D Manning. 2014. Global Belief Recursive Neural Networks. In *Advances in Neural Information Processing Systems 27*, pages 2888–2896.

[Pennington et al.2014] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

[Pham et al.2014] Vu Pham, Theodore Bluche, Christopher Kermorvant, and Jerome Louradour. 2014. Dropout improves Recurrent Neural Networks for Handwriting Recognition. *CoRR*, abs/1312.4569.

[Pham et al.2015] Nghia The Pham, Germán Kruszewski, Angeliki Lazaridou, and Marco Baroni. 2015. Jointly optimizing word representations for lexical and sentential tasks with the C-PHRASE model. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 971–981.

[Polajnar et al.2014] Tamara Polajnar, Laura Rimell, and Stephen Clark. 2014. Using Sentence Plausibility to Learn the Semantics of Transitive Verbs. In *Proceedings of Workshop on Learning Semantics at the 2014 Conference on Neural Information Processing Systems*.

[Polajnar et al.2015] Tamara Polajnar, Laura Rimell, and Stephen Clark. 2015. An Exploration of Discourse-Based Sentence Spaces for Compositional Distributional Semantics. In *Proceedings of the First Workshop on Linking Computational Models of Lexical, Sentential and Discourse-level Semantics*, pages 1–11.

[Press and Wolf2017] Ofir Press and Lior Wolf. 2017. Using the Output Embedding to Improve Language Models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163.

[Ramachandran et al.2017] Prajit Ramachandran, Peter Liu, and Quoc Le. 2017. Unsupervised Pretraining for Sequence to Sequence Learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 383–391.

[Reddy et al.2011] Siva Reddy, Diana McCarthy, and Suresh Manandhar. 2011. An Empirical Study on Compositionality in Compound Nouns. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 210–218.

[Rink and Harabagiu2010] Bryan Rink and Sanda Harabagiu. 2010. UTD: Classifying Semantic Relations by Combining Lexical and Semantic Resources. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 256–259.

[Rusu et al.2016] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive Neural Networks. *CoRR*, abs/1606.04671.

[Sennrich and Haddow2016] Rico Sennrich and Barry Haddow. 2016. Linguistic Input Features Improve Neural Machine Translation. In *Proceedings of the First Conference on Machine Translation*, pages 83–91.

[Sennrich et al.2016] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725.

[Socher et al.2011a] Richard Socher, Eric H. Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y. Ng. 2011a. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. In *Advances in Neural Information Processing Systems 24*, pages 801–809.

[Socher et al.2011b] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. 2011b. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 151–161.

[Socher et al.2012] 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.

[Socher et al.2013] Richard Socher, John Bauer, Christopher D. Manning, and Ng Andrew Y. 2013. Parsing with Compositional Vector Grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 455–465.

[Socher et al.2014] 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. *Transactions of the Association of Computational Linguistics*, 2:207–218.

[Søgaard and Goldberg2016] Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 231–235.

[Søgaard2011] Anders Søgaard. 2011. Semi-supervised condensed nearest neighbor for part-of-speech tagging. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 48–52.

[Srihari and Li2000] Rohini Srihari and Wel Li. 2000. A Question Answering System Supported by Information Extraction. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, pages 166–172.

[Srivastava et al.2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

[Stahlberg et al.2016] Felix Stahlberg, Eva Hasler, Aurelien Waite, and Bill Byrne. 2016. Syntactically Guided Neural Machine Translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 299–305.

[Stenetorp2013] Pontus Stenetorp. 2013. Transition-based Dependency Parsing Using Recursive Neural Networks. In *Proceedings of Deep Learning Workshop at the 2013 Conference on Neural Information Processing Systems*.

[Sutskever et al.2014] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112.

[Suzuki and Isozaki2008] Jun Suzuki and Hideki Isozaki. 2008. Semi-Supervised Sequential Labeling and Segmentation Using Giga-Word Scale Unlabeled Data. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 665–673.

[Tai et al.2015] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566.

[Tang et al.2014] Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. 2014. Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1555–1565.

[Toutanova et al.2003] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 173–180.

[Tsubaki et al.2013] Masashi Tsubaki, Kevin Duh, Masashi Shimbo, and Yuji Matsumoto. 2013. Modeling and Learning Semantic Co-Compositionality through Prototype Projections and Neural

Networks. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 130–140.

[Tsuruoka et al.2011] Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Kazama. 2011. Learning with Lookahead: Can History-Based Models Rival Globally Optimized Models? In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pages 238–246.

[Turian et al.2010] Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word Representations: A Simple and General Method for Semi-Supervised Learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394.

[Van de Cruys et al.2013] Tim Van de Cruys, Thierry Poibeau, and Anna Korhonen. 2013. A Tensor-based Factorization Model of Semantic Compositionality. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1142–1151.

[Van de Cruys2014] Tim Van de Cruys. 2014. A Neural Network Approach to Selectional Preference Acquisition. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 26–35.

[Venkatapathy and Joshi2005] Sriram Venkatapathy and Aravind Joshi. 2005. Measuring the Relative Compositionality of Verb-Noun (V-N) Collocations by Integrating Features. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 899–906.

[Wang and Manning2012] Sida Wang and Christopher Manning. 2012. Baselines and Bigrams: Simple, Good Sentiment and Topic Classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 90–94.

[Weiss et al.2015] David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured Training for Neural Network Transition-Based Parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333.

[Wieting et al.2016] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016. CHARAGRAM: Embedding Words and Sentences via Character n-grams. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1504–1515.

[Wu1997] Dekai Wu. 1997. Stochastic Inversion Transduction Grammars and Bilingual Parsing of Parallel Corpora. *Computational Linguistics*, 23(3):377–404.

