RECURSIVE DEEP LEARNING
FOR NATURAL LANGUAGE PROCESSING
AND COMPUTER VISION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Richard Socher
August 2014

This dissertation is online at: http://purl.stanford.edu/xn618dd0392

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Christopher Manning, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Percy Liang

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Andrew Ng

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Christopher D. Manning) Principal Co-Advisor

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Andrew Y. Ng) Principal Co-Advisor

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Percy Liang)

Approved for the University Committee on Graduate Studies

Abstract

As the amount of unstructured text data that humanity produces overall and on the Internet grows, so does the need to intelligently process it and extract different types of knowledge from it. My research goal in this thesis is to develop learning models that can automatically induce representations of human language, in particular its structure and meaning in order to solve multiple higher level language tasks.

There has been great progress in delivering technologies in natural language processing such as extracting information, sentiment analysis or grammatical analysis. However, solutions are often based on different machine learning models. My goal is the development of general and scalable algorithms that can jointly solve such tasks and learn the necessary intermediate representations of the linguistic units involved. Furthermore, most standard approaches make strong simplifying language assumptions and require well designed feature representations. The models in this thesis address these two shortcomings. They provide effective and general representations for sentences without assuming word order independence. Furthermore, they provide state of the art performance with no, or few manually designed features.

The new model family introduced in this thesis is summarized under the term *Recursive Deep Learning*. The models in this family are variations and extensions of unsupervised and supervised recursive neural networks (RNNs) which generalize deep and feature learning ideas to hierarchical structures. The RNN models of this thesis obtain state of the art performance on paraphrase detection, sentiment analysis, relation classification, parsing, image-sentence mapping and knowledge base completion, among other tasks.

Chapter 2 is an introductory chapter that introduces general neural networks.

The main three chapters of the thesis explore three recursive deep learning modeling choices. The first modeling choice I investigate is the overall objective function that crucially guides what the RNNs need to capture. I explore unsupervised, supervised and semi-supervised learning for structure prediction (parsing), structured sentiment prediction and paraphrase detection.

The next chapter explores the recursive composition function which computes vectors for longer phrases based on the words in a phrase. The standard RNN composition function is based on a single neural network layer that takes as input two phrase or word vectors and uses the same set of weights at every node in the parse tree to compute higher order phrase vectors. This is not expressive enough to capture all types of compositions. Hence, I explored several variants of composition functions. The first variant represents every word and phrase in terms of both a meaning vector and an operator matrix. Afterwards, two alternatives are developed: The first conditions the composition function on the syntactic categories of the phrases being combined which improved the widely used Stanford parser. The most recent and expressive composition function is based on a new type of neural network layer and is called a recursive neural tensor network.

The third major dimension of exploration is the tree structure itself. Variants of tree structures are explored and assumed to be given to the RNN model as input. This allows the RNN model to focus solely on the semantic content of a sentence and the prediction task. In particular, I explore dependency trees as the underlying structure, which allows the final representation to focus on the main action (verb) of a sentence. This has been particularly effective for grounding semantics by mapping sentences into a joint sentence-image vector space. The model in the last section assumes the tree structures are the same for every input. This proves effective on the task of 3d object classification.

Acknowledgments

This dissertation would not have been possible without the support of many people. First and foremost, I would like to thank my two advisors and role models Chris Manning and Andrew Ng. You both provided me with a the perfect balance of guidance and freedom. Chris, you helped me see the pros and cons of so many decisions, small and large. I admire your ability to see the nuances in everything. Thank you also for reading countless drafts of (often last minute) papers and helping me understand the NLP community. Andrew, thanks to you I found and fell in love with deep learning. It had been my worry that I would have to spend a lot of time feature engineering in machine learning, but after my first deep learning project there was no going back. I also want to thank you for your perspective and helping me pursue and define projects with more impact. I am also thankful to Percy Liang for being on my committee and his helpful comments.

I also want to thank my many and amazing co-authors (in chronological order) Jia Deng, Wei Dong, Li-Jia Li, Kai Li, Li Fei-Fei, Sam J. Gershman, Adler Perotte, Per Sederberg, Ken A. Norman, and David M. Blei, Andrew Maas, Cliff Lin, Jeffrey Pennington, Eric Huang, Brody Huval, Bharath Bhat, Milind Ganjoo, Hamsa Sridhar, Osbert Bastani, Danqi Chen, Thang Luong, John Bauer, Will Zou, Daniel Cer, Alex Perelygin, Jean Wu, Jason Chuang, Milind Ganjoo, Quoc V. Le, Romain Paulus, Bryan McCann, Kai Sheng Tai, JiaJi Hu and Andrej Karpathy. It is due to the friendly and supportive environment in the Stanford NLP, machine learning group and the overall Stanford CS department that I was lucky enough to find so many great people to work with. I really enjoyed my collaborations with you. It is not only my co-authors who helped make my Stanford time more fun and productive, I

also want to thank Gabor Angeli, Angel Chang and Ngiam Jiquan for proof-reading a bunch of papers drafts and brainstorming. Also, thanks to Elliot English for showing me all the awesome bike spots around Stanford!

I also want to thank Yoshua Bengio for his support throughout. In many ways, he has shown the community and me the path for how to apply, develop and understand deep learning. I somehow also often ended up hanging out with the Montreal machine learning group at NIPS; they are an interesting, smart and fun bunch!

For two years I was supported by the Microsoft Research Fellowship for which I want to sincerely thank the people in the machine learning and NLP groups in Redmond. A particular shout-out goes to John Platt. I was amazed that he could give so much helpful and technical feedback, both in long conversations during my internship but also after just a 3 minute chat in the hallway at NIPS.

I wouldn't be where I am today without the amazing support, encouragement and love from my parents Karin and Martin Socher and my sister Kathi. It's the passion for exploration and adventure combined with determination and hard work that I learned from you. Those values are what led me through my PhD and let me have fun in the process. And speaking of love and support, thank you Eaming for our many wonderful years and always being on my side, even when a continent was between us.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

As the amount of unstructured text data that humanity produces overall and on the Internet grows, so does the need to intelligently process it and extract different types of knowledge from it. My research goal in this thesis is to develop learning models that can automatically induce representations of human language, in particular its structure and meaning in order to solve multiple higher level language tasks.

There has been great progress in delivering technologies in natural language processing (NLP) such as extracting information from big unstructured data on the web, sentiment analysis in social networks or grammatical analysis for essay grading. One of the goals of NLP is the development of general and scalable algorithms that can jointly solve these tasks and learn the necessary intermediate representations of the linguistic units involved. However, standard approaches towards this goal have two common shortcomings.

1. **Simplifying Language Assumptions**: In NLP and machine learning, we often develop an algorithm and then force the data into a format that is compatible with this algorithm. For instance, a common first step in text classification or clustering is to ignore word order and grammatical structure and represent texts in terms of unordered lists of words, so called *bag of words*. This leads to obvious problems when trying to understand a sentence. Take for instance the

two sentences: "Unlike the surreal Leon, this movie is weird but likeable." and "Unlike the surreal but likeable Leon, this movie is weird." The overall sentiment expressed in the first sentence is positive. My model learns that while the words compose a positive sentiment about Leon in the second sentence, the overall sentiment is negative, despite both sentences having exactly the same words. This is in contrast to the above mentioned *bag of words* approaches that cannot distinguish between the two sentences. Another common simplification for labeling words with, for example, their part of speech tag is to consider only the previous word's tag or a fixed sized neighborhood around each word. My models do not make these simplifying assumptions and are still tractable.

2. **Feature Representations**: While a lot of time is spent on models and inference, a well-known secret is that the performance of most learning systems depends crucially on the feature representations of the input. For instance, instead of relying only on word counts to classify a text, state of the art systems use part-of-speech tags, special labels for each location, person or organization (so called named entities), parse tree features or the relationship of words in a large taxonomy such as WordNet. Each of these features has taken a long time to develop and integrating them for each new task slows down both the development and runtime of the final algorithm.

The models in this thesis address these two shortcomings. They provide effective and general representations for sentences without assuming word order independence. Furthermore, they provide state of the art performance with no, or few manually designed features. Inspiration for these new models comes from combining ideas from the fields of natural language processing and *deep learning*. I will introduce the important basic concepts and ideas of deep learning in the second chapter. Generally, deep learning is a subfield of machine learning which tackles the second challenge by automatically learning feature representations from raw input. These representations can then be readily used for prediction tasks.

There has been great success using deep learning techniques in image classification (Krizhevsky et al., 2012) and speech recognition (Hinton et al., 2012). However, an

import aspect of language and the visual world that has not been accounted for in deep learning is the pervasiveness of recursive or hierarchical structure. This is why deep learning so far has not been able to tackle the first of the two main shortcomings. This thesis describes new deep models that extend the ideas of deep learning to structured inputs and outputs, thereby providing a solution to the first shortcoming mentioned above. In other words, while the methods implemented here are based on deep learning they extend general deep learning ideas beyond classifying fixed sized inputs and introduce recursion and computing representations for grammatical language structures.

The new model family introduced in this thesis is summarized under the term *Recursive Deep Learning*. The models in this family are variations and extensions of unsupervised and supervised recursive neural networks. These networks parse natural language. This enables them to find the grammatical structure of a sentence and align the neural network architecture accordingly. The recursion comes applying the same neural network at every node of the grammatical structure. Grammatical structures help, for instance, to accurately solve the so called *prepositional attachment problem* illustrated in the parse of Fig 1.1. In this example, the "with" phrase in "eating spaghetti with a spoon" specifies a way of eating whereas in "eating spaghetti with some pesto" specifies a dish. The recursive model captures that the difference is due to the semantic content of the word following the preposition "with." This content is captured in the distributional word and phrase representations. These representations capture that utensils are semantically similar or that pesto, sauce and tomatoes are all food related.

Recursive deep models do not only predict these linguistically plausible phrase structures but further learn how words compose the meaning of longer phrases inside such structures. They address the fundamental issue of learning feature vector representations for variable sized inputs without ignoring structure or word order. Discovering this structure helps us to characterize the units of a sentence or image and how they compose to form a meaningful whole. This is a prerequisite for a complete and plausible model of language. In addition, the models can learn *compositional semantics* often purely from training data without a manual description of

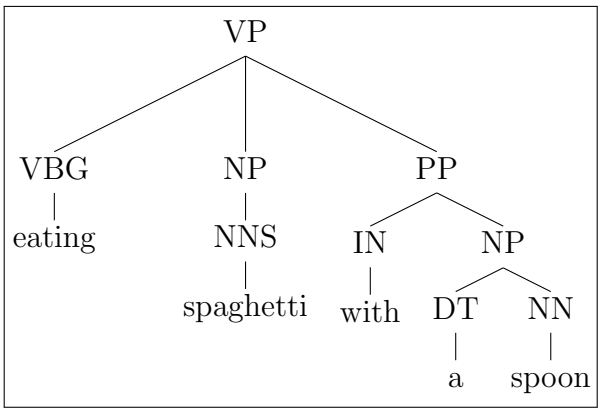


Figure 1.1: Example of how a recursive neural network model correctly identifies that "with a spoon" modifies the verb in this parse tree.

features that are important for a prediction task.

1.2 Contributions and Outline of This Thesis

The majority of deep learning work is focused on pure classification of fixed-sized flat inputs such as images. In contrast, the recursive deep models presented in this thesis can predict an underlying hierarchical structure, learn similarity spaces for linguistic units of any length and classify phrase labels and relations between inputs. This constitutes an important generalization of deep learning to structured prediction and makes these models suitable for natural language processing.

The syntactic rules of natural language are known to be recursive, with noun phrases containing relative clauses that themselves contain noun phrases, e.g., "the church which has nice windows." In Socher et al. (2011b), I introduced a max-margin, structure prediction framework based on Recursive Neural Networks (RNNs) for finding hierarchical structure in multiple modalities. *Recursion* in this case pertains to the idea that the same neural network is applied repeatedly on different components of a sentence. Since this model showed much promise for both language and image understanding, I decided to further investigate the space of recursive deep learning models. In this thesis, I explore model variations along three major axes in order to gain insights into hierarchical feature learning, building fast, practical, state of

the art NLP systems and *semantic compositionality*, the important quality of natural language that allows speakers to determine the meaning of a longer expression based on the meanings of its words and the rules used to combine them (Frege, 1892). The RNN models of this thesis obtain state of the art performance on paraphrase detection, sentiment analysis, relation classification, parsing, image-sentence mapping and knowledge base completion, among other tasks.

Chapter 2 is an introductory chapter that introduces general neural networks. It loosely follows a tutorial I gave together with Chris Manning and Yoshua Bengio at ACL 2012. The next three chapters outline the main RNN variations: objective functions, composition functions and tree structures. A final conclusion chapter distills the findings and discusses shortcomings, advantages and potential future directions.

Summary Chapter 3: Recursive Objective Functions

The first modeling choice I investigate is the overall objective function that crucially guides what the RNNs need to capture. I explore unsupervised learning of word and sentence vectors using reconstruction errors. Unsupervised deep learning models can learn to capture distributional information of single words (Huang et al., 2012) or use morphological analysis to describe rare or unseen words (Luong et al., 2013). Recursive reconstruction errors can be used to train compositional models to preserve information in sentence vectors, which is useful for paraphrase detection (Socher et al., 2011a). In contrast to these unsupervised functions, my parsing work uses a simple linear scoring function and sentiment and relation classification use softmax classifiers to predict a label for each node and phrase in the tree (Socher et al., 2011c, 2012b, 2013d). This chapter is based on the following papers (in that order): Socher et al. (2011b,c,a), each being the basis for one section.

Summary Chapter 4: Recursive Composition Functions

The composition function computes vectors for longer phrases based on the words in a phrase. The standard RNN composition function is based on a single neural network layer that takes as input two phrase or word vectors and uses the same set of weights

at every node in the parse tree to compute higher order phrase vectors. While this type of composition function obtained state of the art performance for paraphrase detection (Socher et al., 2011a) and sentiment classification (Socher et al., 2011c), it is not expressive enough to capture all types of compositions. Hence, I explored several variants of composition functions. The first variant represents every word and phrase in terms of both a meaning vector and an operator matrix (Socher et al., 2012b). Each word's matrix acts as a function that modifies the meaning of another word's vector, akin to the idea of lambda calculus functions. In this model, the composition function becomes completely input dependent. This is a very versatile functional formulation and improved the state of the art on the task of identifying relationships between nouns (such as message-topic or content-container). But it also introduced many parameters for each word. Hence, we developed two alternatives, the first conditions the composition function on the syntactic categories of the phrases being combined (Socher et al., 2013a). This improved the widely used Stanford parser and learned a soft version of head words. In other words, the model learned which words are semantically more important in the representation of a longer phrase. The most recent and expressive composition function is based on a new type of neural network layer and is called a recursive neural tensor network (Socher et al., 2013d). It allows both additive and mediated multiplicative interactions between vectors and is able to learn several important compositional sentiment effects in language such as negation and its scope and contrastive conjunctions like *but*. This chapter is based on the following papers (in that order): Socher et al. (2013a, 2012b, 2013d), each being the basis for one section.

Summary Chapter 5: Tree Structures

The third major dimension of exploration is the tree structure itself. I have worked on constituency parsing, whose goal it is to learn the correct grammatical analysis of a sentence and produce a tree structure (Socher et al., 2013a). Another approach allowed the actual task, such as sentiment prediction or reconstruction error, to determine the tree structure (Socher et al., 2011c). In my recent work, I assume the

tree structure is provided by a parser already. This allows the RNN model to focus solely on the semantic content of a sentence and the prediction task (Socher et al., 2013d). I have explored dependency trees as the underlying structure, which allows the final representation to focus on the main action (verb) of a sentence. This has been particularly effective for grounding semantics by mapping sentences into a joint sentence-image vector space (Socher et al., 2014). The model in the last section assumes the tree structures are the same for every input. This proves effective on the task of 3d object classification. This chapter is based on the following papers (in that order): Socher et al. (2014, 2012a), each being the basis for one section.

The next chapter will give the necessary mathematical background of neural networks and their training, as well as some motivation for why these methods are reasonable to explore.

Chapter 2

Deep Learning Background

Most current machine learning methods work well because of human-designed representations and inputs features. When machine learning is applied only to the input features it becomes merely about optimizing weights to make the best final prediction. Deep learning can be seen as putting back together representation learning with machine learning. It attempts to jointly learn good features, across multiple levels of increasing complexity and abstraction, and the final prediction.

In this chapter, I will review the reasons for the resurgence of deep, neural network based models, define the most basic form of a neural network, explain how deep learning methods represent single words. Simple neural nets and these word representation are then combined in single window based methods for word labeling tasks. I will end this chapter with a short comparison of optimization methods that are frequently used. The flow of this chapter loosely follows a tutorial I gave together with Chris Manning and Yoshua Bengio at ACL 2012.

2.1 Why Now? The Resurgence of Deep Learning

In this section, I outline four main reasons for the current resurgence of deep learning.

Learning Representations

Handcrafting features is time-consuming and features are often both over-specified and incomplete. Furthermore, work has to be done again for each modality (images, text, databases), task or even domain and language. If machine learning could learn features automatically, the entire learning process could be automated more easily and many more tasks could be solved. Deep learning provides one way of automated feature learning.

Distributed Representations

Many models in natural language processing, such as PCFGs (Manning and Schütze, 1999), are based on counts over words. This can hurt generalization performance when specific words during testing were not present in the training set. Another characterization of this problem is the so-called "curse of dimensionality." Because an index vector over a large vocabulary is very sparse, models can easily overfit to the training data. The classical solutions to this problem involve either the above mentioned manual feature engineering, or the usage of very simple target functions as in linear models. Deep learning models of language usually use distributed word vector representation instead of discrete word counts. I will describe a model to learn such word vectors in section 2.4. While there are many newer, faster models for learning word vectors (Collobert et al., 2011; Huang et al., 2012; Mikolov et al., 2013; Luong et al., 2013; Pennington et al., 2014) this model forms a good basis for understanding neural networks and can be used for other simple word classification tasks. The resulting vectors capture similarities between single words and make models more robust. They can be learned in an unsupervised way to capture distributional similarities and be fine-tuned in a supervised fashion. Sections 2.3 and 2.4 will describe them in more details.

Learning Multiple Levels of Representation

Deep learning models such as convolutional neural networks (LeCun et al., 1998) trained on images learn similar levels of representations as the human brain. The

first layer learns simple edge filters, the second layer captures primitive shapes and higher levels combine these to form objects. I do not intend to draw strong connections to neuroscience in this thesis but do draw motivation for multi-layer architectures from neuroscience research. I will show how both supervised, semi-supervised and unsupervised intermediate representations can be usefully employed in a variety of natural language processing tasks. In particular, I will show that just like humans can process sentences as compositions of words and phrases, so can recursive deep learning architectures process and compose meaningful representations through compositionality.

