Learning Vector Representations for Sentences

The Recursive Deep Learning Approach

Phong Lê

Learning Vector Representations for Sentences

The Recursive Deep Learning Approach

ILLC Dissertation Series DS-2016-05

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

For further information about ILLC-publications, please contact

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Science Park 107
1098 XG Amsterdam
phone: +31-20-525 6051
e-mail: illc@uva.nl
homepage: http://www.illc.uva.nl/

The investigations were supported by the Faculty of Humanities of the University of Amsterdam, through a Digital Humanities fellowship.

Cover image by Raquel Garrido Alhama.
Cover design by Phong Lê.
Printed and bound by Ipskamp Printing.

ISBN: 978-94-028-0181-1

Learning Vector Representations for Sentences

The Recursive Deep Learning Approach

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. D.C. van den Boom
ten overstaan van een door het College voor Promoties ingestelde
commissie, in het openbaar te verdedigen in de Aula der Universiteit
op vrijdag 3 juni 2016, te 11.00 uur

door Phong Lê

geboren te Thai Binh, Vietnam

Promotiecommissie:

Promotor:	Prof. dr. L.W.M. Bod	Universiteit van Amsterdam
Co-promotor:	Dr. W.H. Zuidema	Universiteit van Amsterdam
Overige leden:	Dr. M. Baroni	University of Trento
	Prof. dr. M. Lapata	University of Edinburgh
	Dr. R. Fernandez Rovira	Universiteit van Amsterdam
	Prof. dr. K. Sima'an	Universiteit van Amsterdam
	Prof. dr. M. Welling	Universiteit van Amsterdam

Faculteit der Geesteswetenschappen

to my father

who showed me the beauty of science and technology

Contents

Acknowledgments

First of all, I am very grateful to my promotors, Rens Bod and Jelle Zuidema, for offering me a position, their patience, and their great support. I am especially thankful to Jelle, who has played a very important role in most of my work in the last 5 years, and gave me invaluable advice.

I want to thank Remko Scha whose insights helped me shape my initial ideas in the beginning of my PhD, Khalil Sima'an who showed me the beauty of NLP when I was a Master's student, and Ivan Titov whose job offer helped me write this dissertation in peace. I thank Federico Sangati, Gideon Borensztajn, and Tim Van de Cruys for helpful discussions. I also want to thank the staff members at the ILLC, Jenny Batson, Karine Gigengack, Tanja Kassenaar, Fenneke Kortenbach, for their support.

I thank all of my friends with whom I shared my PhD adventure: Sophie Arnoult, Joost Bastings, Andreas van Cranenburgh, Desmond Elliott, Stella Frank, Wilker Ferreira Aziz, LiFeng Han, Cuong Hoang, Dieuwke Hupkes, Ehsan Khoddammohammadi, Corina Koolen, Gideon Maillette de Buy Wenniger, Diego Marcheggiani, Philip Schulz, Ke Tran, Sara Veldhoen,... I especially thank Raquel Garrido Alhama, Milos Stanojevic, and Joachim Daiber for what they taught me about Western life, for endless discussions, for good beer, food, movies, and for many other things that I cannot name.

Finally, I couldn't thank my family enough for their endless love and unconditional support. Thank you so much, my parents, brother, sister in law, and parents in law. Last but not least, a million thanks to my wife, Nguyen Dang, for being with me during the hardest times in my life.

Amsterdam
April, 2016.

Phong Le

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

Artificial neural networks are mathematical tools that abstract the computation of (a part of) the human brain. Theoretical results, such as that two-layer neural networks are universal approximators [Cybenko, 1989] and that Turing machines are recurrent neural networks [Hyötyniemi, 1996], mean that their expressiveness is undeniable. However, in practice, how to train large, deep neural networks effectively and efficiently has been for a long time one of the most challenging problems in machine learning. In spite of a lot of effort in solving this challenge (see Schmidhuber [2015] for a general review), only recently, since Hinton et al. [2006] succeeded in training deep belief networks, *deep learning*, a subfield of machine learning dealing with training deep models, has become a "promised land". What makes deep learning so appealing is that it significantly reduces (or, in many cases, avoids) the need for manual feature engineering, which is to use expertise knowledge to extract useful features. Deep learning systems can now learn to recognize cats in photos [Le et al., 2012], play video games [Mnih et al., 2015] just by raw pixel inputs, and beat a professional human Go player [Silver et al., 2016].

Because of the notorious complexity and irregularity of human language, traditional approaches in Natural Language Processing (NLP) rely heavily on expertise and domain knowledge to design helpful features. Deep *representation* learning, therefore, also holds great promise for NLP. And, in fact, we did not need to wait for long to see outcomes of this "marriage": just two years after the publication of Hinton et al. [2006], SENNA[1] [Collobert and Weston, 2008], the first deep learning NLP system, got (nearly) state-of-the-art performance on several tasks (part-of-speech tagging, chunking, named entity recognition, and semantic role labelling). The following years have seen a surprising number of publications proposing deep learning approaches to tackle a wide range of NLP tasks, such as syntactic parsing [Socher et al., 2013a, Chen and

[1] `http://ronan.collobert.com/senna/`

Manning, 2014, Dyer et al., 2015], sentiment analysis [Socher et al., 2013b, Kalchbrenner et al., 2014, Tai et al., 2015], machine translation [Sutskever et al., 2014, Luong et al., 2015, Bahdanau et al., 2015], dialog processing [Vinyals and Le, 2015, Wen et al., 2015], language modelling [Mikolov et al., 2010, Jozefowicz et al., 2016], and question answering [Weston et al., 2015, Kumar et al., 2015].

However, the road is not always smooth. The impact of deep learning in reducing the dependency on linguistic knowledge is so significant that some in the NLP community have started to worry [Manning, 2016], and get increasingly vocal about their criticism of the approach, using arguments that were also used in debates about connectionism in cognitive science [Fodor and Pylyshyn, 1988] and in linguistics [Jackendoff, 2001]. One of the main lines of such criticism relates to the *binding* problem, often defined as the problem of representing conjunctions of properties. For instance, when processing the phrase "fat mice and small cats", how can a connectionist system know that "fat" related to "mice" and "small" to "cats"? How could it avoid combining "fat" with "cats"?

Another main line relates to the criticism that a neural network is a "black box", in the sense that, in spite of the universal approximation property, studying the structure of a neural network does not give us any understanding about the structure of the function being approximated. Besides, examining neurons' activations hardly tells us why and how neural networks work. In contrast, many of the advances in NLP of the last 40 years were based on developing meaningful *representations* (Figure 1.1). A representation could be a string of part-of-speech tags or a parse tree, if we are interested in syntactic structures. For semantics, logic has since the Ancient Greece era been used as a tool to represent meaning. Traditional representations conveying meanings of sentences are therefore logical expressions such as first-order logical forms. Those types of representations have in common that they are readable for human beings, in the sense that we can understand them and analyse their components directly. For example, looking at a parse tree of a sentence, we can detect the tense of its main verb, whether the sentence is complex, and so on. The vector representations used in deep learning systems look, at first sight, like a major step back: it is very difficult (or sometimes impossible) to fully interpret its components.

Despite those difficulties, in this dissertation, I do develop deep neural network models to tackle the problem of transforming sentences into vector representations. Like Manning [2016], I view the marriage of deep learning and NLP as a great opportunity to solve many NLP challenges: the use of linguistic knowledge can provide (1) insights for building proper neural network models, and (2) priors for boosting the learning process. (1) leads me to a hybrid approach of symbolic NLP and connectionist deep learning to avoid the binding problem, based on the *principle of compositionality*, which says that:

> The meaning of a whole is a function of the meanings of the parts and of the way they are *syntactically* combined

[Partee, 1995]. Technically speaking, the principle suggests a general mechanism to

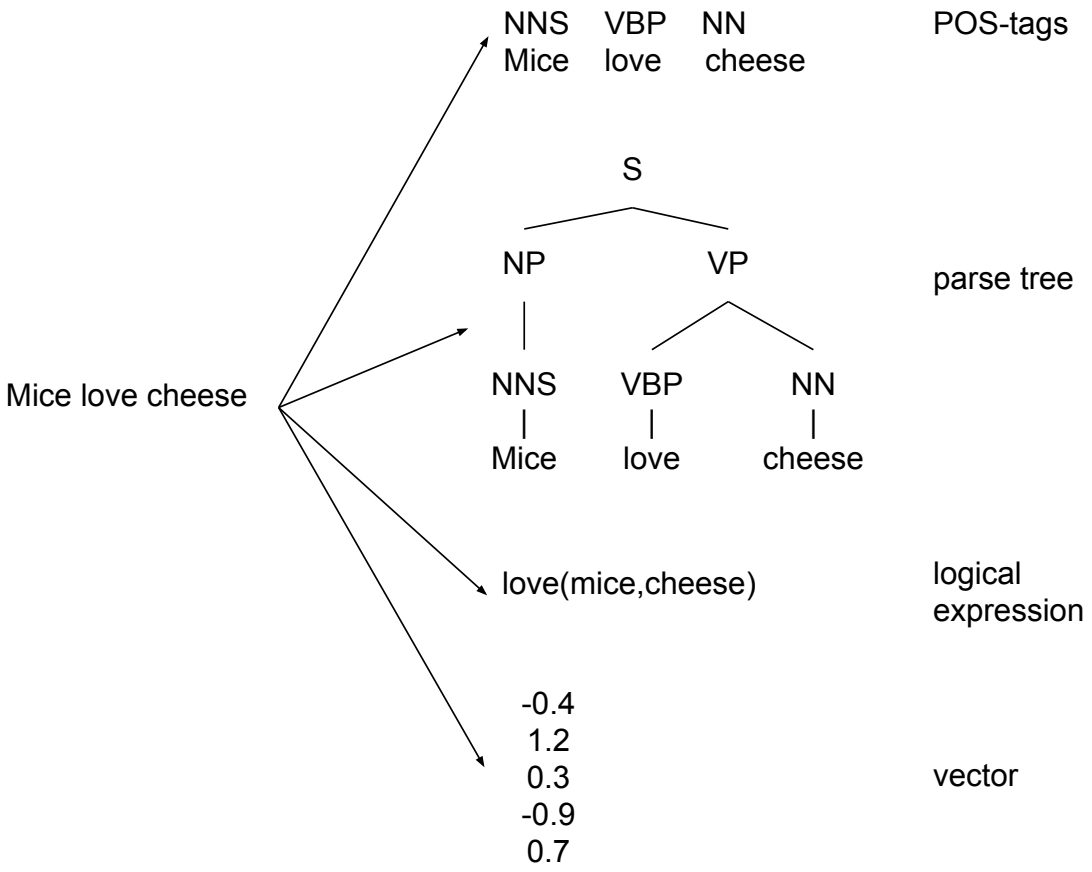


Figure 1.1: A sentence can have different representations. The symbolics representations (POS-tags, parse tree, logical expression) are readable whereas the vector representation is not.

represent a sentence: a syntactic structure gives us a skeleton for combining tokens and the remaining task is recursively applying *composition functions* along this skeleton in a bottom-up manner. The role of symbolic NLP is to provide syntactic structures whereas composition functions are implemented (and trained) by connectionist deep learning. The works of Pollack [1988] and Socher et al. [2010] are important inspirations: they recursively apply a neural network at every internal node of a parse tree, given by an external parser, in a bottom-up manner. Thanks to this hybrid, I can safely avoid the binding problem.

Although the "black box problem" is real, the vector representation type brings great advantages that make it worth the trouble:

- Computers are exceptionally good at numerical computation. Thanks to algebra, we therefore can build systems employing vectors and matrices (and tensors, in general) as core components that run efficiently on the current computer architecture, especially with the help of advanced GPUs.

- Classical symbolic representations are powerful for performing tasks requiring reasoning. However, there is a wide range of semantic tasks that this form is not appropriate, such as measuring semantic similarity and classification (e.g. sentiment analysis). The vector form, in contrast, fits such tasks well. This is due to the fact that we can measure the distance between two arbitrary vectors, and that there are an enormous number of machine learning techniques defining instances as vectors. In addition, as shown in Bowman et al. [2015a] and in other recent works, reasoning with vector representations is not impossible.

- We can consider human brain as a dynamical system the state of which changes over time. At a time the system's state, which is a set of neuron activations, can be represented by a point in a vector space. It is therefore reasonable to say that the vector form is, to some extent, biologically plausible.

In fact, the idea of using vectors to represent linguistic items could be traced back to the birth of distributional semantics [Firth, 1957a]. The emergence of the vector type for sentences could be credited to first neural-network-based attempts in replacing symbolic approaches on sentence processing tasks: the Recursive auto-associative model of Pollack [1988], the Simple recurrent network model of Elman [1990], and the work of St John and McClelland [1990], to name a few.

1.2 Contributions and Outline

This dissertation focuses on how to extract information from sentences, which is a crucial step for further processes, such as classification and translation. It draws a broad picture in which vector representations are the central representations, featuring in a hybrid of symbolic NLP and connectionist deep learning approaches, in turn based on the principle of compositionality.

Socher et al. [2010] show that the Recursive neural network (RNN) model has the potential to transform sentences into vectors, forming a basis for several recursive deep network models proposed in this dissertation. The outline of the rest of the dissertation is given as follows:

Chapter 2 This chapter presents the background for the whole dissertation. In the first section, I will introduce neural networks and how to train them with the back-propagation algorithm. In the second section, I review the literature on representing words and sentences using vectors.

Chapter 3 I propose to integrate context and more syntactic information into the computation of the RNN model. The vector representation of an internal node is now computed by taking into account, in addition to the representations of its children, representations of its syntactic label (e.g. N, VP), context words (e.g. one word before and one word after the phrase the node covers), and its head word. This idea is instantiated by a novel RNN-based reranker which achieves a significant improvement over the baseline, the Berkeley parser, on the Penn Treebank corpus. The content of this chapter is based on the following publication:

Le, Zuidema, and Scha [2013] Learning from errors: Using vector-based compositional semantics for parse reranking. *Proceedings ACL Workshop on Continuous Vector Space Models and their Compositionality*.

Chapter 4 The RNN model, like traditional compositional semantics approaches, focuses on how to represent phrases and sentences. That, in fact, limits its applicability because contexts, which play an important role in solving many NLP tasks, are not represented. In this chapter, I point out that the meaning of the context of a phrase can also be computed recursively, leading to a natural extension for the traditional compositional semantics framework: the Inside-Outside Semantic framework. I then propose its neural-net-based instance, called the Inside-outside recursive neural network (IORNN), which, in principle, is applicable to both supervised and unsupervised learning, in both a bottom-up and top-down manner. The new network model will be shown to perform on par with or better than the state-of-the-art (neural) models in word prediction, phrase-similarity judgements and semantic role labelling. The content of this chapter is based on the following publication:

Le and Zuidema [2014b] Inside-Outside Semantic: A Framework for Neural Models of Semantic Composition. *NIPS 2014 Workshop on Deep Learning and Representation Learning*.

Chapter 5 This chapter shows a case study illustrating that the IORNN model proposed in Chapter 4 can handle top-down processes. Using this model, I introduce an implementation for an ∞-order generative dependency grammar model which can overcome the problem of data sparsity. A reranker employing this model,

with the help of the MSTParser, will be shown to perform on par with state-of-the-art parsers on the Penn Treebank corpus. The content of this chapter is based on the following publication:

Le and Zuidema [2014a] The Inside-Outside Recursive Neural Network model for Dependency Parsing. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Chapter 6 For each parse tree the RNN model will build a network shaped according to a given tree. If the tree is deep, information from deep leaves propagating to the root is obscured, leading to the challenge of capturing long range dependencies. Besides, error signals propagating backward from the root to those leaves can vanish so quickly that learning has little effect. In this chapter, I address these two problems and propose a novel extension of the Long short term memory architecture that can work on trees. Empirical results show that the proposed model, the Recursive LSTM, significantly outperforms the traditional RNN model on the Stanford Sentiment Treebank. The content of this chapter is based on the following publication:

Le and Zuidema [2015c] Compositional Distributional Semantics with Long Short Term Memory. *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics (*SEM).*

Chapter 7 According to the principle of compositionality, syntax guides semantic composition. However, automatic parsers are far-from-perfect and, in many cases, are not aware of tasks to which they are applied. As a result, it raises the challenge of how to deal with the uncertainty resulting from automatic parsers. Moreover, constituents can be formed by different grammar rules with different branching factors, and a semantic composition model should take these issues into account. Being aware of the fact that the traditional RNN model is not able to solve any of these problems, in this chapter, I propose two key ideas: (i) a system should take as input a forest of parses instead of a single parse tree as in the traditional compositional semantics framework, and (ii) we can employ a convolutional network to handle varying branching factors, which also to some extent provide us with a solution to handle different grammar rules. The resulting model, the Forest Convolutional Network, will be shown to perform on par with state-of-the-art models on the Stanford Sentiment Treebank and on the TREC question dataset. The content of this chapter is based on the following publication:

Le and Zuidema [2015a] The Forest Convolutional Network: Compositional Distributional Semantics with a Neural Chart and without Binarization. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP).*

All the research and all implementations described in this dissertation were carried out by Phong Le. Willem Zuidema (and Remko Scha) helped shape the key ideas, edited

the publications referred to above and contributed some text to the introduction and conclusion sections of those papers.

Chapter 2

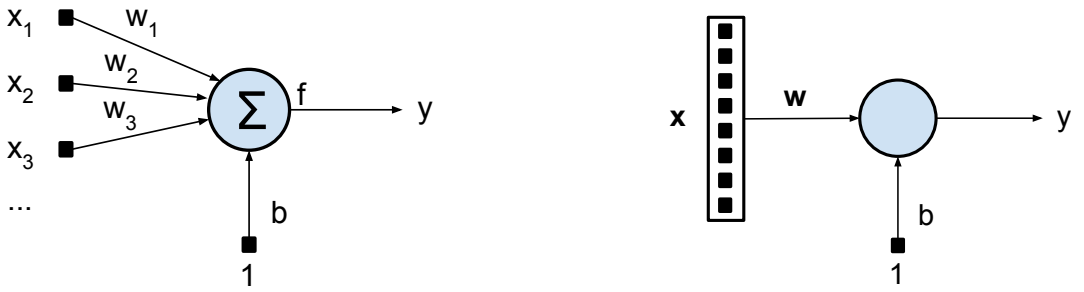
Background

The ultimate goal of this dissertation is to represent sentences by vectors based on the principle of compositionality and using neural networks. It is thus related to two research areas: artificial neural networks and vector space modelling. In this chapter, I will give a brief introduction to them. More specifically, Section 2.1 presents neural networks at a basic level such as artificial neurons, multi-layer feed-forward neural networks, the back-propagation algorithm. Section 2.2 introduces vector space modelling of language for representing words and phrases, and summarizes recent related work.

2.1 Artificial Neural Networks

2.1.1 Neurons

An artificial neuron (Figure 2.1a) is a computation unit which is loosely inspired by a biological neuron: an output is a function of a sum of weighted inputs. Technically speaking, let $x_1, x_2, ..., x_n \in \mathbb{R}$ be n scalar inputs associated with weights $w_1, w_2, ..., w_n \in \mathbb{R}$ and a bias $b \in \mathbb{R}$. The corresponding output y of the neuron is



(a) Classic form

(b) Vector form

Figure 2.1: Artificial neuron.

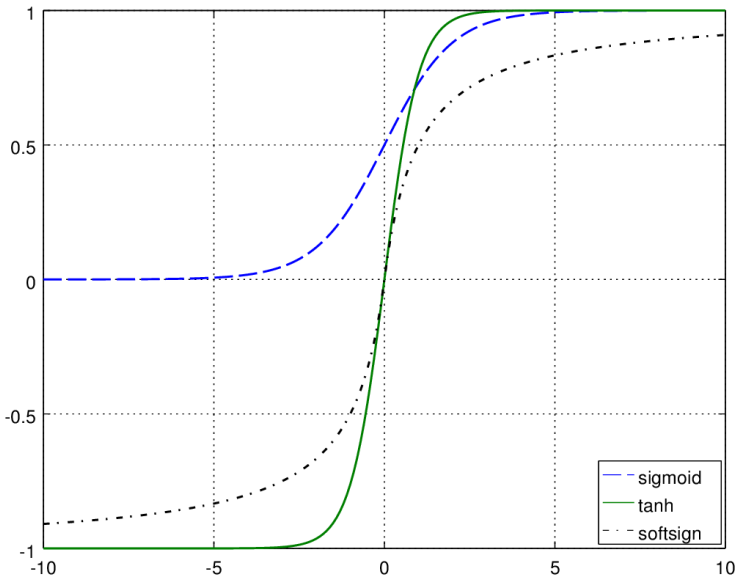


Figure 2.2: Activation functions: $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$, $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$, $\text{softsign}(x) = \frac{x}{1+|x|}$.

computed by:

$$z = \sum_{i=1}^{n} w_i x_i + b = w_1 x_1 + w_2 x_2 + ... + w_n x_n + b$$
$$y = f(z)$$

where f is an activation function (see Figure 2.2).

Another form which uses the vector notation is given as follows. Let $\mathbf{x} \in \mathbb{R}^n$ be an n-dim vector input, associated with $\mathbf{w} \in \mathbb{R}^n$ an n-dim weight vector and a bias $b \in \mathbb{R}$, the output y is computed by:

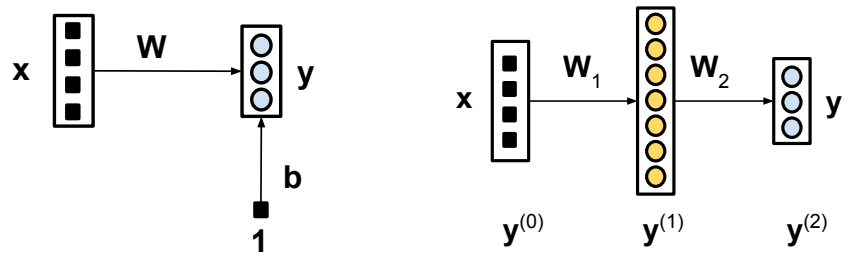
$$z = \mathbf{w}^T \mathbf{x} + b \; ; \quad y = f(z)$$

Because this form is more compact than the classic form, it will be used in the rest of this dissertation.

2.1.2 Multi-layer Neural Networks

If m neurons share the same input, we have a one-layer neural network (Figure 2.3a) which is described mathematically by:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \; ; \quad \mathbf{y} = f(\mathbf{z})$$



(a) (b)

Figure 2.3: (a) One-layer neural network. (b) Two-layer neural network (biases are removed for simplicity).

where $\mathbf{W} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$; f is now an element-wise activation function:

$$f\left(\begin{pmatrix} z_1 \\ z_2 \\ \dots \\ z_m \end{pmatrix}\right) = \begin{pmatrix} f(z_1) \\ f(z_2) \\ \dots \\ f(z_m) \end{pmatrix}$$

We call $\mathbf{x}$ and $\mathbf{y}$ respectively the input and output layer of a one-layer neural network.

Now, if we insert one layer of neurons right before the output layer of a one-layer neural network, we have a two-layer neural network (Figure 2.3b) in which the computation is given below:

$$\begin{aligned} & & \mathbf{y}^{(0)} &= \mathbf{x} \\ \mathbf{z}^{(1)} &= \mathbf{W}_1\mathbf{y}^{(0)} + \mathbf{b}_1 \; ; & \mathbf{y}^{(1)} &= f(\mathbf{z}^{(1)}) \\ \mathbf{z}^{(2)} &= \mathbf{W}_2\mathbf{y}^{(1)} + \mathbf{b}_2 \; ; & \mathbf{y}^{(2)} &= f(\mathbf{z}^{(2)}) \\ & & \mathbf{y} &= \mathbf{y}^{(2)} \end{aligned}$$

The inserted layer is called a *hidden layer*.

This hidden layer in fact decides the expressiveness of the whole neuron network. As pointed out by Minsky and Papert [1969], one-layer networks can not solve problems with non-linearly separable data (e.g., exclusive-or). A two-layer network, in contrast, can: the hidden layer can project the data onto another vector space in which they are now linearly separable (Figure 2.4). Interestingly, several theoretical proofs (e.g., Cybenko [1989]) show that two-layer neural networks are universal approximators: they are capable of approximating any continuous function to any desired degree of accuracy.

Do we need more hidden layers?

Although theoretically one hidden layer is enough, adding more hidden layers can be beneficial. This is because of two advantages of deep architectures (including deep neural networks, i.e., networks with many hidden layers)

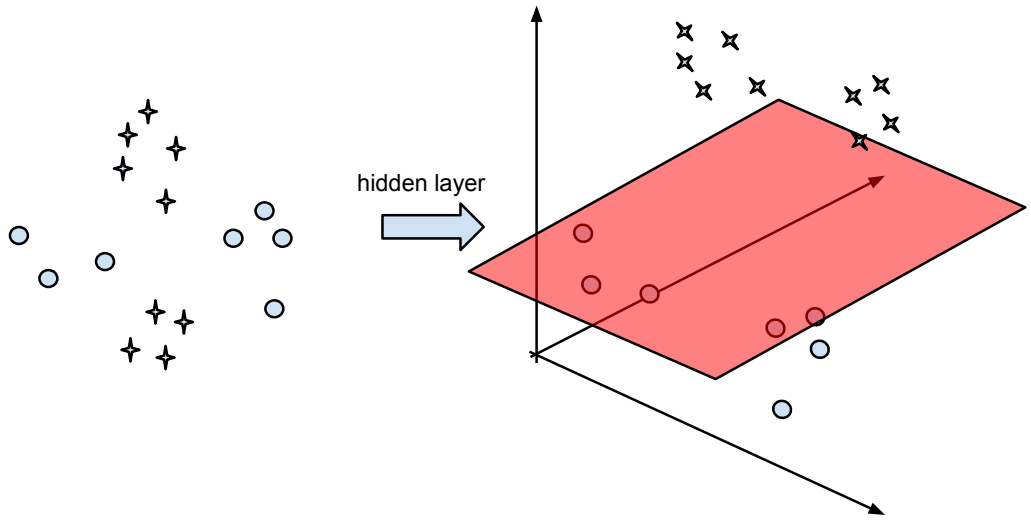


Figure 2.4: The role of the hidden layer in a two-layer feed-forward neural network is to project the data onto another vector space in which they are now linearly separable.

> (1) deep architectures promote the *re-use* of features, and (2) deep architectures can potentially lead to progressively more *abstract* features at higher layers of representations.

[Bengio et al., 2013b].

To understand these two advantages, let us take Figure 2.5 as an example.[1] Here, we build a four-layer neural network for face recognition: we feed in the network an image at the input layer and receive a distribution over a predefined set of objects at the output layer. This distribution tells us how likely the given image is of a person. After properly training this network on a large dataset, we will find that the first hidden layer detects low level features such as edges and simple shapes; the second layer can identify more complex shapes like noses, eyes; and so on. This means that higher layers (i.e. closer to the output layer) can extract more abstract features.

Now, let us look at neurons in the second hidden layer. Because they receive the same input which is the activation of the first layer, features extracted from the first hidden layer are actually *re-used* many times in the second hidden layer. This is desirable because, to identify noses and eyes, detecting edges and curves is necessary. It is therefore more efficient to have a lower layer extract common features which are shared among neurons in higher layers. For formal arguments, see Bengio et al. [2013b].

[1]Note that this example merely serves as an illustration. A deep network classifier could be more complicated, employing more kinds of layers such as convolutional layers and pooling layers [Lee et al., 2009]. For an online demo, see `http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html`.

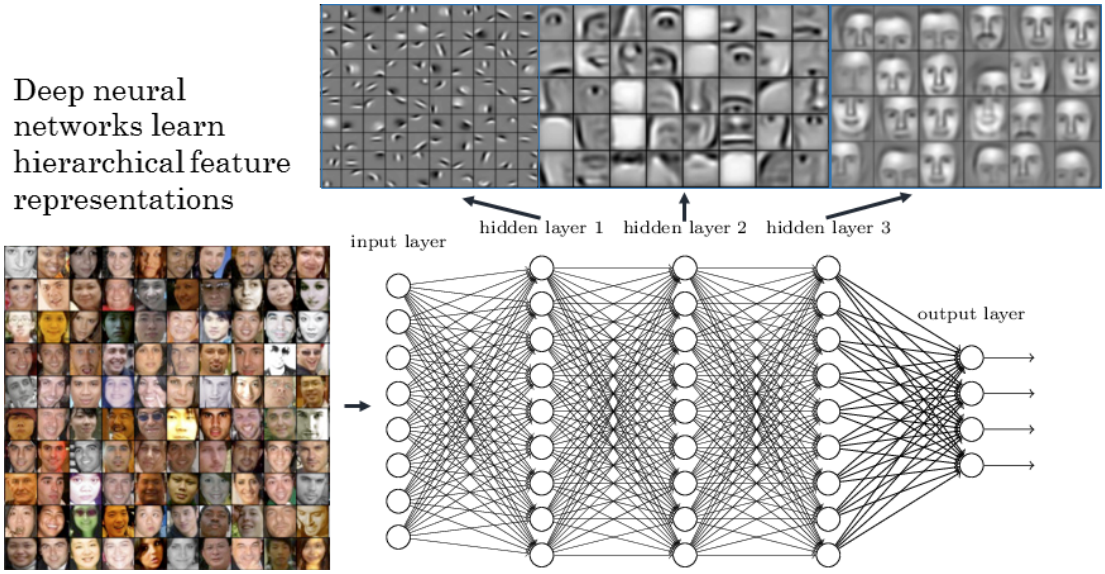


Figure 2.5: A four-layer neural network for face recognition. Higher layers (i.e. closer to the output layer) can extract more abstract features: the first hidden layer detects low level features such as edges and simple shapes; the second layer can identify more complex shapes like noses, eyes; and so on. Reproduced from `http://www.rsipvision.com/exploring-deep-learning/`.

Classification

Binary classification can be done with a one-neuron output layer of which the activation function is a sigmoid (or any function whose range is $[0, 1]$). An output can be interpreted as the probability $Pr(true|\mathbf{x})$ of assigning a predefined class to an input $\mathbf{x}$.

For N-class classification (with $N > 2$), a special N-neuron output layer called *softmax*[2] is often used: each neuron estimates the probability of assigning a class c to an input $\mathbf{x}$ by:

$$Pr(c|\mathbf{x}) = \text{softmax}(c) = \frac{e^{u(c,\mathbf{y}_{L-1})}}{\sum_{c' \in C} e^{u(c',\mathbf{y}_{L-1})}}$$

$$[u(c_1, \mathbf{y}_{L-1}), ..., u(c_N, \mathbf{y}_{L-1})]^T = \mathbf{W}\mathbf{y}_{L-1} + \mathbf{b}$$

where $\mathbf{y}_{L-1} \in \mathbb{R}^{n_{L-1}}$ is the activation of the layer right before the output layer; C is the set of all possible classes; $\mathbf{W} \in \mathbb{R}^{N \times n_{L-1}}, \mathbf{b} \in \mathbb{R}^N$ are a weight matrix and a bias vector.

Cost Function

For regression, given a training dataset $D_{train} = \{(\mathbf{x}_1, \mathbf{t}_1), .., (\mathbf{x}_n, \mathbf{t}_n)\}$, the squared error (average squared distance from predictions to ground truths) is often used as a

[2]Interestingly, a softmax function with $N = 2$ turns out to be equivalent to a sigmoid function.

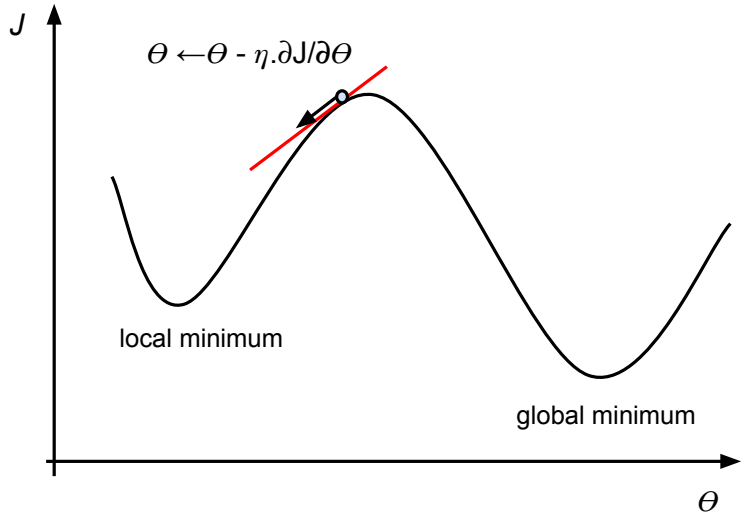


Figure 2.6: The gradient descent method. We iteratively update θ by adding to it an amount of $-\eta\frac{\partial J}{\partial \theta}$ until J converges. In this way, J gets smaller and smaller until it reaches a local minimum. This is similar to rolling a ball downhill.

cost function:

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} ||\mathbf{y}_i - \mathbf{t}_i||_2^2 + \underbrace{\frac{\lambda}{2}||\theta||_2^2}_{L_2 \text{ regularization}}$$

where $\mathbf{y}_i$ is the output of the network when the input is $\mathbf{x}_i$. The regularization term is added to avoid overfitting (by forcing parameters to remain close to 0).

For classification with $D_{train} = \{(\mathbf{x}_1, c_1), .., (\mathbf{x}_n, c_n)\}$, the cross-entropy, which is negative likelihood, is preferred

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} -\log y_{i,c_i} + \frac{\lambda}{2}||\theta||_2^2 \equiv \underbrace{-\frac{1}{n}\sum_{i=1}^{n} \log Pr(c_i|\mathbf{x}_i)}_{\text{log likelihood}} + \underbrace{\frac{\lambda}{2}||\theta||_2^2}_{L_2 \text{ regularization}}$$

where y_{i,c_i} is the c_i-th entry of $\mathbf{y}_i$. It is obvious that we can also use the squared error objective in this case. However, as pointed out by Golik et al. [2013], training using the squared error often ends up in a worse local optimum.

Training

Let θ be the parameter set of a neural network model, and $J(\theta)$ be a cost function that should be minimized with respect to θ. A traditional way to train a neural network is to employ the minibatch stochastic gradient descent method (Figure 2.6): given a training dataset D and a learning rate η,

- step 1: sample a minibatch D' of m examples from the training set D,
- step 2: compute $J(\theta)$ on D',
- step 3: update the parameters: $\theta \leftarrow \theta - \eta \frac{\partial J}{\partial \theta}$,
- step 4: if stopping criteria are not met, jump to step 1.

Thanks to the back-propagation algorithm [Werbos, 1974], which is a breakthrough in artificial neural network research, gradients can be computed efficiently. The key idea is to back-propagate errors from the output layer to the input layer using the chain rule in calculus: if $y = f(z)$ and $z = g(x)$ then

$$\frac{dy}{dx} = \frac{dy}{dz}\frac{dz}{dx}$$

In cases of using vectors, i.e. $\mathbf{y} = f(\mathbf{z})$ and $\mathbf{z} = g(\mathbf{x})$, we have a similar formula:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}}\frac{\partial \mathbf{y}}{\partial \mathbf{z}}$$

where for any vector $\mathbf{u} \in \mathbb{R}^m$, $\mathbf{v} \in \mathbb{R}^n$, [3]

$$\frac{\partial \mathbf{u}}{\partial \mathbf{v}} = \begin{pmatrix} \frac{\partial u_1}{\partial v_1} & \frac{\partial u_2}{\partial v_1} & \cdots & \frac{\partial u_m}{\partial v_1} \\ \frac{\partial u_1}{\partial v_2} & \frac{\partial u_2}{\partial v_2} & \cdots & \frac{\partial u_m}{\partial v_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial u_1}{\partial v_n} & \frac{\partial u_2}{\partial v_n} & \cdots & \frac{\partial u_m}{\partial v_n} \end{pmatrix}$$

Without loss of generality, let us take a two-layer neural network (Figure 2.7) as an example. We first back-propagate the error to the hidden layer as follows:

$$\frac{\partial J}{\partial \mathbf{y}^{(2)}} = \frac{\partial J}{\partial \mathbf{y}} \tag{2.1}$$

$$\frac{\partial J}{\partial \mathbf{z}^{(2)}} = \frac{\partial \mathbf{y}^{(2)}}{\partial \mathbf{z}^{(2)}}\frac{\partial J}{\partial \mathbf{y}^{(2)}} \; ; \quad \frac{\partial J}{\partial \mathbf{W}_2} = \frac{\partial J}{\partial \mathbf{z}^{(2)}}\mathbf{y}^{(1)T} \; ; \quad \frac{\partial J}{\partial \mathbf{y}^{(1)}} = \mathbf{W}_2^T\frac{\partial J}{\partial \mathbf{z}^{(2)}} \tag{2.2}$$

and then to the input layer:

$$\frac{\partial J}{\partial \mathbf{z}^{(1)}} = \frac{\partial \mathbf{y}^{(1)}}{\partial \mathbf{z}^{(1)}}\frac{\partial J}{\partial \mathbf{y}^{(1)}} \; ; \quad \frac{\partial J}{\partial \mathbf{W}_1} = \frac{\partial J}{\partial \mathbf{z}^{(1)}}\mathbf{y}^{(0)T} \; ; \quad \frac{\partial J}{\partial \mathbf{y}^{(0)}} = \mathbf{W}_1^T\frac{\partial J}{\partial \mathbf{z}^{(1)}}$$

$$\frac{\partial J}{\partial \mathbf{x}} = \frac{\partial J}{\partial \mathbf{y}^{(0)}}$$

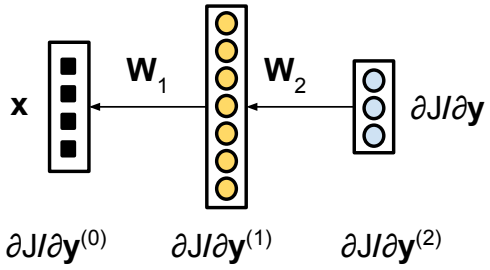


Figure 2.7: Back-propagation on a two-layer neural network.

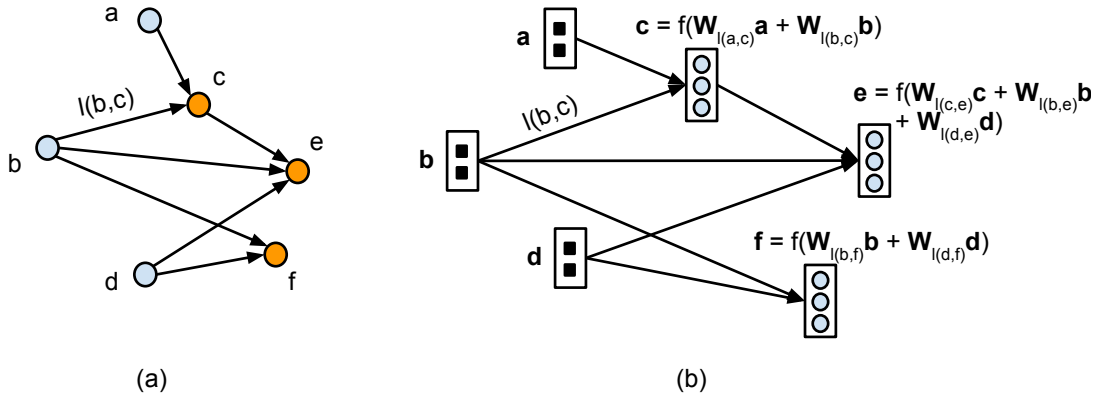


Figure 2.8: (a) A directed acyclic graph (DAG) is a graph without rings ($l(x, y)$ is the label of the edge from x to y). (b) A DAG neural network has a DAG topology (biases are removed for simplicity).

2.1.3 Directed-acyclic-graph Neural Networks

The topology of a multi-layer neural network is a directed sequence in which each node corresponds to a layer. In general, we can build neural networks with more sophisticated topologies, such as directed acyclic graphs (DAGs). We define a DAG neural network as a neural network whose topology is a DAG [Goller and Küchler, 1996] (see Figure 2.8).