[Yamada and Knight2001] Kenji Yamada and Kevin Knight. 2001. A Syntax-based Statistical Translation Model. In *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics*, pages 523–530.

[Yang et al.2017] Baosong Yang, Derek F. Wong, Tong Xiao, Lidia S. Chao, and Jingbo Zhu. 2017. Towards Bidirectional Hierarchical Representations for Attention-Based Neural Machine Translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1443–1452.

[Yazdani et al.2015] Majid Yazdani, Meghdad Farahmand, and James Henderson. 2015. Learning Semantic Composition to Detect Non-compositionality of Multiword Expressions. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1733–1742.

[Yin et al.2016] Wenpeng Yin, Hinrich Schutze, Bing Xiang, and Bowen Zhou. 2016. ABCNN: Attention-Based Convolutional Neural Network for Modeling Sentence Pairs. *Transactions of the Association for Computational Linguistics*, 4:259–272.

[Yogatama et al.2017] Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. 2017. Learning to Compose Words into Sentences with Reinforcement Learning. In *Proceedings of the 5th International Conference on Learning Representations*.

[Yu et al.2014] Mo Yu, Matthew R. Gormley, and Mark Dredze. 2014. Factor-based Compositional Embedding Models. In *Proceedings of Workshop on Learning Semantics at the 2014 Conference on Neural Information Processing Systems*.

[Zeng et al.2014] Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao. 2014. Relation Classification via Convolutional Deep Neural Network. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2335–2344.

[Zhang and Weiss2016] Yuan Zhang and David Weiss. 2016. Stack-propagation: Improved Representation Learning for Syntax. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1557–1566.

[Zhang et al.2006] Min Zhang, Jie Zhang, Jian Su, and GuoDong Zhou. 2006. A Composite Kernel to Extract Relations between Entities with Both Flat and Structured Features. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 825–832.

[Zhang et al.2017] Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2017. Dependency Parsing as Head Selection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 665–676.

[Zhou et al.2016] Yao Zhou, Cong Liu, and Yan Pan. 2016. Modelling Sentence Pairs with Tree-structured Attentive Encoder. In *Proceedings of the 26th International Conference on Computational Linguistics*, pages 2912–2922.

[Zhu2015] Zhongyuan Zhu. 2015. Evaluating Neural Machine Translation in English-Japanese Task. In *Proceedings of the 2nd Workshop on Asian Translation*, pages 61–68.

[Zoph et al.2016] Barret Zoph, Ashish Vaswani, Jonathan May, and Kevin Knight. 2016. Simple, Fast Noise-Contrastive Estimation for Large RNN Vocabularies. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1217–1222.

Publications

1. Kazuma Hashimoto, Pontus Stenetorp, Makoto Miwa, and Yoshimasa Tsuruoka. 2015. Task-Oriented Learning of Word Embeddings for Semantic Relation Classification. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pp. 268–278.

2. Kazuma Hashimoto and Yoshimasa Tsuruoka. 2015. Learning Embeddings for Transitive Verb Disambiguation by Implicit Tensor Factorization. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pp. 1–11.

3. Kazuma Hashimoto and Yoshimasa Tsuruoka. 2016. Adaptive Joint Learning of Compositional and Non-Compositional Phrase Embeddings. In *Proceedings of the 54th Annual Meeting of Association for Computational Linguistics*, pp. 205–215.

4. Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. 2017. A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 446–456.

5. Kazuma Hashimoto and Yoshimasa Tsuruoka. 2017. Neural Machine Translation with Source-Side Latent Graph Parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 125–135.

Other Publications

1. Yuchen Qiao, Kazuma Hashimoto, Akiko Eriguchi, Haxia Wang, Dongsheng Wang, Yoshimasa Tsuruoka, and Kenjiro Taura. 2017. Cache Friendly Parallelization of Neural Encoder-Decoder Models without Padding on Multi-core Architecture. In *Proceedings of the 6th International Workshop on Parallel and Distributed Computing for Large Scale Machine Learning and Big Data Analytics*.

2. Yusuke Watanabe, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Domain Adaptation for Neural Networks by Parameter Augmentation. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pp. 249—257. **(Best Paper Award)**

3. Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-Sequence Attentional Neural Machine Translation. In *Proceedings of the 54th Annual Meeting of Association for Computational Linguistics*, pp. 823—833.

4. Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Character-based Decoding in Tree-to-Sequence Attention-based Neural Machine Translation. In *Proceedings of the 3rd Workshop on Asian Translation*, pp. 175—183.

5. Kazuma Hashimoto, Akiko Eriguchi, and Yoshimasa Tsuruoka. 2016. Domain Adaptation and Attention-Based Unknown Word Replacement in Chinese-to-Japanese Neural Machine Translation. In *Proceedings of the 3rd Workshop on Asian Translation*, pp. 75—83.

6. Kazuma Hashimoto, Georgios Kontonatsios, Makoto Miwa, and Sophia Ananiadou. 2016. Topic detection using paragraph vectors to support active learning in systematic reviews. *Journal of Biomedical Informatics*, Volume 62, pp. 59—65.

7. Kazuma Hashimoto, Pontus Stenetorp, Makoto Miwa, and Yoshimasa Tsuruoka. 2014. Jointly Learning Word Representations and Composition Functions Using Predicate-Argument Structures. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1544—1555.

8. Kazuma Hashimoto, Makoto Miwa, Yoshimasa Tsuruoka, and Takashi Chikayama. 2013. Simple Customization of Recursive Neural Networks for Semantic Relation Classification. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1372—1376.