Recent Advances

Neural networks have been around for many decades (Rumelhart et al., 1986; Hinton, 1990). However, until 2006, deep, fully connected neural networks were commonly outperformed by shallow architectures that used feature engineering. In that year, however, Hinton and Salakhutdinov (2006) introduced a novel way to pre-train deep neural networks. The idea was to use restricted Boltzmann machines to initialize weights one layer at a time. This greedy procedure initialized the weights of the full neural network in basins of attraction which lead to better local optima (Erhan et al., 2010). Later, Vincent et al. (2008) showed that similar effects can be obtained by using autoencoders. These models try to train the very simple function $f(x) = x$. A closer look at this function reveals a common form of $f(x) = g^2(g^1(x)) = x$ where g^1 introduces an informational bottleneck which forces this function to learn useful bases to reconstruct the data with. While this started a new wave of enthusiasm about deep models it has now been superseded by large, purely supervised neural network models in computer vision (Krizhevsky et al., 2012).

In natural language processing three lines of work have created excitement in the community. First, a series of state of the art results in speech recognition have been obtained with deep architectures (Dahl et al., 2010). For an overview of deep speech processing see Hinton et al. (2012). Second, Collobert and Weston (2008) showed that a single neural network model can obtain state of the art results on multiple language tasks such as part-of-speech tagging and named entity recognition.

Since their architecture is very simple, we describe it in further detail and illustrate the backpropagation algorithm on it in section 2.4. Lastly, neural network based language models (Bengio et al., 2003; Mikolov and Zweig, 2012) have outperformed traditional counting based language models alone.

Three additional reasons have recently helped deep architectures obtain state of the art performance: large datasets, faster, parallel computers and a plethora of machine learning insights into sparsity, regularization and optimization. Because deep learning models learn from raw inputs and without manual feature engineering they require more data. In this time of "big data" and crowd sourcing, many researchers and institutions can easily and cheaply collect huge datasets which can be used to train deep models with many parameters. As we will see in the next section, neural networks require a large number of matrix multiplications which can easily be parallelized on current multi-core CPU and GPU computing architectures. In section 2.6, I will give some up-to-date and practical tips for optimizing neural network models.

It is on this fertile ground that my research has developed. The next section will describe the related basics of prior neural network work.

2.2 Neural Networks: Definitions and Basics

In this section, I will give a basic introduction to neural networks. Fig. 2.1 shows a single neuron which consists of inputs, an activation function and the output. Let the inputs be some n-dimensional vector $x \in \mathbb{R}^n$. The output is computed by the following function:

$$a = f(w^T x + b), \tag{2.1}$$

where f defines the activation function. This function is also called a nonlinearity and commonly used examples are the sigmoid function:

$$f(x) = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}}, \tag{2.2}$$

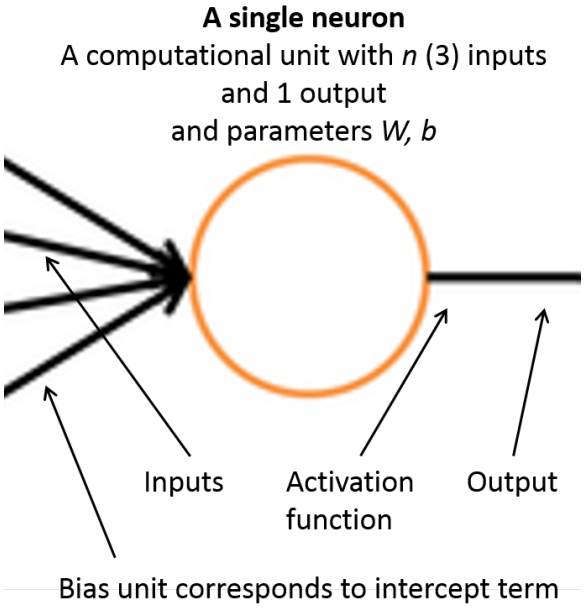


Figure 2.1: Definition of a single neuron with inputs, activation function and outputs.

or the hyperbolic tangent function:

$$f(x) = \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}. \tag{2.3}$$

The sigmoid activation function maps any real number to the $[0, 1]$ interval. With this unit, the activation can be interpreted as the probability for the "neural unit" parameterized by w and the bias b to be on. Despite the loss of a probabilistic interpretation, the tanh function is often preferred in practice due to better empirical performance (possibly due to having both negative and positive values inside the recursion). Both nonlinearities are shown in Fig. 2.2.

Various other recent nonlinearities exist such as the hard tanh (which is faster to compute) or rectified linear: $f(x) = \max(0, x)$ (which does not suffer from the vanishing gradient problem as badly). The choice of nonlinearity should largely by selected by cross-validation over a development set.

A neural network now stacks such single neurons both horizontally (next to each

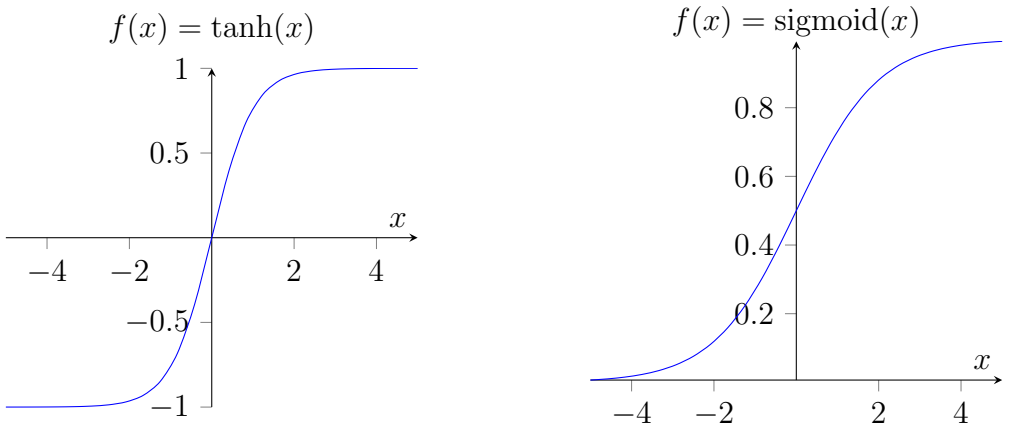


Figure 2.2: These are two commonly used nonlinearities. All RNN models in this thesis use the tanh function.

other) and vertically (on top of each other) and is then followed by a final output layer. For multiple units, I will describe this as follows: each neuron's activation a_i will be computed by one inner product with its parameters, followed by an addition with its bias: $a_i = f(W_{i\cdot}x + b_i)$, where each parameter vector $W_{i\cdot} \in \mathbb{R}^n$. We can disentangle the multiplication and the nonlinearity and write this in matrix notation for m many neurons stacked horizontally as:

$$z = Wx + b \tag{2.4}$$

$$a = f(z), \tag{2.5}$$

where $W \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ and the f is applied element-wise:

$$f(z) = f([z_1, z_2, \ldots, z_m]) = [f(z_1), f(z_2), \ldots, f(z_m)]. \tag{2.6}$$

The output of such a neural network layer can be seen as a transformation of the input that captures various interactions of the original inputs. A practitioner has now two options: (i) stack another layer on top of this one or (ii) directly compute an error based on this layer. When multiple layers are stacked on top of each other we will use a superscript to denote the various matrices that parameterize each layer's

connections:

$$\begin{aligned} z^{(2)} &= W^{(1)}x + b^{(1)} \\ a^{(2)} &= f(z^{(2)}) \\ z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\ a^{(3)} &= f(z^{(3)}) \\ &\dots \\ z^{(l+1)} &= W^{(l)}a^{(l)} + b^{(l)} \\ a^{(l+1)} &= f(z^{(l+1)}). \end{aligned} \tag{2.7}$$

In this equation we use the nomenclature of the inputs being the first layer. The last layer (here the lth layer) is then given to an output layer on which an error function is minimized. Standard examples are a softmax classifier (as defined below in Eq. 2.20) and training with the cross-entropy error, or a linear output layer with a least squares error.

Before describing the general training procedure for neural networks, called back-propagation or backprop, the next section introduces word vector representations which often constitute the inputs x of the above equations.

2.3 Word Vector Representations

The majority of rule-based and statistical language processing algorithms regard words as atomic symbols. This translates to a very sparse vector representation of the size of the vocabulary and with a single 1 at the index location of the current word. This, so-called "one-hot" or "one-on" representations has the problem that it does not capture any type of similarity between two words. So if a model sees "hotel" in a surrounding context it cannot use this information when it sees "motel" at the same location during test time.

Instead of this simplistic approach, Firth (1957) introduced the powerful idea of

representing each word by means of its neighbors.[1] This is one of the most successful ideas of modern statistical natural language processing (NLP) and has been used extensively in NLP, for instance Brown clustering (Brown et al., 1992) can be seen as an instance of this idea. Another example is the soft clustering induced by methods such as LSA (Deerwester et al., 1990) or latent Dirichlet allocation (Blei et al., 2003). The related idea of neural language models (Bengio et al., 2003) is to jointly learn such context-capturing vector representations of words and use these vectors to predict how likely a word is to occur given its context.

Collobert and Weston (2008) introduced a model to compute an embedding with a similar idea. This model will be described in detail in Sec. 2.4. In addition, Sec. 2.5 on backpropagation directly following it describes in detail how to train word vectors with this type of model. At a high level, the idea is to construct a neural network that outputs high scores for windows that occur in a large unlabeled corpus and low scores for windows where one word is replaced by a random word. When such a network is optimized via gradient ascent the derivatives backpropagate into a word embedding matrix $L \in \mathbb{R}^{n \times V}$, where V is the size of the vocabulary.

The main commonality between all unsupervised word vector learning approaches is that the resulting word vectors capture distributional semantics and co-occurrence statistics. The model by Collobert and Weston (2008) described in the next section achieves this via sampling and contrasting samples from the data distribution with corrupted samples. This formulation transforms an inherently unsupervised problem into a supervised problem in which unobserved or corrupted context windows become negative class samples. This idea has been used in numerous word vector learning models (Collobert and Weston, 2008; Bengio et al., 2009; Turian et al., 2010; Huang et al., 2012; Mikolov et al., 2013). When models are trained to distinguish true from corrupted windows they learn word representations that capture the good contexts of words. This process works but is slow since each window is computed separately. Most recently, Pennington et al. (2014) showed that predicting the word co-occurrence matrix (collected over the entire corpus) with a weighted regression loss achieves better performance on a range of evaluation tasks than previous negative sampling

[1] The famous quote is "You shall know a word by the company it keeps."

approaches. Despite this success, I will describe the window-based model below since its backpropagation style learning can be used for both unsupervised and supervised models.

The above description of the embedding matrix L assumes that its column vectors have been learned by a separate unsupervised model to capture co-occurrence statistics. Another commonly used alternative is to simply initialize L to small uniformly random numbers and then train this matrix as part of the model. The next section assumes such a randomly initialized L and then trains it. Subsequent chapters mention the different pre-training or random initialization choices. Generally, if the task itself has enough training data, a random L can suffice. For further details and evaluations of word embeddings, see Bengio et al. (2003); Collobert and Weston (2008); Bengio et al. (2009); Turian et al. (2010); Huang et al. (2012); Mikolov et al. (2013); Pennington et al. (2014).

In subsequent chapters, a sentence (or any n-gram) is represented as an ordered list of m words. Each word has an associated vocabulary index k into the embedding matrix which is used to retrieve the word's vector representation. Mathematically, this look-up operation can be seen as a simple projection layer where a binary vector $b \in \{0,1\}^V$ is used which is zero in all positions except at the kth index,

$$x_i = Lb_k \in \mathbb{R}^n. \tag{2.8}$$

The result is a sentence representation in terms of a list of vectors $(x_1, \ldots, x_m)$. In this thesis, I will use the terms *embedding*, *vector representation* and *feature representation* interchangeably.

Continuous word representations are better suited for neural network models than the binary number representations used in previous related models such as the recursive autoassociative memory (RAAM) (Pollack, 1990; Voegtlin and Dominey, 2005) or recurrent neural networks (Elman, 1991) since the sigmoid units are inherently continuous. Pollack (1990) circumvented this problem by having vocabularies with only a handful of words and by manually defining a threshold to binarize the resulting vectors.

2.4 Window-Based Neural Networks

In this section, I will describe in detail the simplest neural network model that has obtained great results on a variety of NLP tasks and was introduced by Collobert et al. (2011). It has commonalities with the neural network language model previously introduced by Bengio et al. (2003), can learn word vectors in an unsupervised way and can easily be extended to other tasks.

The main idea of the unsupervised window-based word vector learning model is that a word and its context is a positive training sample. A context with a random word in the center (or end position) constitutes a negative training example. This is similar to implicit negative evidence in contrastive estimation (Smith and Eisner, 2005). One way to formalize this idea is to assign a score to each window of k words, then replace the center word and train the model to identify which of the two windows is the true one. For instance, the model should correctly score:

$$\text{s(obtained great results on a)} > \text{s(obtained great Dresden on a)}, \tag{2.9}$$

where the center word "results" was replaced with the random word "Dresden." I define any observed window as x and a window corrupted with a random word as x_c.

This raises the question on how to compute the score. The answer is by assigning each word a vector representation, giving these as inputs to a neural network and have that neural network output a score. In the next paragraphs, I will describe each of these steps in detail.

Words are represented as vectors and retrieved from the L matrix as described in the previous section. For simplicity, we will assume a window size of 5. For each window, we retrieve the word vectors of words in that window and concatenate them to a $5n$-dimensional vector x. So, for the example score of Eq. 2.9, we would have:

$$x = [x_{\text{obtained}} x_{\text{great}} x_{\text{results}} x_{\text{on}} x_{\text{a}}]. \tag{2.10}$$

This vector is then given as input to a single neural network layer as described in section 2.2. Simplifying notation from the multilayer network, we have the following

equations to compute a single hidden layer and a score:

$$\begin{aligned} z &= Wx + b \\ a &= f(z) \\ s(x) &= U^T a, \end{aligned} \tag{2.11}$$

where I define the size of the hidden layer to be h and hence $W \in \mathbb{R}^{h \times 5n}$ and $U \in \mathbb{R}^h$. More compactly, this entire scoring function can be written as

$$s(x) = U^T f(Wx + b). \tag{2.12}$$

The above equation described the so-called feedforward process of the neural network. The final missing piece for training is the objective function that incorporates the scores of the observed and corrupted windows such that Eq. 2.9 becomes true. To this end, the following function J is minimized for each window we can sample from a large text corpus:

$$J = \max(0, 1 - s(x) + s(x_c)). \tag{2.13}$$

Since this is a continuous function, we can easily use stochastic gradient descent methods for training. But first, we define the algorithm to efficiently compute the gradients in the next section.

2.5 Error Backpropagation

The backpropagation algorithm (Bryson et al., 1963; Werbos, 1974; Rumelhart et al., 1986) follows the inverse direction of the feedforward process in order to compute the gradients of all the network parameters. It is a way to efficiently re-use parts of the gradient computations that are the same. These similar parts become apparent when applying the chain rule for computing derivatives of the composition of multiple functions or layers. To illustrate this, we will derive the backpropagation algorithm

for the network described in Eq. 2.12.

Assuming that for the current window $1 - s(x) + s(x_c) > 0$, I will describe the derivative for the score $s(x)$. At the very top, the first gradient is for the vector U and is simply:

$$\begin{aligned} \frac{\partial s}{\partial U} &= \frac{\partial}{\partial U} U^T a \\ \frac{\partial s}{\partial U} &= a, \end{aligned} \tag{2.14}$$

where a is defined as in Eq. 2.11.

Next, let us consider the gradients for W, the weights that govern the layer below the scoring layer:

$$\frac{\partial s}{\partial W} = \frac{\partial}{\partial W} U^T a = \frac{\partial}{\partial W} U^T f(z) = \frac{\partial}{\partial W} U^T f(Wx + b). \tag{2.15}$$

For ease of exposition, let us consider a single weight W_{ij}. This only appears in the computation of a_i, the ith element of a. Hence, instead of looking at the entire inner product, we can only consider:

$$\frac{\partial}{\partial W_{ij}} U^T a \quad \rightarrow \quad \frac{\partial}{\partial W_{ij}} U_i a_i.$$

That partial derivative can easily be computed by using the chain rule:

$$\begin{aligned} U_i \frac{\partial}{\partial W_{ij}} a_i &= U_i \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i \frac{\partial f(z_i)}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i f'(z_i) \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i f'(z_i) \frac{\partial W_{i.} x + b_i}{\partial W_{ij}} \\ &= U_i f'(z_i) \frac{\partial}{\partial W_{ij}} \sum_k W_{ik} x_k \\ &= \underbrace{U_i f'(z_i)}_{\delta_i} x_j \\ &= \delta_i \quad x_j, \end{aligned}$$

where δ_i can be seen as the local error signal and x_j as the local input signal. In summary, a single partial derivative is:

$$\begin{aligned} \frac{\partial J}{\partial W_{ij}} &= \underbrace{U_i f'(z_i)}_{\delta_i} x_j \\ &= \delta_i \quad x_j. \end{aligned} \tag{2.16}$$

Since all combination of $i = 1, \ldots, h$ cross product with $j = 1, \ldots, n$ are needed, the outer product of the vectors δ and x gives the full gradient of W:

$$\frac{\partial J}{\partial W} = \delta x^T,$$

where $\delta \in \mathbb{R}^{h \times 1}$ is the "responsibility" coming from each activation in a. The bias derivatives are computed similarly:

$$\begin{aligned} & U_i \frac{\partial}{\partial b_i} a_i \\ = \; & U_i f'(z_i) \frac{\partial W_{i.} x + b_i}{\partial b_i} \\ = \; & \delta_i \end{aligned} \tag{2.17}$$

So far, backpropagation was simply taking derivatives and using the chain rule. The only remaining trick now is that one can re-use derivatives computed for higher layers when computing derivatives for lower layers. In our example, the main goal of this was was to train the word vectors. They constitute the last layer. For ease of exposition, let us consider a single element of the full x vector. Depending on which index of x is involved, this node will be part of a different word vector. Generally, we again take derivatives of the score with respect to this element, say x_j. Unlike for W, we cannot simply take into consideration a single a_i because each x_j is connected to all the hidden neurons in the above activations a and hence x_j influences the overall score $s(x)$ through all a_j's. Hence, we start with the sum and and again apply the chain rule:

$$\begin{aligned}
\frac{\partial s}{\partial x_j} &= \sum_{i=1}^{2} \frac{\partial s}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
&= \sum_{i=1}^{2} \frac{\partial U^T a}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
&= \sum_{i=1}^{2} U_i \frac{\partial f(W_{i\cdot} x + b)}{\partial x_j} \\
&= \sum_{i=1}^{2} \underbrace{U_i f'(W_{i\cdot} x + b)}_{} \frac{\partial W_{i\cdot} x}{\partial x_j} \\
&= \sum_{i=1}^{2} \delta_i W_{ij} \\
&= \delta^T W_{\cdot j}
\end{aligned} \tag{2.18}$$

Notice the underbrace that highlights part of the equation that is identical to what we needed for W and b derivatives? This is the main idea of the backpropagation algorithm: To re-use these so-called *delta* messages or error signals.