Given a DAG, we build a neural network by assigning a layer to each node and information flows in this network are decided by the edges of the graph. Technically speaking, the activation of the layer at node p is a function of the activations of the layers at its children nodes (biases are removed for simplicity):

$$\mathbf{p} = f(\sum_{c:c\rightarrow p} \mathbf{W}_{l(c,p)}\mathbf{c})$$

where $\mathbf{W}_{l(x,y)} \in \mathbb{R}^{dim(\mathbf{y})\times dim(\mathbf{x})}$ and $\mathbf{x} \in \mathbb{R}^{dim(\mathbf{x})}$ is the activation of the layer at node x. For example, as shown in Figure 2.8, we can compute the activation of the layer at node c by taking into account the activations of the layers at node a and b as follows:

$$\mathbf{c} = f(\mathbf{W}_{l(a,c)}\mathbf{a} + \mathbf{W}_{l(b,c)}\mathbf{b})$$

We do the same to compute the activation of the layer at node f:

$$\mathbf{f} = f(\mathbf{W}_{l(b,f)}\mathbf{b} + \mathbf{W}_{l(d,f)}\mathbf{d})$$

Finally, we now can compute the activation of the layer at node e:

$$\mathbf{e} = f(\mathbf{W}_{l(c,e)}\mathbf{c} + \mathbf{W}_{l(b,e)}\mathbf{b} + \mathbf{W}_{l(d,e)}\mathbf{d})$$

Training a DAG neural network is similar to training a multi-layer neural network. We also use the stochastic gradient descent method where gradients are efficiently computed thanks to the back-propagation through structure algorithm [Goller and Küchler, 1996], which is a variant of the back-propagation algorithm but for DAGs. Let us take the example above to illustrate how this algorithm works (see Figure 2.9). Given that $\mathbf{e}$ and $\mathbf{f}$ are used in further computations, say classification with an objective J. We firstly compute $\frac{\partial J}{\partial \mathbf{e}}$ and $\frac{\partial J}{\partial \mathbf{f}}$. We then back-propagate the errors to their child nodes and grandchild nodes as follows:

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{c}} &= \frac{\partial \mathbf{e}}{\partial \mathbf{c}}\frac{\partial J}{\partial \mathbf{e}} \\
\frac{\partial J}{\partial \mathbf{a}} &= \frac{\partial \mathbf{c}}{\partial \mathbf{a}}\frac{\partial J}{\partial \mathbf{c}} \\
\frac{\partial J}{\partial \mathbf{b}} &= \frac{\partial \mathbf{c}}{\partial \mathbf{b}}\frac{\partial J}{\partial \mathbf{c}} + \frac{\partial \mathbf{e}}{\partial \mathbf{b}}\frac{\partial J}{\partial \mathbf{e}} + \frac{\partial \mathbf{f}}{\partial \mathbf{b}}\frac{\partial J}{\partial \mathbf{f}} \\
\frac{\partial J}{\partial \mathbf{d}} &= \frac{\partial \mathbf{e}}{\partial \mathbf{d}}\frac{\partial J}{\partial \mathbf{e}} + \frac{\partial \mathbf{f}}{\partial \mathbf{d}}\frac{\partial J}{\partial \mathbf{f}}
\end{aligned}$$

[3]Here I use the so called denominator-layout notation.

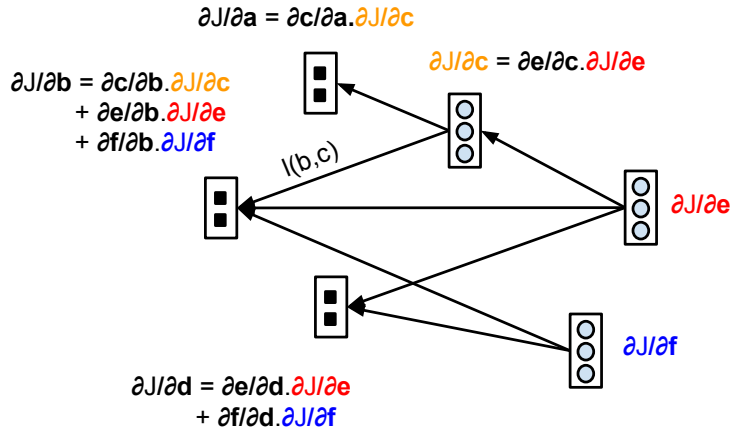


Figure 2.9: The back-propagation through structure.

In general, for each child node c, we have:

$$\frac{\partial J}{\partial \mathbf{c}} = \sum_{p:c \to p} \frac{\partial \mathbf{p}}{\partial \mathbf{c}} \frac{\partial J}{\partial \mathbf{p}}$$

Given those gradients, computing $\frac{\partial J}{\partial \mathbf{W}_{l(x,y)}}, \forall l(x,y)$ is now straightforward, as in Equation 2.2.

2.2 Vector Space Modelling for NLP

The artificial intelligence field, since its inception, has been struggling how to build machines that can understand human language. Traditional approaches are using formal languages (e.g., first order logic) to eliminate ambiguities and to do reasoning. For instance,

$$\text{All men are mortal :- } \forall x.man(x) \to mortal(x)$$
$$\text{Socrates is a man :- } man(Socrates)$$
$$\text{therefore } mortal(Socrates)$$

From late 20th century, the idea of using vectors to represent linguistic items, firstly at the lexical level, emerged. Firth introduced the concept of distributional semantics [Firth, 1957b] which is to use *context* as the key factor to determine the meaning of a word. This concept leads to a new philosophy of meaning representation: using vectors. Although there is controversy in calling it "semantics" (because it lacks compositionality and reasoning ability [Lenci, 2008]), the ability of measuring *semantic similarity* is the crucial reason for its wide-spread use in NLP. In addition, unlike formal semantics, building word meaning with distributional semantics is always in an unsupervised learning fashion; and thus we can exploit a huge amount of raw texts available on the Internet.

2.2.1 Representing Words

Many approaches that tackle the problem of how to represent word meaning by vectors, from simple counting (see Erk [2012] for a survey) to recent advanced machine learning techniques [Collobert et al., 2011b, Mikolov et al., 2013b], are based on the insight of Firth

> You shall know a word by the company it keeps

[Firth, 1957b]. It means that: the meaning of a word can be determined by words come with it (i.e., context). To make it clear, let us consider the following *bardiwac* example (by Stefan Evert). Imagine that we are reading an English book and encounter the strange word "bardiwac" in some sentences:

- He handed her her glass of *bardiwac*.
- Beef dishes are made to complement the *bardiwac*.
- Nigel staggered to his feet, face flushed from too much *bardiwac*.
- Malbec, one of the lesser-known *bardiwac* grapes, responds well to Australia's sunshine.
- I dined off bread and cheese and this excellent *bardiwac*.
- The drinks were delicious: blood-red *bardiwac* as well as light, sweet Rhenish.

Although we have no idea about the meaning of the word "bardiwac", we can guess it using contexts. From the first sentence, we can guess that "bardiwac" is a kind of (drinkable) liquid. The second sentence suggests that it is drank with beef dishes. With the third sentence, it seems that "bardiwac" can make us drunk. And so on. So finally, we can guess (with a very high confidence) that "bardiwac" is a kind of red strong wine.

Because we are human beings, we can guess the meaning of a unknown word easily. The question is: how can a computer do that? Below two popular approach classes will be introduced: *co-occurrence count* and *prediction*.

Co-occurrence count

For simplicity, we define a *context* of a target word in a sentence as the set of all words standing before or after the target word not exceeding k positions. This is called a window context. (If $k = \infty$, the context is the whole sentence.)

Let $\mathcal{V}$ be a vocabulary. A very simple method is to count how many times a word $u \in \mathcal{V}$ co-occurs with the target word $w \in \mathcal{V}$. This is equivalent to estimating the conditional distribution $Pr(U = u|W = w)$ using maximum likelihood. An illustration is shown in Figure 2.10. First, we collect sentences containing the target word w.

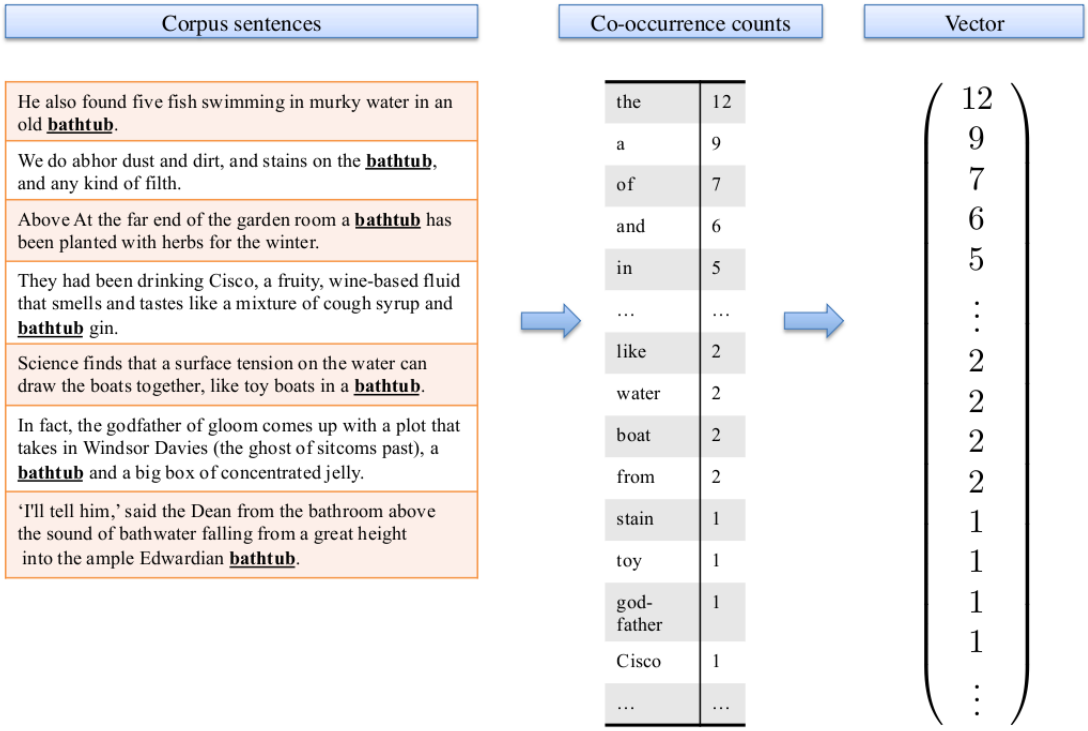


Figure 2.10: Step by step to build a vector for the word "bathtub" by simple count. Reproduced from Erk [2012]. Copyright 2012 by John Wiley and Sons.

We then for each word u count how many times u co-occurs with w within a defined window size k. Finally, we extract a vector containing all of those counts, which is used to represent the target word w. Estimating the conditional distribution is simple by:

$$P(U = u|W = w) = \frac{\#\text{u and w co-occur}}{\#w} \tag{2.3}$$

However, in order to come up with meaningful vectors, there are some details that should be concerned with. First, which u's should be counted. It is clear that function words (e.g., "a", "an", "how", "what"), which are extremely frequent, carry little information about the meanings of target words. With this simple count, every word plays an equal role. By contrast, rare words tend to contribute more important information. We thus need one more step called *weighting scheme*. One of popular schemes is point-wise mutual information:

$$PMI(u, w) = \frac{P(U = u|W = w)}{P(U = u)} \tag{2.4}$$

where $P(U = u)$ is the probability that u appears in the corpus. Intuitively speaking, rare words should get higher weights than frequent words.

In practice, the resulting vectors are high dimensional (e.g. 2000 or more). Therefore, a dimension reduction method like PCA or non-negative matrix factorization is often employed in the end (see Dinu and Lapata [2010] for an example).

Prediction

The key idea of this approach class is very similar to how we answer "filling in blanks" questions in TOEFL and IELTS tests:

> _____ is the study of numbers, equations, functions, and geometric shapes and their relationships.
>
> a. physics
>
> b. mathematics
>
> c. geography
>
> d. theology

If an examinee gets a good result (i.e. an accuracy substantially higher than 25% of random guess), she is supposed to know the meanings of correct words as well as their contexts. Similarly, if resulting word vectors can help us to do well on such questions, those vectors are supposed to capture the meanings of the corresponding words.

Let us assume that each word $v \in \mathcal{V}$ is represented by a vector $\mathbf{v} \in \mathbb{R}^d$ and there is a mechanism to compute the probability $P(W = w|\mathcal{U} = (u_1, u_2, ..., u_l))$ of the event that a target word w appears in a context $(u_1, u_2, ..., u_l)$. We are going to find a vector $\mathbf{v}$ for each word v such that those probabilities are as high as possible for each w and its context $(u_1, u_2, ..., u_l)$.

There are many approaches sharing this idea. Here, I introduce the one proposed by Bengio et al. [2003], which is to my knowledge, the very first approach in this class. First, we compose words $u_1, ..., u_l$ into a vector $\mathbf{x}$ as below:

$$\mathbf{x} = tanh(\mathbf{b} + \mathbf{V}_1\mathbf{u}_1 + ... + \mathbf{V}_l\mathbf{u}_l) \tag{2.5}$$

where $\mathbf{V}_i \in \mathbb{R}^{n \times d}$ and $\mathbf{b} \in \mathbb{R}^n$. We then project $\mathbf{x}$ onto a $|\mathcal{V}|$-dim vector space:

$$\mathbf{a} = \mathbf{c} + \mathbf{W}\mathbf{x} \tag{2.6}$$

Finally, using a $softmax$, we estimate the probability that w_k, the k-th word in $\mathcal{V}$, appears in the context $(u_1, ..., u_l)$

$$P(W = w_k|\mathcal{U} = (u_1, u_2, ..., u_l)) = softmax(k, \mathbf{a}) = \frac{e^{a_k}}{\sum_{i=1}^{|\mathcal{V}|} e^{a_i}} \tag{2.7}$$

Now, let $\theta = \langle \mathbf{w}_1, ..., \mathbf{w}_{|\mathcal{V}|}, \mathbf{V}_1, ..., \mathbf{V}_l, \mathbf{b}, \mathbf{W}, \mathbf{c} \rangle$. Training this neural network is to maximize the log-likelihood (i.e., minimizing the cross entropy objective function) of the training data with respect to θ:

$$L(\theta) = \sum_{w,(u_1,...,u_l)} \log P(W = w|\mathcal{U} = (u_1, ..., u_l)) \tag{2.8}$$

where the sum is over all possible ⟨target word w, context $(u_1, ..., u_l)$ ⟩ pairs found in the corpus.

Recent approaches use different neural network models. Mikolov et al. [2010] employ (simple) recurrent network networks of which resulting word vectors are shown to capture linguistic regularities [Mikolov et al., 2013c] like

$$\overrightarrow{king} - \overrightarrow{man} + \overrightarrow{woman} = \overrightarrow{queen}$$

Another model is the skip-gram model [Mikolov et al., 2013a] which is trying to predict contextual words using target word vectors. Huang et al. [2012]'s model employs global information captured at the document level.

Because the computational cost of a softmax function (Equation 2.8) is linearly proportional to the vocabulary size, with a very large vocabulary naively using a softmax function leads to a huge demand of computation. To overcome this problem, one can use a hierarchical softmax [Morin and Bengio, 2005] whose complexity is a log of the vocabulary size. Sampling techniques like noise contrastive estimation can also be used [Mnih and Teh, 2012].

Maximizing log-likelihood is not the only way to train a model. Collobert et al. [2011b] propose to use a ranking criterion: for every word w' which is different from the target word w, w should be ranked higher than w'.

2.2.2 Representing Sentences

Those techniques for representing words in theory can be applied to larger items like phrases and sentences. However, in practice, lack of data is so serious that they are mostly useless. In this section, I introduce two classes of approaches: *composition* and *non-composition*.[4]

Composition

The solution that formal semantics research found is to rely on the principle of compositionality. With the lambda calculus, composition turns out to be very elegant: lambda function application and reduction. For instance,

$$\begin{aligned} \text{John :- } & john \\ \text{run :- } & \lambda x.run(x) \\ \text{John run :- } & \big(\lambda x.run(x)\big)(john) = run(john) \end{aligned}$$

[4]It is worth mentioning that, in this dissertation, I draw a border between composition and non-composition: any approach that is not explicitly based on the principle of compositionality is considered non-compositional. This border however is not absolute because some models, such as Kalchbrenner et al. [2014], are non-compositional according to this definition, but are shown to be able to find out hidden structures similar to syntactic trees.

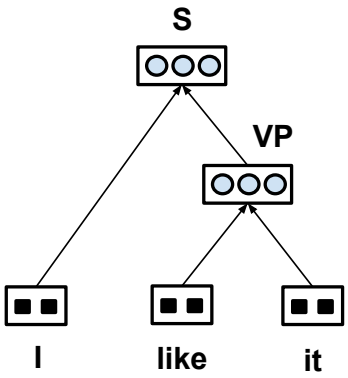


Figure 2.11: Compute sentence vector in a bottom manner. Firstly we compute a vector for the VP and then compute a vector for the S .

For vectorial semantics, because there is unfortunately no such a beautiful mathematical solution like that, looking for universal composition functions is still very challenging. However, we can build a general framework based on that principle as follows.

Given that $\mathbf{x}$ and $\mathbf{y}$ are vectors representing two items x and y, respectively. The task is to find a composition function f so that the output is a vector representing constituent $p = xy$:

$$\mathbf{p} = f(\mathbf{x}, \mathbf{y}, R, K) \tag{2.9}$$

where R is the syntactic rule that combines x and y, K is background knowledge [Mitchell and Lapata, 2008].

Computing a vector for a sentence is thus, like in formal semantics, carried out in a bottom up manner on a given parse tree. An illustration is given in Figure 2.11. Assuming that there are vectors representing words (we can randomly initialize them or use any technique in Section 2.2.1). We first compute a vector for the VP:

$$\overrightarrow{like\ it} = f(\overrightarrow{like}, \overrightarrow{it}, VP \rightarrow ADJ\ PRP)$$

and then go higher up in the hierarchy to compute a vector for the whole sentence

$$\overrightarrow{I\ like\ it} = f(\overrightarrow{I}, \overrightarrow{like\ it}, S \rightarrow NP\ VP)$$

Approaches in this class mainly differ by the composition functions they make use of. Below I will introduce three of them.

Mixture-based composition The simplest approach is to use vector element-wise operations like addition and multiplication [Mitchell and Lapata, 2008]

- Additive: $\mathbf{p} = \mathbf{x} + \mathbf{y}$
- Multiplicative: $\mathbf{p} = \mathbf{x} \odot \mathbf{y}$ (i.e., $p_i = x_i \times y_i$)

where $p = xy$ is formed by combined x and y. The name of this model comes from the fact that an entry of the resulting vector can be considered as a mixture (via addition or multiplication) of the two corresponding entries of the two input vectors.

Despite its simplicity, this approach works surprisingly well in composing noun-noun and adjective-noun phrases [Blacoe and Lapata, 2012]. However, from the theoretical point of view, this approach is unsatisfactory. Because those operations are commutative, the word order is not taken into account, i.e.,

$$\overrightarrow{man\ bit\ dog} = \overrightarrow{dog\ bit\ man}$$

A more sophisticated way is to linearly combine those two operations:

$$\mathbf{p} = \alpha\mathbf{x} + \beta\mathbf{y} + \gamma\mathbf{x}_i \odot \mathbf{y}$$

Function-application-based Composition The idea is to resemble composition in formal semantics. In formal semantics, words such as adjectives and verbs are *functors* which are to modify *arguments* representing other words like nouns; and composition is simply function application. Based on this, with the slogan "composition is largely a matter of *function application*" Baroni et al. [2013] propose using tensors[5] to represent words:

- first-order tensors (i.e., vectors $\in \mathbb{R}^n$) are to represent nouns,
- second-order tensors (i.e., matrices $\in \mathbb{R}^{n\times m}$) are to represent adjectives, intransitive verbs,
- third-order tensors (i.e., $\in \mathbb{R}^{n\times m\times k}$) are for transitive verbs, and so on.

Composition is carried out through tensor production. For example composing an adjective and a noun is a matrix-vector multiplication like:

$$\overrightarrow{old\ dog} = f_{old}(\overrightarrow{dog}) = \mathbf{OLD} \times \overrightarrow{dog} \tag{2.10}$$

[5] Tensors describe linear relations between vectors, scalars, and other tensors. An n-dim vector is a first-order n-shape tensor, an $n \times m$ matrix is a second-order $n \times m$-shape tensor. A third-order $(n \times m) \times k$-shape tensor $\mathbf{T}$ describes a linear map from a k-dim vector $\mathbf{v}$ to an $n \times m$ matrix $\mathbf{M}$

$$\mathbf{M} = \mathbf{T} \times \mathbf{v}$$

where

$$M_{m,n} = \sum_k T_{m,n,k} v_k$$

Generally, a $(I_1 \times ... \times I_m) \times (J_1 \times ... \times J_n)$-shape tensor $\mathbf{T}$ describes a linear map from a $J_1 \times ... \times J_n$-shape tensor $\mathbf{V}$ to an $I_1 \times ... \times I_m$-shape tensor $\mathbf{M}$ by

$$M_{i_1,...,i_m} = \sum_{j_1,...,j_n} T_{i_1,...,i_m,j_1,...,j_n} V_{j_1,...,j_n}$$

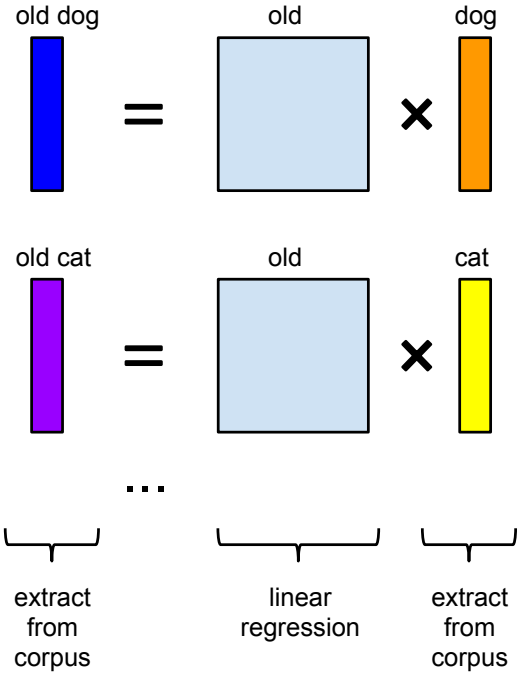


Figure 2.12: Learning a matrix for an adjective using linear regression.

Training this model is to find proper tensors for words. Given the fact that approaches for word vectors are not applicable to second and higher-order tensors, Baroni et al. [2013] and Grefenstette et al. [2013] employ linear regression with co-occurrence counts as training signals. First, co-occurrence count is used to find vectors for nouns like "dog", vectors for noun phrases "old X" such as "old dog", "old cat". From those vectors representing "old X" and "X", linear regression is used to estimate a matrix for "old" (see Figure 2.12). Higher-order tensors are estimated with the same mechanism but with a technique called multi-step linear regression.

From theoretical point of view, the function-application-based approach is a beautiful model, which could be considered as inheriting the philosophy of formal semantics. However, learning tensors for lexical items is the most serious weakness. This is because the model requires a very large number of parameters (a single transitive verb needs n^3, i.e., $1M$ if $n = 100$, parameters). Besides, we need to collect context distributions of phrases like 'old dog' as in the above example. This becomes problematic when phrases are rare.

Recursive neural networks Because two-layer feed forward neural networks are universal approximators, they can theoretically approximate *true* composition functions (if they exist and are measurable). Therefore, it is natural to think of using a feed forward neural net as a composition function. This leads to a model called Recursive

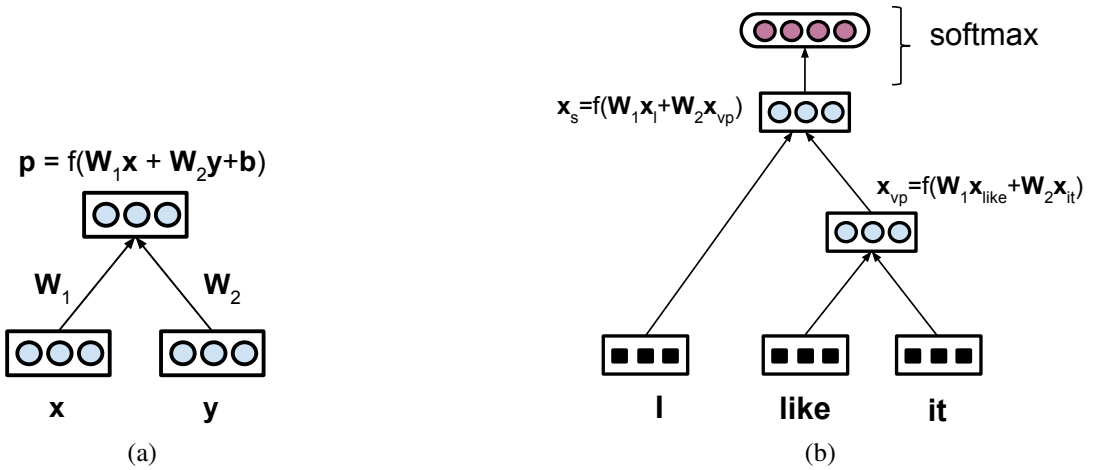


Figure 2.13: (a) A feed-forward neural network as a composition function. (b) Recursive neural network for sentence classification.

neural network (RNN).[6] This idea is first proposed by Pollack [1990] and Goller and Küchler [1996]. It then became popular in NLP thanks to the work series of Socher and colleagues [Socher et al., 2010, 2011c, 2012].

A composition function for this model is a network consisting of two layers (Figure 2.13a): an input layer with $2n$ nodes for pairs of two children, an output layer with n nodes for parents. Technically speaking, given a production $p \rightarrow x\ y$, the computation is given by

$$\mathbf{p} = f(\mathbf{W}_1\mathbf{x} + \mathbf{W}_2\mathbf{y} + \mathbf{b}) \tag{2.11}$$

where $\mathbf{x}, \mathbf{y}, \mathbf{p} \in \mathbb{R}^n$ are vectors representing x, y, p. $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{n \times n}$, $\mathbf{b} \in \mathbb{R}^n$ are weight matrices and a bias vector. f is an activation function (e.g., *tanh*). It is easy to recognize that an RNN is in fact a DAG neural network because a tree is a directed acyclic graph. Therefore, we can employ the method presented in Section 2.1.3 to efficiently train an RNN.

The RNN model differs from the above two models in that it is nonlinear (if the activation function f in Equation 2.11 is nonlinear, e.g. $sigmoid$, $tanh$). Although it is not clear whether the true composition functions are linear or nonlinear, the nonlinearity obviously makes the RNN model more expressive and a wide range of choices for f make it more flexible. Another difference lies in motivations. Because the mixture-based and function-application-based models emerged from the expectation of capturing *general* compositionality based on raw texts only like the way word vectors are built, they are considered as task-independent. The RNN model, on the other hand, is task-oriented, i.e., the resulting composition functions are for specific tasks (e.g.,

[6]The acronym 'RNN' is often referred to *recurrent* neural network in the neural network community. However, because this work is mainly concerned with *recursive* neural networks, I reserve this acronym for *recursive* neural network, and always state the full name for *recurrent* ones.

sentiment prediction). Annotated data are thus required. It is however not true that we can not make the other two models task-oriented nor that we can make the RNN model task-independent. In the former case, injecting task-oriented training signals as in supervised learning can help. In the latter case, training signals can be taken from contexts (see Dinu et al. [2013] for an example).

Although the RNN model is simple, its elegance in terms of modelling and training makes it popular in the NLP community. A bundle of its extensions have been proposed, mainly along two directions: (i) proposing more expressive composition functions and (ii) extending the topology. In the rest of this part, I will introduce three such extensions: the Matrix-Vector RNN model and the Recursive Neural Tensor Network model for (i) and the Bidirectional RNN model for (ii).

Matrix-Vector RNNs (MV-RNN) The beauty of the function-application-based model is that it explicitly point out two main roles of an item (word or phrase): an item (e.g., adjective) could be used to modify the meaning of another item (e.g., noun), and could be modified by another item (e.g., adverb). The MV-RNN model [Socher et al., 2012] is an extension of the traditional RNN model such that the two roles are explicitly employed. The key idea is to represent any item a by a matrix-vector pair $\langle \mathbf{A}, \mathbf{a} \rangle$ where $\mathbf{a} \in \mathbb{R}^n$ represents the core meaning of a and $\mathbf{A} \in \mathbb{R}^{n \times n}$ captures a's ability of modifying another item. The composition for a production $p \rightarrow x\ y$ is given by

$$\mathbf{p} = f\big(\mathbf{W}_1(\mathbf{Y}\mathbf{x}) + \mathbf{W}_2(\mathbf{X}\mathbf{y}) + \mathbf{b}\big) \tag{2.12}$$

$$\mathbf{P} = \mathbf{U}_1\mathbf{X} + \mathbf{U}_2\mathbf{Y} \tag{2.13}$$

where $\mathbf{W}_1, \mathbf{W}_2, \mathbf{U}_1, \mathbf{U}_2 \in \mathbb{R}^{n \times n}$, and $\mathbf{b} \in \mathbb{R}^n$.

Because now each word in the vocabulary must be represented by a matrix-vector pair, the size of a MV-RNN is about $n + 1$ times larger than the size of an RNN where n is the dimension of the vector space. Socher et al. [2012], to reduce the model size, thus propose to use a low-rank matrix approximation method

$$\mathbf{A} = \mathbf{U}\mathbf{V} + diag(\mathbf{c})$$

where if $\mathbf{A} \in \mathbb{R}^{n \times n}$ then $\mathbf{U} \in \mathbb{R}^{n \times r}$, $\mathbf{V} \in \mathbb{R}^{r \times n}$, $\mathbf{c} \in \mathbb{R}^n$, and $r \ll n$.

Empirical results show that the MV-RNN model outperforms the traditional RNN model in predicting sentiments and classifying sentiment relationships [Socher et al., 2012].

Recursive Neural Tensor Networks (RNTN) The MV-RNN model above requires a large number of parameters (even after employing the low-rank matrix approximation method) and this number grows rapidly regarding to the vocabulary size. Socher et al. [2013b] thus suggest that it is cognitively more plausible to have one powerful composition function with a fixed number of parameters. Their proposal is a tensor-based

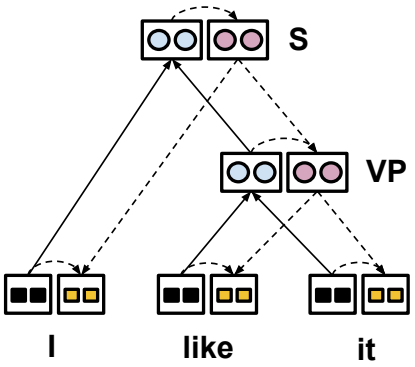


Figure 2.14: Bidirectional Recursive neural network. The dashed lines correspond to the top-down component.

composition[7] which employs a tensor product (see the definition in Footnote 5)

$$\mathbf{p} = f\Big(\underbrace{\mathbf{V}\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix}}_{\text{tensor product}} \begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} + \underbrace{\mathbf{W}_1\mathbf{x} + \mathbf{W}_2\mathbf{y} + \mathbf{b}}_{\text{standard RNN}} \Big)$$

where $\mathbf{V} \in \mathbb{R}^{n\times 2n\times 2n}$ is a third-order tensor which is to transform a $2n$-dim vector to an $n \times 2n$ matrix. Comparing this to Equation 2.13, we can see that this RNTN model can be considered as a compact form of the MV-RNN model where a vector for each item now captures both its core meaning and modification ability. Multiplying the tensor $\mathbf{V}$ to $\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix}$ is thus to extract the modifying matrices $\mathbf{X}$ and $\mathbf{Y}$.

Tested on the Stanford Sentiment Treebank[8] for sentiment analysis, this RNTN model outperforms the MV-RNN model, and becomes a strong baseline for this task.

Bidirectional RNNs One restriction of the RNN model is that the information flows on parse trees bottom-up only, a node in a parse tree is thus not aware of its surrounding context. The RNN model therefore is not applicable to several NLP tasks such as semantic role labelling in which contexts play an important role to determine some desired property of constituents. The solution proposed by Irsoy and Cardie [2013] is to add a top-down component to, at every node in a parse tree, compute one more vector capturing information of the surrounding structure (see Figure 2.14).

Given a parse tree, we assign two vectors to each node p: $\mathbf{p}^{\uparrow} \in \mathbb{R}^n$ representing the node itself and $\mathbf{p}^{\downarrow} \in \mathbb{R}^n$ representing the structure surrounding it. Let x and y be p's left and right child, respectively. We compute $\mathbf{p}^{\uparrow}$ exactly like in the RNN model. Now, if p is the root node then

$$\mathbf{p}^{\downarrow} = f(\mathbf{V}\mathbf{p}^{\uparrow} + \mathbf{b}^{\downarrow})$$

[7] I present an equivalent form in order to connect this model to the MV-RNN model.

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

Otherwise, let r be p's parent, then

$$\mathbf{p}^{\downarrow} = \begin{cases} f(\mathbf{W}_L^{\downarrow}\mathbf{r}^{\downarrow} + \mathbf{V}\mathbf{p}^{\uparrow} + \mathbf{b}^{\downarrow}) & \text{if } p \text{ is the left child of } r \\ f(\mathbf{W}_R^{\downarrow}\mathbf{r}^{\downarrow} + \mathbf{V}\mathbf{p}^{\uparrow} + \mathbf{b}^{\downarrow}) & \text{if } p \text{ is the right child of } r \end{cases}$$

where $\mathbf{V}, \mathbf{W}_L^{\downarrow}, \mathbf{W}_R^{\downarrow} \in \mathbb{R}^{n\times n}$ and $\mathbf{b}^{\downarrow} \in \mathbb{R}^n$. Because of the way information flows, it is easy to see that $\mathbf{p}^{\downarrow}$ represents both p and its surrounding structure.

Fusing information of both p and its context in $\mathbf{p}^{\downarrow}$ is a weak point. In fact, the Bidirectional RNN model is not applicable to top-down generation as well as tasks concerning incomplete sequences. In Chapter 4, I will discuss this in more details and propose a network model, the Inside-outside Recursive neural network, to overcome this problem.

Non-composition

Some approaches try to compute vectors for sentences without (explicitly) using syntactic structures as glue to combine sub-components. They are thus non-compositional.

Recurrent Neural Networks A neural network is *recurrent* if it has at least one directed ring in its structure. In NLP, the simple recurrent neural network (SRN) model proposed by Elman [1990] (see Figure 2.15a) and its extensions are used to tackle sequence-learning problems, such as machine translation [Sutskever et al., 2014] and language modelling [Mikolov et al., 2010].

In an SRN, an input $\mathbf{x}_t$ is fed to the network at each time t. The hidden layer $\mathbf{h}$, which has activation $\mathbf{h}_{t-1}$ right before $\mathbf{x}_t$ comes in, plays a role as a memory capturing the whole history $\big(\mathbf{x}_0, ..., \mathbf{x}_{t-1}\big)$. When $\mathbf{x}_t$ comes in, the hidden layer updates its activation by:

$$\mathbf{h}_t = g\big(\mathbf{W}_x\mathbf{x}_t + \mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{b}\big)$$

where $\mathbf{W}_x \in \mathbb{R}^{n\times dim(\mathbf{x}_t)}$, $\mathbf{W}_h \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$ are weight matrices and a bias vector; g is an activation function. This network model thus, in theory, can be used to estimate conditional probabilities with long histories. Besides, computing gradients is efficient thanks to the back-propagation through time algorithm [Werbos, 1990].

Because the SRN can handle sequences with different lengths, we can use it to process sentences. For instance, Figure 2.15b shows how we can use the SRN model for language modelling: the idea is to try to predict which word will come next. In this way, when we reach the end of a sentence, the hidden layer activation captures the whole sentence. In other words, we can use the activation of the hidden layer at that time point to represent the whole sentence. Figure 2.15b also shows that the SRN model is in fact a special case of the RNN model: the input tree is always left-branching.

Although in theory recurrent neural networks are, at least, as expressiveness as Turing machines [Hyötyniemi, 1996], training them is difficult because of the vanishing

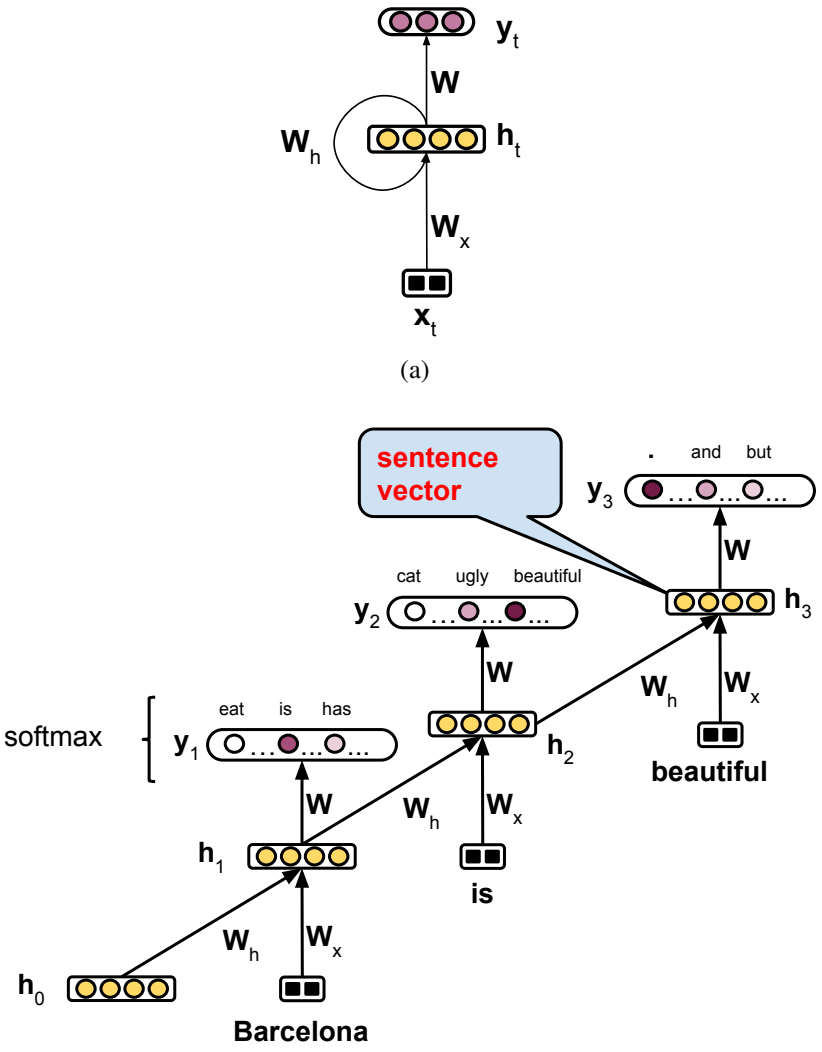


Figure 2.15: (a) Simple recurrent network (SRN) and (b) when we unfold it. Notice that $\mathbf{h}_0$ can be simply $\vec{0}$.

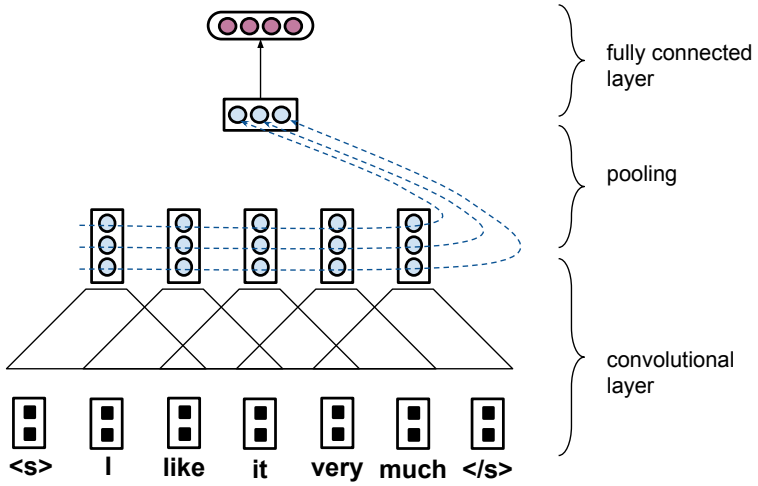


Figure 2.16: Convolutional neural network (one convolutional layer and one fully connected layer) with a window-size-3 kernel.

gradient problem and the problem of how to capture long range dependencies [Hochreiter et al., 2001]. Two alternatives, which become popular recently, are the Long short term memory architecture [Hochreiter and Schmidhuber, 1997] (which is discussed in more detail in Chapter 6) and gated recurrent neural networks [Chung et al., 2014].

Convolutional Networks A very popular class of neural-network-based models employs convolutional networks. A convolutional neural network (CNN) [LeCun et al., 1998] is also a feed-forward neural network; it consists of one or more convolutional layers (often with a pooling operation) followed by one or more fully connected layers. This architecture was invented for computer vision. It then has been widely applied to solve NLP tasks [Collobert et al., 2011b, Kalchbrenner et al., 2014, Kim, 2014].

To illustrate how a CNN works, the following example uses a simplified version of the model proposed by Collobert et al. [2011b], which consists of one convolutional layer with a max pooling operation, followed by one fully connected layer (Figure 2.16). This CNN uses a *kernel* with the window size 3. When sliding this kernel along the sentence "⟨s⟩ I like it very much ⟨/s⟩", we get five vectors:

$$\begin{aligned}
\mathbf{u}^{(1)} &= \mathbf{W}_1\mathbf{x}_{<s>} + \mathbf{W}_2\mathbf{x}_I + \mathbf{W}_3\mathbf{x}_{like} + \mathbf{b}_c \\
\mathbf{u}^{(2)} &= \mathbf{W}_1\mathbf{x}_I + \mathbf{W}_2\mathbf{x}_{like} + \mathbf{W}_3\mathbf{x}_{it} + \mathbf{b}_c \\
&\ldots \\
\mathbf{u}^{(5)} &= \mathbf{W}_1\mathbf{x}_{very} + \mathbf{W}_2\mathbf{x}_{much} + \mathbf{W}_3\mathbf{x}_{</s>} + \mathbf{b}_c
\end{aligned}$$

where $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3 \in \mathbb{R}^{d\times m}$ are weight matrices, $\mathbf{b}_c \in \mathbb{R}^d$ is a bias vector. A max pooling operation (i.e., element-wise max) is then applied to those resulting vectors:

$$\mathbf{x} = \left[\max_{1\le i\le 5} \mathbf{u}_1^{(i)}, ..., \max_{1\le i\le 5} \mathbf{u}_j^{(i)}, ...\right]^T$$

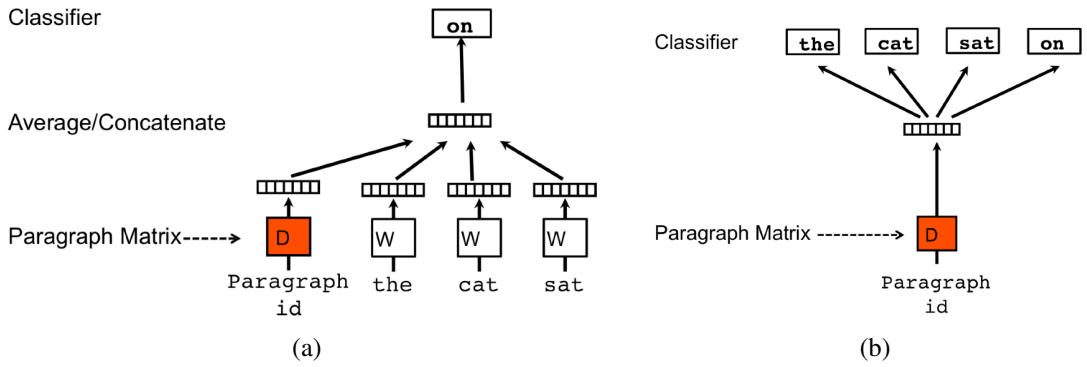


Figure 2.17: ParagraphVector. (a) Distributed Memory (b) Distributed Bag-of-words. Reproduced from Le and Mikolov [2014].

Finally, a fully connected layer is employed:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

where $\mathbf{W}$, $\mathbf{b}$ are a real weight matrix and a bias vector, respectively; f is an activation function.

Intuitively, a window-size-k kernel extracts (local) features from k-grams, and is thus able to capture k-gram composition. The max pooling operation is for reducing dimension and forcing the network to discriminate important features from others by assigning high values to them. For instance, if the network is used for sentiment analysis, local features corresponding to k-grams containing the token "like" should receive high values in order to be propagated to the top layer.

ParagraphVector Differing from all of the above models where the information direction is from word to sequence (i.e., the vectors of the words in a sequence are composed to form a vector representing the whole sequence), the ParagraphVector method proposed by Le and Mikolov [2014] is in the opposite direction: from sequence to word. For each sequence, it searches for a vector that can help to predict which words occur in the sequence. If the prediction is successful, this vector is used to represent the whole sequence.

From the technical perspective, this method is very much similar to the *prediction* approach presented in Section 2.2.1. As a result, any word vector model in that class can be used to produce word-sequence vectors. Le and Mikolov [2014] propose two models: distributed memory (Figure 2.17a) and distributed bag-of-words (Figure 2.17b). In Figure 2.17, the i-th column of the matrix $\mathbf{D}$ represents the i-th document (which could be a phrase, a sentence, or a paragraph) in corpora. The distributed memory model thus considers a whole document as a part of the context of every word in that document. The distributed bag-of-words model, on the other hand, considers a document as a special token whose context consists of all of its words. Concate-

nating word-sequence vectors produced by the two models are shown to be useful for sentiment analysis and information retrieval.

One can recognize that this method is similar to reading comprehension tests: a student is asked to read a paragraph and then answer questions about the content of that paragraph. If her answers are correct, she is supposed to understand the paragraph. Here in the model, questions like "what did the cat sit on?" or "tell me which information can be extracted from the paragraph!" are rephrased by "what should come after *the cat sat on*?" or "tell me what you read in the paragraph!" Although an exact equivalent of the reading comprehension test would have the model learn from paraphrases, paraphrasing is eliminated in this case for simplicity.

Chapter 3

Discriminative Reranking with Recursive Neural Networks

In this chapter, I address the problem of how to use vector representations to improve syntactic parsing, by using a hybrid reranking method: a k-best list generated by a statistical symbolic parser is reranked based on acceptability scores given by a compositional, connectionist classifier. This classifier uses a recursive neural network to construct vector representations for phrases in a candidate parse tree in order to classify it as syntactically correct or not. Tested on the WSJ23, my proposed method achieves a statistically significant improvement of 0.20% in F-score (2% error reduction) and 0.95% in exact match, compared with the Berkeley parser. This result shows that vector representations can be usefully applied to syntactic parsing, and demonstrates the benefits of combining symbolic and connectionist approaches.

3.1 Introduction

According to the principle of compositionality, composing the meaning of a phrase or a sentence requires a syntactic parse tree, which is, in most current systems, given by a statistical parser. This parser, in turn, is trained on syntactically annotated corpora.

However, there are good reasons to also consider information flowing in the opposite direction: from semantics to syntactic parsing. Performance of parsers trained and evaluated on the Penn WSJ treebank has reached a plateau, as many ambiguities cannot be resolved by syntactic information alone. Further improvements in parsing may depend on the use of additional sources of information, including semantics. In this chapter, I study the use of some form of semantics for syntactic parsing.

The currently dominant approach to syntactic parsing is based on extracting probabilistic symbolic grammars from a treebank and defining appropriate probability distributions over the parse trees that they license [Charniak, 2000, Collins, 2003, Klein and Manning, 2003, Petrov et al., 2006, Bod et al., 2003, Sangati and Zuidema, 2011, van Cranenburgh et al., 2011]. An alternative approach, with promising recent de-

velopments [Socher et al., 2010, Collobert, 2011], is based on using neural networks. In the present chapter, I combine the *symbolic* and *connectionist* approaches through reranking: a statistical symbolic parser is used to generate a k-best list which is then reranked based on acceptability scores given by a connectionist classifier.

The idea of reranking is motivated by analyses of the results of state-of-the-art symbolic parsers such as the Brown and Berkeley parsers, which have shown that there is still considerable room for improvement: oracle results on 50-best lists display a dramatic improvement in accuracy (96.08% vs. 90.12% in F-score and 65.56% vs. 37.22% in exact match with the Berkeley parser – shown later in the chapter). This suggests that parsers that rely on syntactic corpus statistics, though not sufficient by themselves, may very well serve as a basis for systems that integrate other sources of information by means of reranking.

One important complementary source of information is the semantic plausibility of the constituents of the syntactically viable parses. The exploitation of this kind of information is the topic of the research I present here. In this work, I follow up on a proposal by Steedman [1999], who suggests that the realm of semantics lacks the clear-cut hierarchical structures that characterise syntax, and that semantic information may therefore be profitably treated by the classificatory mechanisms of neural nets—while the treatment of syntactic structures is best left to symbolic parsers. I thus develop a hybrid system, which parses its input sentences on the basis of a probabilistic symbolic grammar, and reranks the candidate parses based on scores given by a neural network.

My work is inspired by the work of Socher and colleagues [Socher et al., 2010, 2011b]. They propose a parser using a recursive neural network (RNN) for encoding parse trees, representing phrases in a vector space, and scoring them. Their experimental result (only 1.92% lower than the Stanford parser in unlabelled bracketing F-score for sentences up to a length of 15 words) shows that an RNN is expressive enough for syntactic parsing. Additionally, their qualitative analysis indicates that the learnt phrase features capture some aspects of phrasal semantics, which could be useful to resolve semantic ambiguity that syntactic information alone cannot. My work in this chapter differs from theirs in that I replace the parsing task by a reranking task, and thus reduce the search space significantly to a set of parses generated by a symbolic parser rather than the space of all parse trees. As a result, I can apply my method to sentences which are much longer than 15 words.

Reranking a k-best list is not a new idea. Collins [2000], Charniak and Johnson [2005], and Johnson and Ural [2010] have built reranking systems with state-of-the-art performance. In order to achieve such high F-scores, those rerankers rely on a very large number of features selected on the basis of expert knowledge. Unlike them, my feature set is selected automatically, yet the reranker achieves a statistically significant improvement in both F-score and exact match.

Closest to my work is the work of Menchetti et al. [2005] and the work, carried out concurrently with mine, of Socher et al. [2013a]: both also rely on symbolic parsers to reduce the search space and use RNNs to score candidate parses. However, my work differs in the way the feature set for reranking is selected. In their methods, only the

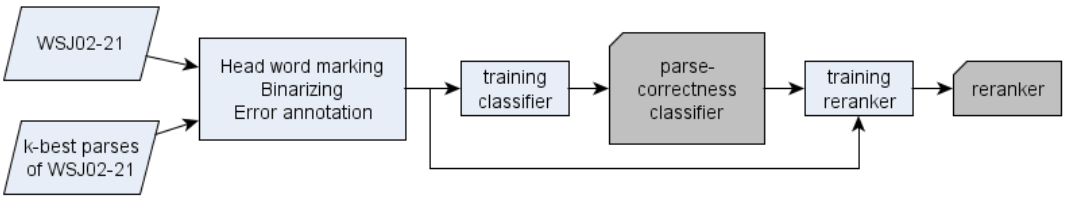


Figure 3.1: An overview of the proposed method.

score at the tree root is considered whereas in my method the scores at all internal nodes are taken into account. Selecting the feature set like that gives us a flexible way to deal with errors accumulated from the leaves to the root.

Figure 3.1 shows a diagram of the proposed method. First, a parser (in this chapter: the Berkeley parser) is used to generate k-best lists of the Wall Street Journal (WSJ) sections 02-21. Then, all parse trees in these lists and the WSJ02-21 are preprocessed by marking head words, binarising, and performing error-annotation (Section 3.2). After that, I use the annotated trees to train the classifier (Section 3.3). Finally, those trees and the classifier are used to train the reranker (Section 3.4).

3.2 Experimental Setup

The experiments presented in this chapter have the following setting. I used the WSJ corpus with the standard splits: sections 2-21 for training, section 22 for development, and section 23 for testing. The latest implementation (version 1.7) of the Berkeley parser[1] [Petrov et al., 2006] was used for generating 50-best lists. I marked head words and binarised all trees in the WSJ and the 50-best lists as in Section 3.2.1, and annotated them as in Section 3.2.2 (see Figure 3.2).

3.2.1 Pre-processing Trees

I pre-processed trees by marking head words and binarising the trees. For head word marking, I used the head finding rules of Collins [Collins, 1999] which are implemented in the Stanford parser. To binarise a k-ary branching, e.g. $P \rightarrow C_1 \ ... \ H \ ... \ C_k$ where H is the top label of the head constituent, I used the following method. If H is not the left-most child, then

$$P \rightarrow C_1 \ @P \ ; \ @P \rightarrow C_2 \ ... \ H \ ... \ C_k$$

otherwise,

$$P \rightarrow @P \ C_k \ ; \ @P \rightarrow H \ ... \ C_{k-1}$$

[1] `https://code.google.com/p/berkeleyparser`

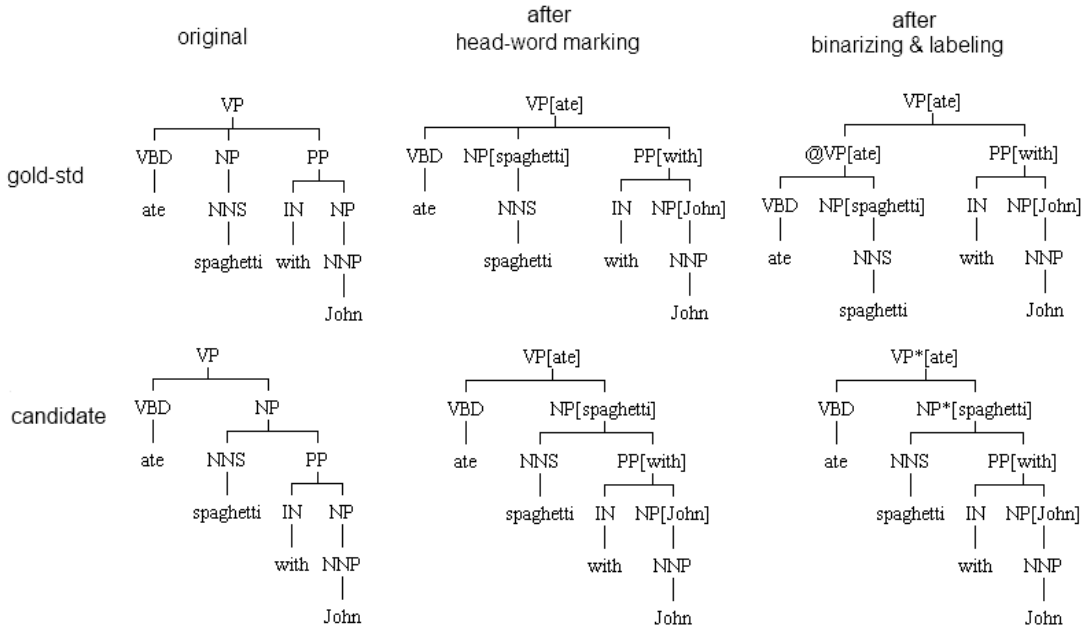


Figure 3.2: Example for preprocessing trees. Nodes marked with (*) are labelled *incorrect* whereas the other nodes are labelled *correct*.

where $@P$, which is called dummy-P, now is the head of P. I then recursively applied this transformation to the children until reaching terminal nodes. In this way, I ensure that every internal node has one and only one head word.

3.2.2 Error Annotation

I annotated nodes (as *correct* or *incorrect*) as follows. Given a parse tree T in a 50-best list and its corresponding gold-standard tree G in the Penn treebank, I first attempted to align their terminal nodes according to the following criterion: a terminal node t is aligned to a terminal node g if they are at the same position counting from left to right and they have the same label. Then, a non-terminal node $P[w_h]$ with children $C_1, ..., C_k$ is aligned to a gold-standard non-terminal node $P^*[w_h^*]$ with children $C_1^*, ..., C_l^*$ ($1 \leq k, l \leq 2$ in our case) if they have the same head word, the same syntactic category, and their children are all aligned in the right order. In other words, the following conditions have to be satisfied

$$P = P^* \ ; \ w_h = w_h^* \ ; \ k = l$$
$$C_i \textit{ is aligned to } C_i^*, \text{ for all } i = 1..k$$

Aligned nodes were annotated as *correct* whereas the other nodes were annotated as *incorrect*.

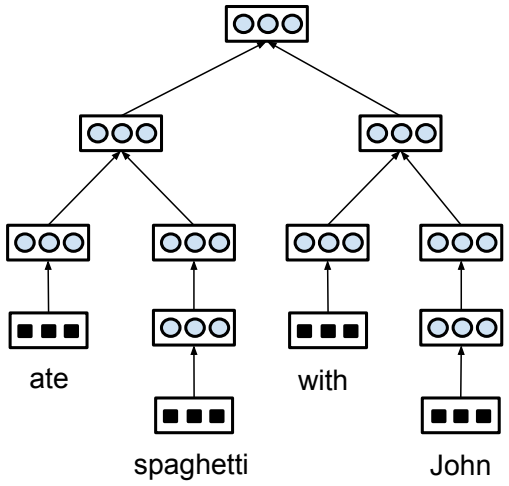


Figure 3.3: An RNN attached to the parse tree shown in the top-right of Figure 3.2. All unary branchings share a set of weight matrices, and all binary branchings share another set of weight matrices (see Figure 3.4).

3.3 Acceptability Classification

This section describes how to employ a neural network to construct vector representations for phrases given parse trees and to identify whether those trees are syntactically correct or not. In order to encode tree structures, inspired by the work of Socher et al. [2010], I employ an RNN (see Figure 3.3 and Figure 3.4). However, unlike traditional RNNs, my RNN

- can handle unary branchings,
- takes into account head words and syntactic tags, and
- takes into account contextual words (one word before and one word after the constituent), as in the second model of Socher et al. [2010].

It is worth noting that, although we can use some transformation to remove unary branchings, handling them is helpful in our case because the system avoids dealing with the many syntactic tags that would result from the transformation. In addition, using a new set of weight matrices for unary branchings makes my RNN more expressive without facing the problem of sparsity thanks to a large number of unary branchings in the treebank.

The RNN processes a tree structure by repeatedly applying itself at each internal node. Thus, walking bottom-up from the leaves of the tree to the root, I compute for every node a vector based on the vectors of its children. Because of this process, those vectors have to have the same dimension. It is worth noting that, because information

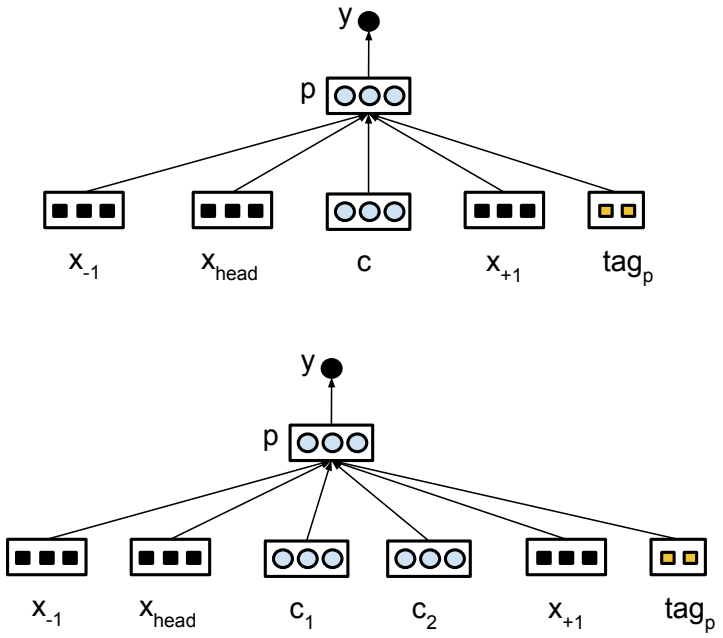


Figure 3.4: Details about the employed RNN for a unary branching (top) and a binary branching (bottom). The biases are not shown for simplicity.

at leaves, i.e. lexical semantics, is composed according to a given syntactic parse, what a vector at each internal node captures is some aspects of compositional semantics of the corresponding phrase. In the remainder of this subsection, I describe in more detail how to construct vector representations geared towards the acceptability classification task.

Similarly to Socher et al. [2010], and Collobert [2011], given a sequence of words $(w_1, ..., w_l)$, I first compute a sequence of vectors $(\mathbf{x}_1, ..., \mathbf{x}_l)$ representing those words by using a lookup table (i.e., word embeddings) $\mathbf{L} \in \mathbb{R}^{n \times |V|}$, where $|V|$ is the size of the vocabulary and n is the dimension of the vectors. This lookup table $\mathbf{L}$ could be seen as a storage of lexical semantics where each column is a vector representation of a word. Hence, let $\mathbf{u}_i$ be the one-hot binary representation of word w_i (i.e., all of the entries of $\mathbf{u}_i$ are zero except the one corresponding to the index of the word w_i in the dictionary), then

$$\mathbf{x}_i = \mathbf{L}\mathbf{u}_i \in \mathbb{R}^n \tag{3.1}$$

I also encode syntactic tags by binary vectors but put an extra bit at the end of each vector to mark whether the corresponding node is a dummy node (i.e., $@P$ or P) resulting from the binarization pre-processing.

Then, given a unary branching $P[w_h] \rightarrow C$, I can compute the vector at the node P by (see Figure 3.4-top):

$$\mathbf{p} = f\big(\mathbf{W}_u\mathbf{c} + \mathbf{W}_h\mathbf{x}_h + \mathbf{W}_{-1}\mathbf{x}_{-1} + \mathbf{W}_{+1}\mathbf{x}_{+1} + \mathbf{W}_t\mathbf{t}_p + \mathbf{b}_u\big) \tag{3.2}$$

where $\mathbf{c}, \mathbf{x}_h$ are vectors representing the child C and the head word; $\mathbf{x}_{-1}, \mathbf{x}_{+1}$ are vector representations of the left and right neighbouring words of P; $\mathbf{t}_p$ encodes the syntactic tag of P, $\mathbf{W}_u$, $\mathbf{W}_h$, $\mathbf{W}_{-1}$, $\mathbf{W}_{+1} \in \mathbb{R}^{n\times n}$; $\mathbf{W}_t \in \mathbb{R}^{n\times(|T|+1)}$; $|T|$ is the size of the set of syntactic tags; $\mathbf{b}_u \in \mathbb{R}^n$; and f can be any activation function ($tanh$ is used in this case). For a binary branching $P[w_h] \rightarrow C_1\ C_2$, I simply change the way the children's vectors are added (see Figure 3.4-bottom):

$$\mathbf{p} = f\big(\mathbf{W}_{b1}\mathbf{c}_1 + \mathbf{W}_{b2}\mathbf{c}_2 + \mathbf{W}_h\mathbf{x}_h + \mathbf{W}_{-1}\mathbf{x}_{-1} + \mathbf{W}_{+1}\mathbf{x}_{+1} + \mathbf{W}_t\mathbf{t}_p + \mathbf{b}_b\big) \tag{3.3}$$

Finally, I put a sigmoid neural unit on the top of each internal node (except pre-terminal nodes because we are not concerned with POS-tagging) to detect the correctness of the subparse tree rooted at that node:

$$y = sigmoid(\mathbf{w}_{cat}^T\mathbf{p} + b_{cat}) \tag{3.4}$$

where $\mathbf{w}_{cat} \in \mathbb{R}^n, b_{cat} \in \mathbb{R}$.

3.3.1 Learning

The cost of a parse tree is computed as a sum of classification errors of all subparses. Hence, learning is to minimise the following objective:

$$J(\theta) = \frac{1}{N}\sum_T \sum_{(y(\theta),t)\in T} \frac{1}{2}(t - y(\theta))^2 + \lambda\|\theta\|^2 \tag{3.5}$$

w.r.t. θ, where θ is the parameter set, N is the number of trees, λ is a regularisation hyper-parameter, T is a parse tree, $y(\theta)$ is given by Equation 3.4, and t is the class of the corresponding subparse ($t = 1$ means *correct*). Gradients $\frac{\partial J}{\partial \theta}$ are computed efficiently thanks to the back-propagation through the structure algorithm [Goller and Küchler, 1996]. L-BFGS [Liu and Nocedal, 1989] is used to minimise the objective function.

3.3.2 Experiments

I implemented the classifier in Torch7[2] [Collobert et al., 2011a], which is a powerful Matlab-like environment for machine learning. The 50-dim word embeddings given by Collobert et al. [2011b][3] were used as $\mathbf{L}$ in Equation 3.1. Note that these embeddings,

[2] `http://www.torch.ch/`

[3] `http://ronan.collobert.com/senna/`

	pos-rate (%)	neg-rate (%)	%-Pos
gold-std	75.31	-	100
1-best	90.58	64.05	71.61
10-best	93.68	71.24	61.32
50-best	95.00	73.76	56.43

Table 3.1: Classification results on the WSJ22 and the k-best lists.

which resulted from training a neural language model on the English Wikipedia and Reuters, have been shown to capture several interesting semantic similarities between words. I train the classifier on 10-best parses of the WSJ02-21, with a minibatch size of 5000, $\lambda = 10^{-4}$, $n = 50$.

I evaluated the classifier on the development set WSJ22, which contains 1700 sentences, and measured its performance in positive rate and negative rate, which are defined by:

$$\text{pos-rate} = 100 \times \frac{\#\text{true\_pos}}{\#\text{true\_pos} + \#\text{false\_neg}}$$
$$\text{neg-rate} = 100 \times \frac{\#\text{true\_neg}}{\#\text{true\_neg} + \#\text{false\_pos}}$$

The positive/negative rate indicates rate at which positive/negative (sub)trees are correctly labelled *positive/negative*. In order to achieve a high performance in the reranking task, the classifier must have a high positive rate as well as a high negative rate. In addition, the percentage of positive examples is also interesting because it shows the unbalancedness of the data. Because accuracy is not a reliable measurement when the dataset is highly unbalanced, I do not show it here. Table 3.1, Figure 3.5, and Figure 3.6 show the classification results.

3.3.3 Discussion

Table 3.1 shows the classification results on the gold standard, 1-best, 10-best, and 50-best lists. On the 1-best data, the classifier missed less than one tenth positive subtrees and correctly found nearly two third of the negative ones. That is, my classifier might be useful for avoiding many of the mistakes made by the Berkeley parser, whilst not introducing too many new mistakes of its own. This fact gives me hope to improve parsing performance when using this classifier for reranking.

Figure 3.5 shows the positive rate, the negative rate, and the percentage of positive examples w.r.t. sub-tree depth on the 50-best data. We can see that the positive rate is inversely proportional to the subtree depth, unlike the negative rate. This is the case because the deeper a subtree is, the less likely it is positive (we can see this via

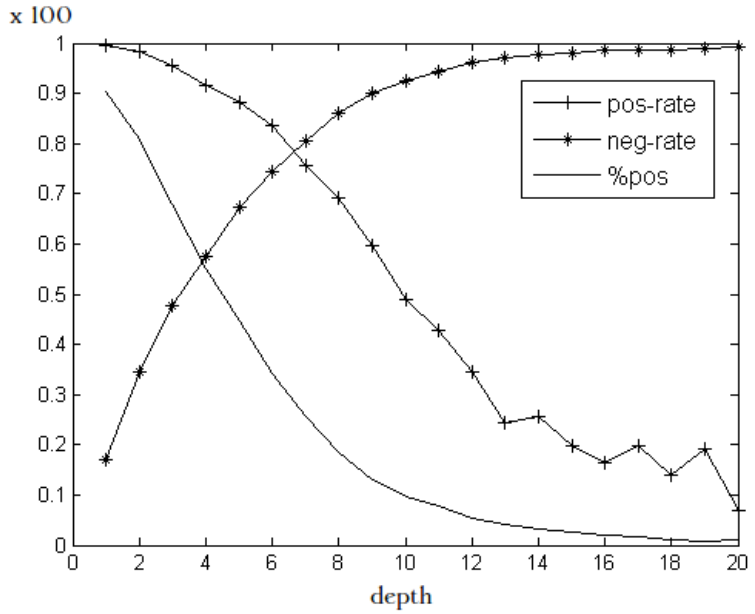


Figure 3.5: Positive rate, negative rate, and percentage of positive examples w.r.t. subtree depth.

the percentage-of-positive-examples curve). In addition, deep subtrees are difficult to classify because uncertainty is accumulated when propagating from bottom to top.

3.4 Reranking

In this section, I describe how to apply the above classifier to the reranking task. First, I need to construct a feature vector for each parse tree in one vector space, i.e., $\mu(T) = \big(\mu_1(T), ..., \mu_v(T)\big)^T$ for any arbitrary parse tree T. Collins [2000], Charniak and Johnson [2005], and Johnson and Ural [2010] choose as the first feature the original model's score and as other features the frequencies of specific discrete handcrafted properties of the tree (e.g., how many times the word "pizza" comes after the word "eat"). I here do the same with a method to discretise results from the classifier; however, unlike the approaches just mentioned, I do not use any expert knowledge for feature selection: instead, I use a fully automatic process. I use a 2D histogram to store predicted scores w.r.t. subtree depths. This gives me a flexible way to penalise low score subtrees and reward high score subtrees based on the performance of the classifier at different depths (see Section 3.3.3).

Technically speaking, a feature vector $\mu(T)$ is computed as follows. $\mu_1(T)$ is the model score (i.e., max-rule-sum score) given by a third-party parser, $\big(\mu_2(T), ..., \mu_v(T)\big)$ is a histogram of a set of (y, h)'s where y is given by Equation 3.4 and h is the depth of the corresponding subtree (see Figure 3.7). The domain of y (i.e., $[0, 1]$) is split into γ_y

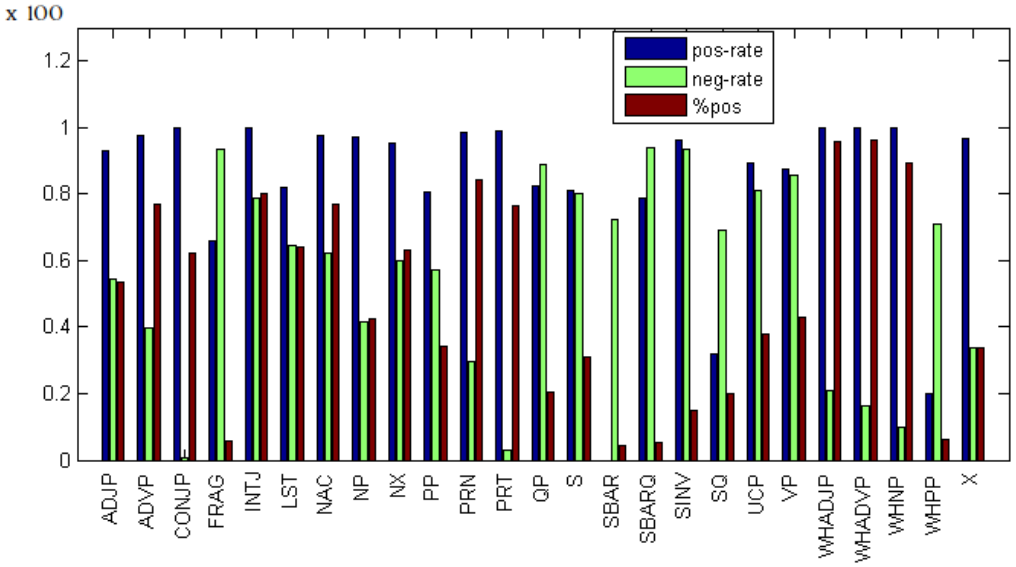


Figure 3.6: Positive rate, negative rate, and percentage of positive samples w.r.t. syntactic categories (excluding POS tags).

equal bins whereas the domain of h (i.e., $\{1, 2, 3, ...\}$) is split into γ_h bins such that the i-th ($i < \gamma_h$) bin corresponds to all subtrees of depth i and the γ_h-th bin corresponds to all subtrees of depths equal or greater than γ_h. The parameters γ_y and γ_h are then tuned on the development set.

After extracting feature vectors for parse trees, I then find a linear ranking function

$$f(T) = \mathbf{w}^\top \mu(T)$$

such that

$$f(T_1) > f(T_2) \text{ iff } \textit{fscore}(T_1) > \textit{fscore}(T_2)$$

where $\textit{fscore}(.)$ is the function calculating F-score, and $\mathbf{w} \in \mathbb{R}^v$ is a weight vector, which is efficiently estimated by SVM ranking [Yu and Kim, 2012]. SVMs were initially used for binary classification. Their goal is to find the hyperplane which has the largest margin to best separate two example sets. They were then proved to be efficient in solving ranking tasks in information retrieval, and in syntactic parsing [Shen and Joshi, 2003, Titov and Henderson, 2006]. In the below experiments, I employ SVM-Rank[4] [Joachims, 2006], which runs extremely fast (less than two minutes on about 38000 10-best lists).

[4] www.cs.cornell.edu/people/tj/svm_light/svm_rank.html

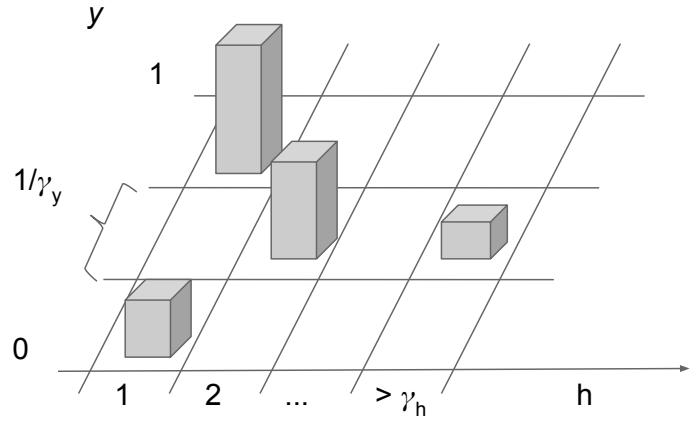


Figure 3.7: Histogram of a set of (y, h) where y is given by Equation 3.4 and h is the depth of the corresponding subtree.

3.4.1 Experiments

Using the classifier in Section 3.3, I implemented the reranker in Torch7, and trained it on the WSJ02-21. I used WSJ22 to estimate the parameters γ_y and γ_h: grid search determined that $\gamma_y = 9$ and $\gamma_h = 4$ yielded the highest F-score.

Table 3.2 shows the results of my reranker on 50-best WSJ23 generated by the Berkeley parser, using the standard evalb. My method improves 0.20% in F-score for sentences of all lengths, and 0.22% for sentences with ≤ 40 words. These differences are statistically significant[5] with *p-value* < 0.003. My method also improves exact match (0.95% for all sentences as well as for sentences with ≤ 40 words).

Table 3.3 shows the comparison of the three parsers that use the same hybrid reranking approach. My method achieves 0.1% in F-score lower than Socher et al. [2013a]'s, and 1.5% better than Menchetti et al. [2005]'s. However, it achieves the least improvement in F-score over its corresponding baseline. That could be because my baseline parser (i.e., the Berkeley parser) performs much better than the other two baseline parsers; and hence, detecting errors it makes on candidate parse trees is more difficult.

3.5 Conclusions

This chapter describes a new reranking method which uses vector representations of words and phrases in syntactic parsing: a symbolic parser generates a k-best list which

[5]I use the "Significance testing for evaluation statistics" software (`http://www.nlpado.de/~sebastian/software/sigf.shtml`) described in Padó [2006].

Parser	LR	LP	LF	EX
all				
Berkeley parser	89.98	90.25	90.12	37.22
This work	**90.10**	**90.54**	**90.32**	**38.17**
Oracle	95.94	96.21	96.08	65.56
≤ 40 words				
Berkeley parser	90.43	90.70	90.56	39.65
This work	**90.57**	**91.01**	**90.78**	**40.50**
Oracle	96.47	96.73	96.60	68.51

Table 3.2: Reranking results on 50-best lists of WSJ23 (LR is labelled recall, LP is labelled precision, LF is labelled F-score, and EX is exact match.)

Parser	LF (all)	k-best parser
Menchetti et al. [2005]	88.8 (0.6)	Collins [1999]
Socher et al. [2013a]	**90.4 (3.8)**	PCFG Stanford parser
My proposed reranker	90.3 (0.2)	Berkeley parser

Table 3.3: Comparison of parsers using the same hybrid reranking approach. The numbers in the blankets indicate the improvements in F-score over the corresponding baselines (i.e., the k-best parsers).

is later reranked with acceptability scores given by a connectionist classifier. The proposed classifier uses an RNN, like Socher et al., [Socher et al., 2010, 2011b], to not only represent phrases in a vector space given parse trees, but also identify if these parse trees are grammatically correct or not.

Tested on WSJ23, my method achieves a statistically significant improvement in F-score (0.20%) as well as in exact match (0.95%). This result, although not comparable to the results reported by Collins [2000], Charniak and Johnson [2005], and Johnson and Ural [2010], shows the potential of using vector representations to support available state-of-the-art parsers.

One of the limitations of the work in this chapter is the lack of a qualitative analysis of how learnt vector representations have affected the reranking results. Therefore, the need for "compositional semantics" in syntactic parsing is not yet proven. One possible solution is to use vector representations together with non-semantic features (e.g., the ones of Charniak and Johnson [2005]) to find out whether the vector representations are truly helpful or they just resemble non-semantic features.

Chapter 4

Inside-Outside Semantics: A Framework

The Recursive Neural Network (RNN) model and its extensions have been shown to be powerful tools for semantic composition with successes in many natural language processing (NLP) tasks. However, in this chapter, I argue that the RNN model is restricted to only a subset of the NLP tasks where semantic compositionality plays a role. I propose a novel extension called Inside-Outside Semantics. In this framework every node in a parse tree is associated with a pair of representations, the inside representation for representing the content under the node, and the outside representation for representing its surrounding context. I demonstrate how this allows me to develop a neural network model for a much broader class of NLP tasks and for supervised as well as unsupervised learning. The neural-net model, Inside-Outside Recursive Neural Network, performs on par with or better than the state-of-the-art (neural) models in word prediction, phrase-similarity judgements and semantic role labelling.

4.1 Introduction

For several decades, the most successful approaches for modelling word meaning, on the one hand, and those for modelling semantic composition (needed to compute the meaning of phrases and sentences), on the other hand, seemed incompatible. Word meanings can be well described with numerical vectors, often reflecting co-occurrence frequencies with other words in corpora. A rich and diverse literature has emerged on *distributional semantics*, with many successes in tasks like similarity judgements, word sense disambiguation and information retrieval (see Lenci [2008] for a review).

Semantic composition, on the other hand, has since Montague [1970] been modelled using the lambda-calculus, which is straightforwardly employed in symbolic logic of various sorts, but seems incompatible with vector representations. As recently as 2008, Lenci [2008] wrote in a review of distributional semantics that the issue of how to do semantic composition with vectorial word representations remained unsolved. Since then, however, a series of papers have appeared proposing a range of technical solutions for this problem and creating much excitement about the prospects

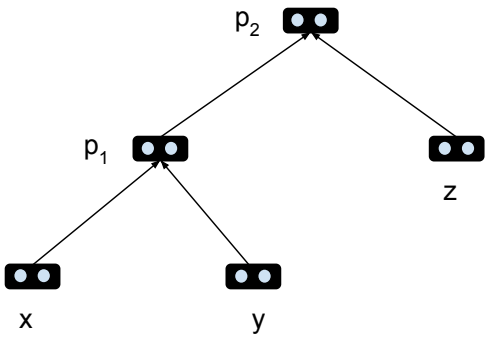


Figure 4.1: Recursive Neural Network

of finally bridging the two fields. One of the most successful of these new approaches, the Recursive Neural Network (RNN) model, originally proposed by Pollack [1990] and employed by Socher et al. [Socher et al., 2010, 2012, 2011c], seems to lend itself well for neural network-based technology for language processing.

In this chapter, for convenience, I re-introduce the traditional RNN model and it extensions, discuss them in some detail, and show that, at a closer look, they leave some major challenges unsolved. One such challenge concerns its bottom-up definition of composition, which reduces the RNN's usefulness as a framework for NLP. In Section 4.3 I propose a solution to this challenge, consisting of an extension, which I call Inside-Outside Semantics. I show that this allows information to flow top-down as well as bottom-up. As a result, its neural implementation, the Inside-Outside Recursive Neural Network, introduced in Section 4.4, can be applied to a wide range of NLP tasks. In Sections 4.5, 4.6, I report results on a word prediction task, a phrase similarity judgement task, and a semantic role labelling task, and show in each task that the new model is on par with the best n-gram or neural network approach for that specific domain.

4.2 Recursive Neural Networks

In compositional semantics, meaning representations are typically computed in a bottom-up manner: given a constituent $p = xy$ and a grammar rule R that combines the two linguistic items x and y, then

$$\mathbf{p} = f_R(\mathbf{x}, \mathbf{y})$$

where f_R is a composition function. There are different ways to instantiate meaning representations and composition functions. In formal semantics, meaning representations are typically λ-expressions and the composition function is function application [Bos et al., 2004]. In recent models of compositional vectorial semantics, the words meanings are vectors, matrices or tensors, and a variety of algebraic operators (e.g., vector addition) have been proposed for composition.

Socher et al. [2010], amongst others, propose to learn composition functions from data, rather than handcraft them, and to implement them using feed-forward neural networks. The basic idea of their RNN is simple. Assume that there is a parse tree $(p_2\ (p_1\ x\ y)\ z)$ (Figure 4.1), and that $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^n$ are vector representations of the three words x, y and z. We can then use a standard feedforward neural network which consists of a weight matrix $\mathbf{W}_1 \in \mathbb{R}^{n \times n}$ for left children and a weight matrix $\mathbf{W}_2 \in \mathbb{R}^{n \times n}$ for right children to compute the vector for a parent node in a bottom up manner. The RNN model is *recursive* in the sense that the same weight matrices are used recursively to compute vector representations for nodes higher up in the tree, until we reach the root node. Thus, we compute representations for p_1:

$$\mathbf{p}_1 = f(\mathbf{W}_1\mathbf{x} + \mathbf{W}_2\mathbf{y} + \mathbf{b})$$

and for p_2:

$$\mathbf{p}_2 = f(\mathbf{W}_1\mathbf{p}_1 + \mathbf{W}_2\mathbf{z} + \mathbf{b})$$

where $\mathbf{b} \in \mathbb{R}^n$ is a bias vector and f is an activation function (e.g., *tanh*).

In the classic setup [Goller and Küchler, 1996], the sequence of computations the RNN model performs is guided by an external, symbolic control structure that is provided with a parse tree of the sentence. Socher et al. integrate the RNN's vector computations with the (greedy) search for the best tree structure for a sentence, but still need a symbolic control structure to guide the applications of the neural network. Using Goller & Kuechler's back-propagation through structure algorithm [Goller and Küchler, 1996], and training on the syntactic annotations of the Penn WSJ treebank, Socher et al. [2010] obtain promising results on the syntactic parsing task for short sentences. A straightforward variant of the model, called compositional vector grammars [Socher et al., 2013a], introduces separate weight matrices for grammar rules, and obtains nearly state-of-the-art parsing scores on the Penn WSJ benchmark test.

Some extensions of the RNN have been introduced to further improve the accuracy of the composition functions, such as the Matrix-Vector RNN model [Socher et al., 2012], and the Recursive Neural Tensor Network model [Socher et al., 2013b]. Thanks to the successes in NLP tasks such as constituency parsing [Socher et al., 2013a], sentiment analysis [Socher et al., 2013b], the RNN models are now firmly established as a powerful tool for semantic (and syntactic) composition.

However, the RNN framework can still only be applied to a specific subset of NLP problems that involve compositional structure, because of

- the need for supervision, i.e. syntactic and semantic annotations needed as training signals, and
- the bottom-up definition of semantic composition.

A partial solution to the first problem (regarding to the need for semantic annotations) is provided by an elegant extension called Recursive Auto-encoders (RAE) [Socher et al., 2011c], which replace the traditional feed-forward neural networks by auto-encoders.