With the above equations one can put together the full gradients for all parameters in the objective function of Eq. 2.13. In particular, the derivatives for the input word vector layer can be written as

$$\frac{\partial s}{\partial x} = W^T \delta. \tag{2.19}$$

Minimizing this objective function results in very useful word vectors that capture syntactic and semantic similarities between words. There are in fact other even simpler (and faster) models to compute word vectors that get rid of the hidden layer entirely (Mikolov et al., 2013; Pennington et al., 2014). However, the above neural network architecture is still very intriguing because Collobert et al. (2011) showed that one can replace the last scoring layer with a standard softmax classifier[2] and classify the center word into different categories such as part of speech tags or named entity tags. The probability for each such class can be computed simply by replacing the inner product $U^T a$ with a softmax classifier:

$$p(c|x) = \frac{\exp(S_{c\cdot}a)}{\sum_{c'} \exp(S_{c'\cdot}a)}, \tag{2.20}$$

where $S \in \mathbb{R}^{C \times h}$ has the weights of the classification.

2.6 Optimization and Subgradients

The objective function above and in various other models in this thesis is not differentiable due to the hinge loss. Therefore, I generalize gradient ascent via the subgradient method (Ratliff et al., 2007) which computes a gradient-like direction. Let $\theta \in \mathbb{R}^{M \times 1}$ be the vector of all model parameters (in the above example these are the vectorized L, W, U). Normal stochastic subgradient descent methods can be used successfully in most models below.

However, an optimization procedure that yields even better accuracy and faster convergence is the diagonal variant of AdaGrad (Duchi et al., 2011). For parameter updates, let $g_\tau \in \mathbb{R}^{M \times 1}$ be the subgradient at time step τ and $G_t = \sum_{\tau=1}^{t} g_\tau g_\tau^T$. The parameter update at time step t then becomes:

$$\theta_t = \theta_{t-1} - \alpha \left(\operatorname{diag}(G_t)\right)^{-1/2} g_t, \tag{2.21}$$

where α is the global learning rate. Since I use the diagonal of G_t, only M values have

[2] Also called logistic regression or MaxEnt classifier.

to be stored and the update becomes fast to compute: At time step t, the update for the i'th parameter $\theta_{t,i}$ is:

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t,i}. \tag{2.22}$$

Hence, the learning rate is adapting differently for each parameter and rare parameters get larger updates than frequently occurring parameters. This is helpful in this model since some word vectors appear in only a few windows. This procedure finds much better optima in various experiments of this thesis, and converges more quickly than L-BFGS which I used in earlier papers (Socher et al., 2011a).

Furthermore, computing one example's gradient at a time and making an immediate update step is not very efficient. Instead, multiple cores of a CPU or GPU can be used to compute the gradients of multiple examples in parallel and then make one larger update step with their sum or average. This is called training with minibatches, which are usually around 100 in the models below.

The next chapter introduces recursive neural networks and investigates the first axis of variation: the main objective function.

Chapter 3

Recursive Objective Functions

In this chapter I will motivate and introduce recursive neural networks inside a max-margin structure prediction framework. While this thesis focuses largely on natural language tasks, the next section shows that the model is general and applicable to language and scene parsing. After the max-margin framework I will explore two additional objective functions: unsupervised and semi-supervised reconstruction errors.

3.1 Max-Margin Structure Prediction with Recursive Neural Networks

Recursive structure is commonly found in different modalities, as shown in Fig. 3.1. As mentioned in the introduction The syntactic rules of natural language are known to be recursive, with noun phrases containing relative clauses that themselves contain noun phrases, e.g., *... the church which has nice windows ...*. Similarly, one finds nested hierarchical structuring in scene images that capture both part-of and proximity relationships. For instance, cars are often on top of street regions. A large car region can be recursively split into smaller car regions depicting parts such as tires and windows and these parts can occur in other contexts such as beneath airplanes or in houses. I show that recovering this structure helps in understanding and classifying scene images. In this section, I introduce recursive neural networks (RNNs)

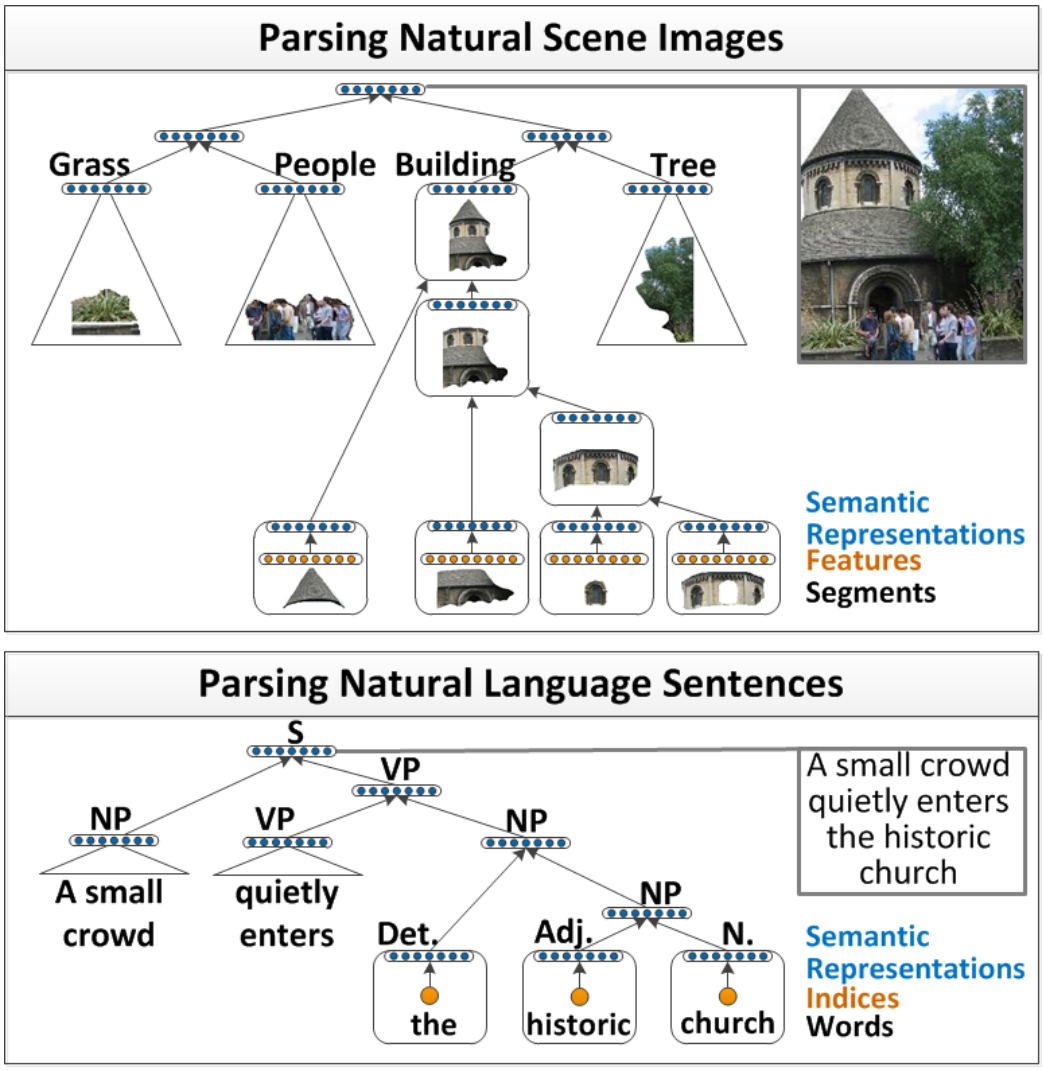


Figure 3.1: Illustration of a recursive neural network architecture which parses images and natural language sentences. Segment features and word indices (orange) are first mapped into semantic feature space (blue). Then they are recursively merged by the same neural network until they represent the entire image or sentence. Both mappings and mergings are learned.

for predicting recursive structure in multiple modalities. I primarily focus on scene understanding, a central task in computer vision often subdivided into segmentation, annotation and classification of scene images. I show that my algorithm is a general tool for predicting tree structures by also using it to parse natural language sentences.

Fig. 3.1 outlines my approach for both modalities. Images are oversegmented into small regions which often represent parts of objects or background. From these regions I extract vision features and then map these features into a "semantic" space using a neural network. Using these semantic region representations as input, my RNN computes (i) a score that is higher when neighboring regions should be merged into a larger region, (ii) a new semantic feature representation for this larger region, and (iii) its class label. Class labels in images are visual object categories such as *building* or *street*. The model is trained so that the score is high when neighboring regions have the same class label. After regions with the same object label are merged, neighboring objects are merged to form the full scene image. These merging decisions implicitly define a tree structure in which each node has associated with it the RNN outputs (i)-(iii), and higher nodes represent increasingly larger elements of the image.

The same algorithm is used to parse natural language sentences. Again, words are first mapped into a semantic space and then they are merged into phrases in a syntactically and semantically meaningful order. The RNN computes the same three outputs and attaches them to each node in the parse tree. The class labels are phrase types such as noun phrase (NP) or verb phrase (VP).

Contributions. This is the first deep learning method to achieve state-of-the-art results on segmentation and annotation of complex scenes. My recursive neural network architecture predicts hierarchical tree structures for scene images and outperforms other methods that are based on conditional random fields or combinations of other methods. For scene classification, my learned features outperform state of the art methods such as Gist descriptors (Oliva and Torralba, 2001a). Furthermore, my algorithm is general in nature and can also parse natural language sentences obtaining competitive performance on length 15 sentences of the Wall Street Journal dataset. Code for the RNN model is available at `www.socher.org`.

3.1.1 Mapping Words and Image Segments into Semantic Space

This section contains an explanation of the inputs used to describe scene images and natural language sentences (segments and words, respectively) and how they are mapped into the semantic space in which the RNN operates.

Word vectors are processed as described in Sec. 2.3. For image segments I closely follow the procedure described in Gould et al. (2009) to compute image features. First, I oversegment an image x into superpixels (also called segments) using the public implementation[1] of the algorithm from Comaniciu and Meer (2002). Instead of computing multiple oversegmentations, I only choose one set of parameters. In my dataset, this results in an average of 78 segments per image. I compute 119 features for the segments as described in Sec. 3.1 of Gould et al. (2009). These features include color and texture features (Shotton et al., 2006), boosted pixel classifier scores (trained on the labeled training data), as well as appearance and shape features.

Next, I use a simple neural network layer to map these features into the "semantic" n-dimensional space in which the RNN operates. Let F_i be the features described above for each segment $i = 1, \ldots, N_{segs}$ in an image. I then compute the representation:

$$a_i \quad = \quad f(W^{sem} F_i + b^{sem}), \tag{3.1}$$

where $W^{sem} \in \mathbb{R}^{n \times 119}$ is the matrix of parameters I want to learn, b^{sem} is the bias and f is applied element-wise and can be any sigmoid-like function. In my vision experiments, I use the original sigmoid function $f(x) = 1/(1 + e^{-x})$.

3.1.2 Recursive Neural Networks for Structure Prediction

In my discriminative parsing architecture, the goal is to learn a function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ is the set of all possible binary parse trees. An input x consists of two parts: (i) A set of activation vectors $\{a_1, \ldots, a_{N_{segs}}\}$, which represent input elements such

[1] `http://coewww.rutgers.edu/riul/research/code/EDISON`

as image segments or words of a sentence. (ii) A symmetric adjacency matrix A, where $A(i, j) = 1$, if segment i neighbors j. This matrix defines which elements can be merged. For sentences, this matrix has a special form with 1's only on the first diagonal below and above the main diagonal.

I denote the set of all possible trees that can be constructed from an input x as $A(x)$. When training the visual parser, I have labels l for all segments. Using these labels, I can define an equivalence set of correct trees $Y(x, l)$. A visual tree is correct, if all adjacent segments that belong to the same class are merged into one super segment before merges occur with super segments of different classes. This equivalence class over trees is oblivious to how object parts are internally merged or how complete, neighboring objects are merged into the full scene image. For training the language parser, the set of correct trees only has one element, the annotated ground truth tree: $Y(x) = \{y\}$. Fig. 3.2 illustrates these ideas.

Max-Margin Estimation

Similar to Taskar et al. (2004), I define a structured margin loss Δ. For the visual parser, the function $\Delta(x, l, \hat{y})$ computes the margin for proposing a parse $\hat{y}$ for input x with labels l. The loss increases when a segment merges with another one of a different label before merging with all its neighbors of the same label. I can formulate this by checking whether the subtree $subTree(d)$ underneath a nonterminal node d in $\hat{y}$ appears in any of the ground truth trees of $Y(x, l)$:

$$\Delta(x, l, \hat{y}) = \sum_{d \in N(\hat{y})} \kappa \mathbf{1}\{subTree(d) \notin Y(x, l)\}, \tag{3.2}$$

where $N(\hat{y})$ is the set of non-terminal nodes and κ is a parameter. The loss of the language parser is the sum over incorrect spans in the tree, see Manning and Schütze (1999). Given the training set, I search for a function f with small expected loss on unseen inputs. I consider the following functions:

$$f_\theta(x) = \arg \max_{\hat{y} \in A(x)} \mathrm{s}(\mathrm{RNN}(\theta, x, \hat{y})), \tag{3.3}$$

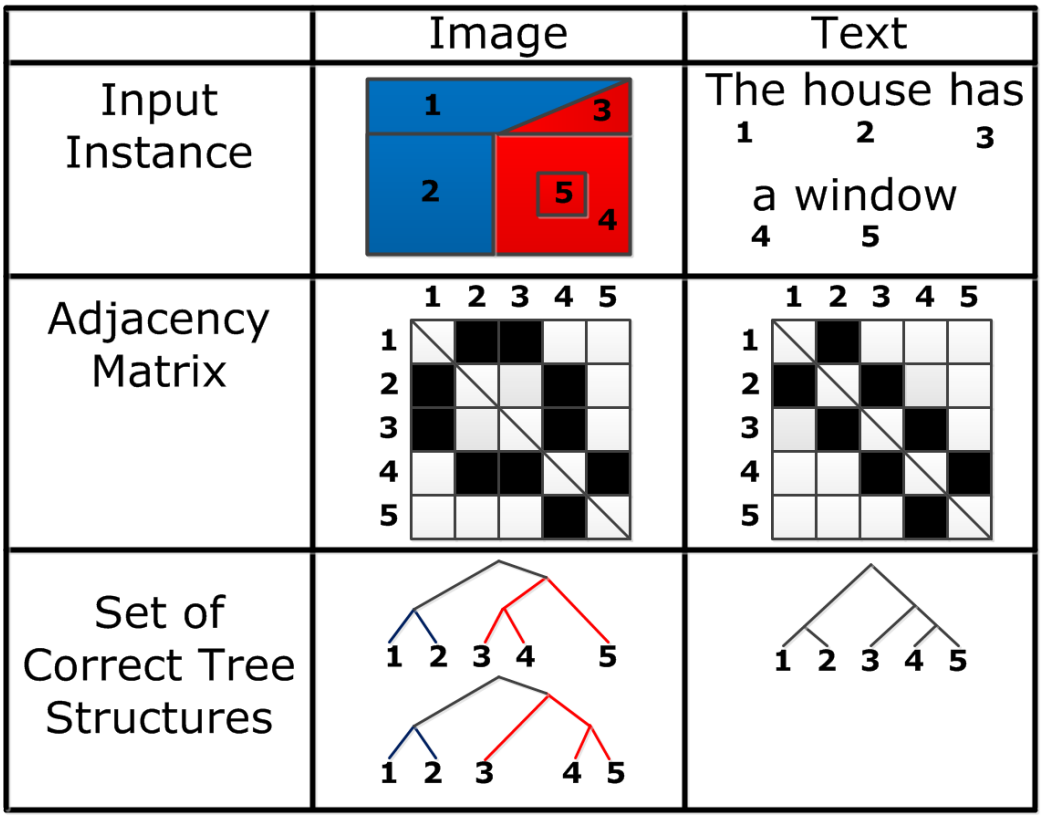


Figure 3.2: Illustration of the RNN training inputs: An adjacency matrix of image segments or words. A training image (red and blue are differently labeled regions) defines a set of correct trees which is oblivious to the order in which segments with the same label are merged. See text for details.

where θ are all the parameters needed to compute a score s with an RNN. The score of a tree y is high if the algorithm is confident that the structure of the tree is correct. Tree scoring with RNNs will be explained in detail below. In the max-margin estimation framework (Taskar et al., 2004; Ratliff et al., 2007), I want to ensure that the highest scoring tree is in the set of correct trees: $f_\theta(x_i) \in Y(x_i, l_i)$ for all training instances $i = 1, \ldots, n$: (x_i, l_i, Y_i). Furthermore, I want the score of the highest scoring correct tree y_i to be larger up to a margin defined by the loss Δ. $\forall i, \hat{y} \in A(x_i)$:

$$\mathrm{s}(\mathrm{RNN}(\theta, x_i, y_i)) \geq \mathrm{s}(\mathrm{RNN}(\theta, x_i, \hat{y})) + \Delta(x_i, l_i, \hat{y}).$$

These desiderata lead us to the following regularized risk function:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} r_i(\theta) + \frac{\lambda}{2}||\theta||^2, \quad \text{where} \tag{3.4}$$

$$r_i(\theta) = \max_{\hat{y}\in A(x_i)} \Big(s(\text{RNN}(\theta, x_i, \hat{y})) + \Delta(x_i, l_i, \hat{y})\Big) - \max_{y_i \in Y(x_i, l_i)} \Big(s(\text{RNN}(\theta, x_i, y_i))\Big) \tag{3.5}$$

Minimizing this objective maximizes the correct tree's score and minimizes (up to a margin) the score of the highest scoring but incorrect tree.

Now that I defined the general learning framework, I will describe in detail how I predict parse trees and compute their scores with RNNs.