The underlying principle of the RAE model is that *composition is compression*, such that an input is able to be recovered from the output by a decoding function. In this chapter, however, I present an alternative extension of RNNs for unsupervised learning that outperforms the RAE model on the popular phrase-similarity judgement task by a large margin.

The next problem, that information is allowed to flow bottom-up only, turns out to be greatly limiting the RNN models' usefulness to a number of important domains in NLP. First, the RNN models are not able to be used in ways that parallel major parsing strategies, such as Earley parsing [Stolcke, 1995] and left-corner parsing [Manning and Carpenter, 2000], that make use of top-down predictions. Second, the RNNs cannot be used either in ways that parallel important classes of top-down generative probabilistic models such as Eisner's dependency models [Eisner, 1996b] and Collins' phrase-structure models [Collins, 1997]. Third, the RNN models do not compute representations for the contexts of constituents, even though contexts are essential for many tasks, such as (i) next or missing word prediction based on preceding or surrounding context, or (ii) semantic role labelling, where the goal is to assign semantic roles to constituents based on properties of both the slot (context) and the filler (content).

4.3 The Inside-Outside Semantics Framework

Believing that a natural solution for the difficulties mentioned above is to be found in the way information *flows* in linguistic structures such as parse trees, I propose a framework, *Inside-Outside Semantics*. The crucial innovation of this framework is to introduce an architecture that allows information to flow in parse trees not only bottom-up, as in traditional compositional semantics frameworks, but also top-down. I accomplish this by adding a second vector to every node that represents the *context* surrounding that node (rather than the *content* under that node). These context representations can, like content representations, be computed recursively. But where content representations are computed bottom-up from a node's children, context representations are computed top-down from a node's parent and siblings. The interaction between content and context representations in turn makes it possible to devise a new unsupervised learning scheme for estimating composition functions from raw texts. The interaction, moreover, can be formed based on training signals in supervised learning tasks such as semantic role labelling, and named entity recognition.

I denote with $\mathbf{i}_p$ and $\mathbf{o}_p$ the representations of p's content and context (see Figure 4.2). Henceforth, I will call $\mathbf{i}$ the *inside representation*, and $\mathbf{o}$ the *outside representation*. Similarly to the traditional compositional semantics framework, we have:

$$\mathbf{i}_p = f_R^i(\mathbf{i}_x, \mathbf{i}_y) \tag{4.1}$$

where f_R^i is a composition function for inside representations. To compute outside

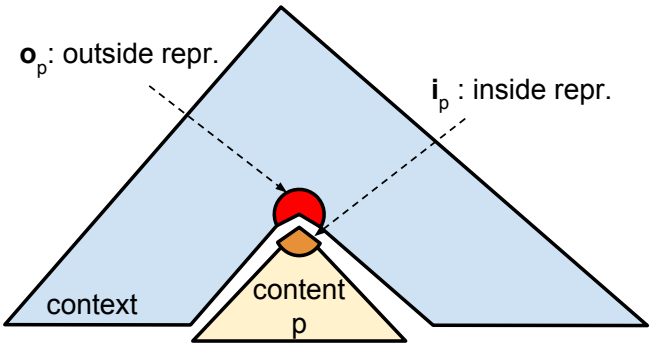
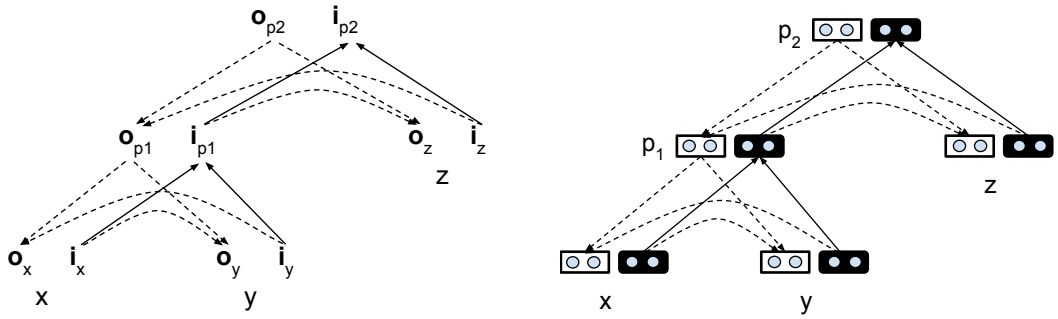


Figure 4.2: Inside ($\mathbf{i}_p$) and outside ($\mathbf{o}_p$) representations at the node that covers constituent p. They are vector representations of p's content and context, respectively.



(a) Inside-Outside Semantics framework

(b) Inside-Outside Recursive Neural Network

Figure 4.3: Inside-Outside framework. Black rectangles correspond to inside representations, white rectangles correspond to outside representations.

representations (i.e., the associative profile of context), I use the following formulas:

$$\mathbf{o}_x = f_R^{o1}(\mathbf{o}_{xy}, \mathbf{i}_y) \tag{4.2}$$

$$\mathbf{o}_y = f_R^{o2}(\mathbf{o}_{xy}, \mathbf{i}_x) \tag{4.3}$$

where f_R^{oj} is a composition function for the outside representation of the j-th child. The rationale for these equations is that the context of the constituent x consists of the context of the parent constituent $p = xy$ and the content of the sibling constituent y; the outside representation of x is thus a function of the outside representation of p and the inside representation of y. [1]

The whole process for the parse tree $(p_2\ (p_1\ x\ y)\ z)$ is depicted graphically in Figure 4.3a, where an arrow indicates that the representation at its head is a function of the representation at its tail, and solid arrows are for computing inside representations

[1] The reader may recognise that equations 4.1, 4.2, and 4.3 bear a resemblance to respectively inside and outside probabilities in the Inside-Outside algorithm for stochastic grammars [Lari and Young, 1990].

whereas dotted arrows are for outside representations. Looking at the graph, we can see that there are two information flows in the tree structure: one from bottom up and the other from top down. The interaction between these two flows may occur at any node and depends on the purpose of usage.

This framework shares some properties with Irsoy and Cardie's recently proposed *bidirectional* RNN model [Irsoy and Cardie, 2013], and Paulus et al. [2014]'s global-belief RNN model, which also add a second representation, the *down vector*, to every node and allow for information to flow top-down. However, because these down vectors can only be computed after *all* up vectors (corresponding to inside representations) for a tree have been computed first, I doubt that these two models are able to solve the three problems mentioned in Section 4.2. In my framework, by contrast, inside and outside representations are strictly complementary: the outside representation for a node x can be computed without knowing the contents under that node.

There are different ways to instantiate representations $\mathbf{i}, \mathbf{o}$ and composition functions f_R^i, f_R^{oj}. In the next section, I describe IORNN, a neural-network-based instance of the framework. The proposed framework, moreover, can also work with λ-expressions, as in formal semantics. The reader can find details in Appendix A.

4.4 The Inside-Outside Recursive Neural Network Model

I explain the operation of the IORNN model in terms of the same example parse tree $(p_2\ (p_1\ x\ y)\ z)$ as I used before (see Figure 4.3b). For the present chapter I assume that an independently given parser assigns a syntactic structure to an input string. Each node u in the syntactic tree carries two vectors $\mathbf{o}_u$ and $\mathbf{i}_u$, the outside representation and the inside representation of the constituent that is covered by the node.

Computing inside representations Inside representations are computed from the bottom up. I assume for every word w an inside representation $\mathbf{i}_w \in \mathbb{R}^n$. The inside representation of a non-terminal node, e.g. p_1, is given by:

$$\mathbf{i}_{p_1} = f(\mathbf{W}_1^i \mathbf{i}_x + \mathbf{W}_2^i \mathbf{i}_y + \mathbf{b}^i)$$

where $\mathbf{W}_1^i, \mathbf{W}_2^i \in \mathbb{R}^{n \times n}$, $\mathbf{b}^i \in \mathbb{R}^n$, and f is an activation function, e.g. $tanh$. The inside representation of a parent node is thus a function of the inside representations of its children. This is similar to the traditional RNN model.

Computing outside representations Outside representations are computed from the top down. For a node which is not the root, e.g. p_1, the outside representation is given by

$$\mathbf{o}_{p_1} = g(\mathbf{W}_p^o \mathbf{o}_{p_2} + \mathbf{W}_2^o \mathbf{i}_z + \mathbf{b}^o)$$

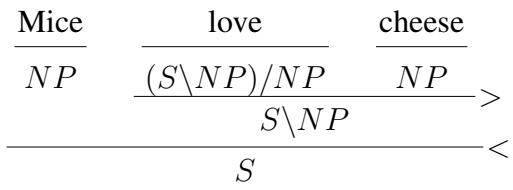


Figure 4.4: A CCG derivation for the sentence "Mice love cheese".

where $\mathbf{W}^o_p, \mathbf{W}^o_{\{1,2\}} \in \mathbb{R}^{n \times n}$, $\mathbf{b}^o \in \mathbb{R}^n$, and g is an activation function.[2] The outside representation of a node is a function of the outside representation of its parent and the inside representations of its siblings. If there is information about the external context of the sentence that is being processed, this information determines the outside representation of the root node $\mathbf{o}_{root}$. In my first experiments reported here, no such information is assumed to be available. In this case, a random value $\mathbf{o}_\emptyset$ is chosen and assigned to the root nodes of all sentences. Note that this value is adjusted by the learning process discussed below.

Training

Training an IORNN is to minimise an objective function $J(\theta)$ which depends on the purpose of usage (see Section 4.5, 4.6) where θ is the parameter set. To do so, I compute gradients $\partial J / \partial \theta$ and apply the stochastic gradient descent method. The gradients are efficiently computed thanks to the back-propagation through structure algorithm [Goller and Küchler, 1996]. I use AdaGrad [Duchi et al., 2011] to update the parameters.

Incorporating Grammatical Information

The IORNN above uses just five matrices (left child/right child for inside, parent/left sibling/right sibling for outside) for computing inside/outside representations in a binary tree, ignoring syntactic labels. However, Socher et al. [2013a] and Hermann and Blunsom [2013] find that using different matrices for different grammar rules/categories greatly improves results. In most of my experiments, I follow this suggestion, using one of the following types of syntactic representations (without binarisation):

- CFG-k: context-free grammar (CFG) derivation with the k most frequent rules (according to the Penn Treebank section 2-21), the other rules are replaced by abstract rules $X \to X \ ... \ X$;
- CCG: combinatory categorial grammar (CCG) derivation with all combinatory rules.

[2] Notice that outside representations and inside representations do not need to have the same dimension.

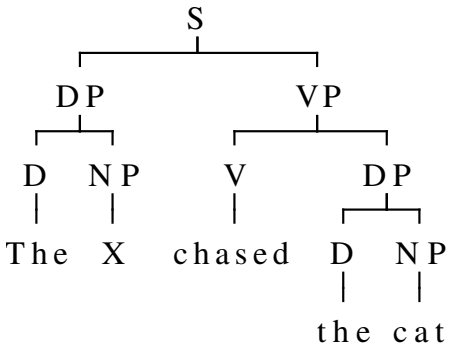


Figure 4.5: A fluent speaker needs to understand the meaning of the whole constituent "chased the cat" to guess the meaning of the unknown word "X" (here should be "dog"). Therefore, if a model can fulfill the task, it is expected to learn appropriate composition functions for $VP \rightarrow V \;\; DP$ and $DP \rightarrow D \;\; NP$.

CCG [Steedman, 2000] is a widely used grammar formalism for semantic applications thanks to its transparent interface between syntax and semantics [Bos et al., 2004]. In CCG, a word or constituent is assigned a category, which could be basic (e.g., NP, S) or complex (e.g., NP/NP, $S\backslash NP$). A complex category defines selectional and directional combination. For instance, NP/NP implies that any constituent or word in this category can combine with a noun phrase to its right-hand side in order to form a new noun phrase. The standard CCG specifies five basic rules for combination: forward application (fa), backward application (ba), composition (comp), coordination (conj), and type raising (tr). Figure 4.4 shows a CCG derivation for the sentence "Mice love cheese". (For a complete introduction, see Steedman [2000].)

4.5 Unsupervised Learning with IORNNs: word prediction and phrase similarity

In this section, I employ the IORNN model to learn from data that contain no annotations for the semantics of constituents, sentences or other combinations of words. In that sense, the learning is *unsupervised*, but note that I will make use of syntactic structures, both at train time and at test time (using a third party parser).

To demonstrate that the capabilities of the IORNN model go beyond those of the standard bottom-up RNN model, I apply it to the task of predicting missing words in a sentence. I reason that any fluent speaker of a language can guess the meaning of an unknown word by making use of the meaning of its context, and that when she correctly predicts the meaning of the unknown word, we can assume that she comprehends the meaning of the context (see Figure 4.5). Following this reasoning, I base the composition function learning task on the word prediction task, which opens up a novel (and much needed) scheme for unsupervised training of semantic composition models.

The IORNN model is expected to learn appropriate composition functions through learning to predict words. To do so, at every leaf, I let the inside and outside representations interact with each other such that what ends up in the outside representation is some expectation about what the inside representation should be. Because the outside representation is computed based on the context word representations with the composition functions, the more accurately the IORNN model performs on the word prediction task is, the more semantically plausible the composition functions should be. I therefore evaluate the IORNN model both on the quality of the missing word predictions and on the quality of the semantic composition functions that it learned for the task (by measuring the correlation of semantic similarity judgements of compounds to those of human subjects).

4.5.1 Word Prediction

Using a method proposed by Collobert et al. [2011b], I train the network such that it assigns the correct target word a score higher than scores given to other words. The score $s(x, \mathbf{o}_w)$ given to a candidate word x for a specific context $\mathbf{o}_w$ is computed by:

$$u(x, \mathbf{o}_w) = f(\mathbf{W}_1^u \mathbf{o}_w + \mathbf{W}_2^u \mathbf{i}_x + \mathbf{b}^u) \tag{4.4}$$

$$s(x, \mathbf{o}_w) = \mathbf{v}^T u(x, \mathbf{o}_w) \tag{4.5}$$

where $\mathbf{W}_1^u, \mathbf{W}_2^u \in \mathbb{R}^{n \times k}$, $\mathbf{v}, \mathbf{b}^u \in \mathbb{R}^k$. (I fix $k = 2n$.) Now, the objective function is a ranking criterion with respect to θ:

$$J(\theta) = \frac{1}{|D|} \sum_{s \in D} \sum_{w \in s} \sum_{x \in V} \max\{0, 1 - s(w, \mathbf{o}_w) + s(x, \mathbf{o}_w)\} + \frac{\lambda}{2} ||\theta||^2 \tag{4.6}$$

where λ is a regularization parameter. The outermost sum is over all sentences s in a corpus D, the next sum is over all observed words w in a sentence, and the innermost sum is over all candidate words x in the vocabulary V. The term inside the innermost sum will give a value higher than 1 if the score of the candidate is higher than that of the observed word, and smaller than 1 (but at least 0) if the observed word's score is higher. In this way, the network is forced to assign to the correct word a score higher than scores given to any other words by, at least, a margin of 1. To minimize this objective function, I randomly, from the uniform distribution, pick words in the vocabulary as 'corrupted' examples and follow the method given in Section 4.4.

4.5.2 Experiments

I built three IORNNs, with $n = 50$, for three types of derivations [CFG-0], [CFG-200], and [CCG] (see Section 4.4). All of them were initialised with the 50-dim word embeddings[3] from Collobert et al. [2011b] and trained on around 3M sentences longer than 10 words[4] in the British National Corpus (BNC), parsed by the C&C parser [Curran

[3] `http://ronan.collobert.com/senna/`

[4] Long contexts are needed for successfully guessing missing words.

Model	NAR
5-gram	1.8%
IORNN [CFG-0]	1.6%
IORNN [CFG-200]	**0.8%**

Table 4.1: Test results on the word prediction task. (Smaller is better.)

et al., 2007] and by the Berkeley Parser [Petrov et al., 2006].

Word Prediction

I run the trained networks on the Penn Treebank,[5] section 22, and compared their performances with a baseline formed by the Kneser-Ney smoothed 5-gram model[6] trained on 300M-word text from the Wikipedia[7] (3 times larger than the BNC). It is worth noting that this model is state-of-the-art in the language modelling area, outperforms syntax-based language models and thus is a strong baseline to compare against (even if it, unlike the IORNN's, does not make use of the syntactic structure of sentences at either train or test time). Predicting the actual word at any location in the sentence is often next to impossible. I therefore based the evaluation on the rank of the correct word in the list of candidate words (all 130k words in the vocabulary); and propose a metric which is called *normalized average rank* (NAR):

$$NAR = \frac{1}{|V|}\frac{1}{\sum_{s\in D}|s|}\sum_{s\in D}\sum_{w\in s} rank(w)$$

where $|V|$ is the size of the vocabulary, D is the test set, and $rank(w)$ is the rank of the correct word w. [8]

Table 4.1 shows test results. Both IORNNs achieve considerably lower NARs than the 5-gram model, showing that the IORNN model is more capable of judging how different words fit the context created by a surrounding utterance.

Phrase Similarity

To evaluate the composition functions that were learned on the word prediction task, I made use of a phrase similarity judgement task. In this task the system considers

[5] I did not use sentences from the BNC corpus because I wanted to examine the generality of the IORNN model.

[6] I used the BerkeleyLM at `http://code.google.com/p/berkeleylm/`.

[7] `http://nlp.stanford.edu/data/WestburyLab.wikicorp.201004.txt.bz2`

[8] Veldhoen [2015] criticizes the proposed metric because it is biased by frequent words. For instance, a uni-gram model can achieve an NAR as low as 3.1%. In other words, one can achieves an NAR of 3.1% just by sorting words by their frequencies. The problem here, in fact, lies in the vocabulary size $|V|$: the larger the vocabulary V is, the more infrequent words it has. Despite that, using the NAR metric we can compare models on how well they distinguish correct words from wrong ones. Therefore, I still use the NAR metric in this experiment.

type	phrase 1	phrase 2	rating
v-obj	remember name	pass time	3
adj-n	dark eye	left arm	5
n-n	county council	town hall	4

Table 4.2: Items in the dataset from Mitchell and Lapata [2010]. The ratings range from 1 (very low similarity) to 7 (very high similarity).

model	adj-n	n-n	v-obj
RAE	0.20	0.18	0.14
IORNN [CFG-0]	**0.38**	**0.36**	**0.32**
CCAE-B	0.38	**0.44**	0.34
IORNN [CCG]	**0.39**	0.40	**0.37**

Table 4.3: Spearman's correlation coefficients of model predictions on the phrase similarity task.

pairs of (short) phrases and estimates the semantic similarity within each pair. Its estimations are compared with human judgements. I used the dataset from Mitchell and Lapata [2010] which contains 5832 human judgements on 108 phrase pairs.[9] The pairs include noun-noun, verb-object, and adjective-noun combinations. The ratings range from 1 (very low similarity) to 7 (very high similarity) (see Table 4.2). The cosine distance was used to measure semantic similarity $d(a, b) = \cos(\mathbf{i}_a, \mathbf{i}_b)$ and the Spearman's correlation coefficient $\rho \in [-1, 1]$ was used to measure the distance between model scores and human judgements.

I compare the IORNN model against two models: the Recursive Auto-encoder (RAE) replicated by Blacoe and Lapata [2012], and the Combinatory Categorial Autoencoders (CCAE)-B model proposed by Hermann and Blunsom [2013]. The CCAE-B model is similar to the RAE model but uses separate weight matrices for each CCG rule. Table 4.3 shows results. The IORNN [CFG-0] (which, like the basic RAE, does not make use of syntactic category information) outperforms the RAE by a large margin. When syntactic category information is used in the IORNN [CCG] and CCAE-B, we can observe improvements across the board, with the IORNN [CCG] outperforming all other models on verb-obj and adj-noun compounds.

4.6 Supervised Learning with IORNNs: Semantic Role Labelling

Differing from the previous section, in this section I focus on a supervised learning task, namely semantic role labelling (SRL), where we learn from data that do contain

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

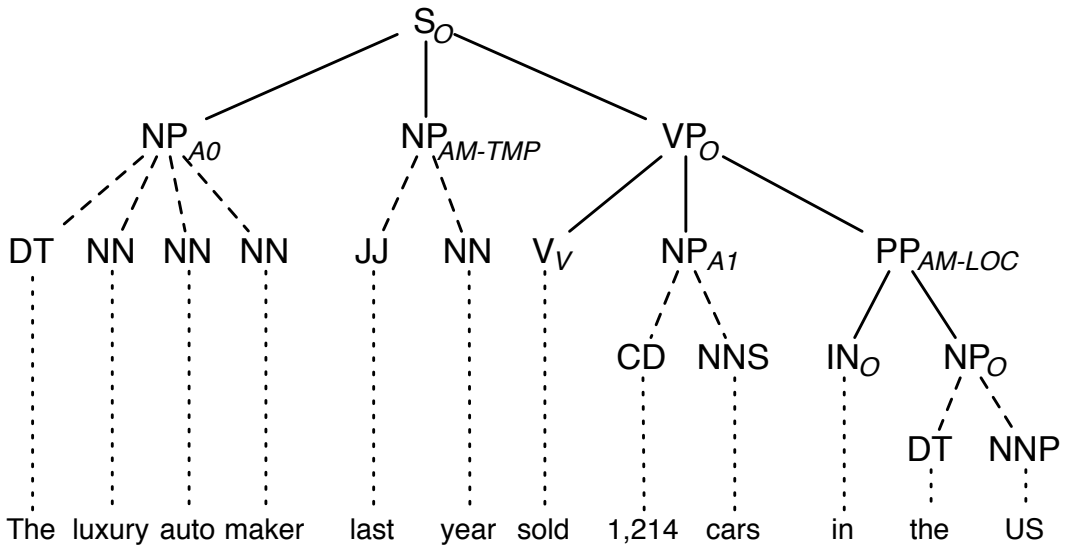


Figure 4.6: An annotated tree in the CoNLL2005 shared task training dataset. The subscripts show the role labels with respect to the predicate "sold". Reproduced from Cohn and Blunsom [2005].

semantic annotations. Semantic role labelling is the task of detecting semantic arguments for a verb or predicate in a sentence and identifying their roles. For instance, in the sentence

Raquel *baked* a cake.

"Raquel" serves as the *agent* for the action "baked" whereas "a cake" is its *theme*, which identifies the entity undergoing change in the event. The interaction between the outside and inside representations at a node is now determined by the semantic role assigned to the constituent the node covers.

The used benchmark is the CoNLL2005 shared task [Carreras and Màrquez, 2005]. In this task, given a sentence and a verb in this sentence, the system is required to classify every constituent into one of 54 classes, corresponding to 53 roles or class '-NONE-' (which means the constituent is not an argument). Figure 4.6 shows an example sentence and its parse tree, where all arguments of the verb "sold" are labeled with their roles. Following the PropBank formalism, roles A0-5 are assigned to words that are arguments of verbs (e.g., A0 for agent, experiencer; A1 for patient, theme; and so on.) In addition, there are several modifier tags such as AM-LOC (locational) and AM-TMP (temporal). The given dataset consists of a training set (sections 2-21 of the Penn WSJ treebank), a development set (section 24), and a test set (section 23 plus three sections from the Brown corpus). All the sentences were parsed by the Charniak parser [Charniak, 2001]. The evaluation metric is the F1 measure.

4.6.1 Model

The new IORNN model is very similar to the one proposed in Section 4.4 except two changes. First, I concatenate the inside representation of every word with an extra bit, indicating whether this word is a target verb or not. Second, the inside representation of the head word of a constituent is used to compute the constituent's inside representation. This is consistent with the head-driven parsing theory of Collins [1997], and is proposed in Chapter 3. At the same time, to force the model to focus on searching for arguments for the target verb, the inside representation of the target verb is used to compute the outside representation of every constituent. Therefore, I compute inside and outside representations for the node p_1 (Figure 4.3) as follows:

$$\mathbf{i}_{p_1} = f(\mathbf{W}_1^i \mathbf{i}_x + \mathbf{W}_2^i \mathbf{i}_y + \mathbf{W}_h \mathbf{i}_h + \mathbf{b}^i)$$
$$\mathbf{o}_{p_1} = g(\mathbf{W}_p^o \mathbf{o}_{p_2} + \mathbf{W}_2^o \mathbf{i}_z + \mathbf{W}_v \mathbf{i}_v + \mathbf{b}^o)$$

where $\mathbf{W}_h, \mathbf{W}_v \in \mathbb{R}^{n \times n}$; $\mathbf{i}_h, \mathbf{i}_v \in \mathbb{R}^n$ are the inside representations of the head word of the constituent p_1 and of the target verb, respectively. Finally, I use a *softmax* function to compute the probability that a class c is assigned to a constituent p:

$$P(c|p) = \frac{e^{u(p,c)}}{\sum_{c' \in C} e^{u(p,c')}} \tag{4.7}$$

where

$$[u(p, c_1), ..., u(p, c_{|C|})]^T = \mathbf{W}^o \mathbf{o}_p + \mathbf{W}^i \mathbf{i}_p + \mathbf{b}$$

C is the set of all classes, $\mathbf{W}^o, \mathbf{W}^i \in \mathbb{R}^{|C| \times n}$, $\mathbf{b} \in \mathbb{R}^{|C|}$. I use the method presented in Section 4.4 to train the model, where the objective function is the regularized cross-entropy over all constituents, i.e.,

$$J(\theta) = -\frac{1}{|D|} \sum_{T \in D} \sum_{p \in T} \sum_{i=1}^{|C|} t_i \times log(P(c_i|p)) + \frac{\lambda}{2} ||\theta||^2$$

where $t_i = 1$ if c_i is the correct class of p, $t_i = 0$ otherwise; D is the set of training parse trees; and λ is a regularization parameter.

IORNN-mixture

Combining neural networks is a simple way to increase the general performance [Mikolov et al., 2011]. To combine m IORNNs, I simply compute an average of the probabilities that IORNNs assign to a class c by

$$P(c|p) = \frac{1}{m} \sum_{i=1}^{m} P_i(c|p)$$

where P_i is the probability given by the i-th IORNN.

Model	F1-valid	F1-test
SENNA (w.o. syntax)	72.29	74.15
SENNA	–	75.49
IORNN	74.04	73.88
IORNN-mixture	–	75.36
IORNN+Charniak.RS	75.78	75.14
IORNN-mixture+Charniak.RS	–	**77.14**

Table 4.4: Performance on the SRL task of the IORNN compared with SENNA. IORNN+Charniak.RS is the IORNN model evaluated on the sentences parsed by the self-trained re-ranked Charniak parser.

4.6.2 Experiment: CoNLL 2005 Shared Task

I built an IORNN [CFG-200], with $n = 50$, initialised with the 50-dim Collobert & Weston word-embeddings [Collobert et al., 2011b]. Because the number of constituents in the class '-NONE-' is significantly larger than the numbers of constituents in other classes, I employed the simple pruning algorithm proposed by Xue and Palmer [2004] to eliminate those constituents that are very unlikely arguments. I also built an IORNN-mixture which is a combination of ten IORNNs that have the same configuration.

In this experiment, I compare the IORNN with SENNA [Collobert et al., 2011b], a state-of-the-art neural net system for POS tagging, chunking, named entity recognising and semantic role labelling. Differing from the IORNN, SENNA's neural net uses a convolutional layer to combine representations of the words in a sentence given the target verb position. In addition, SENNA is biased to produce valid tag sequences, by a mechanism that combines network scores with transition scores. Finally, SENNA may also make use of syntactic constraints (computed by its own network). By contrast, in my IORNN semantic tags are predicted solely on the basis of the computed inside and outside representations for a given node.

Table 4.4 reports results. We can see that SENNA without using syntax performs comparably with the (single) IORNN; when syntactic information is included, SENNA outperforms the IORNN. I conjecture that this is because SENNA is relatively insensitive to errors introduced by automatic parsers, thanks to its convolutional layer and transition matrix. My IORNN, on the other hand, is very sensitive to such errors because it relies on the principle of compositionality.[10] The performance of my system, however, is boosted (by 1.48%) by combining ten IORNNs, yielding comparable performance with SENNA. Furthermore, by using a better parser (I used the state-of-the-art self-trained and discriminative re-ranked Charniak-Johnson parser [Charniak and Johnson, 2005, McClosky et al., 2006]) I increased the performance even further (1.26% for IORNN and 1.78% for IORNN-mixture). This confirms the importance of

[10] In Chapter 7 I propose a solution to this.

accurate syntactic parsing in semantic role labelling [Punyakanok et al., 2008].

4.7 Other Applications

As I emphasize in Section 4.1, the IORNN model has a wider range of applications than the traditional RNN model. In the two sections above, I present how the IORNN model can be applied to three tasks: word prediction, learning composition functions from raw texts, and semantic role labelling. In this section, I introduce two more tasks to which the RNN model is not applicable but the IORNN model is.

4.7.1 Top-down prediction with IORNNs: Dependency parsing

I briefly consider the use of the IORNN model in top-down prediction tasks, such as needed for Earley-style parsing and left-corner parsing, but also for defining neural models to replace top-down generative probabilistic models such as those of Collins [1997] (generating phrase-structure trees) and Eisner [1996b] (generating dependency structures). I limit myself here to Eisner's generative model, but similar observations hold for the other applications.

Eisner's generative model is simple. Every sentence is assumed to have a single head word, generated from ROOT. This head word then first generates 0 or more dependents to its left, and then generates 0 or more dependents to its right. Each of the dependents recursively generates left and right dependents as well, until no more words are generated. Eisner considers a number of variants of this generative story, and defines ways to estimate the probabilities of each step in generating a full dependency structure from counts in a training corpus, smoothed using back-off. At each of these steps we have a partial dependency tree (the "conditioning context", representing the results of the generation process up to this point), and a representation of the dependent we are about to generate (the "event").

Can we define a neural network model to estimate these probabilities, to replace the counting and back-off? This requires a neural representation of the conditioning context, and a neural representation of the event. In a standard RNN the latter is available (the vector representation of the word or subtree), but the former is not. In an IORNN, on the other hand, we can use the outside representation as a representation for the conditioning context, and train a softmax classifier to compute the required conditional probabilities. In Chapter 5 I describe the technical details of such a model, the approximations needed to make it efficient, and state-of-the-art results in dependency parsing using an IORNN as a reranker on the output of the third-party MST-parser [McDonald et al., 2006]. For the purposes of this chapter, where I discuss the range of applications that the Inside-Outside Semantics opens up for neural models, it suffices to note that the IORNN model can successfully implement top-down prediction.

4.7.2 Discourse Relation Classification

At a discourse level, there is a challenging task, namely discourse relation classification, that identifies the relation between two discourse arguments. For instance, the implicit discourse relation between following two sentences

> Milos gave *Raquel* the sangria.
> *She* was thirsty.

should be *because* thanks to the token "sangria" in the first sentence and "thirsty" in the second one. Therefore, a discriminative classification on two bottom-up vector-based representations for two sentences seems to be adequate. It is, however, not true because if we replace the token "she" by "he", i.e.

> *Milos* gave Raquel the sangria.
> *He* was thirsty.

the discourse relation turns to be *though*. The problem, as pointed out by Ji and Eisenstein [2015], is that bottom-up representations are not aware of the entity roles in the discourse. Ji and Eisenstein [2015] thus propose using an IORNN[11] to compute an outside representation for each entity. Integrating those outside representations to a system using bottom-up representations and surface features, they get a 4% improvement in accuracy over the best previous work in this task.

4.8 Conclusion

I propose the Inside-Outside Semantics framework in which every node in a parse tree is associated with a pair of representations, the inside representation for representing the contents under the node, and the outside representation for representing its surrounding context. Thanks to adding context representations, information can flow not only bottom-up but also top-down in parse trees. The Inside-Outside Recursive Neural Network is presented as an instance of the framework, where both representations are instantiated as high-dimensional vectors and composition functions are feed-forward networks trained on large datasets. By exploiting the interaction between context and content representations, I show that the IORNN model is capable of performing a wide range of NLP tasks that the RNN model and its extensions are not.

I demonstrate how the IORNN model can be used for (1) word prediction, (2) learning composition functions from raw texts, and (3) semantic role labelling. It is worth noting that these tasks were chosen because the tradition RNN model alone is not capable of tackling them: (1) and (3) require context awareness, whereas (2) is an unsupervised learning task. The IORNN model outperforms the Kneser-Ney smoothed 5-gram model, which is state-of-the-art in the language modelling area, on

[11] Their network model, although not named, is identical to my IORNN.

the word prediction task. It also achieves better results than the RAE model by a large margin in the phrase similarity evaluation. On the semantic role labelling task, the IORNN model performs comparably with SENNA, a state-of-the-art neural-net-based system for NLP. These empirical results establish that Inside-Outside Semantics offers a promising framework for applying neural networks to NLP tasks that require paying attention to the hierarchical structure of sentences and *predictions* based on contexts.

In this chapter, I assume that sentences stand alone and hence have initialised the outside representation at every root node with the same vector $\mathbf{o}_\emptyset$. To relax this assumption, we can make use of external context such as neighbouring sentences and background knowledge.

Chapter 5

The Inside-Outside Recursive Neural Network model for Dependency Parsing

I propose the first implementation for an ∞-order generative dependency model. This implementation is based on the Inside-outside Recursive Neural Network model, which is introduced in Chapter 4. Empirical results on the English section of the Universal Dependency Treebank show that my ∞-order model achieves a perplexity seven times lower than the traditional third-order model using counting, and tends to choose more accurate parses in k-best lists. In addition, reranking with this model achieves state-of-the-art unlabelled attachment scores and unlabelled exact match scores. [1]

5.1 Introduction

Estimating probability distributions is the core issue in modern, data-driven natural language processing methods. Because of the traditional definition of discrete probability

$$Pr(A) \equiv \frac{\text{the number of times that A occurs}}{\text{the size of the event space}}$$

counting has become a standard method to tackle the problem. When data are sparse, smoothing techniques are needed to adjust counts for non-observed or rare events. However, successful use of those techniques has turned out to be an art. For instance, much skill and expertise is required to create reasonable reduction lists for back-off, and to avoid impractically large count tables, which store events and their counts.

An alternative to counting for estimating probability distributions is to use neural networks. Thanks to recent advances in deep learning, this approach has recently started to look very promising again, with state-of-the-art results in sentiment analysis [Socher et al., 2013b], language modelling [Mikolov et al., 2010], and other tasks. The Mikolov et al. [2010] work, in particular, demonstrates the advantage of neural-network-based approaches over counting-based approaches in language modelling: it

[1] My source code is available at: `github.com/lephong/iornn-depparse`.

shows that recurrent neural networks are capable of capturing long histories efficiently and surpass standard n-gram techniques (e.g., Kneser-Ney smoothed 5-grams).

In this chapter, keeping in mind the success of these models, I compare the two approaches. Complementing recent work that focuses on such a comparison for the case of finding appropriate word vectors [Baroni et al., 2014], I focus here on models that involve more complex, hierarchical structures. Starting with existing generative models that use counting to estimate probability distributions over constituency and dependency parses (e.g., Eisner [1996b], Collins [2003]), I develop an alternative based on the IORNN model that is introduced in Chapter 4. This is a non-trivial task because, to my knowledge, no existing neural network architecture other than the IORNN can be used in this way. For instance, classic recurrent neural networks [Elman, 1990] unfold to left-branching trees, and are not able to process arbitrarily shaped parse trees that the counting approaches are applied to. Recursive neural networks [Socher et al., 2010] and extensions [Socher et al., 2012, Le et al., 2013], on the other hand, do work with trees of arbitrary shape, but process them in a bottom-up manner. The probabilities we need to estimate are, by contrast, defined by top-down generative models, or by models that require information flows in both directions (e.g., the probability of generating a node depends on the whole fragment rooted at its just-generated sister).

Because the IORNN model allows information to flow in any directions, depending on the application, it is a natural tool for estimating probabilities in tree-based generative models. I therefore will apply it to an ∞-order generative dependency model which is impractical for counting due to the problem of data sparsity. Counting, instead, is used to estimate a third-order generative model as in Sangati et al. [2009] and Hayashi et al. [2011]. My empirical results show that the new model not only achieves a seven times lower perplexity than the third-order model, but also tends to choose more accurate candidates in k-best lists. In addition, reranking with this model achieves state-of-the-art scores on the task of supervised dependency parsing.

The outline of the chapter is following. Firstly, I give an introduction to Eisner's generative model in Section 5.2. Then, I present the third-order model using counting in Section 5.3, and the ∞-order model using the IORNN model in Section 5.4. Finally, in Section 5.5 I show empirical results.

5.2 Eisner's Generative Model

Eisner [1996b] proposes a generative model for dependency parsing. The generation process is top-down: starting at the ROOT, it generates left dependents and then right dependents for the ROOT. After that, it generates left dependents and right dependents for each of ROOT's dependents. The process recursively continues until there are no further dependents to generate. The whole process is captured in the following formula

$$P(T(H)) = \prod_{l=1}^{L} P\left(H_l^L | \mathcal{C}_{H_l^L}\right) P\left(T(H_l^L)\right) \times \prod_{r=1}^{R} P\left(H_r^R | \mathcal{C}_{H_r^R}\right) P\left(T(H_r^R)\right)$$

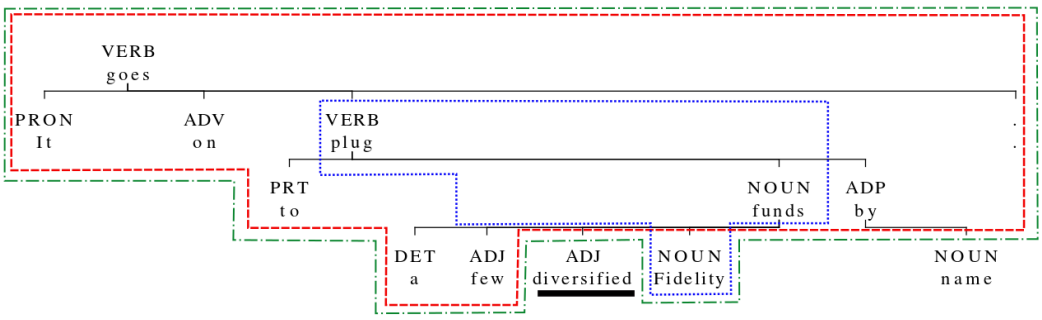


Figure 5.1: Example of different orders of context of "diversified". The blue dotted shape corresponds to the third-order outward context, while the red dashed shape corresponds to the ∞-order left-to-right context. The green dot-dashed shape corresponds to the context to compute the outside representation.

where H is the current head, $T(N)$ is the fragment of the dependency parse rooted at N, and $\mathcal{C}_N$ is the context in which N is generated. H^L, H^R are respectively H's left dependents and right dependents, plus EOC (End-Of-Children), a special token to indicate that there are no more dependents to generate. Thus, $P(T(ROOT))$ is the probability of generating the entire dependency structure T. I refer to $\langle H_l^L, \mathcal{C}_{H_l^L}\rangle$, $\langle H_r^R, \mathcal{C}_{H_r^R}\rangle$ as "events", and $\langle \mathcal{C}_{H_l^L}\rangle$, $\langle \mathcal{C}_{H_r^R}\rangle$ as "conditioning contexts".

In order to avoid the problem of data sparsity, the conditioning context in which a dependent D is generated might capture only part of the fragment generated so far. Based on the amount of information that contexts hold, we can define the *order* of a generative model (see [Hayashi et al., 2011, Table 3] for examples)

- first-order: $\mathcal{C}_D^1$ contains the head H,
- second-order: $\mathcal{C}_D^2$ contains H and the just-generated sibling S,
- third-order: $\mathcal{C}_D^3$ contains H, S, the sibling S' before S (tri-sibling); or H, S and the grand-head G (the head of H) (grandsibling) (the fragment enclosed in the blue dotted contour in Figure 5.1),
- ∞-order: $\mathcal{C}_D^\infty$ contains all of D's ancestors, theirs siblings, and its generated siblings (the fragment enclosed in the red dashed contour in Figure 5.1).

In the original models [Eisner, 1996a], each dependent D is a 4-tuple $\langle dist, w, c, t\rangle$

- $dist(H, D)$ the distance between D and its head H, represented as one of the four ranges 1, 2, 3-6, 7-∞.
- $\underline{w}ord(D)$ the lowercase version of the word of D,
- $\underline{c}ap(D)$ the capitalisation feature of the word of D (all letters are lowercase, all letters are uppercase, the first letter is uppercase, the first letter is lowercase),

- $\underline{t}ag(D)$ the POS-tag of D,

Here, to make the dependency complete, $\underline{d}eprel(D)$, the dependency relation of D (e.g., SBJ, DEP), is also taken into account.

5.3 The third-order Model with Counting

The third-order model I suggest is similar to the grandsibling models proposed by Sangati et al. [2009] and Hayashi et al. [2011]. It defines the probability of generating a dependent $D = \langle dist, d, w, c, t \rangle$ as a product of the distance-based probability and the probabilities of generating each of its components. Each of these probabilities is smoothed using back-off according to the following reduction lists:

$$
\begin{aligned}
P(D|\mathcal{C}_D) =& P(dist(H,D), dwct(D)|H,S,G,dir) \\
=& P(d(D)|H,S,G,dir) \\
& \text{reduction list: } \boxed{\begin{array}{|l|}\hline tw(H),tw(S),tw(G),dir \\ \hline tw(H),tw(S),t(G),dir \\ \hline \left\{\begin{array}{l} tw(H),t(S),t(G),dir \\ t(H),tw(S),t(G),dir \end{array}\right. \\ \hline t(H),t(S),t(G),dir \\ \hline \end{array}} \\
& \times P(t(D)|d(D),H,S,G,dir) \\
& \text{reduction list: } \boxed{\begin{array}{|l|}\hline d(D),dtw(H),t(S),dir \\ \hline d(D),d(H),t(S),dir \\ \hline d(D),d(D),dir \\ \hline \end{array}} \\
& \times P(w(D)|dt(D),H,S,G,dir) \\
& \text{reduction list: } \boxed{\begin{array}{|l|}\hline dtw(H),t(S),dir \\ \hline dt(H),t(S),dir \\ \hline \end{array}} \\
& \times P(c(D)|dtw(D),H,S,G,dir) \\
& \text{reduction list: } \boxed{\begin{array}{|l|}\hline tw(D),d(H),dir \\ \hline tw(D),dir \\ \hline \end{array}} \\
& \times P(dist(H,D)|dtwc(D),H,S,G,dir) \qquad (5.1) \\
& \text{reduction list: } \boxed{\begin{array}{|l|}\hline dtw(D),dt(H),t(S),dir \\ \hline dt(D),dt(H),t(S),dir \\ \hline \end{array}}
\end{aligned}
$$

The reason for generating the dependency relation first is based on the similarity between relation/dependent and role/filler: we generate a role and then choose a filler for that role.

Here, we can notice that a reduction list is a list of conditions sorted by their fineness; for instance in the first list $\big(tw(H), tw(S), t\underline{w}(G), dir\big)$ contains more information than $\big(tw(H), tw(S), t(G), dir\big)$. Because the finer a condition is, the less data we

can find in corpora to estimate the probabilities, coarser conditions are needed to avoid the problem of data sparsity. How to calculate the probabilities based on reduction lists is given below.

Back-off

The back-off parameters are identical to those reported in Eisner [1996b]. To estimate the probability $P(A|context)$ given a reduction list $L = (l_1, l_2, ..., l_n)$ of $context$, let

$$p_i = \begin{cases} \frac{count(A,l_i)+0.005}{count(l_i)+0.5} & \text{if } i = n \\ \frac{count(A,l_i)+3p_{i+1}}{count(l_i)+3} & \text{otherwise} \end{cases}$$

then $P(A|context) = p_1$.

5.4 The ∞-order Model with IORNN

To apply the IORNN architecture to dependency parsing I need to adapt the definitions somewhat. In particular, in the generative dependency model, every step in the generative story involves a decision to generate a specific word while the span of the subtree that this word will dominate only becomes clear when all dependents are generated. I therefore introduce *partial outside representation* as a representation of the current context of a word in the generative process, and compute the final outside representation only after all its siblings have been generated.

Consider an example of head h and its dependents x, y (we ignore directions for simplicity) in Figure 5.2. Assume that we are in the state in the generative process where the generation of h is complete, i.e. we know its inside and outside representations $\mathbf{i}_h$ and $\mathbf{o}_h$. Now, when generating h's first dependent x (see Figure 5.2-a), we first compute x's *partial* outside representation (representing its context at this stage in the process), which is a function of the outside representation of the head (representing the head's context) and the inside representation of the head (representing the content of the head word):

$$\bar{\mathbf{o}}_1 = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o)$$

where $\mathbf{W}_{hi}, \mathbf{W}_{ho} \in \mathbb{R}^{n \times n}$, $\mathbf{b}_o \in \mathbb{R}^n$, f is an activation function.

With the context of the first dependent determined, we can proceed and generate its content. For this purpose, we assume a separate weight matrix $\mathbf{W}$, trained (as explained below) to predict a specific word given a partial outside representation. To compute a proper probability for word x, I use the following softmax function:

$$softmax(x, \bar{\mathbf{o}}_1) = \frac{e^{u(x,\bar{\mathbf{o}}_1)}}{\sum_{w \in V} e^{u(w,\bar{\mathbf{o}}_1)}}$$

where

$$\left[u(w_1, \bar{\mathbf{o}}_1), ..., u(w_{|V|}, \bar{\mathbf{o}}_1)\right]^T = \mathbf{W}\bar{\mathbf{o}}_1 + \mathbf{b}$$

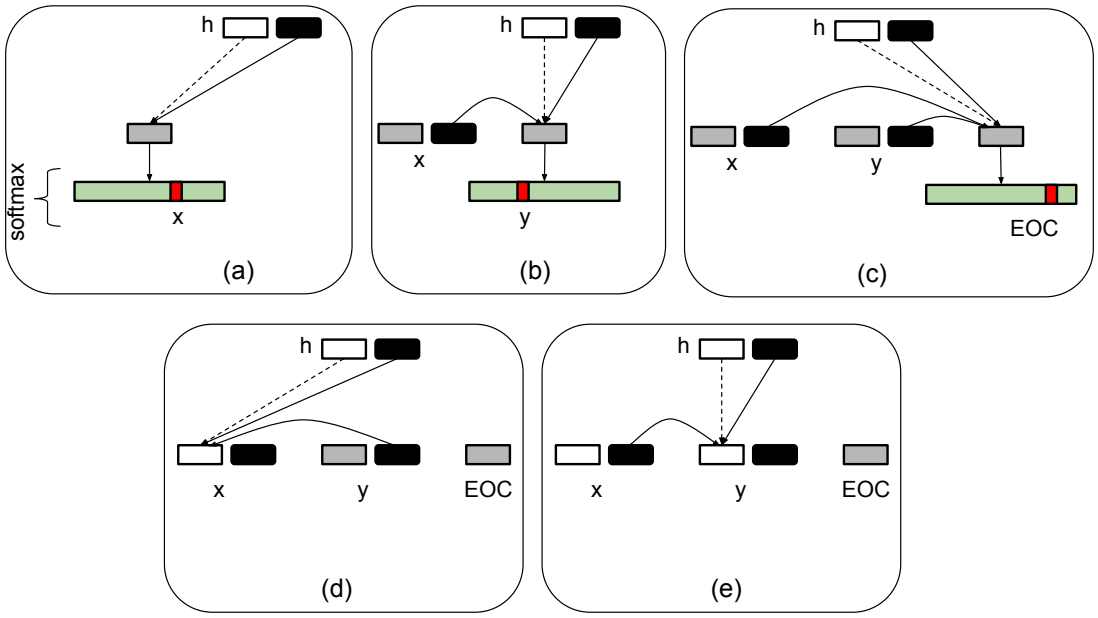


Figure 5.2: Example of applying the IORNN model to dependency parsing. Black, grey, white boxes are respectively inside, partial outside, and outside representations. For simplicity, only links related to the current computation are drawn (see text).

and V is the set of all possible dependents.

Note that since $\mathbf{o}_h$, the outside representation of h, represents the entire dependency structure generated up to that point, $\bar{\mathbf{o}}_1$ is a vector representation of the ∞-order context generating the first dependent (like the fragment enclosed in the red dashed contour in Figure 5.1). The softmax function thus estimates the probability $P(D = x|\mathcal{C}_D^\infty)$.

The next step, now that x is generated, is to compute the partial outside representation for the second dependent (see Figure 5.2-b)

$$\bar{\mathbf{o}}_2 = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{dr(x)}\mathbf{i}_x + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o)$$

where $\mathbf{W}_{dr(x)} \in \mathbb{R}^{n\times n}$ is specific for the dependency relation of x with h.

Next y is generated (using the softmax function above), and the partial outside representation for the third dependent (see Figure 5.2-c) is computed:

$$\bar{\mathbf{o}}_3 = f(\mathbf{W}_{hi}\mathbf{i}_h + \frac{1}{2}\left(\mathbf{W}_{dr(x)}\mathbf{i}_x + \mathbf{W}_{dr(y)}\mathbf{i}_y\right) + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o)$$

Since the third dependent is the End-of-Children symbol (EOC), the process of generating dependents for h stops. We can then return to x and y to replace the partial

outside representations with complete outside representations[2] (see Figure 5.2-d,e):

$$\mathbf{o}_x = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{dr(y)}\mathbf{i}_y + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o)$$
$$\mathbf{o}_y = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{dr(x)}\mathbf{i}_x + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o)$$

Generalizing from this example of computing partial and complete outside representations, we have: if u is the first dependent of h then

$$\bar{\mathbf{o}}_u = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o)$$

otherwise

$$\bar{\mathbf{o}}_u = f(\mathbf{W}_{hi}\mathbf{i}_h + \frac{1}{|\bar{\mathcal{S}}(u)|}\sum_{v\in\bar{\mathcal{S}}(u)}\mathbf{W}_{dr(v)}\mathbf{i}_v + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o)$$

where $\bar{\mathcal{S}}(u)$ is the set of u's sisters generated before it. And, if u is the only dependent of h (ignoring EOC) then

$$\mathbf{o}_u = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o)$$

otherwise

$$\mathbf{o}_u = f(\mathbf{W}_{hi}\mathbf{i}_h + \frac{1}{|\mathcal{S}(u)|}\sum_{v\in\mathcal{S}(u)}\mathbf{W}_{dr(v)}\mathbf{i}_v + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o)$$

where $\mathcal{S}(u)$ is the set of u's siblings.

We then continue this process to generate dependents for x and y until the process stops.

Inside Representations

In the calculation of the probability of generating a word, described above, I assume inside representations of all possible words to be given. These are, in fact, themselves a function of vector representations for the words (in our case, the word vectors are initially borrowed from Collobert et al. [2011b]), the POS-tags and capitalisation features. That is, the inside representation at a node h is given by:

$$\mathbf{i}_h = f\left(\mathbf{W}_w\mathbf{w}_h + \mathbf{W}_p\mathbf{p}_h + \mathbf{W}_c\mathbf{c}_h\right)$$

where $\mathbf{W}_w \in \mathbb{R}^{n\times d_w}$, $\mathbf{W}_p \in \mathbb{R}^{n\times d_p}$, $\mathbf{W}_c \in \mathbb{R}^{n\times d_c}$, $\mathbf{w}_h$ is the word vector of h, and $\mathbf{p}_h, \mathbf{c}_h$ are respectively one-hot vectors representing the POS-tag and capitalisation feature of h.

[2] According to the IORNN architecture, to compute the outside representation of a node, the inside representations of the whole fragments rooted at its sisters must be taken into account. Here, I approximate the inside representation of a fragment by the inside representation of its head since the meaning of a phrase is often dominated by the meaning of its head.

Training

Training this IORNN is to minimise the following objective function, which is the regularised cross-entropy, w.r.t. θ

$$J(\theta) = -\frac{1}{m}\sum_{T\in\mathcal{D}}\sum_{w\in T}\log(P(w|\bar{\mathbf{o}}_w)) + \frac{1}{2}\big(\lambda_W\|\theta_W\|^2 + \lambda_L\|\theta_L\|^2\big)$$

where θ is the parameter set, $\mathcal{D}$ is the set of training dependency parses, m is the number of dependents; θ_W, θ_L are the weight matrix set and the word embeddings ($\theta = (\theta_W, \theta_L)$); λ_W, λ_L are regularisation hyper-parameters.

Implementation

I decompose a dependent D into four features: dependency relation, POS-tag, lowercase version of word, capitalisation feature of word. I then factorise $P(D|\mathcal{C}_D^\infty)$ similarly to Equation 5.1, where each component is estimated by a softmax function instead of a reduction list.

5.5 Experiments

In the following experiments, I converted the Penn Treebank to dependencies using the Universal dependency annotation [McDonald et al., 2013][3]; this yielded a dependency tree corpus I call PTB-U. In order to compare with other systems, I also experimented with an alternative conversion using the head rules of Yamada and Matsumoto [2003][4]; this yielded a dependency tree corpus I call PTB-YM. Sections 2-21 were used for training, section 22 for development, and section 23 for testing. For the PTB-U, the gold POS-tags were used. For the PTB-YM, the development and test sets were tagged by the Stanford POS-tagger[5] trained on the whole training data, whereas 10-way jackknifing was employed to generate tags for the training set.

The vocabulary for both models, the third-order model and the ∞-order model, was given by the list of words occurring more than two times in the training data. All other words were labelled 'UNKNOWN' and every digit was replaced by '0'. For the IORNN used by the ∞-order model, I set $n = 200$, and defined f as the $tanh$ activation function. I initialised it with the 50-dim word embeddings from Collobert et al. [2011b] and trained it with the learning rate 0.1, $\lambda_W = 10^{-4}, \lambda_L = 10^{-10}$.

[3] https://code.google.com/p/uni-dep-tb/
[4] http://stp.lingfil.uu.se/~nivre/research/Penn2Malt.html
[5] http://nlp.stanford.edu/software/tagger.shtml

	Perplexity
3rd-order model	1736.73
∞-order model	**236.58**

Table 5.1: Perplexities of the two models on PTB-U-22.

back-off level	d	t	w	c
0	47.4	61.6	43.7	87.7
1	69.8	98.4	77.8	97.3
2	76.0, 89.5	99.7		
3	97.9			
total	76.1	86.6	60.7	92.5

Table 5.2: Percentages of events extracted from PTB-U-22 appearing more than two times in the training set. Events are grouped according to the reduction lists in Equation 5.1. d, t, w, c stand for dependency relation, POS-tag, word, and capitalisation feature.

5.5.1 Perplexity

I firstly evaluated the two models on PTB-U-22 using the perplexity-per-word metric:

$$ppl(P) = 2^{-\frac{1}{N}\sum_{T\in\mathcal{D}}\log_2 P(T)}$$

where $\mathcal{D}$ is a set of dependency parses, N is the total number of words. It is worth noting that, the better P estimates the true distribution P^* of $\mathcal{D}$, the lower its perplexity is. Because Eisner's model with the $dist(H, D)$ feature (Equation 5.1) is leaky (the model allocates some probability to events that can never legally arise), this feature was discarded (only in this perplexity experiment).

Table 5.1 shows results. The perplexity of the third-order model is more than seven times higher than the ∞-order model. This reflects the fact that data sparsity is more problematic for counting than for the IORNN.

To investigate why the perplexity of the third-order model is so high, I computed the percentages of events extracted from the development set appearing more than two times in the training set. Events are grouped according to the reduction lists in Equation 5.1 (see Table 5.2). We can see that reductions at level 0 (the finest) for dependency relations and words seriously suffer from the data sparsity: more than half of the events occur less than three times, or not at all, in the training data. I thus conclude that counting-based models heavily rely on carefully designed reduction lists for back-off.

5.5.2 Reranking

In the second experiment, I evaluated the two models in the reranking framework proposed by Sangati et al. [2009] on PTB-U. I employed the MSTParser (with the 2nd-

order feature mode) [McDonald et al., 2005] to generate k-best lists. Two evaluation metrics are labelled attachment score (LAS) and unlabelled attachment score (UAS), *including* punctuation.

Rerankers

Given $\mathcal{D}(S)$, a k-best list of parses of a sentence S, I define the *generative* reranker as

$$T^* = \arg\max_{T \in \mathcal{D}(S)} P(T(ROOT))$$

which is identical to the one of Sangati et al. [2009]. Moreover, as in many mixture-model-based approaches, I define the *mixture* reranker as a combination of the generative model and the MST discriminative model, which is similar to the approach of Hayashi et al. [2011]:

$$T^* = \arg\max_{T \in \mathcal{D}(S)} \{\alpha \log P(T(ROOT)) + (1-\alpha) s(S,T)\}$$

where $s(S,T)$ is the score function given by the MSTParser, and $\alpha \in [0,1]$.

Results

Figure 5.3 shows UASs of the generative reranker on the development set. The MSTParser achieves 92.32% and the Oracle achieves 96.23% when $k = 10$. With the third-order model, the generative reranker performs better than the MSTParser when $k < 6$ and the maximum improvement is 0.17%. Meanwhile, with the ∞-order model, the generative reranker always gains higher UASs than the MSTParser, and with $k = 6$, the difference reaches 0.7%. Figure 5.4 shows UASs of the mixture reranker on the same set. α was optimised by searching with the step-size 0.005. Unsurprisingly, we observe improvements over the generative reranker as the mixture reranker can combine the advantages of the two models.

Table 5.3 shows scores of the two rerankers on the test set with the parameters tuned on the development set. Both the rerankers, either using third-order or ∞-order models, outperform the MSTParser. The fact that both gain higher improvements with the ∞-order model shows that the IORNN surpasses counting.

5.5.3 Comparison with other systems

I first compare the mixture reranker using the ∞-order model against the state-of-the-art dependency parser TurboParser (with the full mode) [Martins et al., 2013] on PTB-U-23. Table 5.4 shows test LASs and UASs. When taking labels into account, the TurboParser outperforms the reranker. But without counting labels, the two systems perform comparably, and when ignoring punctuation the reranker even outperforms the TurboParser. This pattern is also observed when the exact match metrics are used (see

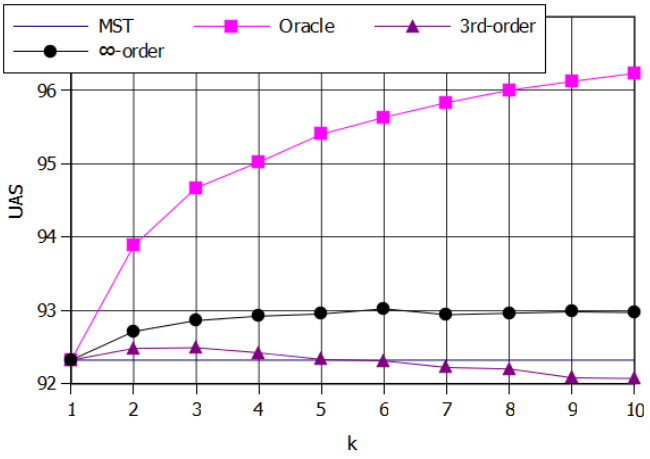


Figure 5.3: Performance of the generative reranker on PTB-U-22.

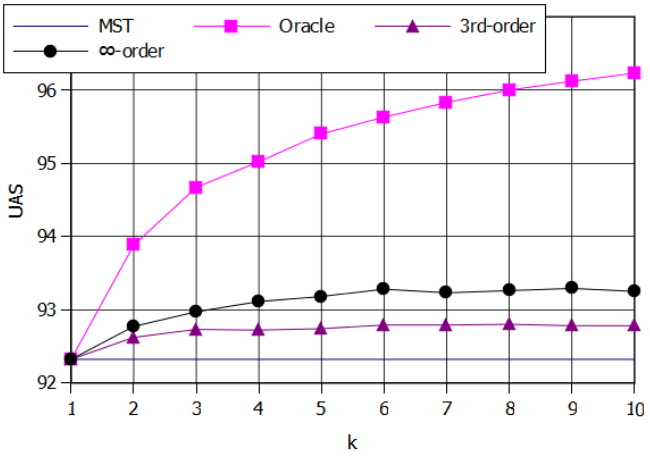


Figure 5.4: Performance of the mixture reranker on PTB-U-22. For each k, α was optimized with the step-size 0.005.

Table 5.4). This is due to the fact that the TurboParser performs significantly better than the MSTParser, which generates k-best lists for the reranker, in labelling: the former achieves a label accuracy score of 96.03% whereas the latter achieves 94.92%.

One remarkable point is that reranking with the ∞-order model helps to improve the exact match scores 4% - 6.4% (see Table 5.4). Because the exact match scores correlate with the ability of handling global structures, I conclude that the IORNN is able to capture large contexts. Figure 5.5 shows distributions of correct-head accuracy over CPOS-tags and Figure 5.6 shows F1-scores over binned HEAD distances (i.e., dependents are grouped w.r.t. the distances to their heads). Reranking with the ∞-order model is clearly helpful for all CPOS-tags and dependent-to-head distances, except a minor decrease on PRT.

I compare the reranker against other systems on PTB-YM-23 using the UAS metric

	LAS	UAS
MSTParser	89.97	91.99
Oracle ($k = 10$)	93.73	96.24
Generative reranker with		
3rd-order ($k = 3$)	90.27 (+0.30)	92.27 (+0.28)
∞-order ($k = 6$)	90.76 (+0.79)	92.83 (+0.84)
Mixture reranker with		
3rd-order ($k = 6$)	90.62 (+0.65)	92.62 (+0.63)
∞-order ($k = 9$)	**91.02 (+1.05)**	**93.08 (+1.09)**

Table 5.3: Comparison based on reranking on PTB-U-23. The numbers in the brackets are improvements over the MSTParser.

	LAS (w/o punc)	UAS (w/o punc)
MSTParser	89.97 (90.54)	91.99 (92.82)
w. ∞-order ($k = 9$)	91.02 (91.51)	**93.08** (**93.84**)
TurboParser	**91.56** (**92.02**)	93.05 (93.70)

(a) Attachment scores

	LEM (w/o punc)	UEM (w/o punc)
MSTParser	32.37 (34.19)	42.80 (45.24)
w. ∞-order ($k = 9$)	37.58 (39.16)	**49.17** (**51.53**)
TurboParser	**40.65** (**41.72**)	48.05 (49.83)

(b) Exact match scores

Table 5.4: Comparison with the TurboParser on PTB-U-23. LEM and UEM are respectively the labelled exact match score and unlabelled exact match score metrics. The numbers in brackets are scores computed excluding punctuation.

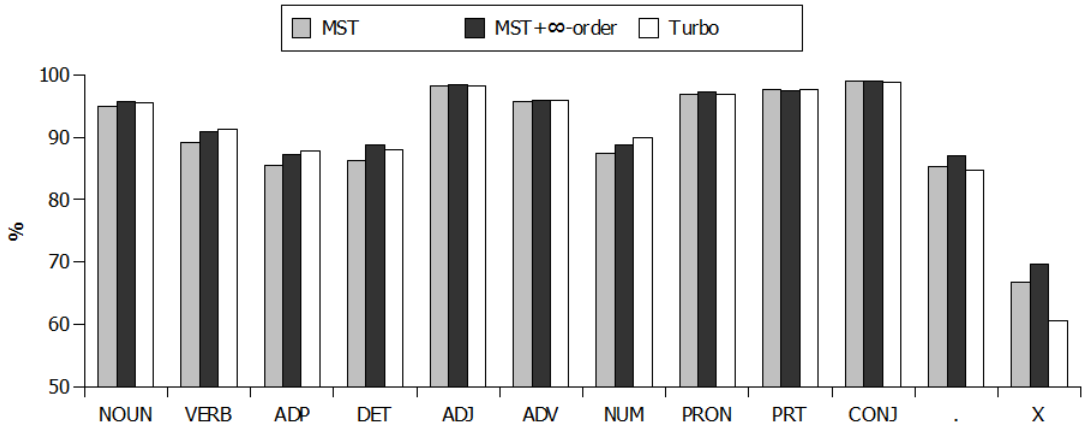


Figure 5.5: Distributions of correct-head accuracy over CPOS-tags (PTB-U-23).

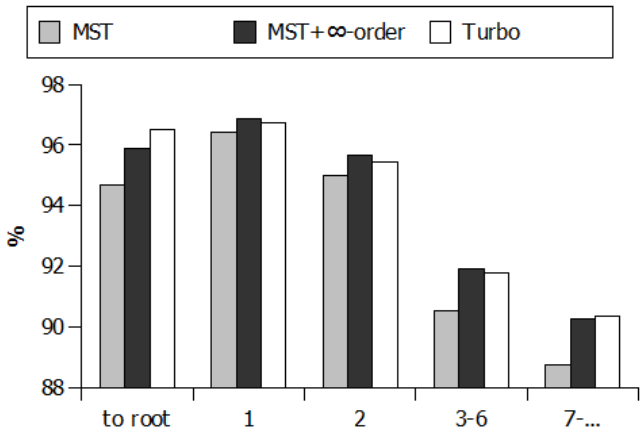


Figure 5.6: F1-score over binned HEAD distance (PTB-U-23).

System	UAS
Huang and Sagae [2010]	92.1
Koo and Collins [2010]	93.04
Zhang and McDonald [2012]	93.06
Martins et al. [2013]	93.07
Bohnet and Kuhn [2012]	93.39
Reranking	
Hayashi et al. [2011]	92.89
Hayashi et al. [2013]	93.12
MST+ ∞-order ($k = 12$)	**93.12**

Table 5.5: Comparison with other systems on PTB-YM-23 (excluding punctuation).

ignoring punctuation (as the standard evaluation for English) (see Table 5.5). My system performs slightly better than many state-of-the-art systems such as Martins et al. [2013] (a.k.a. TurboParser), Zhang and McDonald [2012], Koo and Collins [2010]. It outperforms Hayashi et al. [2011]'s reranker using a combination of third-order generative models with a variational model learnt on the fly; performs on par with Hayashi et al. [2013]'s discriminative reranker using the stacked technique; and slightly worse than Bohnet and Kuhn [2012], which is a hybrid transition-based and graphical-based approach.

5.6 Related Work

The idea of letting parsing decisions depend on arbitrarily long derivation histories is also explored in Borensztajn and Zuidema [2011] and is related to parsing frameworks that allow arbitrarily large elementary trees (e.g., Scha [1990], O'Donnell et al. [2009], Sangati and Zuidema [2011], and van Cranenburgh and Bod [2013]).

Titov and Henderson [2007] are the first proposing to use deep networks for dependency parsing. They introduce a transition-based generative dependency model using incremental sigmoid belief networks and apply beam pruning for searching best trees. Similarly, other recent neural-network-based approaches, such as the works of Stenetorp [2013], Chen and Manning [2014], and Dyer et al. [2015], also rely on transition-based parsing: neural networks are used to estimate distributions over a list of transition decisions. Differing from theirs, the neural network model in my work, the IORNN, is used to estimate probabilities for a top-down generative model. These probabilities are then used to rescore k-best candidates generated by an independent graph-based parser, namely the MSTParser.

Reranking k-best lists was introduced by Collins and Koo [2005] and Charniak and Johnson [2005]. Their rerankers are discriminative and for constituency parsing. Sangati et al. [2009] propose to use a third-order generative model for reranking k-best lists of dependency parses. Hayashi et al. [2011] then follow this idea but combine generative models with a variational model learnt on the fly to rerank forests. In this chapter, I also follow Sangati et al. [2009]'s idea but use an ∞-order generative model, which has never been used before. Recently, Zhu et al. [2015a], returning to Collins and Koo [2005]'s idea, propose to employ a recursive neural network model in a discriminative dependency reranker.

5.7 Conclusions

In this chapter, by applying the IORNN model to dependency parsing, I show that using an ∞-order generative model for dependency parsing, which has never been done before, is practical. My empirical results on the English section of the Universal Dependency Treebanks show that the ∞-order generative model approximates the true dependency distribution better than the traditional third-order model using counting, and tends to choose more accurate parses in k-best lists. In addition, reranking with this model even outperforms the state-of-the-art TurboParser on unlabelled score metrics.

The IORNN model for dependency parsing constitutes a detailed case study of the IOS framework proposed in Chapter 4 and illustrates the power of this framework. This model can, furthermore, be enhanced in different ways. For instance, in Le [2015], I point out that replacing softmax functions by hierarchical softmax functions can significantly speed up the reranker, and taking into account *complete* contexts can improve its performance in terms of accuracy. Moreover, we can even employ the model to tackle the problem of unsupervised dependency parsing. In Le and Zuidema [2015b], we propose an iterated reranking system which uses the reranker introduced in this chapter. The system outperforms the state-of-the-art unsupervised parser of Spitkovsky et al. [2013] on the Penn WSJ corpus by a large margin.

Chapter 6

Composition with Recursive Long Short Term Memory

I propose an extension of the recursive neural network model that makes use of a novel variant of the Long short-term memory (LSTM) architecture. The new model, called Recursive LSTM (RLSTM), allows information low in a parse tree to be stored in a memory register (the *memory cell*) and used much later higher up in the parse tree. This provides a solution to the vanishing gradient problem in the traditional RNN model, and allows the network to capture long range dependencies. Empirical results show that the proposed model outperforms the traditional RNN model on the Stanford Sentiment Treebank.

6.1 Introduction

Moving from lexical to compositional semantics in vectorial semantics requires answers to two difficult questions:

1. What is the nature of the composition functions (given that the lambda calculus for variable binding is no longer applicable)?

2. How do we learn the parameters of those functions (if they have any) from data?

A number of classes of functions have been proposed to answer the first question, including simple linear functions like vector addition [Mitchell and Lapata, 2009], non-linear functions like those defined by multi-layer neural networks [Socher et al., 2010], and vector-matrix multiplication and tensor linear mapping [Baroni et al., 2013]. The matrix and tensor-based functions have the advantage of allowing a relatively straightforward comparison with formal semantics, but the fact that multi-layer neural networks with non-linear activation functions like the sigmoid function can approximate any continuous function [Cybenko, 1989] already make them an attractive choice.

In trying to answer the second question, the advantages of approaches based on neural network architectures, such as the recursive neural network (RNN) model [Socher

et al., 2013b] and the convolutional neural network model of Kalchbrenner et al. [2014], are even clearer. Models in this paradigm can take advantage of general learning procedures based on back-propagation, and with the rise of *deep learning*, of a variety of efficient algorithms and established heuristics (e.g. gradient clipping, dropout) to further improve training. Since the first success of the RNN model [Socher et al., 2011b] in constituency parsing, two classes of extensions have been proposed. One class is to enhance the quality of composition by, for instance, using tensor products [Socher et al., 2013b] or concatenating RNNs horizontally to make a deeper net [Irsoy and Cardie, 2014]. The other class is to extend its topology to fulfill a wider range of tasks, like the IORNN model for dependency parsing (see Chapter 5) and the model of Paulus et al. [2014] for context-dependent sentiment analysis.

My proposal in this chapter is the Recursive Long Short Term Memory (RLSTM) model, an extension of the RNN model to improve the quality of the composition function it implements. My motivation is that, like training recurrent neural networks and as shown later on in experiments with toy data, training RNNs on deep trees can suffer from the *vanishing gradient problem* [Hochreiter et al., 2001], i.e., that error signals propagating backward to leaf nodes shrink quickly [1]. In addition, information sent from a leaf node to the root can be obscured if the path between them is long, thus leading to the problem of how to capture long range dependencies. I therefore adapt the LSTM architecture [Hochreiter and Schmidhuber, 1997] from recurrent neural network research to tackle those two problems. The main idea is to allow information low in a parse tree to be stored in a memory cell and used much later higher up in the parse tree, by recursively *adding* up all memory into a memory cell in a bottom-up manner. In this way, error signals propagating backward through structures do not vanish. And information from leaf nodes is still (loosely) preserved and can be used directly at any higher nodes in the hierarchy. I then demonstrate the contribution of the RLSTM on a sentiment analysis task. Empirical results show that my model outperforms the traditional RNN model in terms of accuracy.

The outline of the rest of the chapter is as follows. I introduce the traditional LSTM architecture in Section 6.2. In Section 6.3 I propose the RLSTM model for tree structures, and examine how the two problems mentioned above affect it and the traditional RNN model on an artificial task. Section 6.4 shows how I apply the RLSTM model to a sentiment analysis task, using the Stanford Sentiment Treebank.

6.2 The Long Short Term Memory Architecture

Recurrent neural networks (Figure 6.1a, Section 2.2), in theory, can be used to estimate conditional probabilities $P(.|x_0...x_t)$ with long histories. In practice, however, training recurrent neural networks with the gradient descent method is challenging because

[1] Error signals can also grow quickly when propagating back to leaf nodes. This problem is called *exploding* gradient. A trick often used to solve this problem and working quite well is to scale gradients into a sphere whenever their norms are larger than a specified threshold.

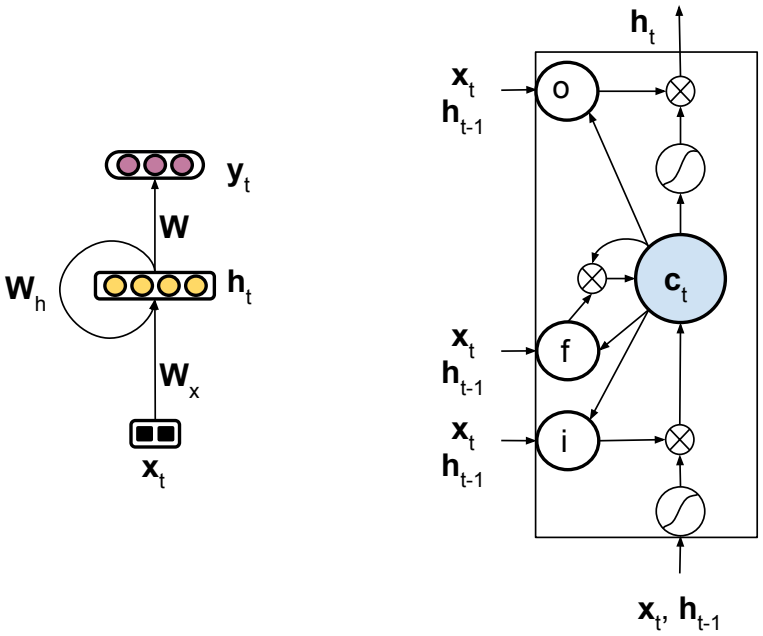


Figure 6.1: (a) Simple recurrent neural network and (b) Long short-term memory. Bias vectors are removed for simplicity.

gradients $\partial J_t / \partial \mathbf{h}_j$ ($j \leq t$, J_t is the objective function at time t) could vanish quickly after a few back-propagation steps [Hochreiter et al., 2001]. In addition, it is difficult to capture long range dependencies, i.e. the output at time t depends on inputs that came in long time ago. One solution, originally proposed by Hochreiter and Schmidhuber [1997] and enhanced by Gers [2001], is the *long short-term memory* (LSTM) architecture.

The main idea of the LSTM architecture is to maintain a memory of *all* inputs that the hidden layer receives over time, by *adding up* all inputs (weighted by a gating function as I will define later) coming in the hidden layer through time to a memory cell. In this way, error signals propagating backward through time do not vanish and even inputs received very long time ago are still (approximately) preserved and can play a role in computing the output of the network at the current time (see the illustration in [Graves, 2012, Chapter 4]).

An LSTM cell (see Figure 6.1b) consists of a memory cell c, an input gate i, a forget gate f, an output gate o. Crucially, the input to the memory cell is gated: the network can learn to use the input gate to decide when to remember coming information, and similarly learn to use the output gate to decide when to access the memory. The forget gate is to reset the memory cell. Computations occurring in this LSTM cell are given below:

$$
\begin{aligned}
\mathbf{i}_t &= \sigma\big(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i\big) \\
\mathbf{f}_t &= \sigma\big(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{W}_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f\big) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh\big(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c\big) \\
\mathbf{o}_t &= \sigma\big(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{W}_{co}\mathbf{c}_t + \mathbf{b}_o\big) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
$$

where σ is the sigmoid function; $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{o}_t$ are the outputs (i.e. activations) of the corresponding gates; $\mathbf{c}_t$ is the state of the memory cell; $\mathbf{h}_t$ is the output of the LSTM cell; $\odot$ denotes the element-wise multiplication operator; $\mathbf{W}$'s and $\mathbf{b}$'s are weight matrices and bias vectors. Note that because the sigmoid function has the range of $(0, 1)$, activations of those gates can be seen as normalized weights.

6.3 The Recursive Long Short-Term Memory Model

I propose the RLSTM model, a novel variant of the LSTM architecture for tree structures. For simplicity, I consider only binary trees, but extending this variant to n-ary trees is straightforward. I then evaluate the proposed RLSTM model and the traditional RNN model on an artificial task to examine how the two problems mentioned above (i.e., the vanishing gradient problem and the problem of how to capture long range dependencies) affect the two models.

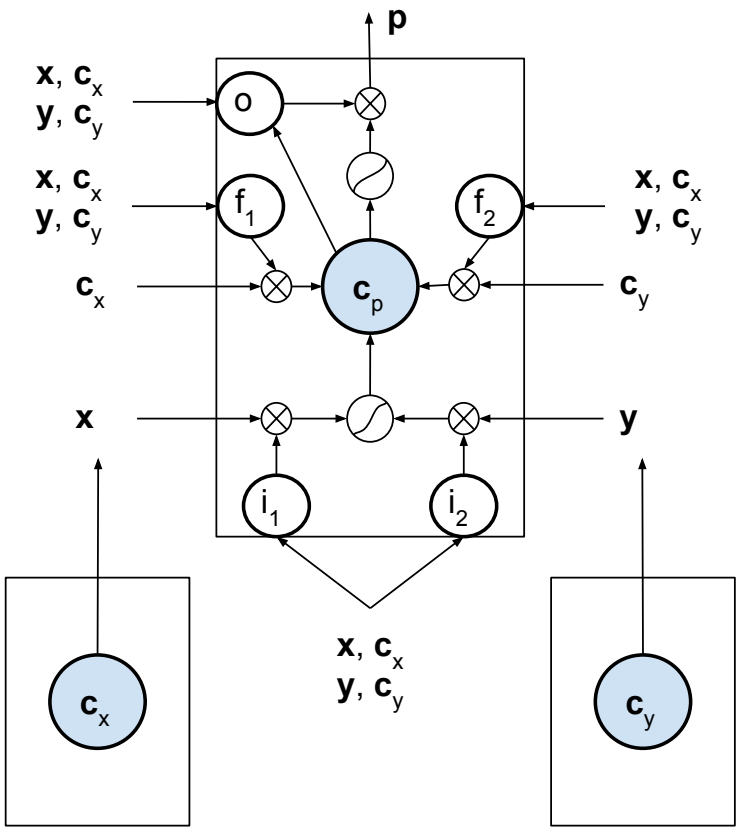


Figure 6.2: Recursive Long short-term memory for tree structures. At each node u in a parse tree, there are two vectors: $\mathbf{u}$ representing the constituent that the node covers, $\mathbf{c}_u$ storing a memory. The parent node p has two input gates i_1, i_2 and two forget gates f_1, f_2 for the two children x and y.

6.3.1 The New Architecture

This variant is inspired by the RNN model, where the key idea is to hierarchically combine vectors of two children to compute a vector representation for their parent: given a production $p \rightarrow x\ y$, the computation of the RNN model is given by

$$\mathbf{p} = f(\mathbf{W}_1\mathbf{x} + \mathbf{W}_2\mathbf{y} + \mathbf{b}) \tag{6.1}$$

where $\mathbf{x}, \mathbf{y}, \mathbf{p} \in \mathbb{R}^n$ are vectors representing x, y, p. $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{n \times n}$, $\mathbf{b} \in \mathbb{R}^n$ are weight matrices and a bias vector. f is an activation function (e.g., *tanh*).

I now, at each node u in a parse tree, use two vectors: $\mathbf{u}$ representing the constituent that the node covers, $\mathbf{c}_u$ storing a memory. In the RLSTM architecture (see Figure 6.2), the parent node p has two input gates i_1, i_2 and two forget gates f_1, f_2 for the two

children x and y. Computations occurring in this RLSTM are:

$$\begin{aligned}
\mathbf{i}_1 &= \sigma\big(\mathbf{W}_{i1}\mathbf{x} + \mathbf{W}_{i2}\mathbf{y} + \mathbf{W}_{ci1}\mathbf{c}_x + \mathbf{W}_{ci2}\mathbf{c}_y + \mathbf{b}_i\big) \\
\mathbf{i}_2 &= \sigma\big(\mathbf{W}_{i1}\mathbf{y} + \mathbf{W}_{i2}\mathbf{x} + \mathbf{W}_{ci1}\mathbf{c}_y + \mathbf{W}_{ci2}\mathbf{c}_x + \mathbf{b}_i\big) \\
\mathbf{f}_1 &= \sigma\big(\mathbf{W}_{f1}\mathbf{x} + \mathbf{W}_{f2}\mathbf{y} + \mathbf{W}_{cf1}\mathbf{c}_x + \mathbf{W}_{cf2}\mathbf{c}_y + \mathbf{b}_f\big) \\
\mathbf{f}_2 &= \sigma\big(\mathbf{W}_{f1}\mathbf{y} + \mathbf{W}_{f2}\mathbf{x} + \mathbf{W}_{cf1}\mathbf{c}_y + \mathbf{W}_{cf2}\mathbf{c}_x + \mathbf{b}_f\big) \\
\mathbf{c}_p &= \mathbf{f}_1 \odot \mathbf{c}_x + \mathbf{f}_2 \odot \mathbf{c}_y + g\big(\mathbf{W}_{c1}\mathbf{x} \odot \mathbf{i}_1 + \mathbf{W}_{c2}\mathbf{y} \odot \mathbf{i}_2 + \mathbf{b}_c\big) \\
\mathbf{o} &= \sigma\big(\mathbf{W}_{o1}\mathbf{x} + \mathbf{W}_{o2}\mathbf{y} + \mathbf{W}_{co}\mathbf{c} + \mathbf{b}_o\big) \\
\mathbf{p} &= \mathbf{o} \odot g(\mathbf{c}_p)
\end{aligned}$$

where $\mathbf{i}_1$, $\mathbf{i}_2$, $\mathbf{f}_1$, $\mathbf{f}_2$, $\mathbf{o}$ are the activations of the corresponding gates; $\mathbf{W}$'s and $\mathbf{b}$'s are weight matrices and bias vectors; and g is an activation function.

Intuitively, I put in p's memory (i.e., $\mathbf{c}_p$) its children's memories, and compute its representation (i.e., $\mathbf{p}$) as the result of an emission of the memory. The input gate i_j lets the RLSTM at the parent node decide how important the representation of the j-th child is. If it is important, the input gate i_j will have an activation close to $\mathbf{1}$, thus allowing the child's representation to have a large contribution to the parent's representation. Moreover, the RLSTM controls, using the forget gate f_j, the degree to which information from the memory of the j-th child should be added to the parent's memory. This way, the network has the option to store information when processing constituents low in the parse tree, and make it available later on when it is processing constituents high in the parse tree.