Greedy RNNs

I can now describe the RNN model that uses the activation vectors and adjacency matrix ((i) and (ii) defined above) as inputs. There are more than exponentially many possible parse trees and no efficient dynamic programming algorithms for my RNN setting. Therefore, I find a greedy approximation. I start by explaining the feed-forward process on a test input.

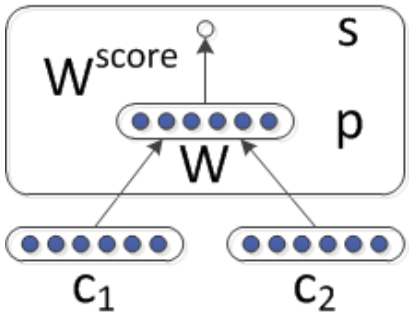
Using the adjacency matrix A, the algorithm finds the pairs of neighboring segments and adds their activations to a set of potential child node pairs:

$$C = \{[a_i, a_j] : \text{A(i,j)=1}\}. \tag{3.6}$$

In the small toy image of Fig. 3.2, I would have the following pairs: $\{[a_1, a_2]$, $[a_1, a_3]$, $[a_2, a_1]$, $[a_2, a_4]$, $[a_3, a_1]$, $[a_3, a_4]$, $[a_4, a_2]$, $[a_4, a_3]$, $[a_4, a_5]$, $[a_5, a_4]\}$. Each pair of activations is concatenated and given as input to a neural network. The network computes the potential parent representation for these possible child nodes:

$$p_{(i,j)} = f(W[c_i; c_j] + b) = f\left(W \begin{bmatrix} c_i \\ c_j \end{bmatrix}\right), \tag{3.7}$$

where the first equality is written in Matlab notation and the second one is a different notation for the vector concatenation. With this representation I can compute a local



$$\begin{aligned} s &= W^{score} p \qquad (3.9) \\ p &= f(W[c_1; c_2] + b) \end{aligned}$$

Figure 3.3: One recursive neural network which is replicated for each pair of possible input vectors. This network is different to the original RNN formulation in that it predicts a score for being a correct merging decision.

score using a simple inner product with a row vector $W^{score} \in \mathbb{R}^{1 \times n}$:

$$s_{(i,j)} = W^{score} p_{(i,j)}. \tag{3.8}$$

The network performing these functions is illustrated in Fig. 3.3. Training will aim to increase scores of good segments pairs (with the same label) and decrease scores of pairs with different labels (unless no more good pairs are left).

After computing the scores for all pairs of neighboring segments, the algorithm selects the pair which received the highest score. Let the score s_{ij} be the highest score, I then (i) Remove $[a_i, a_j]$ from C, as well as all other pairs with either a_i or a_j in them. (ii) Update the adjacency matrix with a new row and column that reflects that the new segment has the neighbors of both child segments. (iii) Add potential child pairs to C:

$$\begin{aligned} C &= C - \{[a_i, a_j]\} - \{[a_j, a_i]\} \qquad (3.10) \\ C &= C \cup \{[p_{(i,j)}, a_k] : a_k \text{ has boundary with } i \text{ or } j\} \end{aligned}$$

In the case of the image in Fig. 3.2, if I merge $[a_4, a_5]$, then $C = \{[a_1, a_2]$, $[a_1, a_3]$, $[a_2, a_1]$, $[a_2, p_{4,5}]$, $[a_3, a_1]$, $[a_3, p_{(4,5)}]$, $[p_{(4,5)}, a_2]$, $[p_{(4,5)}, a_3]\}$.

The new potential parents and corresponding scores of new potential child pairs

are computed *with the same neural network* of Eq. 3.7. For instance, I compute, $p_{2,(4,5)} = f(W[a_2, p_{4,5}] + b)$, $p_{3,(4,5)} = f(W[a_3, p_{4,5}] + b)$, etc.

The process repeats (treating the new $p_{i,j}$ just like any other segment) until all pairs are merged and only one parent activation is left in the set C. This activation then represents the entire image. Hence, the *same* network (with parameters $W, b, W^{score}, W^{label}$) is *recursively* applied until all vector pairs are collapsed. The tree is then recovered by unfolding the collapsing decisions and making each parent a node down to the original segments which are the leaf nodes of the tree. The final score that I need for structure prediction is simply:

$$s(\text{RNN}(\theta, x_i, \hat{y})) = \sum\nolimits_{d \in N(\hat{y})} s_d. \tag{3.11}$$

To finish the example, assume the next highest score was $s_{(4,5),3}$, so I merge the $(4, 5)$ super segment with segment 3, so

$$C = \{[a_1, a_2], [a_1, p_{(45),3}], [a_2, a_1], [a_2, p_{(45),3}], [p_{(4,5),3}, a_1], [p_{(4,5),3}, a_2]\}. \tag{3.12}$$

If I then merge segments $1, 2$, $C = \{[p_{1,2}, p_{(45),3}], [p_{(4,5),3}, p_{1,2}]\}$, leaving us only with the last choice of merging the differently labeled super segments.

Category Classifiers in the Tree

One of the main advantages of my approach is that each node of the tree built by the RNN has associated with it a distributed feature representation (the parent vector p). I can leverage this representation by adding to each RNN parent node (after removing the scoring layer) a simple softmax layer to predict class labels, such as visual or syntactic categories:

$$label_p = softmax(W^{label} p). \tag{3.13}$$

When minimizing the cross-entropy error of this softmax layer, the error will back-propagate and influence both the RNN parameters and the word representations.

Improvements for Language Parsing

Since in a sentence each word only has 2 neighbors, less-greedy search algorithms such as a bottom-up beam search can be used. In my case, beam search fills in elements of the chart in a similar fashion as the CKY algorithm. However, unlike standard CNF grammars, in my grammar each constituent is represented by a continuous feature vector and not just a discrete category. Hence I cannot prune based on category equality. I could keep the k-best subtrees in each cell but initial tests showed no improvement over just keeping the single best constituent in each cell.

Furthermore, since there is only a single correct tree the second maximization in the objective of Eq. 3.4 can be dropped. For further details on this special case, see Socher and Fei-Fei (2010).

3.1.3 Learning

My objective J of Eq. 3.4 is not differentiable due to the hinge loss. As described in Sec. 2.6, I use the subgradient method (Ratliff et al., 2007) which computes a gradient-like direction called the subgradient. Let

$$\theta = (W^{sem}, W, W^{score}, W^{label})$$

be the set of my model parameters,[2] then the gradient becomes:

$$\frac{\partial J}{\partial \theta} = \frac{1}{n} \sum_i \frac{\partial s(\hat{y}_i)}{\partial \theta} - \frac{\partial s(y_i)}{\partial \theta} + \lambda \theta, \tag{3.14}$$

where $s(\hat{y}_i) = s(\text{RNN}(\theta, x_i, \hat{y}_{max(A(x_i))}))$ and $s(y_i) = s(\text{RNN}(\theta, x_i, y_{max(Y(x_i,l_i))}))$. In order to compute Eq. 3.14 I calculate the derivatives by using backpropagation through structure introduced in the next subsection.

[2]In the case of natural language parsing, W^{sem} is replaced by the look-up table L.

3.1.4 Backpropagation Through Structure

The purpose of this section is to give an intuition for backpropagation through structure. It is a modification to the backpropagation algorithm due to the tree structure of recursive neural networks (Goller and Küchler, 1996). There are two main differences resulting from the tree structure:

1. Splitting derivatives at each node.

During forward propagation, the parent is computed using the two children:

$$p = f\left(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}\right). \tag{3.15}$$

Hence, the error message δ that is sent down from the parent node needs to be split with respect to each of them. Assume, for instance, that we classify the parent node in Eq. 3.15 with a softmax classifier as introduced in Eq. 2.20 and in matrix form just above in Eq. 3.13. Hence we will minimize the cross-entropy error and so the standard softmax error message $\delta^p \in \mathbb{R}^{n\times 1}$ at node vector p becomes

$$\delta^p = \left((W^{label})^T(label^p - target^p)\right) \otimes f'(p),$$

where $\otimes$ is the Hadamard product between the two vectors and f' is the element-wise derivative of f. Now, using a similar derivation as in Sec. 2.5, we can see that the error message passed down from vector p to its children is

$$\delta^{p,down} = \left(W^T\delta^p\right) \otimes f'\left(\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}\right).$$

The children of p then each take their half of this error message vector. So the error message of the left child is

$$\delta^{c_1} = \delta^{p,down}[1:n],$$

and

$$\delta^{c_2} = \delta^{p,down}[n+1:2n],$$

where $\delta^{p,down}[d+1:2d]$ indicates that c_2 is the right child of p and hence takes the 2nd half of the error. If each child had its own softmax prediction, then one would add their error message for the complete δ. For the right child the full message would be:

$$\delta^{c_2,combined} = \delta^{c_2,softmax} + \delta^{p,down}[d+1:2d].$$

2. Summing derivatives of all nodes in the tree.

Due to the recursion, the casual reader may assume that derivatives will be very complicated. However, a closer look shows a simplifying equality: Assuming that the W matrices at each node are different and then summing up the derivatives of these different W's turns out to be the same as taking derivatives with respect to the same W inside the recursion. To illustrate this, consider the the following derivative of a minimally recursive neural network (with a single weight):

$$\begin{aligned} & \frac{\partial}{\partial W} f(W(f(Wx)) \\ = \; & f'(W(f(Wx)) \left(\left(\frac{\partial}{\partial W} W \right) f(Wx) + W \frac{\partial}{\partial W} f(Wx) \right) \\ = \; & f'(W(f(Wx)) \left(f(Wx) + W f'(Wx) x \right). \end{aligned}$$

It is easily shown that if we take separate derivatives of each occurrence, we get same result:

$$\begin{aligned} & \frac{\partial}{\partial W_2} f(W_2(f(W_1x)) + \frac{\partial}{\partial W_1} f(W_2(f(W_1x)) \\ = \; & f'(W_2(f(W_1x)) \left(f(W_1x) \right) + f'(W_2(f(W_1x)) \left(W_2 f'(W_1x) x \right) \\ = \; & f'(W_2(f(W_1x)) \left(f(W_1x) + W_2 f'(W_1x) x \right) \\ = \; & f'(W(f(Wx)) \left(f(Wx) + W f'(Wx) x \right). \end{aligned}$$

The same holds true for full RNNs.

In the below experiments, I use L-BFGS over the complete training data to minimize the objective. Generally, this could cause problems due to the non-differentiable objective function. However, I did not observe problems in practice. In subsequent chapters, I found minibatched L-BFGS or AdaGrad to perform much better.

3.1.5 Experiments

I evaluate my RNN architecture on both vision and NLP tasks. The only parameters to tune are n, the size of the hidden layer; κ, the penalization term for incorrect parsing decisions and λ the regularization parameter. I found that my method is robust to both choices, varying in performance only a few percent for some parameter combinations. Furthermore, given the proper regularization, training accuracy was highly correlated with test performance. I chose $n = 100$, $\kappa = 0.05$ and $\lambda = 0.001$.

Scene Understanding: Segmentation and Annotation

The vision experiments are performed on the Stanford background dataset[3]. I first provide accuracy of multiclass segmentation where each pixel is labeled with a semantic class. I follow a similar protocol to Gould et al. (2009) and report pixel level accuracy in table 3.1.5. I train the full RNN model which influences the leaf embeddings through backpropagation of higher node classification and structure learning errors. I can simply label the superpixels by their most likely class based on the multinomial distribution of the softmax layer. Recall that this layer was trained on top of all nodes in the tree as described in Sec. 3.1.2. Here, I only use the response at the leaf nodes. As shown in table 3.1.5, I outperform previous methods that report results on this data, such as the recent methods of Tighe and Lazebnik (2010). I report accuracy of an additional logistic regression baseline to show the improvement using the neural network layer instead of the raw vision features.

On a 2.6GHz laptop my Matlab implementation needs 16 seconds to parse 143 test images. I show segmented and labeled scenes in Fig. 3.4.

[3]The dataset is available at `http://dags.stanford.edu/projects/scenedataset.html`

Method and Semantic Pixel Accuracy in	%
Pixel CRF, Gould et al.(2009)	74.3
Log. Regr. on Superpixel Features	75.9
Region-based energy, Gould et al.(2009)	76.4
Local Labeling,TL(2010)	76.9
Superpixel MRF,TL(2010)	77.5
Simultaneous MRF,TL(2010)	77.5
RNN (my method)	**78.1**

Table 3.1: Pixel level multi-class segmentation accuracy of other methods and my proposed RNN architecture on the Stanford background dataset. TL(2010) methods are reported in Tighe and Lazebnik (2010).

Scene Classification

The Stanford background dataset can be roughly categorized into three scene types: city scenes, countryside and sea-side. I label the images with these three labels and train a linear SVM using the average over all nodes' activations in the tree as features. Hence, I employ the entire parse tree and the learned feature representations of the RNN. With an accuracy of 88.1%, I outperform the state-of-the art features for scene categorization, Gist descriptors (Oliva and Torralba, 2001b), which obtain only 84.0%. I also compute a baseline using my RNN. In the baseline I use as features only the very top node of the scene parse tree. I note that while this captures enough information to perform well above a random baseline (71.0% vs. 33.3%), it does lose some information that is captured by averaging all tree nodes.

Nearest Neighbor Scene Subtrees

In order to show that the learned feature representations (node activations) capture important appearance and label information even for higher nodes in the tree, I visualize nearest neighbor super segments. I parse all test images with the trained RNN. I then find subtrees whose nodes have all been assigned the same class label by my algorithm and save the top nodes' vector representation of that subtree. This also includes initial superpixels. Using this representation, I compute nearest neighbors

Figure 3.4: Results of multi-class image segmentation and pixel-wise labeling with recursive neural networks. Best viewed in color.

across all images and all such subtrees (ignoring their labels). Fig. 3.5 shows the results. The first image is a random subtree's top node and the remaining regions are the closest subtrees in the dataset in terms of Euclidean distance between the vector representations.

Figure 3.5: Nearest neighbors image region trees: The learned feature representations of higher level nodes capture interesting visual and semantic properties of the merged segments below them. Best viewed in color.

Supervised Parsing

In all experiments my word and phrase representations are 100-dimensional, as in the vision experiments. I train all models on the Wall Street Journal section of the Penn Tree Bank using the standard training (2–21), development (22) and test (23) splits.

The final unlabeled bracketing F-measure (see Manning and Schütze (1999) for details) of this single matrix RNN language parser is 90.29% on length 15 sentences, compared to 91.63% for the widely used Berkeley parser (Klein and Manning, 2003a) (development F1 is virtually identical with 92.06% for the RNN and 92.08% for the

Center Phrase and Nearest Neighbors All the figures are adjusted for seasonal variations
1. All the numbers are adjusted for seasonal fluctuations
2. All the figures are adjusted to remove usual seasonal patterns
3. All Nasdaq industry indexes finished lower, with financial issues hit the hardest

Knight-Ridder wouldn't comment on the offer
1. Harsco declined to say what country placed the order
2. Coastal wouldn't disclose the terms
3. Censorship isn't a Marxist invention

Sales grew almost 7% to $UNK m. from $UNK m.
1. Sales rose more than 7% to $94.9 m. from $88.3 m.
2. Sales surged 40% to UNK b. yen from UNK b.
3. Revenues declined 1% to $4.17 b. from$ 4.19 b.

Fujisawa gained 50 to UNK
1. Mead gained 1 to 37 UNK
2. Ogden gained 1 UNK to 32
3. Kellogg surged 4 UNK to 75

The dollar dropped
1. The dollar retreated
2. The dollar gained
3. Bond prices rallied

Figure 3.6: Nearest neighbors phrase trees. The learned feature representations of higher level nodes capture interesting syntactic and semantic similarities between the phrases. (b.=billion, m.=million)

Berkeley parser). Unlike most previous systems, my parser does not provide a parent with information about the syntactic categories of its children. This shows that my learned, continuous representations capture enough syntactic information to make good parsing decisions.

While this parser was not as good as the Berkeley parser, it performed respectably in terms of speed (on a 2.6GHz laptop the Matlab implementation needs 72 seconds to parse 421 sentences of length less than 15) and accuracy (1.3% difference in unlabeled F1). However, for longer sentences the performance dropped off. I introduce a competitive RNN-based parser for sentences of any length in Sec. 4.1.

Nearest Neighbor Phrases

In the same way I collected nearest neighbors for nodes in the scene tree, I can compute nearest neighbor embeddings of multi-word phrases using the Euclidean distance between the node representations. I embed complete sentences from the WSJ dataset into the syntactico-semantic feature space. Especially for longer sentences it is often the case that no semantically and syntactically similar sentences are present in the dataset. In Fig. 3.6 I show several example sentences which had similar sentences in the dataset and list three of their nearest neighbors in embedding space. My examples show that the learned features capture several interesting semantic and syntactic similarities between sentences or phrases.

The next section describes work related to the various aspects of this first thesis section.

3.1.6 Related Work

Five key research areas influence and motivate this first RNN work. I briefly outline connections and differences between them.

Scene Understanding has become a central task in computer vision. The goal is to understand what objects are in a scene (annotation), where the objects are exactly (segmentation) and what general scene type the image shows (classification). Some methods for this task such as Oliva and Torralba (2001b) and Schmid (2006) rely on a global descriptor which can do very well for classifying scenes into broad categories. However, these approaches fail to gain a deeper understanding of the objects in the scene. At the same time, there is a myriad of different approaches for image annotation and semantic segmentation of objects into regions (Rabinovich et al., 2007; Gupta and Davis, 2008). Recently, these ideas have been combined to provide more detailed scene understanding (Hoiem et al., 2006; Li et al., 2009; Gould et al., 2009; Socher et al., 2010).

My algorithm *parses* an image, that is, it recursively merges pairs of segments into *super segments* in a semantically and structurally coherent way. Many other scene understanding approaches only consider a flat set of regions. Some approaches

such as Gould et al. (2009) also consider merging operations. For merged super segments, they compute new features. In contrast, my RNN-based method *learns* a representation for super segments. This learned representation together with simple logistic regression outperforms the original vision features and complex conditional random field models. Furthermore, I show that the image parse trees are useful for scene classification and outperform global scene features such as Gist descriptors (Oliva and Torralba, 2001b). Farabet et al. (2012) introduced a model for scene segmentation that is based on multi-scale convolutional neural networks and learns feature representations.

Syntactic parsing of natural language sentences is a central task in natural language processing (NLP) because of its importance in mediating between linguistic expression and meaning. For example, much work has shown the usefulness of syntactic representations for subsequent tasks such as relation extraction, semantic role labeling (Gildea and Palmer, 2002) and paraphrase detection (Callison-Burch, 2008). My RNN architecture jointly learns how to parse and how to represent phrases in a continuous vector space of features. This allows us to embed both single lexical units and unseen, variable-sized phrases in a syntactically coherent order. The learned feature representations capture syntactic and compositional-semantic information. I show that they can help inform accurate parsing decisions and capture interesting similarities between phrases and sentences.

Using NLP techniques in computer vision. The connection between NLP ideas such as parsing or grammars and computer vision has been explored before (Zhu and Zhang, 2006; Tighe and Lazebnik, 2010), among many others. My approach is similar on a high level, however, more general in nature. I show that the same neural network based architecture can be used for both natural language and image parsing. Furthermore, when directly compared, I outperform other such methods.

Deep Learning in vision applications can find lower dimensional representations for fixed size input images which are useful for classification (Hinton and Salakhutdinov, 2006). Recently, Lee et al. (2009) were able to scale up deep networks to more realistic image sizes. Using images of single objects which were all in roughly the same scale, they were able to learn parts and classify the images into

object categories. My approach differs in several fundamental ways to any previous deep learning algorithm. (i) Instead of learning features from raw, or whitened pixels, I use off-the-shelf vision features of segments obtained from oversegmented full scene images. (ii) Instead of building a hierarchy using a combination of convolutional and max-pooling layers, I recursively apply the same network to merged segments and give each of these a semantic category label. (iii) This is the first deep learning work which learns full scene segmentation, annotation and classification. The objects and scenes vary in scale, viewpoint, lighting etc.

Using deep learning for NLP applications has been investigated by several people (*inter alia* Bengio et al., 2003; Henderson, 2003; Collobert and Weston, 2008). In most cases, the inputs to the neural networks are modified to be of equal size either via convolutional and max-pooling layers or looking only at a fixed size window around a specific word. My approach is different in that it handles variable sized sentences in a natural way and captures the recursive nature of natural language. Furthermore, it jointly learns parsing decisions, categories for each phrase and phrase feature embeddings which capture the semantics of their constituents . In Socher et al. (2010) I developed an NLP specific parsing algorithm based on RNNs. This algorithm is a special case of the one developed here.

Deep learning for phrases Early attempts at using neural networks to describe phrases include Elman (1991), who used recurrent neural networks to create representations of sentences from a simple toy grammar and to analyze the linguistic expressiveness of the resulting representations. Words were represented as one-on vectors, which was feasible since the grammar only included a handful of words. Collobert and Weston (2008) showed that neural networks can perform well on sequence labeling language processing tasks while also learning appropriate features. However, their model is lacking in that it cannot represent the recursive structure inherent in natural language. They partially circumvent this problem by using either independent window-based classifiers or a convolutional layer. RNN-specific training was introduced by Goller and Küchler (1996) to learn distributed representations of given, structured objects such as logical terms. In contrast, my model both predicts the structure and its representation.

This section introduced a first structure prediction objective functions with supervised scores and classification. The next section will also learn structure but will do so in an unsupervised way that does not result in grammatically plausible tree structures.

3.2 Semi-Supervised Reconstruction-Classification Error - For Sentiment Analysis

The next type of objective function is based on autoencoders (Bengio, 2009). An autoencoder is a neural network model that learns the function $f(x) = x$ in order to learn a reduced dimensional representation of fixed-size inputs such as image patches or bag-of-words representations. A closer look at this function reveals a common form of

$$f(x) = g^2(g^1(x)) = x, \tag{3.16}$$

where g^1 introduces an informational bottleneck and, for instance, might output a lower dimensional code.[4] Afterwards, g^2 attempts to reconstruct the original input from this lower dimensional code. This mechanism forces this function to learn useful bases of the original input space.

This section combines the unsupervised objective of such autoencoders with a softmax-based supervised objective as introduced in Sec. 2.5.

The supervised labels can be of any type. I choose sentiment analysis as the first example of supervised recursive prediction because it exhibits interesting linguistic phenomena and variation in predicted classes when multiple words are combined. Furthermore, the ability to identify sentiments about personal experiences, products, movies etc. is crucial to understand user generated content in social networks, blogs or product reviews. Detecting sentiment in these data is a challenging task which has spawned a lot of interest (Pang and Lee, 2008).

Most sentiment analysis baseline methods are based on bag-of-words representations (Pang et al., 2002). As mentioned in the first chapter, ignoring word order

[4]Other forms of informational bottlenecks, such as sparsity (Lee et al., 2007), are also possible

is an oversimplified independence assumption and models using such representations cannot properly capture more complex linguistic phenomena such as negation and its scope in sentiment. More advanced methods such as the one introduced by Nakagawa et al. (2010) could capture such phenomena but requires a lot of manually constructed resources (sentiment lexica, parsers, polarity-shifting rules). This limits the applicability of these methods to a broader range of tasks and languages. Lastly, almost all previous work is based on single categories such as positive/negative or scales such as star ratings, for instance movie reviews (Pang and Lee, 2005), opinions (Wiebe et al., 2005) or customer reviews (Ding et al., 2008). Such a one-dimensional scale does not accurately reflect the complexity of human emotions and sentiments.

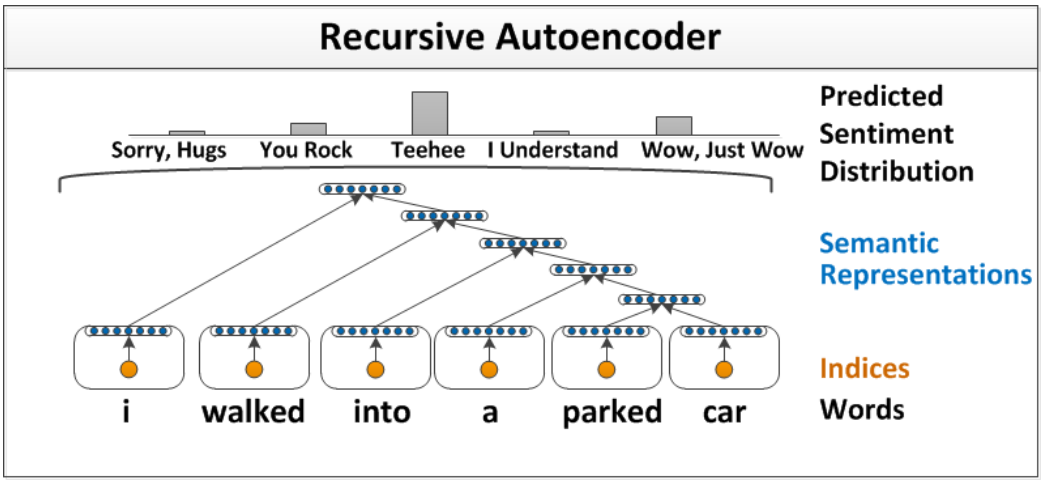


Figure 3.7: Illustration of a recursive autoencoder architecture which learns semantic vector representations of phrases. Word indices (orange) are first mapped into a semantic vector space (blue). Then they are recursively merged by the same autoencoder network into a fixed length sentence representation. The vectors at each node are used to as features to predict a distribution over sentiment labels.

I seek to address these three issues. (i) Instead of using a bag-of-words representation, the RNN-based models of this thesis exploit hierarchical structure and use compositional semantics to understand sentiment. (ii) My system can be trained both on unlabeled domain data and on supervised sentiment data and does not require any language-specific sentiment lexica, parser, etc. (iii) Rather than limiting sentiment

to a positive/negative scale, I predict a multidimensional distribution over several complex sentiments.

My approach is based on semi-supervised, recursive autoencoders (RAE) which use as inputs neural word vector representations (Bengio et al., 2003; Collobert and Weston, 2008). Fig. 3.7 shows an illustration of the model which learns vector representations of phrases and full sentences as well as their hierarchical structure from unsupervised text. I extend this model to jointly learn a distribution over sentiment labels at each node of the hierarchy from sentence-level annotation.

I evaluated the RAE approach on several standard datasets where I achieved state-of-the art performance in 2011. Since then, the more powerful model introduced in Sec. 4.3 has outperformed the RAE. I also show results on the experience project (EP) dataset (Potts, 2010) that captures a broader spectrum of human sentiments and emotions. The dataset consists of very personal confessions anonymously made by people on the experience project website *www.experienceproject.com*. Confessions are labeled with a set of five reactions by other users. Reaction labels are *you rock* (expressing approvement), *tehee* (amusement), *I understand*, *Sorry, hugs* and *Wow, just wow* (displaying shock). For evaluation on this dataset I predict both the label with the most votes as well as the full distribution over the sentiment categories. On both tasks my model outperforms competitive baselines.

After describing the model in detail, I evaluate it qualitatively by analyzing the learned n-gram vector representations and compare quantitatively against other methods on standard datasets and the EP dataset.

3.2.1 Semi-Supervised Recursive Autoencoders

My model aims to find vector representations for variable-sized phrases in either unsupervised or semi-supervised training regimes. These representations can then be used for subsequent tasks. I first review a related recursive model based on autoencoders, introduce my recursive autoencoder (RAE) and describe how it can be modified to jointly learn phrase representations, phrase structure and label predictions of sentiments.

Standard Recursive Autoencoders

The goal of autoencoders is to learn a dimensionality reduction of their inputs. In this section I describe how to obtain a reduced dimensional vector representation for sentences.

In the past autoencoders have only been used in a setting where the tree structure was given a-priori. I review this setting before continuing with my model which can learn the tree structure. Fig. 3.8 shows an instance of a recursive autoencoder (RAE) applied to a given tree. Assume I am given a list of word vectors $x = (x_1, \ldots, x_m)$ as described in Sec. 2.3. I also have a binary tree structure for this input in the form of branching triplets of parents with children: $(p \rightarrow c_1 c_2)$. Each child can be either an input word vector x_i or a nonterminal node in the tree. For the example in Fig. 3.8, I have the following triplets: $((y_1 \rightarrow x_3 x_4), (y_2 \rightarrow x_2 y_1), (y_1 \rightarrow x_1 y_2))$. In order to be able to apply the same neural network to each pair of children, the hidden representations y_i have to have the same dimensionality as the x_i's.

Given this tree structure, I can now compute the parent representations. The first parent vector y_1 is computed from the children $(c_1, c_2) = (x_3, x_4)$:

$$p = f(W_e[c_1; c_2] + b_e), \tag{3.17}$$

where I multiplied a matrix of parameters $W_e \in \mathbb{R}^{n \times 2n}$ by the concatenation of the two children. After adding a bias term I applied element-wise the tanh function to the resulting vector. One way of assessing how well this n-dimensional vector represents its children is to try to reconstruct the children in a reconstruction layer:

$$[c_1'; c_2'] = W_d p + b_d. \tag{3.18}$$

Such a reconstruction function is one possibility for the function g^2 mentioned in the first paragraph of this section and defined in Eq. 3.16.

During training, the goal is to minimize the reconstruction error of this input pair. For each pair, I compute the Euclidean distance between the original input and its

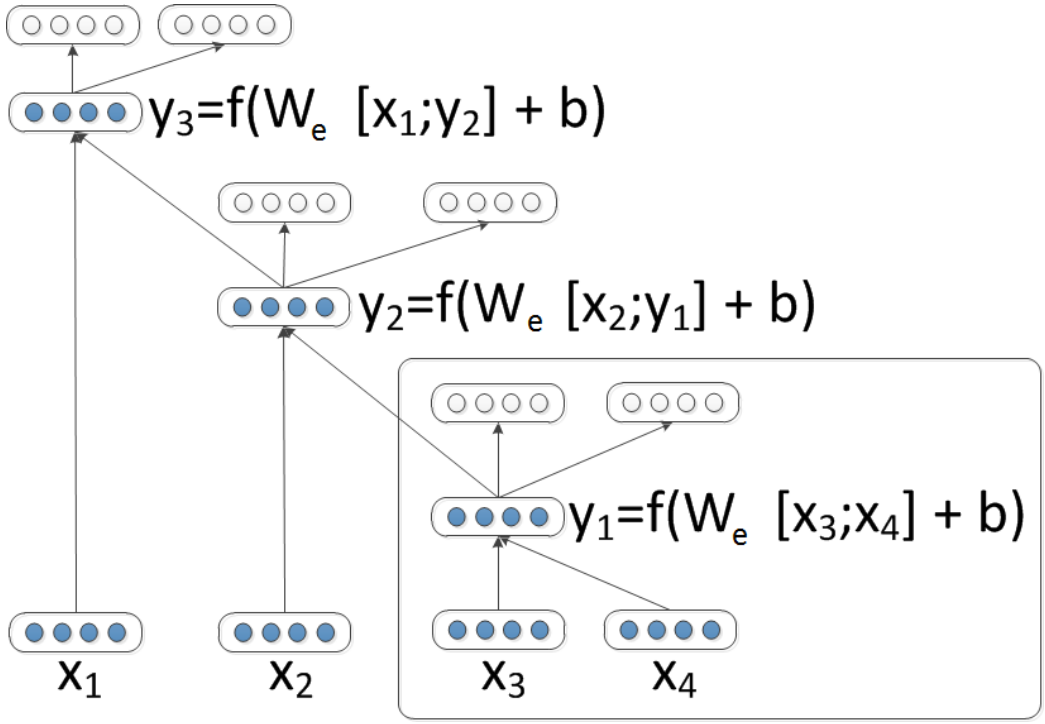


Figure 3.8: Illustration of an application of a recursive autoencoder to a binary tree. The nodes which are not filled are only used to compute reconstruction errors. A standard autoencoder (in box) is re-used at each node of the tree.

reconstruction:

$$E_{rec}([c_1; c_2]) = \frac{1}{2} \left|\left|[c_1; c_2] - [c_1'; c_2']\right|\right|^2 . \tag{3.19}$$

This model of a standard autoencoder is boxed in Fig. 3.8. Now that I have defined how an autoencoder can be used to compute an n-dimensional vector representation (p) of two n-dimensional children (c_1, c_2), I can describe how such a network can be used for the rest of the tree.

Essentially, the same steps repeat. Now that y_1 is given, I can use Eq. 3.17 to compute y_2 by setting the children to be $(c_1, c_2) = (x_2, y_1)$. Again, after computing the intermediate parent vector y_2, I can assess how well this vector capture the content of the children by computing the reconstruction error as in Eq. 3.19. The process repeats until the full tree is constructed and I have a reconstruction error at each nonterminal node. This model is similar to the RAAM model (Pollack, 1990) which also requires a fixed tree structure.

Unsupervised Recursive Autoencoder for Structure Prediction

Now, assume there is no tree structure given for the input vectors in x. The goal of my structure-prediction RAE is to minimize the reconstruction error of all vector pairs of children in a tree. I define $A(x)$ as the set of all possible trees that can be built from an input sentence x. Further, let $T(y)$ be a function that returns the triplets of a tree indexed by s of all the non-terminal nodes in a tree. Using the reconstruction error of Eq. 3.19, I compute

$$RAE_\theta(x) = \arg\min_{y \in A(x)} \sum_{s \in T(y)} E_{rec}([c_1; c_2]_s) \tag{3.20}$$

I now describe a greedy approximation that constructs such a tree.

Greedy Unsupervised RAE. For a sentence with m words, I apply the autoencoder recursively. It takes the first pair of neighboring vectors, defines them as potential children of a phrase $(c_1; c_2) = (x_1; x_2)$, concatenates them and gives them as input to the autoencoder. For each word pair, I save the potential parent node p and the resulting reconstruction error.

After computing the score for the first pair, the network is shifted by one position and takes as input vectors $(c_1, c_2) = (x_2, x_3)$ and again computes a potential parent node and a score. This process repeats until it hits the last pair of words in the sentence: $(c_1, c_2) = (x_{m-1}, x_m)$. Next, it selects the pair which had the lowest reconstruction error (E_{rec}) and its parent representation p will represent this phrase and replace both children in the sentence word list. For instance, consider the sequence (x_1, x_2, x_3, x_4) and assume the lowest E_{rec} was obtained by the pair (x_3, x_4). After the first pass, the new sequence then consists of $(x_1, x_2, p_{(3,4)})$. The process repeats and treats the new vector $p_{(3,4)}$ like any other input vector. For instance, subsequent states could be either: $(x_1, p_{(2,(3,4))})$ or $(p_{(1,2)}, p_{(3,4)})$. Both states would then finish with a deterministic choice of collapsing the remaining two states into one parent to obtain $(p_{(1,(2,(3,4)))})$ or $(p_{((1,2),(3,4))})$ respectively. The tree is then recovered by unfolding the collapsing decisions.

The resulting tree structure captures as much of the semantic information as possible (in order to allow reconstructing the input words) but does not necessarily follow standard syntactic constraints. I also experimented with a method that finds the globally optimal solution to Eq.3.20 based on the CKY-algorithm but the performance is similar and the greedy version is much faster.

Weighted Reconstruction. One problem with simply using the reconstruction error of both children equally as describe in Eq.3.19 is that each child could represent a different number of previously collapsed words and is hence of bigger importance for the overall meaning reconstruction of the sentence. For instance in the case of $(x_1, p_{(2,(3,4))})$ one would like to give more importance to reconstructing p than x_1. I capture this desideratum by adjusting the reconstruction error. Let n_1, n_2 be the number of words underneath a current potential child, I re-define the reconstruction error to be $E_{rec}([c_1; c_2]; \theta) =$

$$\frac{n_1}{n_1 + n_2} ||c_1 - c_1'||^2 + \frac{n_2}{n_1 + n_2} ||c_2 - c_2'||^2 \tag{3.21}$$

Vector Length Normalization. One of the goals of RAEs is to induce semantic vector representations that allow us to compare n-grams of different lengths. The

RAE tries to lower reconstruction error of not only the bigrams but also of nodes higher in the tree. Unfortunately, since the RAE computes the hidden representations it then tries to reconstruct, it can just lower reconstruction error by making the hidden layer very small in magnitude. To prevent such undesirable behavior, I modify the hidden layer such that the resulting parent representation always has length one, after computing p as in Eq.3.17, I simply set: $p = \frac{1}{||p||}$.

Semi-Supervised Recursive Autoencoders

So far, the RAE was completely unsupervised and induced general representations that capture the semantics of multi-word phrases. In this section, I extend RAEs to a semi-supervised setting in order to predict a sentence- or phrase-level target distribution t.[5]

One of the main advantages of the RAE is that each node of the tree built by the RAE has associated with it a distributed vector representation (the parent vector p) which could also be seen as features describing that phrase. I can leverage this representation by adding on top of each parent node a simple softmax layer to predict class distributions:

$$d(p;\theta) = \text{softmax}(W^{label}p). \tag{3.22}$$

Assuming there are K labels, $d \in \mathbb{R}^K$ is a K-dimensional multinomial distribution and $\sum_{k=1} d_k = 1$. Fig. 3.9 shows such a semi-supervised RAE unit. Let t_k be the kth element of the multinomial target label distribution t for one entry. The softmax layer's outputs are interpreted as conditional probabilities $d_k = p(k|[c_1; c_2])$, hence the cross-entropy error is

$$E_{cE}(p, t; \theta) = -\sum_{k=1}^{K} t_k \log d_k(p;\theta). \tag{3.23}$$

Using this cross-entropy error for the label and the reconstruction error from Eq.3.21, the final semi-supervised RAE objective over (sentences,label) pairs (x, t) in a corpus

[5]For the binary label classification case, the distribution is of the form $[1, 0]$ for class 1 and $[0, 1]$ for class 2.