Using one input gate and one forget gate for each child makes the RLSTM flexible in storing memory and computing composition. For instance, in a complex sentence containing a main clause and a dependent clause it could be beneficial if only information about the main clause is passed on to higher levels. This can be achieved by having low values for the input gate and the forget gate for the child node that covers the dependent clause, and high values for the gates corresponding to the child node covering (a part of) the main clause.

Compared to similar models

Independently from and concurrently with this work, Tai et al. [2015] and Zhu et al. [2015b] have developed very similar models. The main difference between my proposed RLSTM and the two other models is that, for a binary tree, my RLSTM has two input gates, corresponding to two children, whereas the other two models have only one input gate. In theory, using one input gate for each child makes the RLSTM more expressive. For instance, this RLSTM can even allow a child to contribute to composition by activating the corresponding input gate, but ignore the child's memory by deactivating the corresponding forget gate. This happens when the information given by the child is of local importance only.

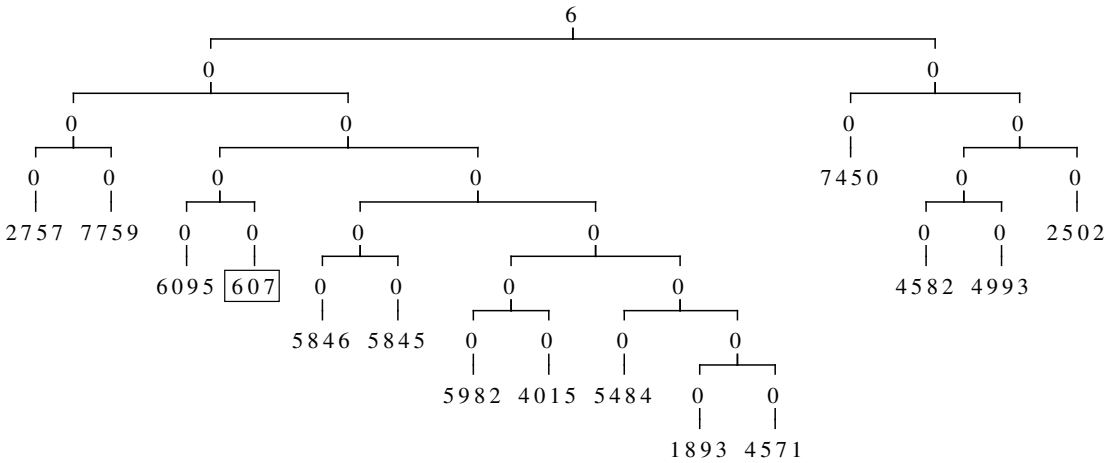


Figure 6.3: Example binary tree for the artificial task. The number enclosed in the box is the *keyword* of the sentence. The label at the root node is the label of the whole sentence. Notice that all internal nodes other than the root are labeled 0, which are not using in the experiments.

6.3.2 Experiments

I now examine how the two problems, the vanishing gradient problem and the problem of how to capture long range dependencies, affect the proposed RLSTM model and the traditional RNN model. To do so, I propose the following artificial task, which requires a model to distinguish useful signals from noise:

- a sentence is a sequence of tokens which are integer numbers in the range $[0, 10000]$,
- a sentence contains one and only one *keyword* token which is an integer number smaller than 1000,
- a sentence is labeled with the integer resulting from dividing the keyword by 100. For instance, if the keyword is 607, the label is 6. In this way, there are 10 classes, ranging from 0 to 9.

The task is to predict the class of a sentence, given its binary parse tree (Figure 6.3). Because the label of a sentence is determined solely by the keyword, the two models need to identify the keyword in the parse tree and allow only the information from the leaf node of the keyword to affect the root node.

The two models, RLSTM and RNN, were implemented with the dimension of vector representations and vector memories $n = 50$. Following Socher et al. [2013b], I used $f = g = tanh$ and initialized word vectors by randomly sampling each value from a uniform distribution $U(-0.0001, 0.0001)$. I trained the two models using the AdaGrad method [Duchi et al., 2011] with a learning rate of 0.05 and a mini-batch size of 20 for the RNN and of 5 for the RLSTM. Development sets were employed for early stopping (training is halted when the accuracy on the development set is not improved after 5 consecutive epochs).

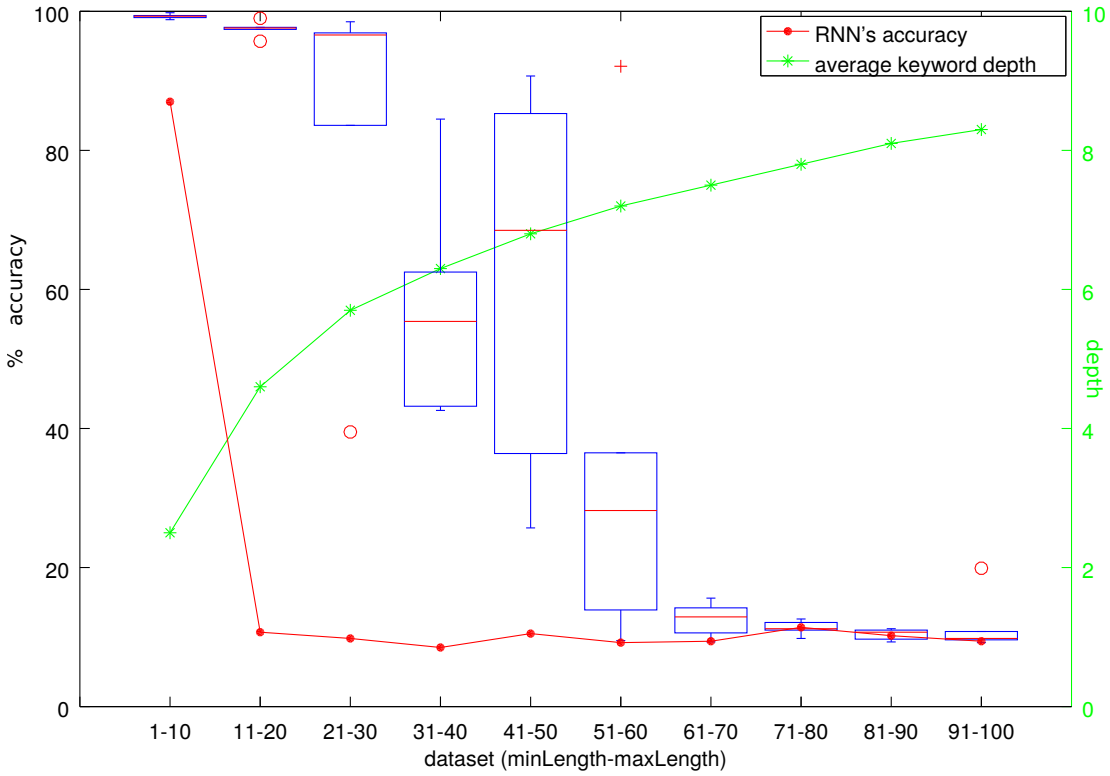


Figure 6.4: Test accuracies of the RNN model (the best among 5 runs) and the RLSTM model (boxplots) on the 10 datasets of different sentence lengths.

Experiment 1

I randomly generated 10 datasets. To generate a sentence of length l, I shuffle a list of randomly chosen $l-1$ non-keywords and one keyword. The i-th dataset contains 12k sentences of lengths from $10i-9$ tokens to $10i$ tokens, and is split into train, dev, test sets with sizes of 10k, 1k, 1k sentences. I parsed each sentence by randomly generating a binary tree whose number of leaf nodes equals to the sentence length.

The test accuracies of the two models on the 10 datasets are shown in Figure 6.4; For each dataset I run each model 5 times and reported the highest accuracy for the RNN model, and the distribution of accuracies (via boxplot) for the RLSTM model. We can see that the RNN model performs reasonably well on very short sentences (less than 11 tokens). However, when the sentence length exceeds 10, the RNN's performance drops so quickly that the difference between it and the random guess' performance (10%) is negligible. Trying different learning rates, mini-batch sizes, and values for n (the dimension of vectors) did not give significant differences. On the other hand, the RLSTM model achieves more than 90% accuracy on sentences shorter than 31 tokens. Its performance drops when the sentence length increases, but is still substantially better than the random guess when the sentence length does not exceed 70. When the sentence length exceeds 70, both the RLSTM and RNN perform

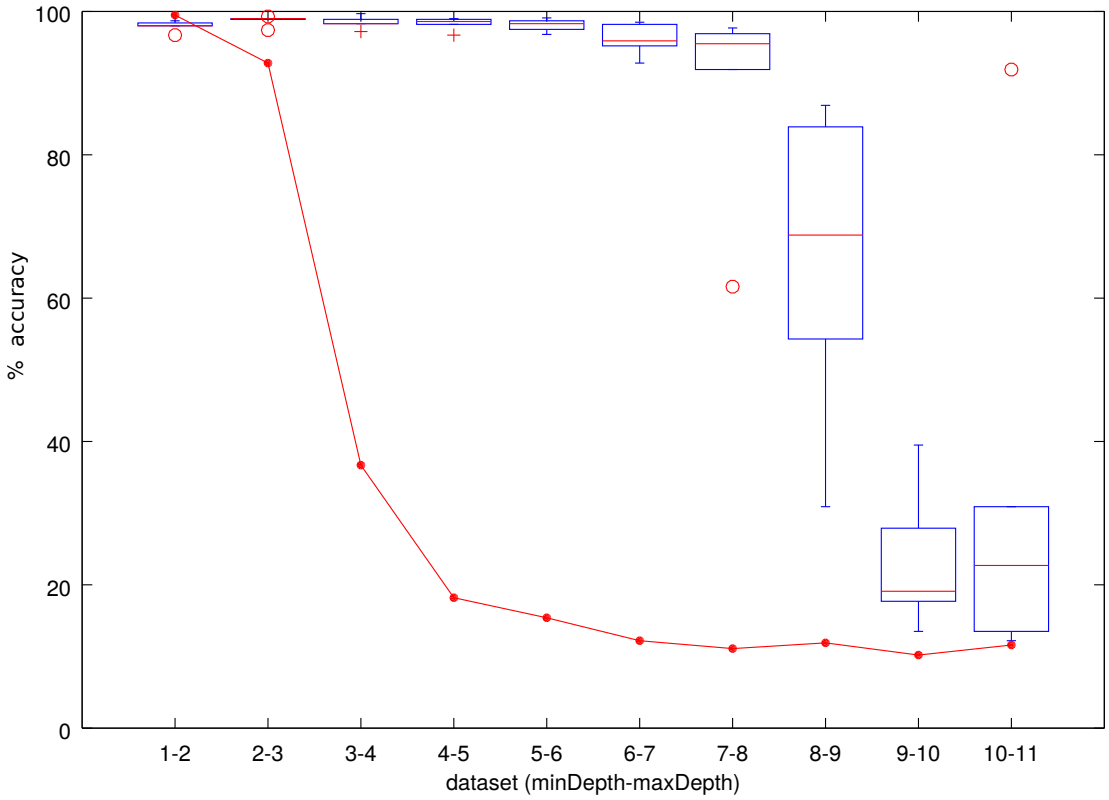


Figure 6.5: Test accuracies of the RNN model (the best among 5 runs) and the RLSTM model (boxplots) on the 10 datasets of different keyword depths.

similarly.

Experiment 2

In Experiment 1, it is not clear whether the tree size or the keyword depth is the main factor of the rapid drop of the RNN's performance. In this experiment, I kept the tree size fixed and vary the keyword depth. I generated a pool of sentences of lengths from 21 to 30 tokens and parsed them by randomly generating binary trees. I then created 10 datasets each of which has 12k trees (10k for training, 1k for development, and 1k for testing). The i-th dataset consists of only trees in which distances from keywords to roots are i or $i + 1$ (to stop the networks from exploiting keyword depths directly).

Figure 6.5 shows test accuracies of the two models on those 10 datasets. Similarly in Experiment 1, for each dataset I run each model 5 times and reported the highest accuracy for the RNN model, and the distribution of accuracies for the RLSTM model. As we can see, the RNN model achieves very high accuracies when the keyword depth does not exceed 3. Its performance then drops rapidly and gets close to the performance of the random guess. This is evidence that the RNN model has difficulty capturing long range dependencies. By contrast, the RLSTM model performs at above 90% accuracy

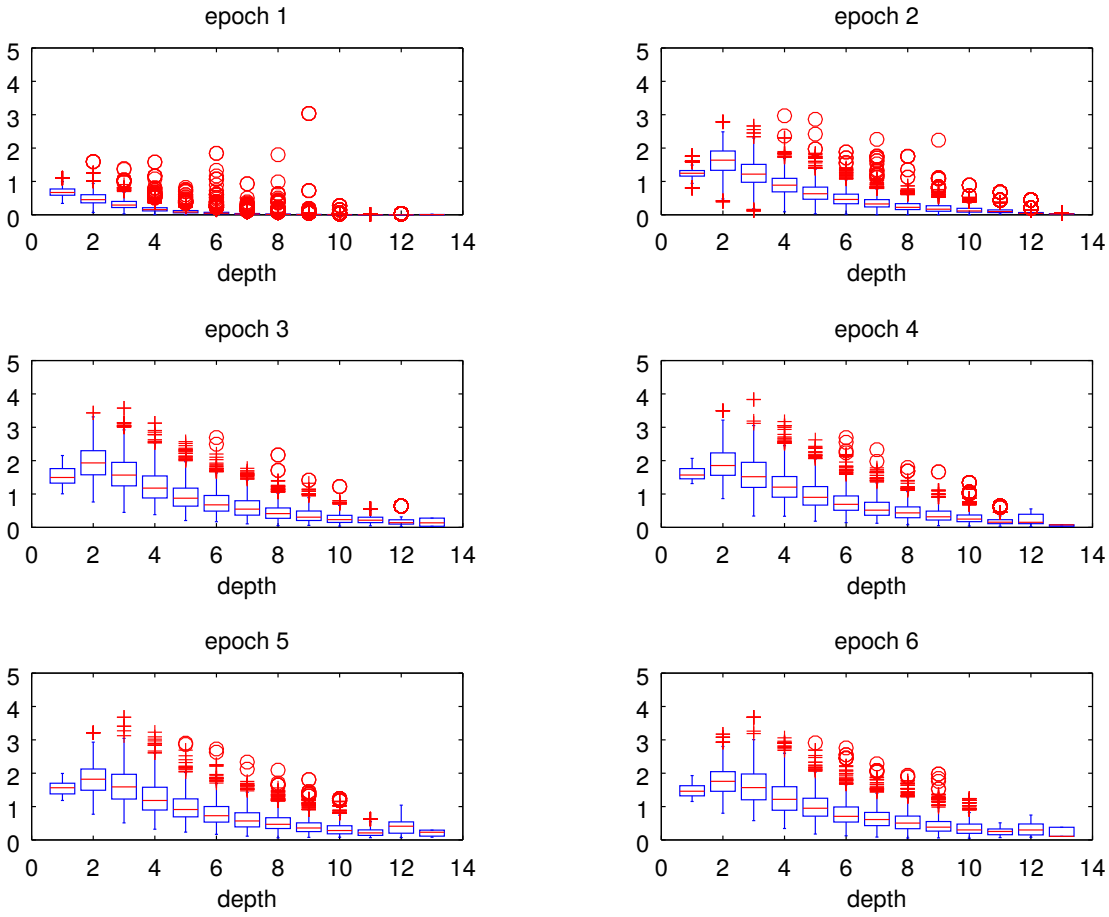


Figure 6.6: Ratios of norms of error vectors at keyword nodes to norms of error vectors at root nodes w.r.t. the keyword node depth in each epoch of training the RNN model.

until the depth of the keyword reaches 8. It has difficulty dealing with larger depths, but the performance is always better than the random guess.

Experiment 3

I now examine whether the two models can encounter the vanishing gradient problem. To do so, I looked at the the back-propagation phase of each model in Experiment 1 on the third dataset (the one containing sentences of lengths from 21 to 30 tokens). For each tree, I calculated the ratio

$$\frac{\|\frac{\partial J}{\partial \mathbf{x}_{keyword}}\|}{\|\frac{\partial J}{\partial \mathbf{x}_{root}}\|}$$

where the numerator is the norm of the error vector at the keyword node and the denominator is the norm of the error vector at the root node. This ratio gives us an intuition how the error signals develop when propagating backward to leaf nodes: if

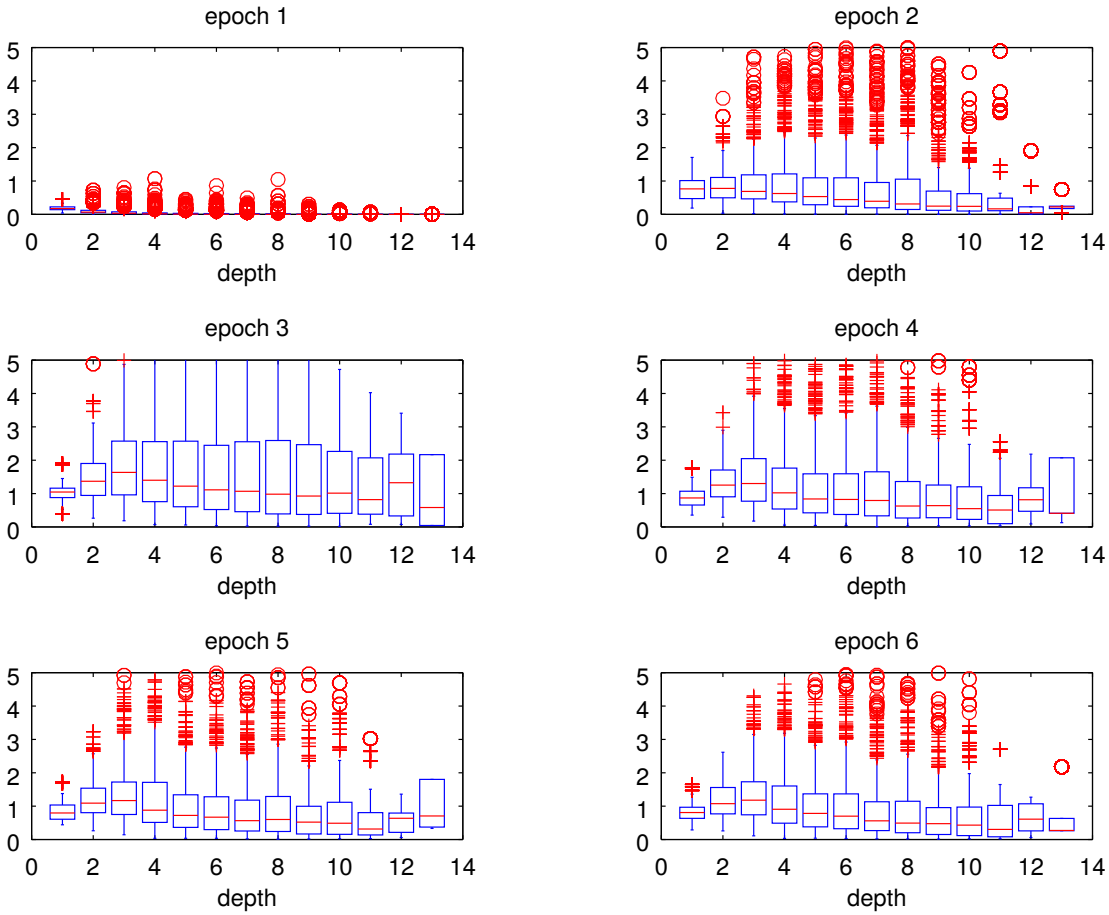
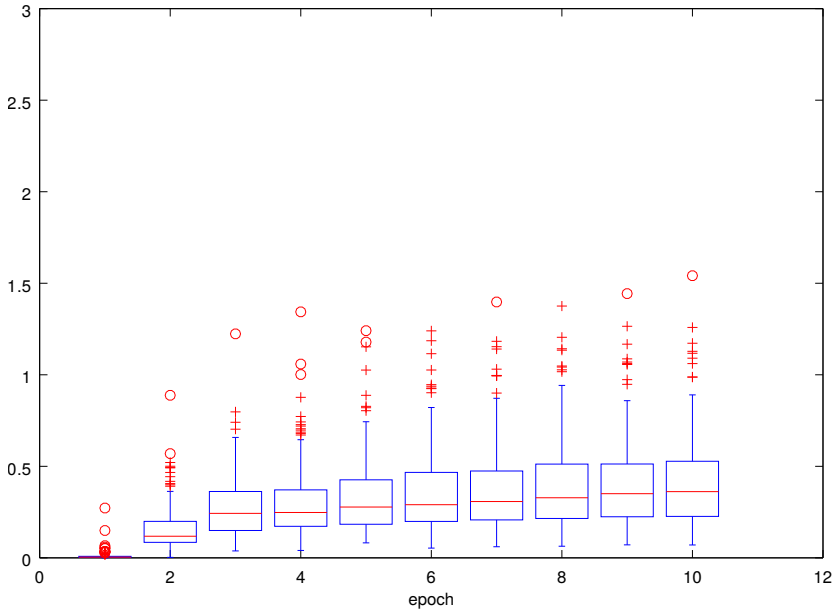
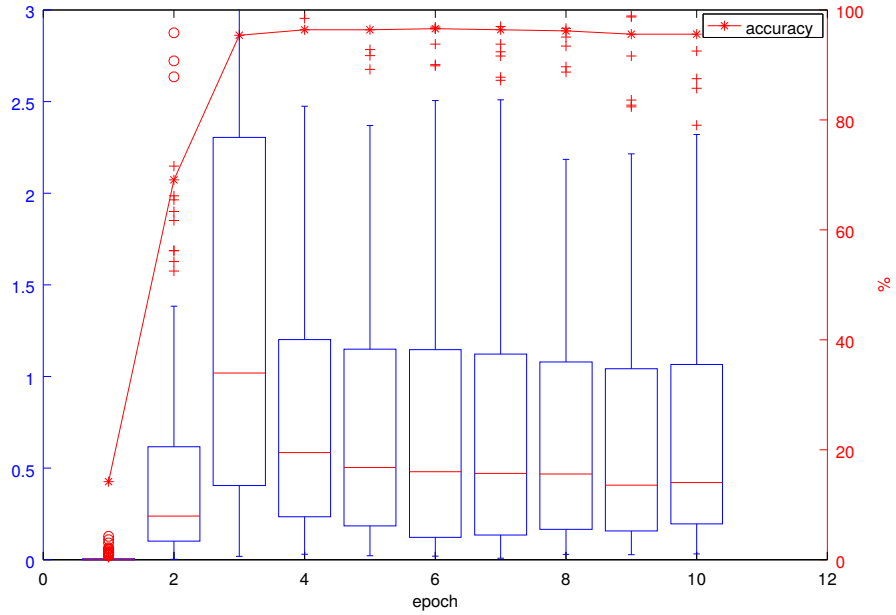


Figure 6.7: Ratios of norms of error vectors at keyword nodes to norms of error vectors at root nodes w.r.t. the keyword node depth in each epoch of training the RLSTM model. (Notice that because the y-axis is cut off at 5, some outliers are not shown.)



(a) RNN



(b) RLSTM (with development accuracies)

Figure 6.8: Ratios at depth 10 in each epoch of training (a) the RNN model and (b) the RLSTM model.

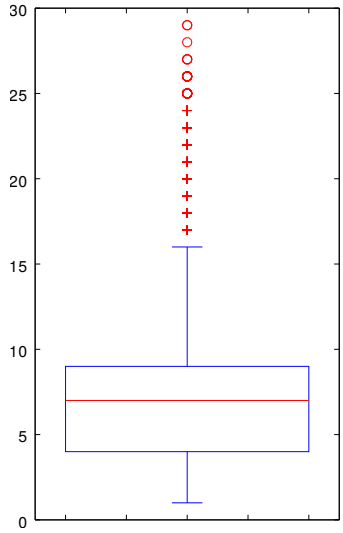


Figure 6.9: Boxplot of leaf node depths in the training set of the of the Stanford Sentiment Treebank. More than three fourth of leaf nodes are at depths less than 10.

the ratio $\ll 1$, the vanishing gradient problem occurs; else if the ratio $\gg 1$, we observe the exploding gradient problem.

Figure 6.6 reports the ratios w.r.t. the keyword node depth in each epoch of training the RNN model. The ratios in the first epoch are always very small, since word vectors were initialized by very small values ($\sim U(-0.0001, 0.0001)$). In each following epoch, the RNN model successfully lifts up the ratios steadily (see Figure 6.8a for a clear picture at the keyword depth 10), but a clear decrease when the depth becomes larger is observable. For the RLSTM model (see Figure 6.7 and 6.8b), the story is somewhat different. The ratios go up after two epochs so rapidly that there are even some exploding error signals sent back to leaf nodes. They are then go down and remain stable with substantially less exploding error signals. This is, interestingly, concurrent with the performance of the RLSTM model on the development set (see Figure 6.8b). It seems that the RLSTM model, after one epoch, can quickly locate the keyword node in a tree and relate it to the root by building a strong bond between them via error signals. After the correlation between the keyword and the label at the root is found, it turns to try to stabilize the training by reducing error signals sent back to the keyword node. Comparing the two models by aligning Figure 6.6 with Figure 6.7, and Figure 6.8a with Figure 6.8b, we can see that the RLSTM model is more capable of transmitting error signals to leaf nodes.

It is worth noting that we do see the vanishing gradient problem happening when training the RNN model in Figure 6.6; but Figure 6.8a suggests that the problem becomes less serious after a few epochs. This might be because the depth 10 is still manageable to the RNN model. The question now is "do we need to handle depths larger than 10?". To answer this, let us look at Figure 6.9, showing the statistics of leaf node depths in the training set of the Stanford Sentiment Treebank, which is used

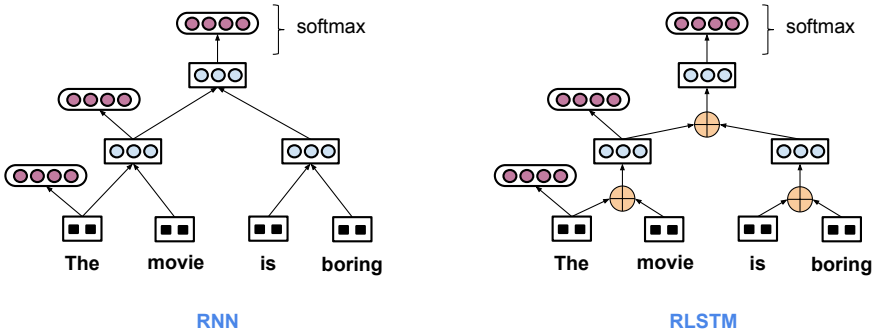


Figure 6.10: The RNN model (left) and the RLSTM model (right) for sentiment analysis.

later in this chapter. (Noting that the average length of those sentences is 19.1, which is quite large.) We can see that more than three quarters of leaf nodes are at depths less than 10. It suggests that handling larger depths might be not so crucial for datasets of average sentence length less than 20.

6.4 The RLSTM model for Sentiment Analysis

6.4.1 The Model

I now apply the proposed RLSTM model to a sentiment analysis task (see Figure 6.10). On the top of each node, if its sentiment class (e.g., positive, negative, or neutral) is given, I put a softmax layer to compute the probability of assigning a class to it. The vector representations of words (i.e. word embeddings) can be initialized randomly, or pre-trained. The memory of any leaf node w, i.e. $\mathbf{c}_w$, is always $\vec{0}$.

Similarly to Irsoy and Cardie [2014], I 'untie' leaf nodes and internal nodes: I use one weight matrix set for leaf nodes and another set for internal nodes. Hence, if word representations are vectors of dimension d_w and phrase representations are vectors of dimension d, all weight matrices from a leaf node to an immediate internal node have a size of $d \times d_w$, and all weight matrices from an internal node to another internal node have a size of $d \times d$.

Training

Training this model is to minimize the following objective function, which is the cross-entropy over a training set $\mathcal{D}$ plus an L2-norm regularization term:

$$J(\theta) = -\frac{1}{|\mathcal{D}|}\sum_{s\in\mathcal{D}}\sum_{p\in s}\log Pr(l_p|\mathbf{p}) + \frac{\lambda}{2}||\theta||^2$$

where θ is the parameter set, l_p is the sentiment class of phrase p, $\mathbf{p}$ is the vector representation at the node covering p, and λ is a regularization parameter. Like training

an RNN, I use the mini-batch gradient descent method to minimize J w.r.t. θ, where gradients $\partial J/\partial\theta$ are computed efficiently using the back-propagation through structure algorithm [Goller and Küchler, 1996]. I use the AdaGrad method [Duchi et al., 2011] to automatically update the learning rate for each parameter.

Complexity

I analyse the complexities of the RNN and RLSTM models in the forward phase where vector representations for internal nodes and classification probabilities are computed. The complexities in the backward phase where gradients $\partial J/\partial\theta$ are computed can be analysed similarly. It is easy to see that the complexities of the two models are dominated by the matrix-vector multiplications that are carried out. Since the number of sentiment classes is very small (5 or 2 in the later experiments) compared to d and d_w, I only consider those matrix-vector multiplications which are for computing vector representations at internal nodes.

For a sentence of length N, let us assume that its parse tree is binarized without any unary branch (as in the data set I used in the later experiments). In the RNN model, because each node (other than the root) requires one matrix multiplication for computing its parent's representation, the complexity in the forward phase is approximately:

$$\underbrace{N \times d \times d_w}_{\text{for } N \text{ leaf nodes}} + \underbrace{(N-2) \times d \times d}_{\text{for } N-2 \text{ internal nodes other than the root}}$$

In the RLSTM model, because each leaf node requires 6 matrix multiplications for computing its parent's gates, each internal node (other than the root) requires 11 matrix multiplications for computing its parent's gates and its own output gate, and the root requires one matrix multiplication for computing its own output gate, the complexity is approximately:

$$\underbrace{N \times 6 \times d \times d_w}_{\text{for } N \text{ leaf nodes}} + \underbrace{(N-2) \times 11 \times d \times d}_{\text{for } N-2 \text{ internal nodes other than the root}} + \underbrace{d \times d}_{\text{for the root node}}$$

If $d_w \approx d$, the complexity of the RLSTM model is about 8.5 times higher than the complexity of the RNN model.

In my experiments, this difference is not a serious problem because training and evaluating the RLSTM model is very fast: it took me, on a single core of a modern computer, about 10 minutes to train the model ($d = 50, d_w = 100$) entirely on 8544 sentences, and about 2 seconds to evaluate it on 2210 sentences.

6.4.2 Experiments

Experiments in this section use the Stanford Sentiment Treebank[2] [Socher et al., 2013b] which consists of 5-way fine-grained sentiment labels (very negative, negative, neutral,

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

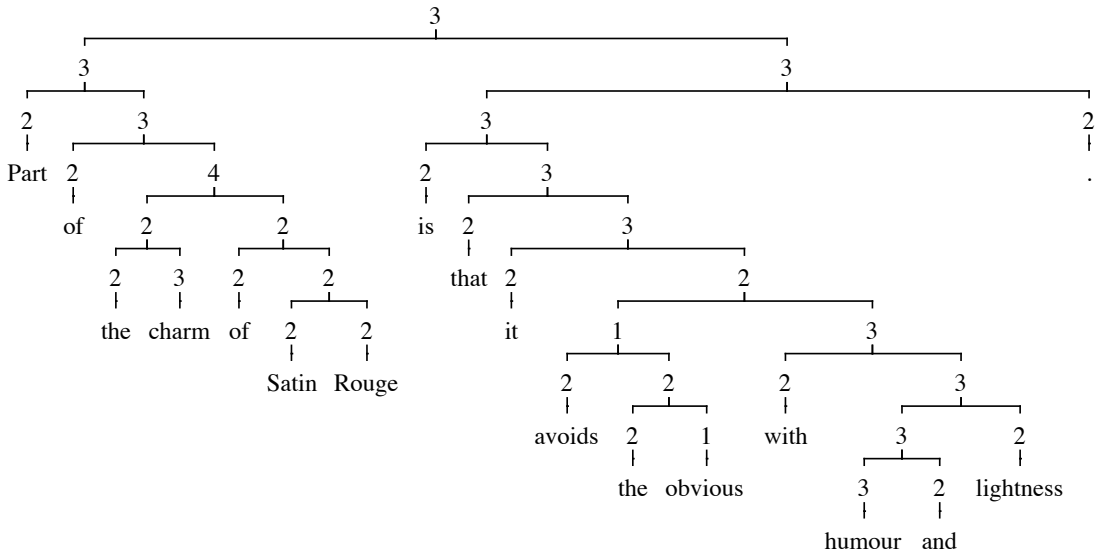


Figure 6.11: Example tree with a sentiment label at every node. (0, 1, 2, 3, 4 mean very negative, negative, neutral, positive, very positive.)

positive, very positive) for 215,154 phrases of 11,855 sentences (see an example tree in Figure 6.11). The standard splitting is also given: 8544 sentences for training, 1101 for development, and 2210 for testing. The average sentence length is 19.1. In addition, the treebank also supports binary sentiment (positive, negative) classification by removing neutral labels, leading to: 6920 sentences for training, 872 for development, and 1821 for testing. The evaluation metric is the accuracy, given by $\frac{100 \times \#correct}{\#total}$.

RLSTM vs. RNN

Setting I initialized the word vectors in both models using the 100-dim GloVe[3] word embeddings [Pennington et al., 2014], which were trained on a 6B-word corpus. The initial values for theirs weight matrices were uniformly sampled from the symmetric interval $\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$ where n is the number of total input units.

For each model (RNN and RLSTM), I employed three activation functions: *softmax*, *tanh*, and *softsign*, leading to six sub-models. Tuning those sub-models on the development set, I chose the dimension of internal nodes' vectors $d = 50$, a learning rate of 0.05, regularization parameter $\lambda = 10^{-3}$, and a mini-batch-size of 5. On each task, I run each sub-model 10 times. Each time, I trained the sub-model in 20 epochs and select the network achieving the highest accuracy on the development set.

Results Figure 6.12 and 6.13 show the statistics of the test accuracies of the resulting networks on the test set on the fine-grained classification task and binary classification task, respectively. It can be seen that the RLSTM model outperforms the RNN model

[3] `http://nlp.stanford.edu/projects/GloVe/`

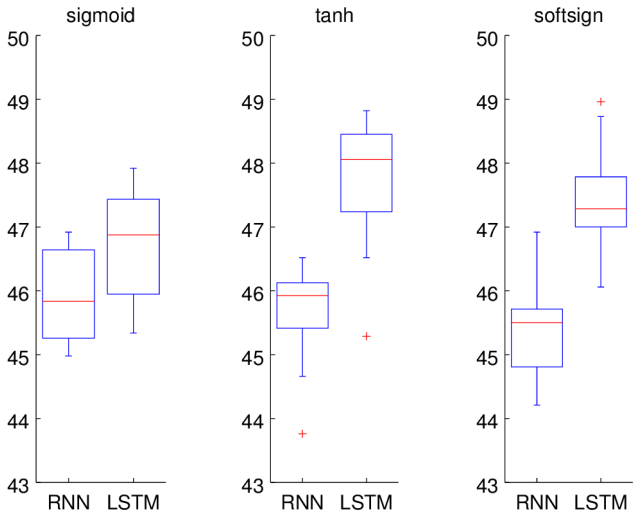


Figure 6.12: Boxplots of test accuracies of 10 runs of the RNN model and the RLSTM model on the fine-grained classification task.

when using *tanh* and *softsign*. With the *sigmoid* activation function, the difference is not as clear, but the RLSTM model performs slightly better. The *tanh*-RLSTM and the *softsign*-RLSTM achieve highest median accuracies (48.1 and 86.4) on the fine-grained classification task and on the binary classification task, respectively.

For the RNN model, the *sigmoid* function performs on par with the other two functions on the fine-grained task, and even better than the *softsign* function on the binary task. This is surprising to see, given that *sigmoid* is not often chosen in recent work. The *softsign* function, which is shown to work better than *tanh* for deep networks [Glorot and Bengio, 2010], however, does not yield improvements in this experiment.

For the RLSTM model, the *tanh* function, in general, works best whereas the *sigmoid* function is the worst. This result agrees with the common choice for activation function for the LSTM architecture in recurrent network research [Gers, 2001, Sutskever et al., 2014].

Compared against other Models

I compare the RLSTM model (using *tanh*) in the previous experiment against existing models: Naive Bayes with bag of bigram features (BiNB), Recursive neural tensor network (RNTN) [Socher et al., 2013b], Convolutional neural network (CNN) [Kim, 2014], Dynamic convolutional neural network (DCNN) [Kalchbrenner et al., 2014], paragraph vectors (PV) [Le and Mikolov, 2014], and Deep RNN (DRNN) [Irsoy and Cardie, 2014].

Among them, the BiNB is the only one that does not employ neural networks. The RNTN and the DRNN are two extensions of the traditional RNN model. Whereas the

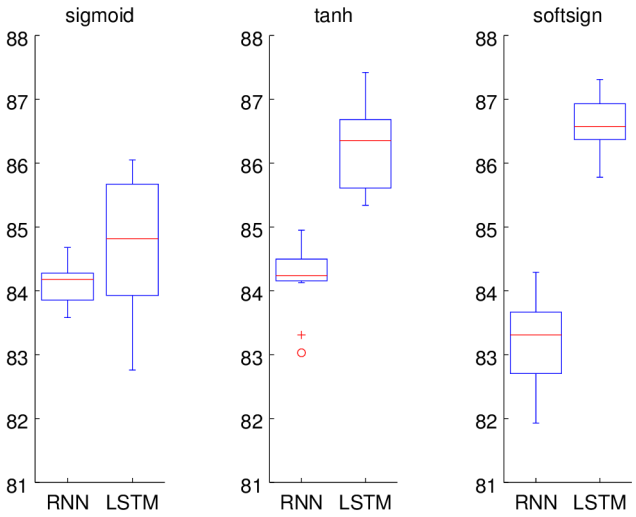


Figure 6.13: Boxplot of test accuracies of 10 runs of the RNN model and the RLSTM model on the binary classification task.

RNTN uses both matrix-vector multiplications and tensor products for the composition purpose, the DRNN has a deeper net structure which results from concatenating two or more RNNs horizontally. The CNN, DCNN and PV do not rely on syntactic trees. The CNN uses a convolutional layer and a max-pooling layer to handle sequences of different lengths. The DCNN is hierarchical in the sense that it stacks two or more convolutional layers with k-max pooling layers in between. In the PV, a sentence (or document) is represented as an input vector to predict which words appear in it.

Table 6.1 (above the dashed line) shows the test accuracies of those models. The accuracy of the RLSTM model is taken from the network achieving the highest perfor-

Model	Fine-grained	Binary
BiNB	41.9	83.1
RNTN	45.7	85.4
CNN	48.0	88.1
DCNN	48.5	86.8
PV	48.7	87.8
DRNN	49.8	86.6
with GloVe-100D		
RLSTM	48.0	86.2
with GloVe-300D		
RLSTM	**49.9**	**88.0**

Table 6.1: Test accuracy of the (tanh) RLSTM compared with other models.

mance out of 10 runs on the development set. The accuracies of the other models are copied from the corresponding papers. The RLSTM clearly performs worse than the DCNN, PV, DRNN on both tasks, and worse than the CNN on the binary task.

Approaching the state-of-the-art with better word embeddings

I now focus on the DRNN model, which is the most similar to the RLSTM model among those four models CNN, DCNN, PV and DRNN. In fact, from the results reported in [Irsoy and Cardie, 2014, Table 1a], the RLSTM performs on par[4] with their one-layer-DRNN ($d = 340$) using dropout (randomly removing some neurons during training). Dropout is a powerful technique to train neural networks, not only because it plays a role as a strong regularization method to prohibit neurons from co-adapting, but it is also considered as a technique to efficiently build an ensemble of a large number of shared weight neural networks [Srivastava et al., 2014]. Thanks to dropout, Irsoy and Cardie [2014] boost the accuracy of a 3-layer-DRNN ($d = 200$) from 46.06 to 49.5 in the fine-grained task.

In the next experiment, I tried to boost the accuracy of the RLSTM model. Inspired by Irsoy and Cardie [2014], I employed dropout and better word embeddings. Dropout, however, does not work with the RLSTM. The reason might be that dropout corrupts the RLSTM's memory, thus making training more difficult. Better word embeddings do pay off, on the other hand. I used the 300-dim GloVe word embeddings trained on a 840B-word corpus. Based on the development set, I chose the same values for the hyper-parameters as in the first experiment, except setting the learning rate 0.01. I also run the model 10 times and selected the network achieving the highest accuracy on the development set. Table 6.1 (below the dashed line) shows the results. Using the 300-dim GloVe word embeddings is very helpful: the RLSTM model now performs on par with the DRNN model on the fine-grained task, and with the CNN model on the binary task. Therefore, taking into account both tasks, the RLSTM model initialized with the 300-dim GloVe word embeddings outperforms all other models. [5]

[4] Irsoy and Cardie [2014] used the 300-dim word2vec word embeddings trained on a 100B-word corpus whereas I used the 100-dim GloVe word embeddings trained on a 6B-word corpus. From the fact that they achieve an accuracy of 46.1 with the RNN ($d = 50$) on the fine-grained task and 85.3 on the binary task, and my implementation of the RNN ($d = 50$) performs worse, I conclude that the 100-dim GloVe word embeddings are not more suitable than the 300-dim word2vec word embeddings.