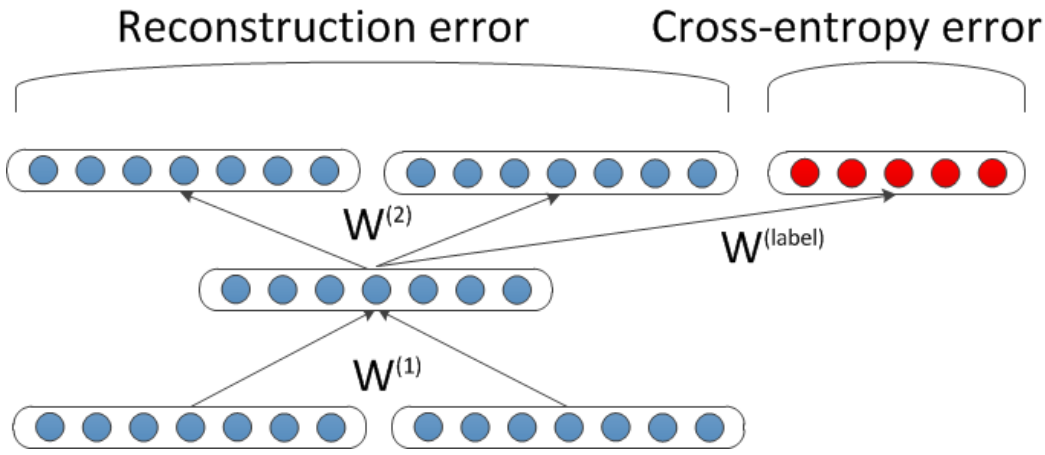


Figure 3.9: Illustration of an RAE unit at a nonterminal tree node. Red nodes show the supervised *softmax* layer for label distribution prediction.

becomes

$$J = \frac{1}{N}\sum_{(x,t)} E(x,t;\theta) + \frac{\lambda}{2}||\theta||^2, \tag{3.24}$$

where I have an error for each entry in the training set that is the sum over the error at the nodes of the tree that is constructed by the greedy RAE:

$$E(x,t;\theta) = \sum_{s \in T(RAE_\theta(x))} E([c_1;c_2]_s, p_s, t, \theta).$$

The error at each nonterminal node is the weighted sum of reconstruction and cross-entropy errors, $E([c_1;c_2]_s, p_s, t, \theta) =$

$$\alpha E_{rec}([c_1;c_2]_s;\theta) + (1-\alpha)E_{cE}(p_s,t;\theta).$$

The hyperparameter α weighs reconstruction and cross-entropy error. When minimizing the cross-entropy error of this softmax layer, the error will backpropagate and influence both the RAE parameters and the word representations. Initially, words such as *good* and *bad* have very similar representations. When learning positive/negative

sentiment, these word embeddings will change and become different.

In order to predict the sentiment distribution of a sentence with this model, I can use the output of the top node's softmax layer. Alternatively, there exist many other options to extract features from the tree, such as taking the average of all the nodes and training an SVM.

3.2.2 Learning

Let $\theta = (W_e, b_e, W_d, b_d, W_{label}, L)$ be the set of my model parameters, then the gradient becomes:

$$\frac{\partial J}{\partial \theta} = \frac{1}{N} \sum_i \frac{\partial E(x, t; \theta)}{\partial \theta} + \lambda \theta. \tag{3.25}$$

To compute this gradient, I first greedily construct all trees and then derivatives for these trees are computed efficiently via backpropagation through structure (Goller and Küchler, 1996). Because the algorithm is greedy and the derivatives of the supervised cross-entropy error also modify the matrix W_e, this objective is not necessarily continuous and a step in the gradient descent direction may not necessarily decrease the objective. However, I found that L-BFGS run over the complete training data (batch mode) to minimize the objective works well in practice, and that convergence is smooth, with the algorithm typically finding a good solution quickly.

3.2.3 Experiments

I first describe the new experience project (EP) dataset, results of standard classification tasks and how to predict sentiment label distributions. I then show results on other commonly used datasets and conclude with an analysis of the important parameters of the model.

In the experiments of this section, I represent words using the 100-dimensional word vectors of the model from Collobert and Weston (2008) provided by Turian et al. (2010).

The Experience Project Dataset

The confessions section of the experience project website `http://www.experienceproject.com/confessions.php` lets people anonymously write short personal stories or "confessions" (Maas et al., 2011). Once a story is on the site, each user can give a single vote to one of five label categories (with my interpretation):

1 Sorry, Hugs: User offers condolences to author.
2. You Rock: Indicating approval, congratulations.
3. Teehee: User found the anecdote amusing.
4. I Understand: Show of empathy.
5. Wow, Just Wow: Expression of surprise,shock.

The EP dataset has 31,676 confession entries, a total number of 74,859 votes for one of the 5 labels above, the average number of votes per entry is 2.4 (with a variance of 33). For the five categories, the numbers of votes are $[14,816; 13,325; 10,073; 30,844; 5,801]$. Since an entry with less than 4 votes is not very well identified, I train and test only on entries with at least 4 total votes. There are 6,130 total such entries. The distribution over total votes in the 5 classes is similar: $[0.22; 0.2; 0.11; 0.37; 0.1]$. The average length of entries is 129 words. Some entries contain multiple sentences. In these cases, I average the predicted label distributions from the sentences. Table 3.2 shows statistics of this and other commonly used sentiment datasets (which I compare on in later experiments). Table 3.3 shows example entries as well as gold and predicted label distributions as described in the next sections.

Corpus	K	Instances	Distr(+/-).	Avg$\|W\|$
MPQA	2	10,624	0.31/0.69	3
MR	2	10,662	0.5/0.5	22
EP	5	31,675	.2/.2/.1/.4/.1	113
EP≥ 4	5	6,129	.2/.2/.1/.4/.1	129

Table 3.2: Statistics on the different datasets. K is the number of classes. Distr. is the distribution of the different classes (in the case of 2, the positive/negative classes, for EP the rounded distribution of total votes in each class). $|W|$ is the average number of words per instance. I use EP≥ 4, a subset of entries with at least 4 votes.

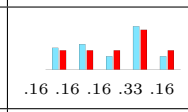
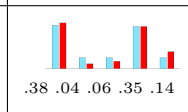
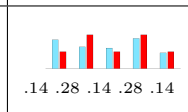
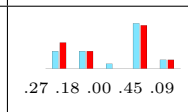
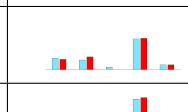
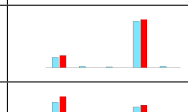
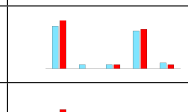
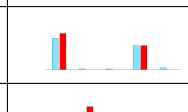
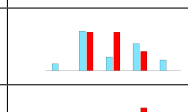
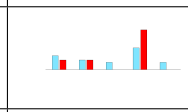
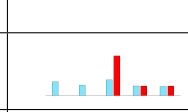
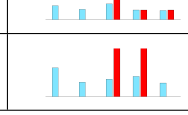
KL	Predicted&Gold	V.	Entry (Shortened if it ends with ...)
.03	 .16 .16 .16 .33 .16	6	I reguarly shoplift. I got caught once and went to jail, but I've found that this was not a deterrent. I don't buy groceries, I don't buy school supplies for my kids, I don't buy gifts for my kids, we don't pay for movies, and I dont buy most incidentals for the house (cleaning supplies, toothpaste, etc.)...
.03	 .38 .04 .06 .35 .14	165	i am a very succesfull buissnes man.i make good money but i have been addicted to crack for 13 years.i moved 1 hour away from my dealers 10 years ago to stop using now i dont use daily but once a week usally friday nights. i used to use 1 or 2 hundred a day now i use 4 or 5 hundred on a friday.my problem is i am a funcational addict...
.05	 .14 .28 .14 .28 .14	7	Hi there, Im a guy that loves a girl, the same old bloody story... I met her a while ago, while studying, she Is so perfect, so mature and yet so lonely, I get to know her and she get ahold of me, by opening her life to me and so did I with her, she has been the first person, male or female that has ever made that bond with me,...
.07	 .27 .18 .00 .45 .09	11	be kissing you right now. i should be wrapped in your arms in the dark, but instead i've ruined everything. i've piled bricks to make a wall where there never should have been one. i feel an ache that i shouldn't feel because i've never had you close enough. we've never touched, but i still feel as though a part of me is missing. ...
.05		23	Dear Love, I just want to say that I am looking for you. Tonight I felt the urge to write, and I am becoming more and more frustrated that I have not found you yet. I'm also tired of spending so much heart on an old dream. ...
.05		5	I wish I knew somone to talk to here.
.06		24	I loved her but I screwed it up. Now she's moved on. I'll never have her again. I don't know if I'll ever stop thinking about her.
.06		5	i am 13 years old and i hate my father he is alwas geting drunk and do's not care about how it affects me or my sisters i want to care but the truthis i dont care if he dies
.13		6	well i think hairy women are attractive
.35		5	As soon as I put clothings on I will go down to DQ and get a thin mint blizzard. I need it. It'll make my soul feel a bit better :)
.36		6	I am a 45 year old divoced woman, and I havent been on a date or had any significant relationship in 12 years...yes, 12 yrs. the sad thing is, Im not some dried up old granny who is no longer interested in men, I just can't meet men. (before you judge, no Im not terribly picky!) What is wrong with me?
.63		6	When i was in kindergarden i used to lock myself in the closet and eat all the candy. Then the teacher found out it was one of us and made us go two days without freetime. It might be a little late now, but sorry guys it was me haha
.92		4	My paper is due in less than 24 hours and I'm still dancing round my room!

Table 3.3: Example EP confessions from the test data with KL divergence between my predicted distribution (light blue, left bar on each of the 5 classes) and ground truth distribution (red bar and numbers underneath), number of votes. The 5 classes are [Sorry, Hugs; You Rock; Teehee; I Understand; Wow, Just Wow]. Even when the KL divergence is higher, my model makes reasonable alternative label choices. Some entries are shortened.

Compared to other datasets, the EP data contains a wider range of human emotions that goes far beyond positive/negative product or movie reviews. The topics range from generic happy statements, daily clumsiness reports, love, loneliness, to relationship abuse and suicidal notes. As is evident from the total number of label votes, the most common user reaction is one of empathy and an ability to relate to the authors experience. However, some stories describe horrible scenarios that are not common and hence receive more offers of condolence. In the following sections I shows some examples of stories with predicted and true distributions but refrain from listing the most horrible experiences.

For all experiments on the EP dataset, I split the data into train (49%), development (21%) and test data (30%).

EP: Predicting the Label with Most Votes

The first task for my evaluation on the EP dataset is to simply predict the single class that receives the most votes. In order to compare my novel joint phrase representation and classifier learning framework to traditional methods, I use the following baselines:

Random Since there are five classes, this gives 20% accuracy.

Most Frequent Selecting the class which most frequently has the most votes (which is the *I understand* class).

Baseline 1: MaxEnt This baseline is a simple MaxEnt model that uses a bag-of-word representation.

Baseline 2: Features This model is similar to traditional approaches to sentiment classification in that it uses many hand-engineered resources. I first used a spell-checker and Wordnet to map words and their misspellings to synsets to reduce the total number of words. I then replaced sentiment words with a sentiment category identifier using the sentiment lexica of the Harvard Inquirer (Stone, 1966) and LIWC (Pennebaker et al., 2007). Lastly, I used tf-idf weighting on the the bag-of-word entry representations and trained an SVM.

Method	Accuracy
Random	20.0
Most Frequent	38.1
Baseline 1: MaxEnt	46.4
Baseline 2: Features	47.0
Baseline 3: Word Vectors	45.5
RAE (my method)	**49.7**

Table 3.4: Accuracy of predicting the class with most votes.

Baseline 3: Word Vectors I can ignore the RAE tree structure and only train softmax layers on the words in order to influence the word vectors. Then train an SVM on the average of the word vectors.

Table 3.4 shows the results for predicting the class with the most votes. Even the approach that is based on sentiment lexica and other resources is outperformed by my model by almost 3%, showing that for tasks involving complex broad-range human sentiment, the often used sentiment lexica lack in coverage and traditional bag-of-words representations are not powerful enough.

EP: Predicting Sentiment Distributions

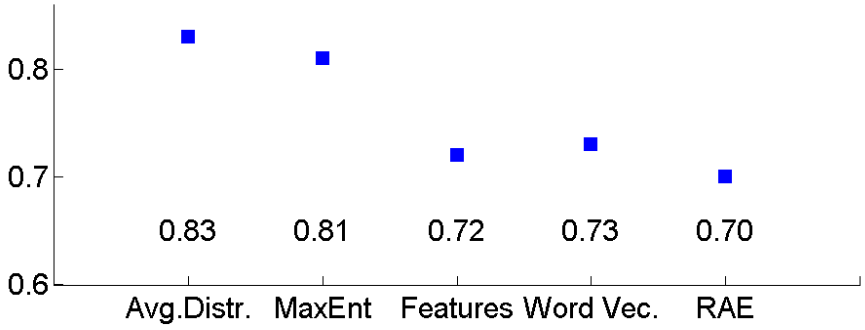


Figure 3.10: Average KL-divergence between gold and predicted sentiment distributions (lower is better).

I now turn to evaluating my distribution-prediction approach. In both this and the previous maximum label task, I backprop using the gold multinomial distribution as a

target. Since I maximize likelihood and because I want to predict a distribution that is closest to the distribution of labels that people would assign to a story, I evaluate using KL divergence: $KL(g||p) = \sum_i g_i \log(g_i/p_i)$, where g is the gold distribution and p is the predicted one. I report the average KL divergence, where a smaller value indicates better predictive power. To get an idea of the values of KL divergence, predicting random distributions gives a an average of 1.2 in KL divergence, predicting simply the average distribution in the training data give 0.83. Fig. 3.10 shows that my RAE-based model outperforms the other baselines. Table 3.3 shows EP example entries with predicted and gold distributions, as well as numbers of votes.

Binary Polarity Classification

In order to compare my approach to other methods I also show results on commonly used sentiment datasets: movie reviews[6] (MR) (Pang and Lee, 2005) and opinions[7] (MPQA) (Wiebe et al., 2005). I give statistical information on these and the EP corpus in Table 3.2.

I compare to the state-of-the-art system of Nakagawa et al. (2010), a dependency tree based classification method that uses CRFs with hidden variables. I use the same training and testing regimen (10-fold cross validation) as well as their baselines: majority phrase voting using sentiment and reversal lexica; rule-based reversal using a dependency tree; Bag-of-Features and their full model Tree-CRF. As shown in Table 3.6, my algorithm outperforms or performs similarly on both compared datasets even though I do not use any hand-designed lexica. An error analysis on the MPQA dataset showed several cases of single words which never occurred during training. Hence, correctly classifying these instances can only be the result of having them in the original sentiment lexicon.

I visualize the semantic vectors that the recursive autoencoder learns by listing n-grams that give the highest probability for each polarity. Table 3.5 shows such n-grams for different lengths when the RAE is trained on the movie review polarity dataset.

[6] www.cs.cornell.edu/people/pabo/movie-review-data/

[7] www.cs.pitt.edu/mpqa/

n	Most negative n-grams	Most positive n-grams
1	bad; boring; dull; flat; pointless; tv; neither; pretentious; badly; worst; lame; mediocre; lack; routine; loud; bore; barely; stupid; tired; poorly; suffers; heavy;nor; choppy; superficial	solid; beautifully; rich; chilling; refreshingly; portrait; thoughtful; socio-political; quiet; beautiful; wonderful; engrossing; provides; flaws; culture; moving; warm; powerful; enjoyable; touching
2	how bad; by bad; dull .; for bad; to bad; boring .; , dull; are bad; that bad; boring ,; , flat; pointless .; badly by; on tv; so routine; lack the; mediocre .; a generic; stupid ,; abysmally pathetic	the beautiful; moving ,; thoughtful and; , inventive; solid and; a beautiful; a beautifully; and hilarious; with dazzling; provides the; provides .; and inventive; as powerful; moving and; a moving; a powerful
3	. too bad; exactly how bad; and never dull; shot but dull; is more boring; to the dull; dull , UNK; it is bad; or just plain; by turns pretentious; manipulative and contrived; bag of stale; is a bad; the whole mildly; contrived pastiche of; from this choppy; stale material .	both the beauty; ... a polished; spare yet audacious; has a solid; cast of solid; with the chilling; , gradually reveals; beautifully acted .; ... a solid; romantic , riveting; culture into a; smart and taut; , fine music; cuts , fast; with a moving; a small gem; funny and touching
5	boring than anything else .; a major waste ... generic; nothing i had n't already; , UNK plotting ; superficial; problem ? no laughs .; , just horribly mediocre .; dull , UNK feel .; there 's nothing exactly wrong; movie is about a boring; essentially a collection of bits	engrossing and ultimately tragic .; with wry humor and genuine; cute , funny , heartwarming; easily the most thoughtful fictional; a solid piece of journalistic; between realistic characters showing honest; engrossing , seldom UNK ,; reminded us that a feel-good
8	loud , silly , stupid and pointless . ; dull , dumb and derivative horror film .; UNK 's film , a boring , pretentious; this film biggest problem ? no laughs .; film in the series looks and feels tired; do draw easy chuckles but lead nowhere .; stupid , infantile , redundant , sloppy	bringing richer meaning to the story 's; film is a riveting , brisk delight .; ... one of the most ingenious and entertaining; , deeply absorbing piece that works as a; an escapist confection that 's pure entertainment .; shot in rich , shadowy black-and-white , devils

Table 3.5: Examples of n-grams ($n = 1, 2, 3, 5, 8$) from the test data of the movie polarity dataset for which my model predicts the most positive and most negative responses.

Method	MR	MPQA
Voting w/Rev.	63.1	81.7
Rule	62.9	82.8
BoF w/ Rev.	76.4	84.1
Tree-CRF (Nakagawa'10)	77.3	**86.1**
RAE (my method)	**77.7**	86.0

Table 3.6: Accuracy of sentiment classification. See text for details on the MPQA dataset.