[5] Tai et al. [2015]'s Tree-LSTM model achieves 51% and 88.0% in accuracy on the fine-grained task and on the binary task, respectively. Zhu et al. [2015b]'s S-LSTM model achieves 50.1% on the fine-grained task. Like my RLSTM in this experiment, both models were initialized with the 300-dim GloVe word embeddings. Finding why the Tree-LSTM outperforms the RLSTM on the fine-grained task is left for future work.

6.5 Conclusions

In this chapter I propose the Recursive Long short term memory (RLSTM) model, a new composition method by extending the LSTM architecture which is widely used in recurrent neural network research. The proposed RLSTM model has been shown to capture long range dependencies and overcome the vanishing gradient problem more effectively than the traditional RNN model. In addition, empirical results on the Stanford Sentiment Treebank benchmark show that the RLSTM model outperforms the RNN model in terms of accuracy.

I now try to answer the question: why does the RLSTM model perform better than the RNN model? Here, I borrow the argument given in [Bengio et al., 2013a, Section 3.2] to answer the question, based on the hypothesis that the RLSTM for tree structures should work very similarly to the traditional LSTM architecture. Bengio explains that LSTMs behave like *low-pass filters* "hence they can be used to focus certain units on different frequency regions of the data". This suggests that an RLSTM plays a role as a lossy compressor which, by focusing on low frequency regions, can capture global information. So composition in this case could be seen as compression, like the Recursive auto-encoder (RAE) [Socher et al., 2011a]. Because pre-training an RNN by an RAE can boost the overall performance [Socher et al., 2011a,c], considering an RLSTM as a compressor might explain why it works better than an RNN without pre-training.

Comparing the RLSTM model against the DRNN model [Irsoy and Cardie, 2014] might give a hint about how to improve the proposed model. From the empirical results, the RLSTM without the 300-dim GloVe word embeddings performs worse than the DRNN, which gains a significant improvement thanks to dropout. Finding a method like dropout that does not corrupt the RLSTM's memory, therefore, might boost the overall performance significantly.

Chapter 7

The Forest Convolutional Network Model

According to the principle of compositionality, the meaning of a sentence is computed from the meanings of its parts and the way they are *syntactically* combined. In practice, however, syntactic structures are generated by automatic parsers which are far-from-perfect and not tuned to the specifics of the task. Current recursive neural network (RNN) approaches for computing sentence meaning therefore run into a number of practical difficulties, including the need to carefully select a parser appropriate for the task, deciding how and to what extent syntactic context modifies the semantic composition function, as well as on how to transform parse trees to conform to the branching settings (typically, binary branching) of the RNN. This chapter introduces a new model, the Forest Convolutional Network, that avoids all of these challenges, by taking as input a parse forest, rather than a single tree, and by allowing arbitrary branching factors. I report improvements over the state-of-the-art in sentiment analysis and question classification.

7.1 Introduction

In the previous chapters I discuss the RNN models and their successes in tackling the challenge of transforming sentences into vectors. In this chapter, I present a new recursive neural network architecture that fits squarely in this tradition, but aims to solve a number of difficulties that have arisen in existing work. In particular, the model I propose addresses three issues:

1. how to make the composition functions adaptive, in the sense that they operate adequately for the many different types of combinations (e.g., adjective-noun combinations are of a very different type than VP-PP combinations);

2. how to deal with different branching factors of nodes in the relevant syntactic trees (i.e., I want to avoid having to binarize syntactic trees,[1] but also do not want ternary productions to be completely independent from binary productions);

[1] [Eisner, 2001, Chapter 2] shows that using flat rules is linguistically beneficial, because "most

3. how to deal with uncertainty about the correct parse inside the neural architecture (i.e., I do not want to work with just the best or k-best parses for a sentence according to an external model, but receive an entire distribution over possible parses).

To solve these challenges I take inspiration from two other traditions: the convolutional neural networks and classic parsing algorithms based on dynamic programming. Including convolution in my network model provides a direct solution to issue (2), and turns out, somewhat unexpectedly, to also provide a solution to issue (1). Introducing the chart representation from classic parsing into the proposed architecture then allows me to tackle issue (3).

The outline of this chapter is as follows. In Section 7.2 I propose the novel Forest Convolutional Network model. I then show empirical results on a sentiment analysis task and a question classification task in Section 7.3. Section 7.4 discusses related work and Section 7.5 gives conclusions.

7.2 The Forest Convolutional Network Model

In this section, I first propose a solution to issues (1) and (2) (i.e., building adaptive composition functions and dealing with different branching factors) using convolutional networks. I then introduce a solution to the third issue (i.e., dealing with uncertainty about the correct parse), called the Chart Neural Network (ChNN). A combination of them, the Forest Convolutional Network (FCN), is presented lastly.

7.2.1 Composition with Convolutional networks

Given a production $p \rightarrow x_1 \ ... \ x_l$, my idea (Figure 7.1), like the CNN model introduced in Section 2.2.2, is to slide a window-size-k kernel along the sequence of children $(x_1, ..., x_l)$ to compute a pool of vectors. A max pooling operation followed by a fully connected layer is then applied to this pool to compute a vector for the parent p.

However, what I propose here differs from that CNN model at two points. First, I use a non-linear kernel: after linearly transforming input vectors, an activation function is applied. Second, I put $k-1$ padding tokens <b> at the beginning of the children sequence and $k-1$ padding tokens <e> at the end. This thus guarantees that all the children contribute equally to the resulting vector pool, which now has $l+k-1$ vectors.

It is obvious that employing convolutional layers for composition can solve the second issue (i.e., dealing with different branching factors), I now show how it can make the composition functions adaptive. We first see what happens if the window

crucially, a flat lexical entry corresponds to the local domain of a headword – the word together with all its semantic arguments and modifiers". From the computational perspective, flat rules make trees less deep, thus reducing the effect of the vanishing gradient problem and the problem of how to capture long range dependencies.

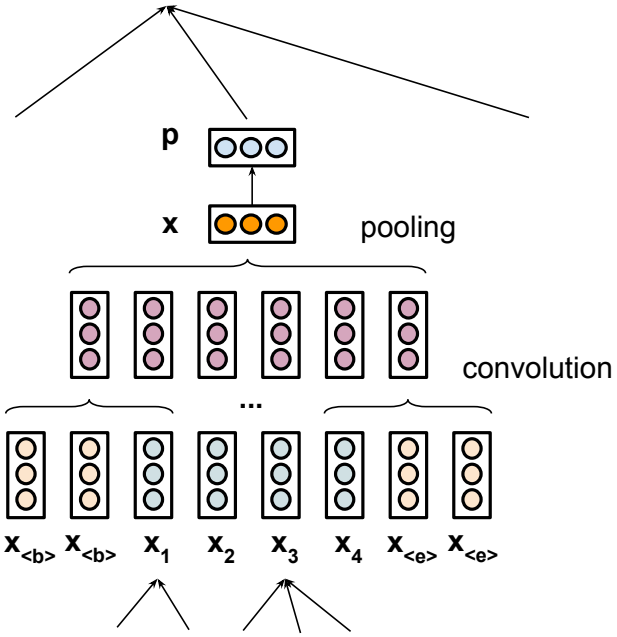


Figure 7.1: Composition with a convolutional network using a nonlinear window-size-3 kernel.

size k is larger than the number of children l, for instance $k = 3$ and $l = 2$. There are four vectors in the pool:

$$\mathbf{u}^{(1)} = f(\mathbf{W}_1\mathbf{x}_{<b>} + \mathbf{W}_2\mathbf{x}_{<b>} + \mathbf{W}_3\mathbf{x}_1 + \mathbf{b}_c)$$
$$\mathbf{u}^{(2)} = f(\mathbf{W}_1\mathbf{x}_{<b>} + \mathbf{W}_2\mathbf{x}_1 + \mathbf{W}_3\mathbf{x}_2 + \mathbf{b}_c)$$
$$\mathbf{u}^{(3)} = f(\mathbf{W}_1\mathbf{x}_1 + \mathbf{W}_2\mathbf{x}_2 + \mathbf{W}_3\mathbf{x}_{<e>} + \mathbf{b}_c)$$
$$\mathbf{u}^{(4)} = f(\mathbf{W}_1\mathbf{x}_2 + \mathbf{W}_2\mathbf{x}_{<e>} + \mathbf{W}_3\mathbf{x}_{<e>} + \mathbf{b}_c)$$

where $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3$ are weight matrices, $\mathbf{b}_c$ is a bias vector, f is an activation function. These four resulting vectors correspond to four ways of composing the two children's vectors:

(1) the first child stands alone (e.g., when the information of the second child is not important, it is better to ignore it),

(2,3) the two children are composed with two different weight matrix sets,

(4) the second child stands alone.

Because the max pooling then chooses features from these four vectors based on their values (i.e., on a row the feature with the highest value will be chosen to pass to the parent), it is reasonable to say that the behaviours of this composition function change according to input vectors. Therefore, this composition function is adaptive.

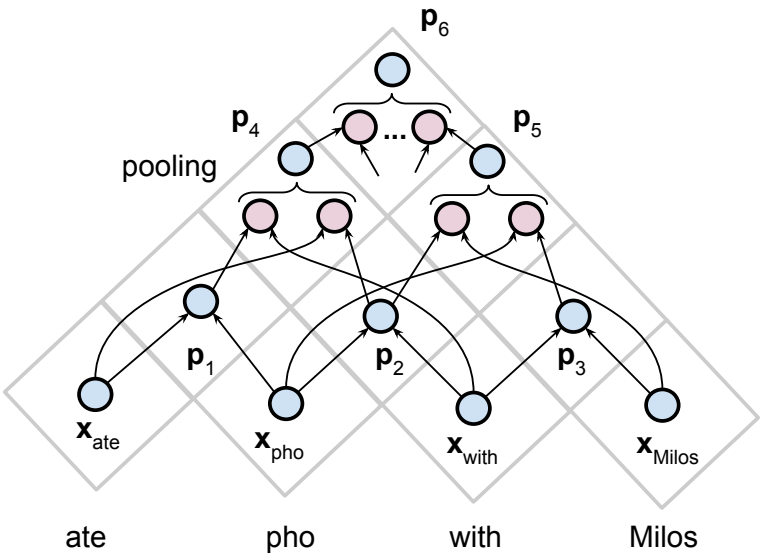


Figure 7.2: Chart Neural Network.

If the window size k is smaller than the number of children l, there are also $l+k-1$ vectors in the resulting pool, which means that we have $l + k - 1$ ways to combine l children. However, now, there is no longer a direct interaction between any two children whose distance is larger than k; still, an indirect interaction can be set up through pooling. In practice, this is not a big limitation because rules with a large number of children are very rare.

7.2.2 The Chart Neural Network model

Unseen sentences are, in the tradition we consider here, parsed by automatic parsers, which are far from perfect and not aware of tasks to which they are applied. Therefore, we would like to give the system a set of parses and let it decide which parse is the best or to combine several of them. The RNN model handles one extreme where this set contains only one parse. I now consider the other extreme where the set contains all possible parses (Figure 7.3). Because the number of all possible binary parse trees of a length-n sentence is the n-th Catalan number, processing individual parses is impractical. I thus propose a new model working on charts in the CKY style [Younger, 1967], called Chart Neural Network (ChNN).

I describe this model by the following example. Given the phrase "ate pho with Milos", the ChNN model will process its parse chart as in Figure 7.2. Because any 2-word constituent only has one parse, the computation for p_1, p_2, p_3 is identical to the computation of the traditional RNN. For the 3-word constituent p_4, because there are two possible productions $p_4 \rightarrow ate\ p_2$ and $p_4 \rightarrow p_1\ with$, we compute one vector for

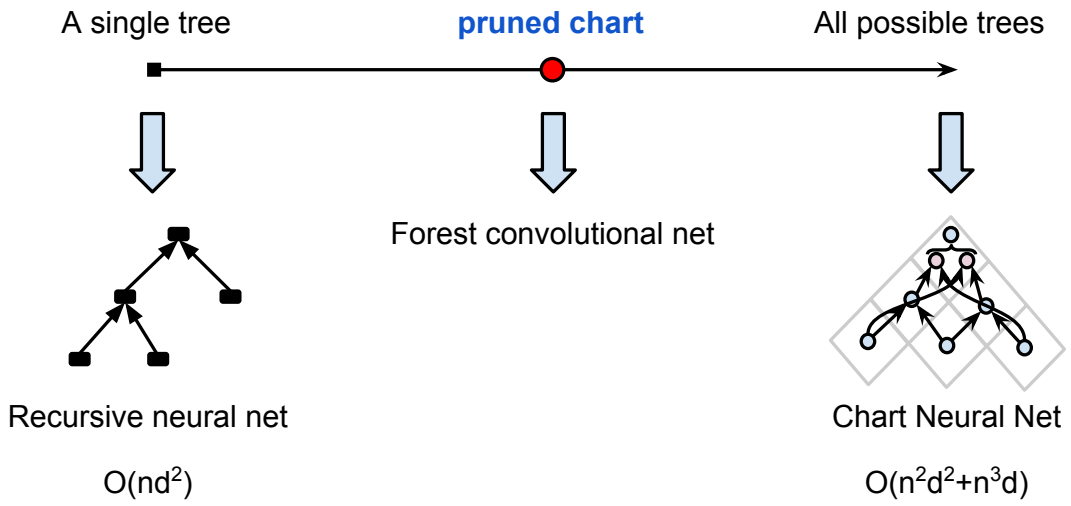


Figure 7.3: The three models, RNN, FCN, and ChNN, are sorted by the number of trees they can handle. The complexities of the RNN and the ChNN are also shown, where n is the sentence length, d is the dimension of vector representations.

each production:

$$\begin{aligned}\mathbf{u}^{(1)} &= f(\mathbf{W}_1\mathbf{x}_{ate} + \mathbf{W}_2\mathbf{p}_2 + \mathbf{b}) \\ \mathbf{u}^{(2)} &= f(\mathbf{W}_1\mathbf{p}_1 + \mathbf{W}_2\mathbf{x}_{with} + \mathbf{b})\end{aligned} \tag{7.1}$$

and then apply a max pooling operation to these two vectors to compute $\mathbf{p}_4$. We do the same to compute $\mathbf{p}_5$. Finally, at the top, there are three productions $p_6 \rightarrow ate\ p_5$, $p_6 \rightarrow p_1\ p_3$ and $p_6 \rightarrow p_4\ Milos$. Similarly, we compute one vector for each production and employ the max pooling operation to compute $\mathbf{p}_6$.

Because the ChNN model processes a chart like the CKY algorithm [Younger, 1967], its time complexity is $O(n^2d^2 + n^3d)$ where n and d are the sentence length and the dimension of vectors, respectively. This formula comes from the fact that in each cell we apply the matrix-vector multiplication two times and (if the cell is not a leaf) apply the max pooling to a pool of maximally n d-dim vectors. As a result, the ChNN model is notably more complex than the traditional RNN model, which has the complexity $O(nd^2)$. Like chart parsing, the complexity can be reduced significantly by pruning the chart before applying the ChNN model. This will be discussed right below.

7.2.3 The Forest Convolutional Network Model

I now introduce the Forest Convolutional Network (FCN) model, which is a combination of the two ideas proposed above. The key idea is to use an automatic parser to prune the chart, debinarize productions (if applicable), and then apply the same mechanism of the ChNN model where the computation in Equation 7.1 is replaced by a

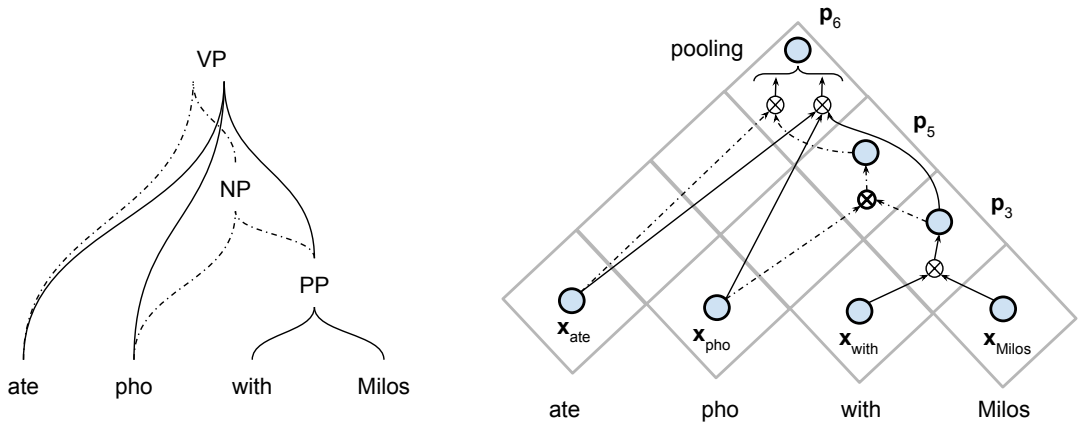


Figure 7.4: Forest of parses (left) and Forest Convolutional Network (right). $\otimes$ denotes a convolutional layer followed by a max pooling operation and a fully connected layer as in Figure 7.1.

convolutional layer followed by a max pooling operation and a fully connected layer as in Figure 7.1. Pruning a chart by an automatic parser is also not flawless. However, the quality of pruned charts can get very close to human annotation. For instance, the chart pruner proposed by Huang [2008] has a forest oracle of 97.8% F-score on section 23 of the Penn Treebank whereas resulting forests are very compact: the average number of hyperedges per forest is 123.1.

Figure 7.4 shows an illustration of how the FCN processes the phrase "ate pho with Milos". The example forest of parses (which is given by an external parser, in practice), comprises two parses:

- $(VP\ ate\ pho\ (PP\ with\ Milos))$ (solid lines) and
- $(VP\ ate\ (NP\ pho\ (PP\ with\ Milos)))$ (dash-dotted lines).

The first parse is the preferred reading if "Milos" is a person, but the second one is a possible reading (for instance, if "Milos" is the name of a sauce). Instead of forcing the external parser to decide which one is correct, we let the FCN model do that because it has more information about the context and domain, which are embedded in training data. What the network should do is depicted in Figure 7.4-right.

Training Training the FCN model is similar to training the traditional RNN model. I use the mini-batch gradient descent method to minimize an objective function J, which depends on which task this model is applied to. For instance, if the task is sentiment analysis, J is the cross-entropy over a training set $\mathcal{D}$ plus an L2-norm regularization term:

$$J(\theta) = -\frac{1}{|\mathcal{D}|}\sum_{s\in\mathcal{D}}\sum_{p\in s}\log Pr(c_p|\mathbf{p}) + \frac{\lambda}{2}||\theta||^2$$

where θ is the parameter set, c_p is the sentiment class of phrase p, $\mathbf{p}$ is the vector representation at the node covering p, $Pr(c_p|\mathbf{p})$ is computed by a softmax function, and λ is a regularization parameter.

Gradients $\partial J/\partial\theta$ are computed efficiently thanks to the back-propagation through structure algorithm [Goller and Küchler, 1996]. I use the AdaGrad method [Duchi et al., 2011] to automatically update the learning rate for each parameter.

7.3 Experiments

I evaluated the FCN model on two tasks: question classification and sentiment analysis. The evaluation metric is the classification accuracy.

The FCN model was initialized with the 300-dim GloVe word embeddings trained on a corpus of 840B words[2] [Pennington et al., 2014]. The initial values for a weight matrix were uniformly sampled from the symmetric interval $\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$ where n is the number of total input units. In each experiment, a development set was used to tune the model. I run the model ten times and chose the run with the highest performance on the development set. I also employed early stopping: training is halted if performance on the development set does not improve after three consecutive epochs.

7.3.1 Sentiment Analysis

In the first experiment, I used the Stanford Sentiment Treebank (SST)[3] [Socher et al., 2013b] which consists of 5-way fine-grained sentiment labels (very negative, negative, neutral, positive, very positive) for 215,154 phrases of 11,855 sentences. I used the standard splitting: 8544 sentences for training, 1101 for development, and 2210 for testing. The average sentence length is 19.1. In addition, the treebank also supports binary sentiment (positive, negative) classification by removing neutral labels, leading to: 6920 sentences for training, 872 for development, and 1821 for testing.

All sentences were parsed by Liang Huang's dependency parser[4] [Huang and Sagae, 2010]. I used this parser because it generates parse forests and because dependency trees are less deep than constituency trees. In addition, because the SST was annotated in a constituency manner, I also employed the Charniak's constituency parser [Charniak and Johnson, 2005] with Huang [2008]'s forest pruner. I found that the beam width 16 for the dependency parser and the log probability beam 10 for the other worked best. Lower values harm the system's performance and higher values are not beneficial.

Testing on the development set, I chose the size of vectors at internal nodes 200, the window size for the convolutional kernel 7, and the activation function $tanh$. The FCN model was trained with the learning rate 0.01, the regularization parameter 10^{-4}, and

[2] `http://nlp.stanford.edu/projects/GloVe/`
[3] `http://nlp.stanford.edu/sentiment/treebank.html`
[4] `http://acl.cs.qc.edu/~lhuang/software`

Model	Fine-grained	Binary
RNTN	45.7	85.4
CNN	48.0	88.1
DCNN	48.5	86.8
PV	48.7	87.8
DRNN	49.8	86.6
RLSTM	49.9	88.0
CT-LSTM	**51.0**	88.0
FCN (dep.)	50.4	88.2
FCN (const.)	**51.0**	**89.1**

Table 7.1: Test accuracies at sentence level on the SST dataset. FCN (dep.) and FCN (const.) denote the FCN with dependency forests and with constituency forests, respectively. The accuracies of the other models are copied from the corresponding papers (see text).

the mini batch size 5. To reduce the average depth of the network, the fully connected layer following the convolutional layer was removed (i.e., $\mathbf{p} = \mathbf{x}$, see Figure 7.1).

I compare the FCN model against other models: the Recursive neural tensor network (RNTN) [Socher et al., 2013b], the Convolutional neural network (CNN) [Kim, 2014], the Dynamic convolutional neural network (DCNN) [Kalchbrenner et al., 2014], the Paragraph vectors (PV) [Le and Mikolov, 2014], the Deep recursive neural network (DRNN) [Irsoy and Cardie, 2014], the Recursive Long short term memory (RLSTM) proposed in Chapter 6 (and in Le and Zuidema [2015c]) and the Constituency Tree LSTM (CT-LSTM) [Tai et al., 2015].[5]

Table 7.1 shows the test results. The FCN model using constituency forests achieves the highest accuracies on both fine-grained task and binary task, 51% and 89.1%. Comparing to the CT-LSTM model, although there are no differences on the fine-grained task, the difference on the binary task is substantially (1.1%). Comparing to the RLSTM, the differences on both tasks are all remarkable (1.1% and 1.1%).

Constituency parsing is clearly more helpful than dependency parsing, which is also confirmed in [Tai et al., 2015, Table 2]: the improvements that the FCN model gets are 0.6% on the fine-grained task and 0.9% on the binary task. This might be because sentences in the treebank were parsed by a constituency parser (here is the Stanford parser). As a result, training on constituency forests is more beneficial.

7.3.2 Question Classification

The second task is question classification. I used the TREC question dataset[6] [Li and Roth, 2002] which contains 5952 questions (5452 questions for training and 500 ques-

[5] See Section 6.3.1 for how the RLSTM and the Tree-LSTM are different.

[6] `http://cogcomp.cs.illinois.edu/Data/QA/QC`

Model	Acc. (%)
DCNN	93.0
MaxEnt$_H$	93.6
CNN-non-static	93.6
SVM$_S$	**95.0**
LSTM-RNN	93.4
FCN	<u>94.8</u>

Table 7.2: Test accuracies on the TREC question type classification dataset. The accuracies of the other models are copied from the corresponding papers (see text).

tions for testing). The task is to assign one of six types to a question: ABBREVIATION, ENTITY, DESCRIPTION, HUMAN, LOCATION, NUMERIC. The average length of the questions in the training set is 10.2 whereas in the test set is 7.5. This difference is due to the fact that those questions are from different sources. All questions were parsed by Liang Huang's dependency parser with the beam width 16.

I randomly picked 5% of the training set (272 questions) for validation. Using this development set, I chose the dimension of vectors at internal nodes 200, the window size for the convolutional kernel 5, and the activation function $tanh$. The FCN was trained with the learning rate 0.01, the regularization parameter 10^{-4}, and the mini batch size 1. The vectors representing the two padding tokens <b>, <e> were fixed to $\mathbf{0}$.

I compare the FCN model against the Convolutional neural network (CNN) [Kim, 2014], the Dynamic convolutional neural network (DCNN) [Kalchbrenner et al., 2014], MaxEnt$_H$ [Huang et al., 2008] (which uses MaxEnt with uni-bi-trigrams, POS, wh-word, head word, word shape, parser, hypernyms, WordNet) and the SVM$_S$ [Silva et al., 2011] (which uses SVM with, in addition to features used by MaxEnt$_H$, 60 hand-coded rules). I also include the RLSTM whose accuracy was computed by running the published source code[7] on binary trees generated by the Stanford Parser[8] [Klein and Manning, 2003]. The RLSTM was also initialized by the 300-dim GloVe word embeddings.

Table 7.2 shows the results.[9] The FCN achieves the second best accuracy, only slightly lower than the SVM$_S$ (0.2%). This is a promising result because the FCN uses only parse forests, unsupervisedly pre-trained word embeddings whereas the SVM$_S$ uses heavily engineered resources. The difference between the FCN and the third best is remarkable (1.2%). Interestingly, the RLSTM does not perform well on this dataset. This is likely because the questions are short and the parse trees quite shallow. As a result, the two problems that the LSTM architecture was invented for (the problem of how to capture long range dependencies and the vanishing gradient problem) are not

[7] `https://github.com/lephong/lstm-rnn`

[8] http://nlp.stanford.edu/software/lex-parser.shtml

[9] While finalizing the current work I discovered a paper by Ma et al. [2015] proposing a convolutional network model for dependency trees. They report a new state-of-the-art accuracy of 95.6% in accuracy.

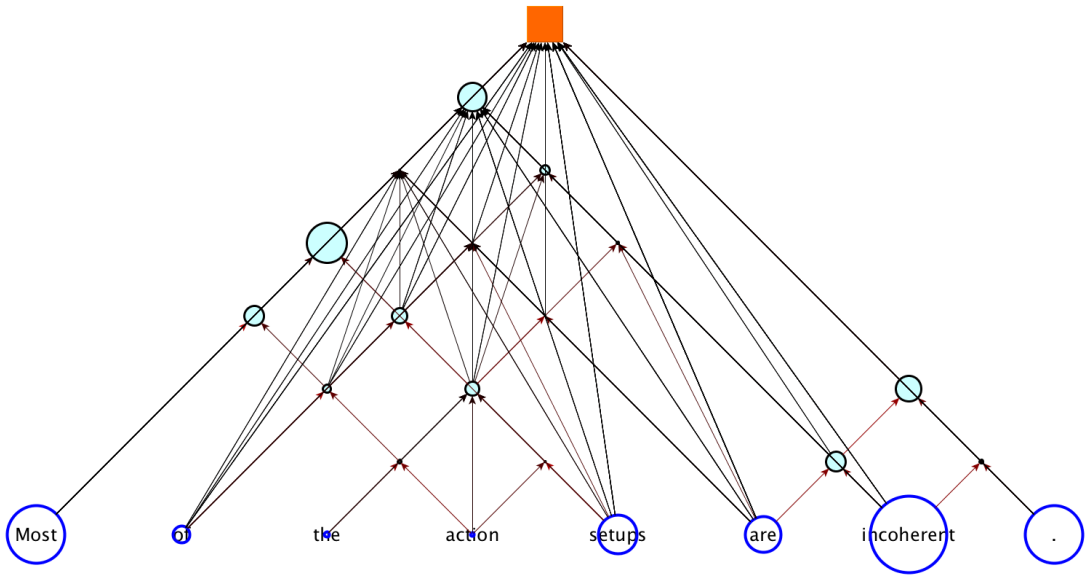


Figure 7.5: Chart of sentence "Most of the action setups are incoherent ." The size of a circle is proportional to the number of the cell's features that are propagated to the root.

crucial.

7.3.3 Visualization

I visualize the charts I obtained on the sentiment analysis task as in Figure 7.5. To identify how important each cell is for determining the final vector at the root, I computed the number of features of each that are actually propagated all the way to the root in the successive max pooling operations. The circles in a graph are proportional to this number. Here, to make the contribution of each individual cell clearer, I have set the window size to 1 to avoid direct interactions between cells.

At the lexical level, we can see that the FCN model can discriminate the important words from the others. The two words "most" and "incoherent" are key to the sentiment of this sentence: if one of them is replaced by another word (e.g. replacing "most" by "few" or "incoherent" by "coherent"), the sentiment will flip. The punctuation "." however also has a high contribution to the root. This happens to other charts as well. I conjecture that the network uses the vector of "." to store neutral features and propagates them to the root whenever it can not find more useful features in other vectors.

At the phrasal level, the network tends to group words in grammatical constituents, such as "most of the action setups", "are incoherent". Ill-formed constituents such as "of the action" and "incoherent ." receive little attention from the network. The evidence is that the circles corresponding to ill-formed constituents are relatively much

smaller than the circles corresponding to well-formed constituents. Interestingly, we can see that the circle of "incoherent" is larger than the circles of any internal cells, suggesting that the network is able to make use of parses containing direct links from that word to the root. This shows that the network has the ability of selecting (or combining) parses that are beneficial to this sentiment analysis task.

7.4 Related Work

The idea that a composition function must be able to change its behaviour on the fly according to input vectors is explored by Socher et al. [2013b] and my work in Chapter 6, to name a few. The tensor in the former is multiplied with the vector representations of the phrases it is going to combine to define a composition function (a matrix) on the fly, which is then multiplied again with these vectors to yield a compound representation. In the RLSTM architecture of the latter, there is one input gate for each child in order to control how the vector of the child affects the composition at the parent node. Because the input gate is a function of the vector of the child, the composition function has an *infinite* number of behaviours. In this chapter, I instead slide a kernel function along the sequence of children to generate different ways of composition. Although the number of behaviours is limited (and depends on the window size), it simultaneously provides us with a solution to handle rules of different branching factors.

Some approaches try to overcome the problem of varying branching factors. My work in Chapter 4 uses different sets of weight matrices for different branching factors, thus requiring a large number of parameters. Because productions of large branching factors are rare, many parameters are infrequently updated during training. Socher et al. [2014], for dependency trees, use a weight matrix for each relative position to the head word (e.g., first-left, second-right). My work in Chapter 5 replaces relative positions by dependency relations (e.g., OBJ, SUBJ). These approaches strongly depend on input parse trees and are very sensitive to parsing errors. The approach presented in this chapter, on the other hand, does not need the information about the head word position and is less sensitive to parsing errors. Moreover, its number of parameters is independent from the maximal branching factor.

Convolutional networks have been widely applied to solve natural language processing tasks. Collobert et al. [2011b], Kalchbrenner et al. [2014], and Kim [2014] use convolutional networks to deal with varying length sequences. Recently, Zhu et al. [2015a] and Ma et al. [2015] integrate syntactic information by employing parse trees. Ma et al. [2015] extend the work of Kim [2014] by taking into account dependency relations so that long range dependencies could be captured. The model proposed by Zhu et al. [2015a], which is very similar to my proposal in Section 7.2.1, is to use a convolutional network for the composition purpose. My work, although also employing a convolutional network and syntactic information, goes beyond all of them: I address the issue of how to deal with uncertainty about the correct parse inside the neural architecture. To do so, instead of using a single parse, the proposed FCN model

takes as input a forest of parses.

Related to the FCN model is the Gated recursive convolutional neural network model proposed by Cho et al. [2014] which stacks $n - 1$ convolutional neural layers using a window-size-2 gated kernel (where n is the sentence length). If we map their network into a chart, we can see that each cell is only connected to the two cells right below it. What makes this network model special is the gated kernel which is a 3-gate switcher for (softly) choosing one of three options: directly transmit the left/right child's vector to the parent node, or compose the vectors of the two children. Thanks to this, the network can capture any binary parse tree by setting those gates properly. However, the network is not able to capture arbitrary forests. For instance, if the switcher prefers the left gate, the combination of the children's vectors cannot have a large effect on cells higher up in the chart. The FCN model, which can handle arbitrary forests, is thus more expressive and flexible than the other.

7.5 Conclusions

I propose the Forest Convolutional Network (FCN) model that tackles the three issues: (1) how to make the composition functions adaptive, (2) how to deal with different branching factors of nodes in the relevant syntactic trees, and (3) how to deal with uncertainty about the correct parse inside the neural architecture. The key principle is to carry out many different ways of computation and then choose or combine some of them. For more details, the two first issues are solved by employing a convolutional net for composition. To the third issue, the network takes as input a forest of parses rather than a single parse as in traditional approaches.

There are different ways to improve the current model. We can focus on how to choose/combine different ways of computation. For instance, we might replace the max pooling by different pooling operations such as mean pooling, k-max pooling [Kalchbrenner et al., 2014], and stochastic pooling [Zeiler and Fergus, 2013]. Another way is to bias the selection/combination toward grammatical constituents by weighing cells by their inside probabilities [Lari and Young, 1990].

Chapter 8

Conclusions

8.1 Summary

In this dissertation I propose several recursive deep network models for learning to transform sentences to vectors, based on the principle of compositionality. Starting with the traditional Recursive neural network (RNN) model, which is successfully exploited in Socher et al. [2010] and Socher et al. [2011b], I explore three different directions.

The first direction focuses on strengthening the composition functions. One way to do that is making use of syntactic information and contexts, as in Chapter 3. In that chapter, I propose composition functions, which are also one-layer feed-forward neural networks, taking into account representations of syntactic labels (e.g. N, VP), context words, and head words. Another way is to replace one-layer neural networks by more advanced networks. In Chapter 6, based on empirical results which show that the Long short term memory (LSTM) architecture can capture long range dependencies and deal with the vanishing gradient problem more effectively than Recurrent neural networks, I introduce a novel variant of the LSTM, called Recursive-LSTM, that works on trees. Empirical results on an artificial task and on the Stanford Sentiment Treebank confirm that the proposed Recursive-LSTM model is superior to the traditional RNN model in terms of accuracy. Furthermore, in Chapter 7, I demonstrate how a convolutional neural network can be used as a composition function.

The second direction to extend the classical RNN is to focus on how information flows in a parse tree. In traditional compositional semantics approaches, including the RNN model, information flows in a bottom-up manner, leading to a situation where there is no way for a node to be aware of its surrounding context. As a result, these approaches are not applicable to top-down processes such as several top-down generative parsing models, and to problems requiring contexts such as semantic role labelling. In Chapter 4, I propose a solution to this, namely the Inside-Outside Semantic framework, in which the key idea is to allow information to flow not only bottom-up but also top-down. In this way, we can recursively compute representations for the content

and the context of the phrase that a node in a parse tree covers. The Inside-Outside RNN model, a neural-net-based instance of this framework, is shown to work well on several tasks, including unsupervised composition function learning from raw texts, supervised semantic role labelling, and dependency parsing (Chapter 5).

The third direction is dealing with the uncertainty of the correct parse. As a result of relying on the principle of compositionality, compositional semantics uses syntactic parse trees to guide composition, which in turn makes compositional semantics approaches vulnerable to the errors of automatic parsers. The problems here are that automatic parsers are not flawless, and that they are not aware of domains to which they are applied. To overcome this problem, in Chapter 7, I propose the Forest Convolutional Network model, which takes as input a forest of parse trees rather than a single tree as in traditional approaches. The key idea is that we should give the model several options and let it select (or combine) ones that best fit its need. Empirical results show that the model performs on par with state-of-the-art models on the Stanford Sentiment Treebank and on the TREC question dataset.

8.2 Do we need trees?

Although it is clear, from a theoretical point of view [Montague, 1970, Bos, 2011], that in many cases syntactic trees are essential for computing meaning on the sentence level,[1] many recent neural-network-based approaches do not make use of syntactic trees. Some of them, such as Kim [2014] and Kalchbrenner et al. [2014], employ convolutional networks. Several use (deep) recurrent neural networks and similar architectures, for instance Sutskever et al. [2014], Luong et al. [2015], Wen et al. [2015].

The rejection of trees as an essential level of representation is clear in neural machine translation, where the key idea is to represent the source sentence by a vector and then use it to generate a sentence in the target language. The majority of recent approaches, such as Sutskever et al. [2014] and Luong et al. [2015], rely on deep recurrent models (such as Figure 8.1) (with or without the help of an attention mechanism [Bahdanau et al., 2015]). The only exception I found is the work of Liu et al. [2014]. This is also true for other research areas, such as language modelling [Kim et al., 2016, Tran et al., 2016] and dialog processing [Wen et al., 2015, Vinyals and Le, 2015].

This state of affairs raises the question "*when are parse trees needed in practice?*" The work of Li et al. [2015] is the first to systematically compare recursive models and recurrent models on four tasks: binary sentiment classification, phrase matching, semantic relation classification, and discourse parsing. They find that the recursive models can mainly help on tasks that "require long distance connection modeling, particularly on very long sequences". Breaking long sentences at punctuations allows recurrent models to perform on par with recursive models. They then conclude that the role of tree structures is to break down long sentences into manageable components.

[1] See Frank et al. [2012] for a paper that challenges this.

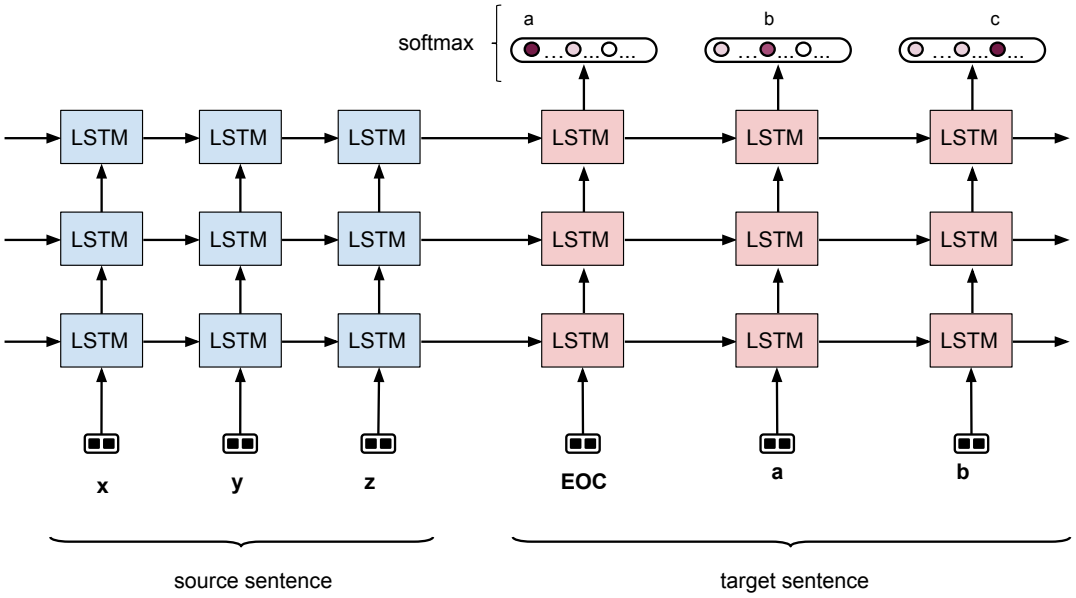


Figure 8.1: An LSTM-based sequence-to-sequence learning model for machine translation.

Concurrently with Li et al. [2015], Bowman et al. [2015b] also compare the two classes of models. They find that recurrent models can discover and implicitly use recursive structure. However, they demonstrate that recurrent models are significantly outperformed on an artificial task (logical relation classification) where recursive structure is crucial.

To my knowledge, no final answers to the question have been presented. On the one hand, several NLP tasks such as sentiment analysis and question classification do not strictly require syntactic structures. For instance, in sentiment analysis, finding out and combining keywords (e.g. "good", "bad") and negation (e.g. "not") is more important than knowing how to compose a noun phrase with a verb phrase. Another problem is that automatic parsers are still far from perfect, leading to the fact that a system relying on parse trees could be "fooled" by errors on automatically generated trees.[2] Besides, automatic parsers are not always available or reliable, especially in machine translation and dialog processing.

On the other hand, it seems that existing deep learning models do not yet exploit syntactic structures effectively. The evidence is that, as stated by Manning [2016], so far the most important factor helping us to gain promising empirical results is word embeddings. I therefore believe that we should not give up on recursive models. Instead, there is the need for more advanced ones, if gains from word embeddings cannot yet be further improved. More importantly, because employing tree structures is not harmful,

[2]This is, of course, the problem I address in Chapter 7 by proposing the Forest Convolutional Network model which takes as input a forest of parses.

according to the empirical results reported by Li et al. [2015], and a complete AI system should be involved in logical operations, recursive models should be considered. Furthermore, the need for external (statistical) symbolic parsers can be eliminated by recent connectionist approaches aiming to tackle the challenge of syntactic parsing (such as Borensztajn [2011]).

8.3 Future work: Unifying the three directions

Since the three directions I explore in this dissertation are not mutually exclusive, I can freely unify them. An obvious step to extend the work in this dissertation is therefore to build a model which takes into account all of these three directions. This model

- implements the Inside-outside semantic framework for representing both contents and contexts,
- uses R-LSTMs as composition functions for capturing long range dependencies and dealing with the vanishing gradient problem, and
- employs the same mechanism as in the FCN model for making use of parse forests.

As a result, this model can be considered as a complete neural implementation for the inside-outside algorithm [Lari and Young, 1990]; and hence, its potential usage covers all applications where chart-based algorithms can be employed, such as modelling language with probabilistic top-down parsing [Roark, 2001].

Appendix A

Inside-Outside Semantics Framework with λ-Expressions

I demonstrate how the IOS framework works with λ-expressions as in formal semantics. First of all, I rely on the observation that the content and context of a constituent must be strongly correlated in the sense that we can predict the content thanks to the context. For instance, given the world model

$$M = \{\text{Tom loves cheese, mice love cheese, mice hate cats}\}$$

and the context "mice love X", we can infer that $X \in \{\textit{cheese}\}$. Therefore, I hypothesize that, given a world model M, $M \models \mathbf{o}_p\mathbf{i}_p$ must hold true for any constituent p, for instance $\mathbf{i}_{cheese} = cheese$ and $\mathbf{o}_{cheese} = \lambda y.love(mice, y)$.

Combinatory Categorial Grammar (CCG) [Steedman, 2000] (see Section 4.4) is a widely used grammar formalism for semantic applications thanks to its transparent surface between syntax and semantics [Bos et al., 2004]. In the terminology of the IOS framework, it provides an elegant method to compute *inside representations* (see the first and second columns of Table A.1. Figure A.1 shows a CCG derivation for the sentence "mice love cheese".) However, it does not specify the operations needed to compute *outside representations*. To do so, I introduce extensions for its combinatory rules.

Extended rules From the hypothesis that $M \models \mathbf{o}_p\mathbf{i}_p$ must hold true for any constituent p, I postulate a more strict constraint: $\mathbf{o}_p\mathbf{i}_p = \mathbf{o}_q\mathbf{i}_q$ for all constituents p, q, which is equivalent to the following constraint:

$$\mathbf{o}_c\mathbf{i}_c = \mathbf{o}_p\mathbf{i}_p \;\; \forall c \in \mathcal{C}(p), \text{ where } \mathcal{C}(p) \text{ is the set of children of } p \tag{A.1}$$

Therefore, I propose 9 formulas shown in the third column of Table A.1. To prove that these formulas satisfy the constraint in Equation A.1, for an arbitrary child node, simply applying the outside representation to the inside representation and then employing the beta reduction, we get $\mathbf{o}_p\mathbf{i}_p$.

Rule	Inside (parent)	Outside (children)
(fa) $X/Y \quad Y \Rightarrow X$	$\mathbf{i}_p = \mathbf{i}_l\mathbf{i}_r$	$\mathbf{o}_l = \lambda P.\mathbf{o}_p\,(P\mathbf{i}_r)$ $\mathbf{o}_r = \lambda P.\mathbf{o}_p\,(\mathbf{i}_l P)$
(ba) $Y \quad X\backslash Y \Rightarrow X$	$\mathbf{i}_p = \mathbf{i}_r\mathbf{i}_l$	$\mathbf{o}_l = \lambda P.\mathbf{o}_p\,(\mathbf{i}_r P)$ $\mathbf{o}_r = \lambda P.\mathbf{o}_p\,(P\mathbf{i}_l)$
(comp) $X/Y \quad Y/Z \Rightarrow X/Z$	$\mathbf{i}_p = \lambda x.(\mathbf{i}_l(\mathbf{i}_r x))$	$\mathbf{o}_l = \lambda P.\mathbf{o}_p\,(\lambda x.(P(\mathbf{i}_r x)))$ $\mathbf{o}_r = \lambda P.\mathbf{o}_p\,(\lambda x.(\mathbf{i}_l(Px)))$
(conj) $X \;\; conj \;\; X' \Rightarrow X''$	$\mathbf{i}_p = \lambda x.(\mathbf{i}_l x \wedge \mathbf{i}_r x)$	$\mathbf{o}_l = \lambda P.\mathbf{o}_p\,(\lambda x.(Px \wedge \mathbf{i}_r x)))$ $\mathbf{o}_r = \lambda P.\mathbf{o}_p\,(\lambda x.(\mathbf{i}_l x \wedge Px)))$
(tr) $A \Rightarrow X/(X\backslash A)$	$\mathbf{i}_p = \lambda R.(R\mathbf{i}_c)$	$\mathbf{o}_c = \lambda P.\mathbf{o}_p\,(\lambda R.(RP))$

Table A.1: The five basic CCG combinatory rules (the first and second columns) and their extensions for computing outside representations (the third column). p, l, r, c respectively denote parent, left-child, right-child, and child.

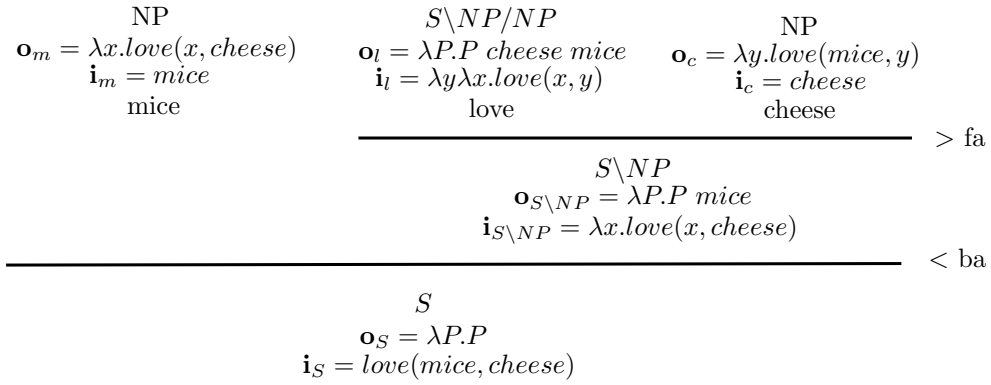


Figure A.1: CCG parse tree and inside, outside representations for the sentence *"Mice love cheese"*.

In order to see how those formulas work, let us consider a simple example of the sentence "Mice love cheese" (Figure A.1). First of all, because this sentence stands alone, $\lambda P.P$ is assigned to the outside representation of the whole sentence, which means that the truth of this sentence solely depends on its own meaning. Then, using the formulas given in Table A.1 we compute the outside and inside representations for each constituent.

The first thing we can see is that, given the world model M above, all $\langle$inside, outside$\rangle$ pairs satisfy the requirement $M \models \mathbf{oi}$. Secondly, it is easy to see that those outside representations are intuitive. For instance, the outside representation of the word "cheese" $\mathbf{o}_c = \lambda x.love(mice, x)$ perfectly intuitively matches the context "mice love –": the *missing* word/constituent must represent an object x that makes the predicate $love(cheese, x)$ true.

One application for this might be to represent questions. Since a question can be seen as a context of an unknown object/information, we can use the IOS framework to

represent a question by converting it to the cloze format (a sentence containing a blank corresponding to the unknown constituent). The outside representation of the unknown constituent therefore can be used as the representation for the question. Another application is to be a base for formal-semantics-liked compositional distributional semantics approaches, e.g. [Baroni et al., 2013], to compute context representations.

Bibliography

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Marco Baroni, Raffaella Bernardi, and Roberto Zamparelli. Frege in space: A program for compositional distributional semantics. In A. Zaenen, B. Webber, and M. Palmer, editors, *Linguistic Issues in Language Technologies*. CSLI Publications, Stanford, CA, 2013.

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 238–247, 2014.

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

Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8624–8628. IEEE, 2013a.

Yoshua Bengio, Aaron Courville, and Pierre Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828, 2013b.

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

Rens Bod, Remko Scha, and Khalil Sima'an. *Data-Oriented Parsing*. CSLI Publications, Stanford, CA, 2003.

Bernd Bohnet and Jonas Kuhn. The best of both worlds: a graph-based completion model for transition-based parsers. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 77–87. Association for Computational Linguistics, 2012.

Gideon Borensztajn. *The neural basis of structure in language Bridging the gap between symbolic and connectionist models of language processing*. PhD thesis, University of Amsterdam, 2011.

Gideon Borensztajn and Willem Zuidema. Episodic grammar: a computational model of the interaction between episodic and semantic memory in language processing. In *Proceedings of the 33d Annual Conference of the Cognitive Science Society (CogSci'11)*, pages 507–512. Lawrence Erlbaum Associates, 2011.

Johan Bos. A survey of computational semantics: Representation, inference and knowledge in wide-coverage text understanding. *Language and Linguistics Compass*, 5(6):336–366, 2011.

Johan Bos, Stephen Clark, Mark Steedman, James R Curran, and Julia Hockenmaier. Wide-coverage semantic representations from a ccg parser. In *Proceedings of the 20th international conference on Computational Linguistics*, page 1240. Association for Computational Linguistics, 2004.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal, September 2015a. Association for Computational Linguistics.

Samuel R Bowman, Christopher D Manning, and Christopher Potts. Tree-structured composition in neural networks without tree-structured architectures. In *Proceedings of the NIPS Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches*, 2015b.

Xavier Carreras and Lluís Màrquez. Introduction to the conll-2005 shared task: Semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 152–164. Association for Computational Linguistics, 2005.

Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics*, pages 132–139. Association for Computational Linguistics, 2000.

Eugene Charniak. Immediate-head parsing for language models. In *ACL*, pages 116–123, 2001.

Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 173–180. Association for Computational Linguistics, 2005.

Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. *Syntax, Semantics and Structure in Statistical Translation*, page 103, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Trevor Cohn and Philip Blunsom. Semantic role labelling with tree conditional random fields. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 169–172. Association for Computational Linguistics, 2005.

Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, pages 16–23, 1997.

Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

Michael Collins. Discriminative reranking for natural language parsing. In *Proceedings of the International Workshop on Machine Learning (then Conference)*, pages 175–182, 2000.

Michael Collins. Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637, 2003.

Michael Collins and Terry Koo. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–66, 2005.

Ronan Collobert. Deep learning for efficient discriminative parsing. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011a.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011b.

James R Curran, Stephen Clark, and Johan Bos. Linguistically motivated large-scale nlp with c&c and boxer. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 33–36. Association for Computational Linguistics, 2007.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Georgiana Dinu and Mirella Lapata. Measuring distributional similarity in context. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1162–1172. Association for Computational Linguistics, 2010.

Georgiana Dinu, Nghia The Pham, and Marco Baroni. General estimation and evaluation of compositional distributional semantic models. In *Workshop on Continuous Vector Space Models and their Compositionality, Sofia, Bulgaria*, 2013.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, pages 2121–2159, 2011.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. Transition-based dependeny parsing with stack long short-term memory. In *Proc. ACL*, 2015.

Jason Eisner. *Smoothing a Probabilistic Lexicon via Syntactic Transformations*. PhD thesis, University of Pennsylvania, July 2001. 318 pages.

Jason M. Eisner. An empirical comparison of probability models for dependency grammar. Technical report, University of Pennsylvania Institute for Research in Cognitive Science, 1996a.

Jason M. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics, 1996b.

Jeffrey L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Katrin Erk. Vector space models of word meaning and phrase meaning: A survey. *Language and Linguistics Compass*, 6(10):635–653, 2012.

John Rupert Firth. *Papers in Linguistics*. Oxford Univeristy Press, 1957a.

John Rupert Firth. A synopsis of linguistic theory 1930-55. In *Studies in Linguistic Analysis (special volume of the Philological Society)*, volume 1952-59, pages 1–32. The Philological Society, Oxford, 1957b.

Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1):3–71, 1988.

Stefan L. Frank, Rens Bod, and Morten H. Christiansen. How hierarchical is language use? *Proceedings of the Royal Society of London B: Biological Sciences*, page rspb20121741, 2012.

Felix Gers. Long short-term memory in recurrent neural networks. *Unpublished PhD dissertation, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland*, 2001.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

Pavel Golik, Patrick Doetsch, and Hermann Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *INTERSPEECH*, pages 1756–1760, 2013.

Christoph Goller and Andreas Küchler. Learning task-dependent distributed representations by backpropagation through structure. In *International Conference on Neural Networks*, pages 347–352. IEEE, 1996.

Alex Graves. *Supervised sequence labelling with recurrent neural networks*. Springer, 2012.

Edward Grefenstette, Georgiana Dinu, Yao-Zhong Zhang, Mehrnoosh Sadrzadeh, and Marco Baroni. Multi-step regression learning for compositional distributional semantics. In *Proceedings of the 10th International Conference on Computational Semantics (IWCS)*, 2013.

Katsuhiko Hayashi, Taro Watanabe, Masayuki Asahara, and Yuji Matsumoto. Third-order variational reranking on packed-shared dependency forests. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1479–1488. Association for Computational Linguistics, 2011.

Katsuhiko Hayashi, Shuhei Kondo, and Yuji Matsumoto. Efficient stacked dependency parsing by forest reranking. *Transactions of the Association for Computational Linguistics*, 1(1):139–150, 2013.

Karl Moritz Hermann and Phil Blunsom. The role of syntax in vector space models of compositional semantics. In *Proceedings of the ACL*, 2013.

Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kremer and Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.

Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882. Association for Computational Linguistics, 2012.

Liang Huang. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of the Annual Meeting on Association for Computational Linguistics*, pages 586–594, 2008.

Liang Huang and Kenji Sagae. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086. Association for Computational Linguistics, 2010.

Zhiheng Huang, Marcus Thint, and Zengchang Qin. Question classification using head words and their hypernyms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 927–936. Association for Computational Linguistics, 2008.

Heikki Hyötyniemi. Turing machines are recurrent neural networks. *Proceedings of STEP*, 96, 1996.

Ozan Irsoy and Claire Cardie. Bidirectional recursive neural networks for token-level labeling with structure. In *NIPS Workshop on Deep Learning*, 2013.

Ozan Irsoy and Claire Cardie. Deep recursive neural networks for compositionality in language. In *Advances in Neural Information Processing Systems*, pages 2096–2104, 2014.

Ray Jackendoff. *Foundations of Language: Brain, Meaning, Grammar, Evolution*. Oxford: Oxford University Press, 2001.

Yangfeng Ji and Jacob Eisenstein. One vector is not enough: Entity-augmented distributed semantics for discourse relations. *Transactions of the Association for Computational Linguistics*, 3:329–344, 2015. ISSN 2307-387X.

Thorsten Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM, 2006.

Mark Johnson and Ahmet Engin Ural. Reranking the Berkeley and Brown parsers. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 665–668. Association for Computational Linguistics, 2010.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, 2016.

Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

Terry Koo and Michael Collins. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11. Association for Computational Linguistics, 2010.

Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. *CoRR*, abs/1506.07285, 2015.

Karim Lari and Steve J Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer speech & language*, 4(1):35–56, 1990.

Phong Le. Enhancing the inside-outside recursive neural network reranker for dependency parsing. In *Proceedings of the 14th International Conference on Parsing*

Technologies, pages 87–91, Bilbao, Spain, July 2015. Association for Computational Linguistics.

Phong Le and Willem Zuidema. The inside-outside recursive neural network model for dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2014a.

Phong Le and Willem Zuidema. Inside-outside semantics: A framework for neural models of semantic composition. In *NIPS 2014 Workshop on Deep Learning and Representation Learning*, 2014b.

Phong Le and Willem Zuidema. The forest convolutional network: Compositional distributional semantics with a neural chart and without binarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1155–1164, Lisbon, Portugal, September 2015a. Association for Computational Linguistics.

Phong Le and Willem Zuidema. Unsupervised dependency parsing: Let's use supervised parsers. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 651–661, Denver, Colorado, May–June 2015b. Association for Computational Linguistics.

Phong Le and Willem Zuidema. Compositional distributional semantics with long short term memory. In *Proceedings of the Joint Conference on Lexical and Computational Semantics (*SEM)*. Association for Computational Linguistics, 2015c.

Phong Le, Willem Zuidema, and Remko Scha. Learning from errors: Using vector-based compositional semantics for parse reranking. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality (51st Annual Meeting of the ACL)*, pages 11–19, 2013.

Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196, 2014.

Quoc Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg Corrado, Jeff Dean, and Andrew Ng. Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning*, 2012.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

Alessandro Lenci. Distributional semantics in linguistic and cognitive research. *Italian journal of linguistics*, 20(1):1–31, 2008.

Jiwei Li, Thang Luong, Dan Jurafsky, and Eduard Hovy. 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, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics, 2002.

Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

Shujie Liu, Nan Yang, Mu Li, and Ming Zhou. A recursive recurrent neural network for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1491–1500, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

Mingbo Ma, Liang Huang, Bowen Zhou, and Bing Xiang. Dependency-based convolutional neural networks for sentence embedding. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 174–179, Beijing, China, July 2015. Association for Computational Linguistics.

Christopher D Manning. Computational linguistics and deep learning. *Computational Linguistics*, 2016.

Christopher D Manning and Bob Carpenter. Probabilistic parsing using left corner language models. In *Advances in probabilistic and other parsing technologies*. Springer, 2000.

Andre Martins, Miguel Almeida, and Noah A. Smith. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *HLT-NAACL*, 2006.

Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 91–98. Association for Computational Linguistics, 2005.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. Multilingual dependency analysis with a two-stage discriminative parser. In *CoNLL*. Association for Computational Linguistics, 2006.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, et al. Universal dependency annotation for multilingual parsing. *Proceedings of ACL, Sofia, Bulgaria*, 2013.

Sauro Menchetti, Fabrizio Costa, Paolo Frasconi, and Massimiliano Pontil. Wide coverage natural language processing using kernel methods and neural networks for structured data. *Pattern Recogn. Lett.*, 26(12):1896–1906, September 2005. ISSN 0167-8655. doi: 10.1016/j.patrec.2005.03.011.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048, 2010.

Tomas Mikolov, Anoop Deoras, Stefan Kombrink, Lukas Burget, and Jan Cernockỳ. Empirical evaluation and combination of advanced language modeling techniques. In *INTERSPEECH*, pages 605–608, 2011.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013c.

Marvin L Minsky and Seymour A Papert. *An Introduction to Computational Geometry*. MIT press Boston, MA:, 1969.

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

Jeff Mitchell and Mirella Lapata. Language models based on semantic composition. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 430–439, 2009.

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

Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1751–1758, 2012.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Richard Montague. English as a formal language. In Bruno Visentini, editor, *Linguaggi nella Società e nella Tecnica*, pages 189–224. Edizioni di Comunità, Milano, 1970.

Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pages 246–252. Citeseer, 2005.

Timothy J O'Donnell, Noah D Goodman, and Joshua B Tenenbaum. Fragment grammar: Exploring reuse in hierarchical generative processes. Technical report, Technical Report MIT-CSAIL-TR-2009-013, MIT, 2009.

Sebastian Padó. *User's guide to* `sigf`*: Significance testing by approximate randomisation*, 2006.

Barbara Partee. Lexical semantics and compositionality. In Lila R. Gleitman and Mark Liberman, editors, *An Invitation to Cognitive Science. Vol.1: Language*, pages 311–360. MIT Press, Cambridge, MA, 1995.

Romain Paulus, Richard Socher, and Christopher D Manning. Global belief recursive neural networks. In *Advances in Neural Information Processing Systems*, pages 2888–2896, 2014.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12:1532–1543, 2014.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the ACL*, pages 433–440, 2006.

Jordan B. Pollack. Recursive auto-associative memory. *Neural Networks*, 1:122, 1988.

Jordan B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1): 77–105, 1990.

Vasin Punyakanok, Dan Roth, and Wen tau Yih. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287, 2008.

Brian Roark. Probabilistic top-down parsing and language modeling. *Computational linguistics*, 27(2):249–276, 2001.

Federico Sangati and Willem Zuidema. Accurate parsing with compact tree-substitution grammars: Double-DOP. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 84–95. Association for Computational Linguistics, 2011.

Federico Sangati, Willem Zuidema, and Rens Bod. A generative re-ranking model for dependency parsing. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 238–241, 2009.

Remko Scha. Taaltheorie en taaltechnologie; competence en performance. In R. de Kort and G.L.J. Leerdam, editors, *Computertoepassingen in de Neerlandistiek*, pages 7–22. LVVN, Almere, the Netherlands, 1990. English translation at http://iaaa.nl/rs/LeerdamE.html.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

Libin Shen and Aravind K Joshi. An SVM based voting algorithm with application to parse reranking. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 9–16. Association for Computational Linguistics, 2003.

Joao Silva, Luísa Coheur, Ana Cristina Mendes, and Andreas Wichert. From symbolic to sub-symbolic information in question classification. *Artificial Intelligence Review*, 35(2):137–154, 2011.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Richard Socher, Christopher D. Manning, and Andrew Y. Ng. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, 2010.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. *Advances in Neural Information Processing Systems*, 24:801–809, 2011a.

Richard Socher, Cliff C. Lin, Andrew Y. Ng, and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 26th International Conference on Machine Learning*, volume 2, 2011b.

Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 151–161, 2011c.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 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, 2012.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 455–465, 2013a.

Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings EMNLP*, 2013b.

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

Valentin I Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. Breaking out of local optima with count transforms and model recombination: A study in grammar induction. In *EMNLP*, pages 1983–1995, 2013.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Mark F. St John and James L. McClelland. Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence*, 46(1):217–257, 1990.

Mark Steedman. Connectionist sentence processing in perspective. *Cognitive Science*, 23(4):615–634, 1999.

Mark Steedman. *The syntactic process*. MIT Press, 2000.

Pontus Stenetorp. Transition-based dependency parsing using recursive neural networks. In *NIPS Workshop on Deep Learning*, 2013.

Andreas Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational linguistics*, 21(2), 1995.

Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 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, Beijing, China, July 2015. Association for Computational Linguistics.

Ivan Titov and James Henderson. Loss minimization in parse reranking. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 560–567. Association for Computational Linguistics, 2006.

Ivan Titov and James Henderson. A latent variable model for generative dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies*, pages 144–155, 2007.

Ke Tran, Arianna Bisazza, and Christof Monz. Recurrent memory network for language modeling. *arXiv preprint arXiv:1601.01272*, 2016.

Andreas van Cranenburgh and Rens Bod. Discontinuous parsing with an efficient and accurate DOP model. In *Proceedings of the International Conference on Parsing Technologies (IWPT'13)*, 2013.

Andreas van Cranenburgh, Remko Scha, and Federico Sangati. Discontinuous Data-Oriented Parsing: A mildly context-sensitive all-fragments grammar. In *Proceedings of the Second Workshop on Statistical Parsing of Morphologically Rich Languages*, pages 34–44. Association for Computational Linguistics, 2011.

Sara Veldhoen. Semantic Adequacy of Compositional Distributed Representations. Master's thesis, University of Amsterdam, the Netherlands, 2015.

Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation

for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

Paul Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, 1974.

Paul Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Nianwen Xue and Martha Palmer. Calibrating features for semantic role labeling. In *EMNLP*, pages 88–94, 2004.

Hiroyasu Yamada and Yuji Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of International Conference on Parsing Technologies (IWPT)*, pages 195–206, 2003.

Daniel H Younger. Recognition and parsing of context-free languages in time n^3. *Information and control*, 10(2):189–208, 1967.

Hwanjo Yu and Sungchul Kim. SVM tutorial: Classification, regression, and ranking. In Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, editors, *Handbook of Natural Computing*, volume 1, pages 479–506. Springer, 2012.

Matthew D Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2013.

Hao Zhang and Ryan McDonald. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 320–331. Association for Computational Linguistics, 2012.

Chenxi Zhu, Xipeng Qiu, Xinchi Chen, and Xuanjing Huang. A re-ranking model for dependency parser with recursive convolutional neural network. 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 1159–1168, Beijing, China, July 2015a. Association for Computational Linguistics.

Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. Long short-term memory over recursive structures. In *Proceedings of International Conference on Machine Learning*, July 2015b.

Summary

Learning Vector Representations for Sentences
– The Recursive Deep Learning Approach

Natural language processing (NLP) systems, until recently, relied heavily on sophisticated representations and carefully designed feature sets. Now with the rise of deep learning, for the first time in the history of NLP, the importance of such manual feature engineering has started to be challenged. Deep learning systems using very few handcrafted features can achieve state-of-the-art (or nearly state-of-the-art) performance on many tasks, such as syntactic parsing, machine translation, sentiment analysis, and language modelling. However, rather than letting deep learning replace linguistically informed approaches, in this dissertation I explore how linguistic knowledge can provide insights for building even better neural network models. I tackle the problem of transforming sentences into vectors by employing a hybrid approach of symbolic NLP and connectionist deep learning based on the principle of compositionality. In this approach, the role of symbolic NLP is to provide syntactic structures whereas composition functions are implemented (and trained) by connectionist deep learning.

All of the models I develop in this dissertation are variants of the Recursive neural network (RNN). The RNN takes a sentence, syntactic tree, and vector representations for the words in the sentence as input, and applies a neural network to recursively compute vector representations for all the phrases in the tree and the complete sentence. The RNN is a popular model because of its elegant definition and promising empirical results. However, it also has some serious limitations: (i) the composition functions it can learn are linguistically impoverished, (ii) it can only be used in a bottom-up fashion, and (iii) it is extremely sensitive to errors in the syntactic trees it is presented with. Starting with the classic RNN, I propose extensions along three different directions that solve each of these problems.

The first direction focuses on strengthening the composition functions. One way to do that is making use of syntactic information and contexts, as in Chapter 3. In that

chapter, I propose composition functions, which are also one-layer feed-forward neural networks, taking into account representations of syntactic labels (e.g. N, VP), context words, and head words. Another way is to replace one-layer neural networks by more advanced networks. In Chapter 6, based on empirical results which show that the Long short term memory (LSTM) architecture can capture long range dependencies and deal with the vanishing gradient problem more effectively than Recurrent neural networks, I introduce a novel variant of the LSTM, called Recursive-LSTM, that works on trees. Empirical results on an artificial task and on the Stanford Sentiment Treebank confirm that the proposed Recursive-LSTM model is superior to the classic RNN model in terms of accuracy. Furthermore, in Chapter 7, I demonstrate how a convolutional neural network can be used as a composition function.

The second direction to extend the classic RNN is to focus on how information flows in a parse tree. In traditional compositional semantics approaches, including the RNN model, information flows in a bottom-up manner, leading to a situation where there is no way for a node to be aware of its surrounding context. As a result, these approaches are not applicable to top-down processes such as several top-down generative parsing models, and to problems requiring contexts such as semantic role labelling. In Chapter 4, I propose a solution to this, namely the Inside-Outside Semantic framework, in which the key idea is to allow information to flow not only bottom-up but also top-down. In this way, we can recursively compute representations for the content and the context of the phrase that a node in a parse tree covers. The Inside-Outside RNN model, a neural-net-based instance of this framework, is shown to work well on several tasks, including unsupervised composition function learning from raw texts, supervised semantic role labelling, and dependency parsing (Chapter 5).

The third direction is dealing with the uncertainty of the correct parse. As a result of relying on the principle of compositionality, compositional semantics uses syntactic parse trees to guide composition, which in turn makes compositional semantics approaches vulnerable to the errors of automatic parsers. The problems here are that automatic parsers are not flawless, and that they are not aware of domains to which they are applied. To overcome this problem, in Chapter 7, I propose the Forest Convolutional Network model, which takes as input a forest of parse trees rather than a single tree as in traditional approaches. The key idea is that we should give the model several options and let it select (or combine) ones that best fit its need. Empirical results show that the model performs on par with state-of-the-art models on the Stanford Sentiment Treebank and on the TREC question dataset.

The dissertation thus proposes solutions to the main shortcomings of the RNN model. It provides all components for a completely neural implementation of a syntactic-semantic parser: the three ideas above essentially yield a neural inside-outside algorithm. This represents an approach to NLP that combines the best of two worlds: all the flexibility and learning power of deep learning without sacrificing the linguistic adequacy of earlier approaches in computational linguistics.

Samenvatting

Het leren van vector-representaties van zinnen
– de 'recursive deep learning'-aanpak

Systemen voor taalverwerking per computer waren tot voor kort grotendeels gebaseerd op complexe, symbolische representaties en, voor zover ze gebruik maken van machinaal leren, toch afhankelijk van met de hand geselecteerde lijstjes van kenmerken. Met de opkomst van 'deep learning' is het, voor het eerst in the geschiedenis van het vakgebied, mogelijk geworden om ook die kenmerk-selectie te gaan automatiseren. In de afgelopen jaren hebben we succesvolle deep learning-systemen zien verschijnen die nauwelijks of geen handmatige kenmerk-selectie behoeven en toch bij de best presterende systemen behoren op taken zoals automatisch ontleden, automatisch vertalen, sentiment-analyse en woordvoorspelling.

Die successen betekenen echter niet dat we alle taalkundig geïnformeerde benaderingen nu aan de kant moeten schuiven. In dit proefschrift exploreer ik op welke manier taalkundige kennis ingezet kan worden om nog betere neurale netwerk-modellen van taal te kunnen bouwen. Ik pak de uitdaging op om vector-representaties voor zinnen uit te rekenen op basis van een hybride symbolisch-connectionistische benadering, uitgaande van het zogeheten compositionaliteitsbeginsel. In mijn aanpak levert de symbolische traditie de syntactische structuur van zinnen, maar gebruik ik neurale netwerken om representaties van woorden, combinaties van woorden en zinnen te leren.

Alle modellen die ik uitwerk in dit proefschrift zijn varianten van het Recursive Neural Network (RNN). Een RNN neemt een zin, een syntactische boom en vector-representaties van de woorden in die zin als input. Vervolgens gebruikt het model een neuraal netwerk om recursief representaties uit te rekenen voor combinaties van woorden, beginnend bij de combinaties van woorden die volgens de syntactische boom een frase vormen, en eindigend met een representatie voor de hele zin. Het RNN is een populair model vanwege de elegante definitie en veelbelovende empirische resultaten. Het model heeft echter ook heel duidelijke beperkingen: (i) de compositie-functies die

het leert zijn taalkundig deficiënt; (ii) het model kan alleen in een bottom-up richting worden toegepast; (iii) het model is extreem gevoelig voor fouten in de aangeboden syntactische bomen. Met het standaard RNN-model als startpunt stel ik daarom een uitbreidingen voor in drie richtingen als oplossingen voor elk van deze drie problemen.

Het eerste type uitbreidingen betreft het verbeteren van de compositie-functies. Eén manier om dat te doen is om gebruik te maken van syntactische en context-informatie, zoals ik dat doe in hoofdstuk 3. De compositie-functies in dat hoofdstuk zijn nog steeds zogeheten 'one-layer feedforward'-netwerken, maar er is een apart netwerk voor iedere combinatie van syntactische categorieën en 'heads'. Een andere manier is om die eenvoudige netwerken te vervangen door complexere. In hoofdstuk 6 rapporteer ik resultaten waaruit blijkt dat het zogeheten Long Short Term Memory-netwerk (LSTM) effectiever omgaat met lange afstandsafhankelijkheden en het 'vanishing gradient'-probleem dan de veelgebruikte recurrente netwerken. Ik werk in dat hoofdstuk een nieuwe variant van het LSTM uit, het 'Recursive LSTM', dat werkt met syntactisch bomen. Empirische resultaten op een kuntmatige taak en op de Stanford Sentiment Treebank laten zien dat dit nieuwe model veel accurater is dan het standaard RNN. In hoofdstuk 7 laat ik tenslotte zien dat ook zogeheten convolutional neural networks succesvol gebruikt kunnen worden om de compositie-functie mee te implementeren.

Het tweede type uitbreidingen betreft de manier waarop informatie stroomt door een syntactische boom. In klassieke compositionele semantiek-benaderingen, waaronder ook de RNN, is die informatie-stroom strikt bottom-up, waardoor een knoop in zo'n boom geen toegang heeft tot informatie over de context van een zin. Zulke benaderingen zijn daarom moeilijk te combineren met technieken die top-down werken, zoals verschillende populaire statistische modellen voor automatisch ontleden, of technieken die gebruik maken van context-informatie, zoals populaire modellen voor semantische rolbepaling. In hoofdstuk 4 stel ik een oplossing voor voor deze problemen, onder de naam 'Inside-Outside Semantics framework', waar het centrale idee is dat informatie zowel bottom-up als top-down moet kunnen stromen. Ik stel voor om voor elke knoop in een syntactische boom twee representaties te berekenen (via recursieve definities): een 'content representation' voor het corresponderende deel van de zin die bottom-up wordt berekend, en een 'context representation' die top-down wordt bepaald. Ik laat zien, in hoofdstuk 5, dat een neurale netwerk-implementatie van dit idee heel goed werkt op een reeks van verschillende taken, inclusief 'unsupervised composition function learning', 'semantic role labeling' en 'dependency parsing'.

Het derde type uitbreidingen betreft de omgang met onzekerheid over de juiste syntactische ontleedboom. Ontleedbomen zijn een cruciaal onderdeel van alle modellen in deze dissertatie, omdat volgens het compositionaliteitsbeginsel de syntactische structuur bepaalt welke semantische composities worden uitgevoerd, en op welk moment. Dat maakt de aanpak gevoelig voor fouten in de ontleedbomen. Dergelijke fouten worden onvermijdelijk door automatische ontleedprogramma's geïntroduceerd, omdat die programma's binnen het domein waar ze voor zijn ontwikkeld al niet foutloos opereren, maar bovendien in veel gevallen buiten dat domein worden ingezet. Om dit probleem het hoofd te bieden stel ik in hoofdstuk 7 het 'Forest Convolutional Net-

work' voor, dat in plaats van een enkele ontleedboom een grote verzameling bomen, een zogeheten 'parse forest', als input krijgt. Het idee achter dit model is dus dat het model uit een variatie aan mogelijkheden de syntactisch structuur kiest (of samenstelt) die het beste past bij de waar het model voor wordt geoptimaliseerd. De empirische resultaten laten zien dat het resulterende model tot de best beschikbare modellen behoort op twee populaire taken: de 'Stanford Sentiment Treebank'-taak en de 'TREC vraag-classificatie'-taak.

In dit proefschrift beschrijf ik dus concrete oplossingen voor de belangrijkste tekortkomingen van het RNN-model. Daarmee bevat dit proefschrift alle ingrediënten voor een volledige neurale implementatie van een syntactisch-semantische parser: de drie beschreven uitbreidingen komen neer op een neurale versie van het 'inside-outside'-algoritme. De aanpak in dit proefschrift biedt daarmee het beste van twee werelden: de enorme flexibiliteit en zelflerende kracht van 'deep learning', zonder de taalkundige principes en uitdrukkingskracht van eerdere benaderingen in de computationele taalkunde op te geven.

Titles in the ILLC Dissertation Series:

ILLC DS-2009-01: **Jakub Szymanik**
Quantifiers in TIME and SPACE. Computational Complexity of Generalized Quantifiers in Natural Language

ILLC DS-2009-02: **Hartmut Fitz**
Neural Syntax

ILLC DS-2009-03: **Brian Thomas Semmes**
A Game for the Borel Functions

ILLC DS-2009-04: **Sara L. Uckelman**
Modalities in Medieval Logic

ILLC DS-2009-05: **Andreas Witzel**
Knowledge and Games: Theory and Implementation

ILLC DS-2009-06: **Chantal Bax**
Subjectivity after Wittgenstein. Wittgenstein's embodied and embedded subject and the debate about the death of man.

ILLC DS-2009-07: **Kata Balogh**
Theme with Variations. A Context-based Analysis of Focus

ILLC DS-2009-08: **Tomohiro Hoshi**
Epistemic Dynamics and Protocol Information

ILLC DS-2009-09: **Olivia Ladinig**
Temporal expectations and their violations

ILLC DS-2009-10: **Tikitu de Jager**
"Now that you mention it, I wonder…": Awareness, Attention, Assumption

ILLC DS-2009-11: **Michael Franke**
Signal to Act: Game Theory in Pragmatics

ILLC DS-2009-12: **Joel Uckelman**
More Than the Sum of Its Parts: Compact Preference Representation Over Combinatorial Domains

ILLC DS-2009-13: **Stefan Bold**
Cardinals as Ultrapowers. A Canonical Measure Analysis under the Axiom of Determinacy.

ILLC DS-2010-01: **Reut Tsarfaty**
Relational-Realizational Parsing

ILLC DS-2010-02: **Jonathan Zvesper**
Playing with Information

ILLC DS-2010-03: **Cédric Dégremont**
The Temporal Mind. Observations on the logic of belief change in interactive systems

ILLC DS-2010-04: **Daisuke Ikegami**
Games in Set Theory and Logic

ILLC DS-2010-05: **Jarmo Kontinen**
Coherence and Complexity in Fragments of Dependence Logic

ILLC DS-2010-06: **Yanjing Wang**
Epistemic Modelling and Protocol Dynamics

ILLC DS-2010-07: **Marc Staudacher**
Use theories of meaning between conventions and social norms

ILLC DS-2010-08: **Amélie Gheerbrant**
Fixed-Point Logics on Trees

ILLC DS-2010-09: **Gaëlle Fontaine**
Modal Fixpoint Logic: Some Model Theoretic Questions

ILLC DS-2010-10: **Jacob Vosmaer**
Logic, Algebra and Topology. Investigations into canonical extensions, duality theory and point-free topology.

ILLC DS-2010-11: **Nina Gierasimczuk**
Knowing One's Limits. Logical Analysis of Inductive Inference

ILLC DS-2010-12: **Martin Mose Bentzen**
Stit, Iit, and Deontic Logic for Action Types

ILLC DS-2011-01: **Wouter M. Koolen**
Combining Strategies Efficiently: High-Quality Decisions from Conflicting Advice

ILLC DS-2011-02: **Fernando Raymundo Velazquez-Quesada**
Small steps in dynamics of information

ILLC DS-2011-03: **Marijn Koolen**
The Meaning of Structure: the Value of Link Evidence for Information Retrieval

ILLC DS-2011-04: **Junte Zhang**
System Evaluation of Archival Description and Access

ILLC DS-2011-05: **Lauri Keskinen**
Characterizing All Models in Infinite Cardinalities

ILLC DS-2011-06: **Rianne Kaptein**
Effective Focused Retrieval by Exploiting Query Context and Document Structure

ILLC DS-2011-07: **Jop Briët**
Grothendieck Inequalities, Nonlocal Games and Optimization

ILLC DS-2011-08: **Stefan Minica**
Dynamic Logic of Questions

ILLC DS-2011-09: **Raul Andres Leal**
Modalities Through the Looking Glass: A study on coalgebraic modal logic and their applications

ILLC DS-2011-10: **Lena Kurzen**
Complexity in Interaction

ILLC DS-2011-11: **Gideon Borensztajn**
The neural basis of structure in language

ILLC DS-2012-01: **Federico Sangati**
Decomposing and Regenerating Syntactic Trees

ILLC DS-2012-02: **Markos Mylonakis**
Learning the Latent Structure of Translation

ILLC DS-2012-03: **Edgar José Andrade Lotero**
Models of Language: Towards a practice-based account of information in natural language

ILLC DS-2012-04: **Yurii Khomskii**
Regularity Properties and Definability in the Real Number Continuum: idealized forcing, polarized partitions, Hausdorff gaps and mad families in the projective hierarchy.

ILLC DS-2012-05: **David García Soriano**
Query-Efficient Computation in Property Testing and Learning Theory

ILLC DS-2012-06: **Dimitris Gakis**
Contextual Metaphilosophy - The Case of Wittgenstein

ILLC DS-2012-07: **Pietro Galliani**
The Dynamics of Imperfect Information

ILLC DS-2012-08: **Umberto Grandi**
Binary Aggregation with Integrity Constraints

ILLC DS-2012-09: **Wesley Halcrow Holliday**
Knowing What Follows: Epistemic Closure and Epistemic Logic

ILLC DS-2012-10: **Jeremy Meyers**
Locations, Bodies, and Sets: A model theoretic investigation into nominalistic mereologies

ILLC DS-2012-11: **Floor Sietsma**
Logics of Communication and Knowledge

ILLC DS-2012-12: **Joris Dormans**
Engineering emergence: applied theory for game design

ILLC DS-2013-01: **Simon Pauw**
Size Matters: Grounding Quantifiers in Spatial Perception

ILLC DS-2013-02: **Virginie Fiutek**
Playing with Knowledge and Belief

ILLC DS-2013-03: **Giannicola Scarpa**
Quantum entanglement in non-local games, graph parameters and zero-error information theory

ILLC DS-2014-01: **Machiel Keestra**
Sculpting the Space of Actions. Explaining Human Action by Integrating Intentions and Mechanisms

ILLC DS-2014-02: **Thomas Icard**
The Algorithmic Mind: A Study of Inference in Action

ILLC DS-2014-03: **Harald A. Bastiaanse**
Very, Many, Small, Penguins

ILLC DS-2014-04: **Ben Rodenhäuser**
A Matter of Trust: Dynamic Attitudes in Epistemic Logic

ILLC DS-2015-01: **María Inés Crespo**
Affecting Meaning. Subjectivity and evaluativity in gradable adjectives.

ILLC DS-2015-02: **Mathias Winther Madsen**
The Kid, the Clerk, and the Gambler - Critical Studies in Statistics and Cognitive Science

ILLC DS-2015-03: **Shengyang Zhong**
Orthogonality and Quantum Geometry: Towards a Relational Reconstruction of Quantum Theory

ILLC DS-2015-04: **Sumit Sourabh**
Correspondence and Canonicity in Non-Classical Logic

ILLC DS-2015-05: **Facundo Carreiro**
Fragments of Fixpoint Logics: Automata and Expressiveness

ILLC DS-2016-01: **Ivano A. Ciardelli**
Questions in Logic

ILLC DS-2016-02: **Zoé Christoff**
Dynamic Logics of Networks: Information Flow and the Spread of Opinion