On a 4-core machine, training time for the smaller corpora such as the movie reviews takes around 3 hours and for the larger EP corpus around 12 hours until convergence. Testing of hundreds of movie reviews takes only a few seconds.

Reconstruction vs. Classification Error

In this experiment, I show how the hyperparameter α influences accuracy on the development set of one of the cross-validation splits of the MR dataset. This parameter essentially trade-off the supervised and unsupervised parts of the objective. Fig. 3.11 shows that a larger focus on the supervised objective is important but that a weight of $\alpha = 0.2$ for the reconstruction error prevents overfitting and achieves the highest performance.

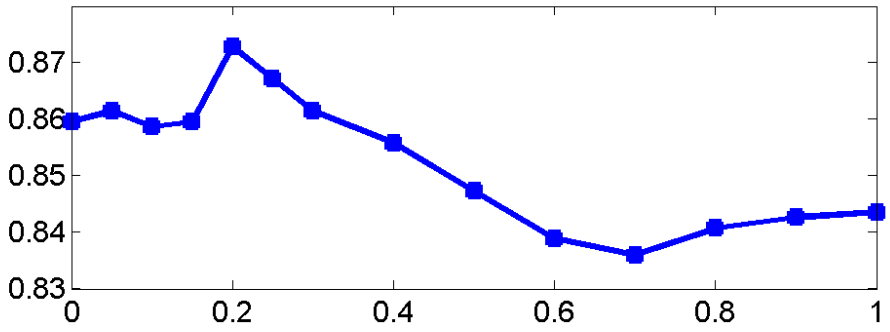


Figure 3.11: Accuracy on the development split of the MR polarity dataset for different weightings of reconstruction error and supervised cross-entropy error: $err = \alpha E_{rec} + (1 - \alpha) E_{cE}$.

3.2.4 Related Work

Autoencoder can be used to efficiently learn feature encodings which are useful for classification. Recently, Mirowski et al. (2010) learn dynamic autoencoders for documents in a bag-of-words format which, like the semi-supervised one in this section, combine supervised and reconstruction objectives.

The idea of applying an autoencoder in a recursive setting was introduced by Pollack (1990). Pollack's recursive auto-associative memories (RAAMs) are similar to

ours in that they are a connectionst, feedforward model. However, RAAMs learn vector representations only for fixed recursive data structures, whereas my RAE builds this recursive data structure. More recently, Voegtlin and Dominey (2005) introduced a linear modification to RAAMs that is able to better generalize to novel combinations of previously seen constituents. One of the major shortcomings of previous applications of recursive autoencoders to natural language sentences was their binary word representation.

Sentiment Analysis Pang et al. (2002) were one of the first to experiment with sentiment classification. They show that simple bag-of-words approaches based on Naive Bayes, MaxEnt models or SVMs are often insufficient for predicting sentiment of documents even though they work well for general topic-based document classification. Even adding specific negation words, bigrams or part-of-speech information to these models did not add significant improvements. Other document-level sentiment work includes Turney (2002); Dave et al. (2003); Beineke et al. (2004); Pang and Lee (2004). For further references, see Pang and Lee (2008).

Instead of document level sentiment classification, Wilson et al. (2005) analyze the contextual polarity of phrases and incorporate many well designed features including dependency trees. They also show improvements by first distinguishing between neutral and polar sentences. My model naturally incorporates the recursive interaction between context and polarity words in sentences in a unified framework while simultaneously learning the necessary features to make accurate predictions. Other approaches for sentence-level sentiment detection include Yu and Hatzivassiloglou (2003); Kim and Hovy (2007); Grefenstette et al. (2004); Ikeda et al. (2008).

Most previous work is centered around given sentiment lexica or building one via heuristics (Kim and Hovy, 2007; Esuli and Sebastiani, 2007), manual annotation (Das and Chen, 2001) or machine learning techniques (Turney, 2002). In contrast, I do not require an initial or constructed sentiment lexicon of positive and negative words. In fact, when training my approach on documents or sentences, it jointly learns such lexica for both single words and n-grams (see Table 3.5). Mao and Lebanon (2007) propose isotonic conditional random fields and differentiate between local, sentence-level and global, document-level sentiment.

The work of Polanyi and Zaenen (2006) and Choi and Cardie (2008) focuses on manually constructing several lexica and rules for both polar words and related content-word negators, such as "prevent cancer", where *prevent* reverses the negative polarity of *cancer*. Like my approach they capture compositional semantics. However, my model does so without manually constructing any rules or lexica.

Recently, Velikovich et al. (2010) showed how to use a seed lexicon and a graph propagation framework to learn a larger sentiment lexicon that also includes polar multi-word phrases such as "once in a life time". While my method can also learn multi-word phrases it does not require a seed set or a large web graph. Nakagawa et al. (2010) introduced an approach based on CRFs with hidden variables with very good performance. I compare to their state-of-the-art system. I outperform them on the standard corpora that I tested on without requiring external systems such as POS taggers, dependency parsers and sentiment lexica. My approach jointly learns the necessary features and tree structure.

The last two sections learned both the tree structure and a related task prediction. In contrast, the next model in the section uses fixed parse trees and learns semantic phrase vectors without any supervised signal.

3.3 Unfolding Reconstruction Errors - For Paraphrase Detection

In this section, I present a different RAE model that incorporates the similarities between both single word vectors as well as multi-word phrase vectors for the important task of paraphrase detection. Unlike all other models of this thesis, the RNN objective function in this section is entirely unsupervised. This is necessary because paraphrase detection requires the ability to deal with a very broad range of semantic phenomena for which it would be very hard to find sufficient labeled training data.

Paraphrase detection determines whether two sentences of arbitrary length and form capture the same meaning. It is used in information retrieval, question answering (Marsi and Krahmer, 2005), plagiarism detection (Clough et al., 2002), evaluation of

machine translation (Callison-Burch, 2008) and text summarization, among others. For instance, in order to avoid adding redundant information to a summary one would like to detect that the following two sentences are paraphrases:

S1 The judge also refused to postpone the trial date of Sept. 29.

S2 Obus also denied a defense motion to postpone the September trial date.

The full model in this section is based on two novel components as outlined in Fig. 3.12. The first component, is an *unfolding recursive autoencoder* (URAE) for unsupervised feature learning from unlabeled parse trees. The U.RAE is a recursive neural network that computes features for each node in an unlabeled parse tree. It is similar to the RAE of the previous section but attempts to reconstruct the entire subtree under each node instead of the direct children only (a more detailed comparison will follow below). After node vectors are trained by the U.RAE, these embeddings are used to compute a similarity matrix that compares both the single words as well as all nonterminal node vectors in both sentences. In order to keep as much of the resulting global information of this comparison and deal with the arbitrary length of the two sentences, I then introduce the second component: a new *variable-sized min-pooling layer which outputs a fixed-size representation.* Any classifier such as logistic regression or a neural network can then be used to classify whether the two sentences are paraphrases or not.

I first describe the unsupervised feature learning with RAEs followed by a description of the pooling and following supervised classification. In experiments I show qualitative comparison of two different RAE models. I then describe my state-of-the-art results on the Microsoft Research Paraphrase (MSRP) Corpus introduced by Dolan et al. (2004) and discuss related work.

3.3.1 Recursive Autoencoders

In this section I contrast two variants of unsupervised recursive autoencoders which can be used to learn features from parse trees.

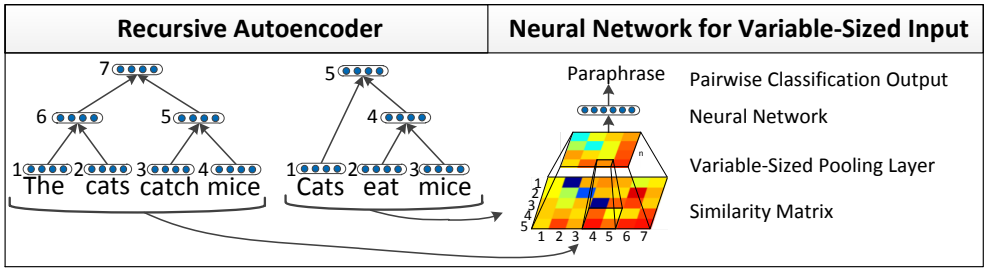


Figure 3.12: An overview of my paraphrase model. The recursive autoencoder learns phrase features for each node in a parse tree. The distances between all nodes then fill a similarity matrix which is given to a neural network architecture. Using a novel min-pooling layer the network can compare variable-sized sentences and classify pairs as being paraphrases or not.

Recursive Autoencoder

Fig. 3.13 (left) shows an instance of a standard recursive autoencoder (RAE, see the previous Sec. 3.2, repeated here for ease of exposition and contrast to the new model) applied to a given parse tree. Here, I assume that such a tree is given for each sentence by a parser. Assume I am given a list of word vectors $x = (x_1, \ldots, x_m)$ as described in the previous section. The binary parse tree for this input is in the form of branching triplets of parents with children: $(p \rightarrow c_1 c_2)$. The trees are given by a syntactic parser. Each child can be either an input word vector x_i or a nonterminal node in the tree. For both examples in Fig. 3.13, I have the following triplets: $((y_1 \rightarrow x_2 x_3), (y_2 \rightarrow x_1 y_1))$, $\forall x, y \in \mathbb{R}^n$.

Given this tree structure, I can now compute the parent representations. The first parent vector $p = y_1$ is computed from the children $(c_1, c_2) = (x_2, x_3)$ by the same standard neural network layer as in Eq. 3.17.

One way of assessing how well this n-dimensional vector represents its direct children is to decode their vectors in a reconstruction layer and then to compute the Euclidean distance between the original input and its reconstruction as in Eqs. 3.17 and 3.18.

In order to apply the autoencoder recursively, the same steps repeat. Now that y_1 is given, I can use Eq. 3.17 to compute y_2 by setting the children to be $(c_1, c_2) =$

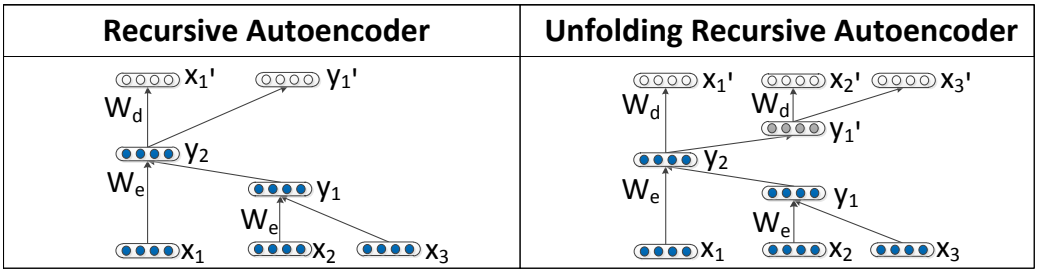


Figure 3.13: Two autoencoder models with details of the reconstruction at node y_2. For simplicity I left out the reconstruction layer at the first node y_1 which is the same standard autoencoder for both models. Left: A standard autoencoder that tries to reconstruct only its direct children. Right: The unfolding autoencoder which tries to reconstruct all leaf nodes underneath each node.

(x_1, y_1). Again, after computing the intermediate parent vector $p = y_2$, I can assess how well this vector captures the content of the children by computing the reconstruction error as in Eqs. 3.18 and 3.19. The process repeats until the full tree is constructed and each node has an associated reconstruction error.

During training, the goal is to minimize the reconstruction error of all input pairs at nonterminal nodes p in a given parse tree $\mathcal{T}$:

$$E_{rec}(\mathcal{T}) = \sum_{p \in \mathcal{T}} E_{rec}(p) \tag{3.26}$$

For the example in Fig. 3.13, I minimize $E_{rec}(\mathcal{T}) = E_{rec}(y_1) + E_{rec}(y_2)$.

Since the RAE computes the hidden representations it then tries to reconstruct, it could potentially lower reconstruction error by shrinking the norms of the hidden layers. In order to prevent this, I add a length normalization layer $p = p/||p||$ to this RAE model (referred to as the standard RAE). Another more principled solution is to use a model in which each node tries to reconstruct its entire subtree and then measure the reconstruction of the original leaf nodes. Such a model is described in the next section.

Unfolding Recursive Autoencoder

The unfolding RAE has the same encoding scheme as the standard RAE. The difference is in the decoding step which tries to reconstruct the entire spanned subtree underneath each node as shown in Fig. 3.13 (right). For instance, at node y_2, the reconstruction error is the difference between the leaf nodes underneath that node $[x_1; x_2; x_3]$ and their reconstructed counter parts. The unfolding produces the reconstructed leaves by starting at y_2 and computing

$$[x_1'; y_1'] = f(W_d y_2 + b_d). \tag{3.27}$$

Then it recursively splits y_1' again to produce vectors

$$[x_2'; x_3'] = f(W_d y_1' + b_d). \tag{3.28}$$

In general, I repeatedly use the decoding matrix W_d to unfold each node with the same tree structure as during encoding. The reconstruction error is then computed from a concatenation of the word vectors in that node's span. For a node y that spans words i to j:

$$E_{rec}(y_{(i,j)}) = \left|\left|[x_i; \ldots; x_j] - \left[x_i'; \ldots; x_j'\right]\right|\right|^2. \tag{3.29}$$

The unfolding autoencoder essentially tries to encode each hidden layer such that it best reconstructs its entire subtree to the leaf nodes. Hence, it will not have the problem of hidden layers shrinking in norm. Another potential problem of the standard RAE is that it gives equal weight to the last merged phrases even if one is only a single word (in Fig. 3.13, x_1 and y_1 have similar weight in the last merge). In contrast, the unfolding RAE captures the increased importance of a child when the child represents a larger subtree.

Deep Recursive Autoencoder

Both types of RAE can be extended to have multiple encoding layers at each node in the tree. Instead of transforming both children directly into parent p, I can have another hidden layer h in between. While the top layer at each node has to have the

same dimensionality as each child (in order for the same network to be recursively compatible), the hidden layer may have arbitrary dimensionality. For the two-layer encoding network, I would replace Eq. 3.17 with the following:

$$h = f(W_e^{(1)}[c_1; c_2] + b_e^{(1)}) \tag{3.30}$$

$$p = f(W_e^{(2)}p + b_e^{(2)}). \tag{3.31}$$

RAE Training

For training I use a set of parsed trees and then minimize the sum of all nodes' reconstruction errors. I compute the gradient efficiently via backpropagation through structure (Goller and Küchler, 1996). Even though the objective is not convex, I found that L-BFGS run with mini-batch training works well in practice. Convergence is smooth and the algorithm typically finds a good locally optimal solution quickly.

After the unsupervised training of the RAE, I can use the learned feature representations for several different tasks. The qualitative experiments below suggest numerous applications for such features.

3.3.2 An Architecture for Variable-Sized Matrices

Now that I have described the unsupervised feature learning, I explain how to use these features to classify sentence pairs as being in a paraphrase relationship or not.

Computing Sentence Similarity Matrices

My method incorporates both single word and phrase similarities in one framework. First, the RAE computes phrase vectors for the nodes in a given parse tree. I then compute Euclidean distances between all word and phrase vectors of the two sentences. These distances fill a similarity matrix $\mathcal{S}$ as shown in Fig. 3.12. For computing the similarity matrix, the rows and columns are first filled by the words in their original sentence order. I then add to each row and column the nonterminal nodes in a depth-first, right-to-left order.

Simply extracting aggregate statistics of this table such as the average distance or a histogram of distances cannot accurately capture the global *gist* of the similarities. For instance, paraphrases often have low or zero Euclidean distances in elements close to the diagonal of the similarity matrix. This happens when similar words align well between the two sentences. However, since the matrix dimensions vary based on the sentence lengths one cannot simply feed the similarity matrix into a standard neural network layer or logistic regression classifier.

Pooling

Consider a similarity matrix $\mathcal{S}$ generated by sentences of lengths n and m. Since the parse trees are binary and I also compare all nonterminal nodes, $\mathcal{S} \in \mathbb{R}^{(2n-1)\times(2m-1)}$. I would like to map S into a matrix S_{pooled} of fixed size, $n_p \times n_p$. My first step in constructing such a map is to partition the rows and columns of $\mathcal{S}$ into n_p roughly equal parts, producing an $n_p \times n_p$ grid.[8] I then define S_{pooled} to be the matrix of minimum values of each rectangular region within this grid, as shown in Fig. 3.14.

The matrix S_{pooled} loses some of the information contained in the original matrix, but it still captures much of its global structure. Since elements of S with small Euclidean distances show that there are similar phrases in both sentences, I keep this information by applying a min function to the pooling regions. Other functions, like averaging, are also possible, but might obscure the presence of similar phrases. This pooling layer could make use of overlapping pooling regions. For simplicity, I consider only non-overlapping pooling regions.

[8]The partitions will only be of equal size if $2n-1$ and $2m-1$ are divisible by n_p. I account for this in the following way, although many alternatives are possible. Let the number of rows of $\mathcal{S}$ be $R = 2n-1$. Each pooling window then has $\lfloor R/n_p \rfloor$ many rows. Let $M = R \mod n_p$, the number of remaining rows. I then evenly distribute these extra rows to the last M window regions which will have $\lfloor R/n_p \rfloor + 1$ rows. The same procedure applies to the number of columns for the windows. In the rare cases when $n_p > R$, the pooling layer needs to first up-sample. I achieve this by simply duplicating pixels row-wise until $R \leq n_p$. This procedure will have a slightly finer granularity for the single word similarities which is desired for my task since word overlap is a good indicator for paraphrases.

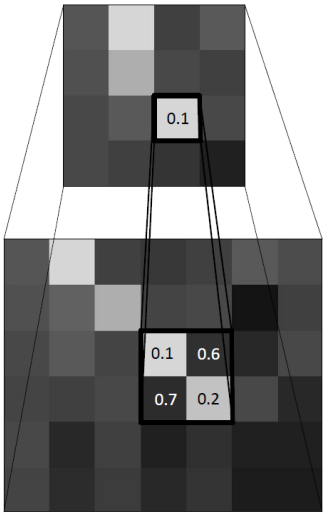


Figure 3.14: Example of the min-pooling layer finding the smallest number in a pooling window region of the original similarity matrix.

3.3.3 Experiments

For unsupervised RAE training I used a subset of 150,000 sentences from the NYT and AP sections of the Gigaword corpus. I used the Stanford parser (Klein and Manning, 2003a) to create the parse trees for all sentences. For initial word embeddings I used the 100-dimensional vectors computed via the unsupervised method of Collobert and Weston (2008) and provided by Turian et al. (2010).

For all paraphrase experiments I used the Microsoft Research paraphrase corpus(MSRP) introduced by Dolan et al. (2004). The dataset consists of 5,801 sentence pairs. The average sentence length is 21, the shortest sentence has 7 words and the longest 36. 3,900 are labeled as being in the paraphrase relationship (technically defined as "mostly bidirectional entailment"). I use the standard split of 4,076 training pairs (67.5% of which are paraphrases) and 1,725 test pairs (66.5 % paraphrases). All sentences were labeled by two annotators who agreed in 83% of the cases. A third annotator resolved conflicts. During dataset collection, negative examples were selected to have high lexical overlap to prevent trivial examples. For more information see Dolan et al. (2004); Das and Smith (2009).

As described in Sec. 3.3.1, I can have deep RAE networks with two encoding layers. In an initial experiment with a validation set I found that two layer RAEs always outperform single layer RAEs. The hidden RAE layer (see h in Eq. 3.31) has 200 units.

Center Phrase	Recursive Average	RAE	Unfolding RAE
the U.S.	the U.S. and German	the Swiss	the former U.S.
suffering low morale	suffering a 1.9 billion baht UNK 76 million	suffering due to no fault of my own	suffering heavy casualties
halted supplies to them	halted supplies to the Bosnian Serbs in August	halted supplies to the Bosnian Serbs in August	resume supplies to them
to watch hockey	to watch one Jordanian border policeman stamp the Israeli passports	to watch television	to watch a video
advance to the next round	advance to final qualifying round in Argentina	advance to the final of the UNK 1.1 million Kremlin Cup	advance to the semis
a prominent political figure	such a high-profile figure	a powerful business figure	the most visible and popular public figure

Table 3.7: Nearest neighbors of randomly chosen phrases of different lengths. The unfolding RAE captures most closely both syntactic and semantic similarities. R.Avg focuses only on the last merged words and incorrectly adds extra information.

Qualitative Evaluation of Nearest Neighbors

In order to show that the learned feature representations capture important semantic and syntactic information even for higher nodes in the tree, I visualize nearest neighbor phrases of varying length. After embedding sentences from the Gigaword corpus, I compute nearest neighbors for all nodes in all trees. In Table 3.7 the first phrase is a randomly chosen phrase and the remaining phrases are the closest phrases in the dataset that are not in the same sentence. I use Euclidean distance between the vector representations. Note that I do not constrain the neighbors to have the same word length. I compare the two autoencoder models above: RAE and unfolding RAE (U.RAE), as well as a recursive averaging baseline (R.Avg). R.Avg recursively takes the average of both child vectors in the tree.

Table 3.7 shows several interesting phenomena. Recursive averaging is almost entirely focused on an exact string match of the last merged words of the current phrase in the tree. This leads the nearest neighbors to incorrectly add various extra information which breaks the paraphrase relationship. The standard RAE does well though it is also somewhat focused on the last merges in the tree. Finally, the unfolding RAE captures most closely the underlying syntactic and semantic structure. However, like all these methods it can suffer from word vectors that are only based on small co-occurrence windows. Namely, some words that appear in similar contexts might be antonyms, for instance to "halt" and "resume".

Reconstructing Phrases via Recursive Decoding

In this section I analyze the information captured by the unfolding RAE's 100-dimensional phrase vectors. I show that these 100-dimensional vector representations can not only capture and memorize single words but also longer, unseen phrases.

In order to show how much of the information can be recovered I recursively reconstruct sentences after encoding them. The process is similar to unfolding during training. It starts from a phrase vector of a nonterminal node in the parse tree. I then unfold the tree as given during encoding and find the nearest neighbor word to each of the reconstructed leaf node vectors. Table 3.8 shows that the RAE can very well reconstruct phrases of up to length five. No other method that I compared had such reconstruction capabilities. Longer phrases retain some correct words and usually the correct part of speech but the semantics of the words get merged.

Evaluation on Full-Sentence Paraphrasing

I now turn to evaluating the unsupervised features and their performance as input to my neural network architecture in my main task of paraphrase detection. My neural network consists of the min-pooling layer, one hidden layer, followed by a softmax classification layer. In the similarity matrix $\mathcal{S}$, I ignore stop words which I set conservatively to be the 10 most frequent words.

Methods which are based purely on vector representations invariably lose some

Encoding Input	Generated Text from Unfolded Reconstruction
a flight	a flight
the safety	the safety
of a flight	of a flight
the first qualifying session	the first qualifying session
Most of the victims	Most of the rebels
the signing of the accord	the signing of the accord
the safety of a flight	the safety of a flight
the U.S. House of Representatives	the Swiss House of Representatives
visit and discuss investment possibilities	visit and postpone financial possibilities
the agreement it made with Malaysia	the agreement it released for Mexico
the safety of a flight and illegal possession of a weapon	the labour of the base and municipal taxpayers of the language
a pocket knife was found in his suitcase in the plane 's cargo hold	a fatal knife was concentrated in the order in the European national gun rally

Table 3.8: Original inputs for encoding and the generated output from decoding and displaying the nearest neighbor word. The unfolding RAE can reconstruct perfectly phrases with up to 5 words. Longer phrases start to get incorrect nearest neighbor words. For other methods good reconstructions are only possible for at most two or three words.

information. For instance, numbers often have very similar representations, but even small differences are crucial to reject the paraphrase relation in this dataset. Hence, I add a feature which is 1 if two sentences contain exactly the same numbers and 0 otherwise. Since my pooling-layer ignores sentence length, I also add the difference in sentence length. Lastly, the number of exact string matches in the parse trees gets lost in the pooling layer, so I add this number as a separate feature to all methods and also report performance without it (only $\mathcal{S}$).

For all of my models, I perform 10-fold cross-validation on the training set to choose the best regularization parameters and the size of the hidden neural network layer. Based on an initial exploration on the training data, I fix the size of the pooling layer to $n_p = 15$, slightly less than the average sentence length.

Unsupervised	Acc.	F1
R.Avg	75.9	82.9
RAE 2L	75.5	82.7
U.RAE 2L	76.4	83.4

Supervised	Acc.	F1
RAE-Hist	72.0	81.7
Only Feat.	73.2	81.8
Only $\mathcal{S}$	72.6	81.1
Logistic Reg.	74.2	82.0

Table 3.9: Test results on the MSRP paraphrase corpus. Comparisons of unsupervised feature learning methods (left) and similarity feature extraction and supervised classification methods (right).

In my first set of experiments shown in Table 3.9 (left) I compare several unsupervised learning methods: Recursive average baseline (R.Avg.) as defined in Sec. 3.3.3, two-layered RAEs and unfolding RAEs (U.RAE). With the help of my powerful pooling-layer architecture and good initial word vectors the simple RAE and recursive averaging both perform well, only the unfolding RAE can learn compositional features beyond the initial word vectors and identify the more complex paraphrase relationships to improve accuracy. The performance improvement of the unfolding RAE is statistically significant over the R.Avg under a paired t-test ($p < 0.06$).

Next, I compare my min-pooling neural network to simpler feature extraction and supervised learning methods as shown in Table 3.9 (right). This table shows that each of the components of my architecture is important for achieving high accuracy. For every setting I exhaustively cross-validate on the training data over all possible

parameters (unsupervised features, logistic regression, hidden layer sizes) and report the best performance. The settings are:
(i) RAE-Hist. This method replaces the min-pooling layer with a histogram of similarities in the matrix $\mathcal{S}$. The low performance shows that my variable-sized min-pooling layer is much better able to capture the global similarity information than simple aggregate statistics of $\mathcal{S}$.
(ii) Only Feat. This method uses only the string matching feature described above and shows that simple binary string and number matching can detect many of the simple paraphrases but fails to detect more complex cases.
(iii) Only $\mathcal{S}$. This baseline gives the performance of the unfolding RAE without the string matching feature showing that some of the string matching information gets lost in $\mathcal{S}_{\text{pooled}}$.
(iv) Logistic Reg. This setup replaces the single-layer neural network with just logistic regression. This cannot capture some of the interactions that are in the similarity matrix.

Table 3.10 shows my results compared to previous approaches (most are explained in Sec. 3.3.4 below). My unfolding RAE achieves state-of-the-art performance without numerous hand-designed semantic features such as WordNet.

Model	Acc.	F1
All Paraphrase Baseline	66.5	79.9
Rus et al. (2008)	70.6	80.5
Mihalcea et al. (2006)	70.3	81.3
Islam and Inkpen (2007)	72.6	81.3
Qiu et al. (2006)	72.0	81.6
Fernando and Stevenson (2008)	74.1	82.4
Wan et al. (2006)	75.6	83.0
Das and Smith (2009)	73.9	82.3
Das and Smith (2009)+Feat.	76.1	82.7
U.RAE 2L (my method)	**76.4**	**83.4**

Table 3.10: Test results on the MSRP paraphrase corpus. Comparison to previous methods. See text for details.

In Table 3.11 I show several examples of correctly classified paraphrase candidate pairs together with their similarity matrix after min-pooling. The first and last pair

are simple cases of paraphrase and not paraphrase. The second example shows a pooled similarity matrix when large chunks are swapped in both sentences. My model is very robust to such transformations and gives a high probability to this pair. Even more complex examples such as the third with very little direct string matches (few blue squares) are correctly classified. The second to last example is highly interesting. Even though there is a clear diagonal with good string matches, the gap in the center shows that the first sentence contains much extra information. This is also captured by my model.

3.3.4 Related Work

Paraphrase Detection The field of paraphrase detection has progressed immensely in recent years. Early approaches were based purely on lexical matching techniques (Barzilay and Lee, 2003; Zhang and Patrick, 2005; Qiu et al., 2006; Kozareva and Montoyo, 2006). Since these methods are often based on exact string matches of n-grams, they fail to detect similar meaning that is conveyed by synonymous words. Several approaches overcome this problem by using Wordnet- and corpus-based semantic similarity measures (Mihalcea et al., 2006; Islam and Inkpen, 2007). In their approach they choose for each open-class word the single most similar word in the other sentence. Fernando and Stevenson (2008) improved upon this idea by computing a similarity matrix that captures all pair-wise similarities of single words in the two sentences. They then threshold the elements of the resulting similarity matrix and compute the mean of the remaining entries. There are two shortcomings of such methods: They ignore (i) the syntactic structure of the sentences (by comparing only single words) and (ii) the global structure of such a similarity matrix (by computing only the mean).

Instead of comparing only single words, Wan et al. (2006) adds features from dependency parses. Most recently, Das and Smith (2009) adopted the idea that paraphrases have related syntactic structure. Their quasi-synchronous grammar formalism incorporates a variety of features from WordNet, a named entity recognizer, a part-of-speech tagger, and the dependency labels from the aligned trees. In order

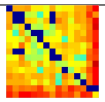
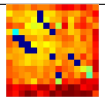
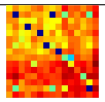
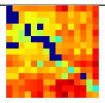
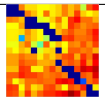
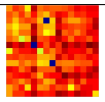
L	Pr	Sentences	Sim.Mat.
P	0.95	(1) LLEYTON Hewitt yesterday traded his tennis racquet for his first sporting passion - Australian football - as the world champion relaxed before his Wimbledon title defence (2) LLEYTON Hewitt yesterday traded his tennis racquet for his first sporting passion-Australian rules football-as the world champion relaxed ahead of his Wimbledon defence	
P	0.82	(1) The lies and deceptions from Saddam have been well documented over 12 years (2) It has been well documented over 12 years of lies and deception from Saddam	
P	0.67	(1) Pollack said the plaintiffs failed to show that Merrill and Blodget directly caused their losses (2) Basically , the plaintiffs did not show that omissions in Merrill's research caused the claimed losses	
N	0.49	(1) Prof Sally Baldwin , 63 , from York , fell into a cavity which opened up when the structure collapsed at Tiburtina station , Italian railway officials said (2) Sally Baldwin , from York , was killed instantly when a walkway collapsed and she fell into the machinery at Tiburtina station	
N	0.44	(1) Bremer , 61 , is a onetime assistant to former Secretaries of State William P. Rogers and Henry Kissinger and was ambassador-at-large for counterterrorism from 1986 to 1989 (2) Bremer , 61 , is a former assistant to former Secretaries of State William P. Rogers and Henry Kissinger	
N	0.11	(1) The initial report was made to Modesto Police December 28 (2) It stems from a Modesto police report	

Table 3.11: Examples of sentence pairs with ground truth labels (P - Paraphrase N - Not Paraphrase), the probabilities my model assigns to them for being a paraphrase ($Pr(S_1, S_2) > 0.5$ is assigned the label Paraphrase) and their similarity matrices after min-pooling. Simple paraphrase pairs have clear diagonal structure due to perfect word matches with Euclidean distance 0 (dark blue). That structure is preserved by my min-pooling layer. Matrices best viewed in color. See text for more details.

to obtain high performance they combine their parsing-based model with a logistic regression model that uses 18 hand-engineered surface features.

I merge these word-based models and syntactic models in one joint framework: The matrix consists of phrase similarities and instead of just taking the mean of the similarities I can capture the global layout of the matrix via the min-pooling layer.

Autoencoder The work related to autoencoders has already been described in Sec. 3.2.4. In addition to the work mentioned in that section, Bottou (2011) recently discussed related ideas of recursive autoencoders and recursive image and text understanding but without experimental results. Larochelle et al. (2009) investigated autoencoders with an unfolded "deep objective".

3.4 Conclusion

This concludes the chapter on RNN objective functions, the choice of which will follow largely from the problem definition.

Every RNN has an associated objective function. The next chapter concentrates on another important aspect: the composition or encoding function which so far has been a standard neural network. The objective functions that train these composition functions will be similar to those of this chapter: scores and softmax classifiers as in Eq. 3.22 in section 3.2.1.

Chapter 4

Recursive Composition Functions

The previous chapter introduced standard RNNs and introduced the main objective functions. This chapter investigates more powerful RNN architectures that move beyond having the same, standard neural network for composing parent vectors as in the previous chapter. The main objective functions, I explored are

1. Syntactically untied RNNs: The composition matrix that is used to compute the parent vector depends on the syntactic category of the children.

2. Matrix Vector RNNs: This is a more extreme form of untying, in which every composition matrix depends on the exact words in all the leaves underneath each node. This is the result of every single word having its own associated matrix that is multiplied with the vector of the sibling node.

3. Recursive Neural Tensor Networks: This section introduces an entirely new type of neural network layer based on the idea that there are multiple multiplicative effects when two word vectors are combined. Empirically, this outperforms all previous neural architectures on tasks such as sentiment analysis.

Each model is motivated by a specific language task, such as parsing or sentiment analysis. The ideas in this chapter are quite general and could also be combined. For instance, one could syntactically untie the recursive neural tensor network or cluster the Matrices and hence arrive at a similar structure as the syntactically untied RNN.

4.1 Syntactically Untied Recursive Neural Networks - For Natural Language Parsing

The models of previous sections did not make much use of linguistically motivated syntactic theory. Motivated by a long history of parsing models, the model in this section differs and combines linguistic knowledge with deep feature learning.

While Sec. 3.1 gave an introduction to general image and sentence parsing this section is more focused on the specifics of parsing natural language. I revisit syntactic language parsing because it is a central task in natural language processing due to its importance in mediating between linguistic expression and meaning. Much work has shown the usefulness of syntactic representations for subsequent NLP tasks such as relation extraction, semantic role labeling (Gildea and Palmer, 2002) and paraphrase detection (Callison-Burch, 2008).

Apart from my work in Sec. 3.1, syntactic descriptions standardly use coarse discrete categories such as NP for noun phrases or PP for prepositional phrases. However, recent work has shown that parsing results can be greatly improved by defining more fine-grained syntactic categories, which better capture phrases with similar behavior, whether through manual feature engineering (Klein and Manning, 2003a) or automatic learning (Petrov et al., 2006). However, subdividing a category like NP into 30 or 60 subcategories can only provide a very limited representation of phrase meaning and semantic similarity. Two strands of work therefore attempt to go further. First, recent work in discriminative parsing has shown gains from careful engineering of features (Taskar et al., 2004; Finkel et al., 2008). Features in such parsers can be seen as defining effective dimensions of similarity between categories. Second, lexicalized parsers (Collins, 2003; Charniak, 2000) associate each category with a lexical item. This gives a fine-grained notion of semantic similarity, which is useful for tackling problems like ambiguous attachment decisions. However, this approach necessitates complex shrinkage estimation schemes to deal with the sparsity of observations of the lexicalized categories.

In many natural language systems, single words and n-grams are usefully described by their distributional similarities (Brown et al., 1992), among many others. But, even

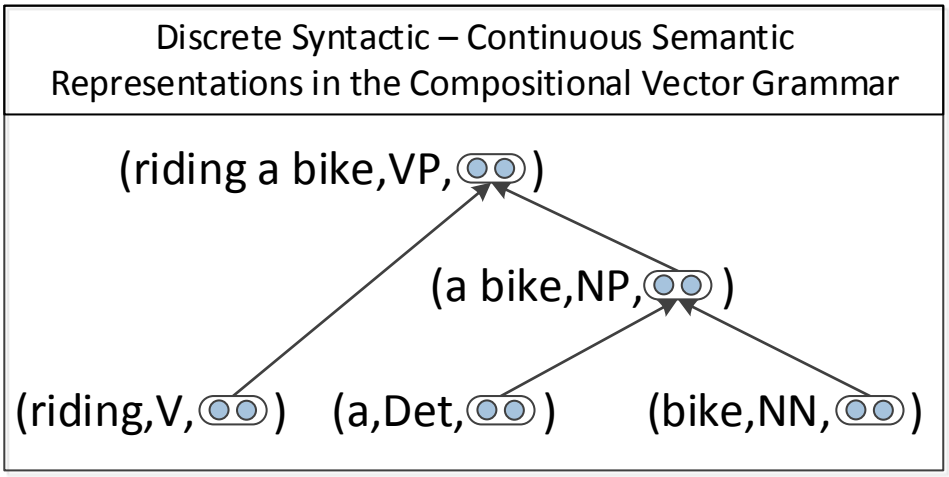


Figure 4.1: Example of a CVG tree with (category,vector) representations at each node. The vectors for nonterminals are computed via a new type of recursive neural network which is conditioned on syntactic categories from a PCFG.

with large corpora, many n-grams will never be seen during training, especially when n is large. In these cases, one cannot simply use distributional similarities to represent unseen phrases.

In this section, I present an RNN-based model to learn features and phrase representations even for very long, unseen n-grams. This model generalizes RNN models from the previous chapter. In particular, I introduce a Compositional Vector Grammar Parser (CVG) for structure prediction. Like the above work on parsing, the model addresses the problem of representing phrases and categories. Unlike them, it jointly learns how to parse and how to represent phrases as both discrete categories and continuous vectors as illustrated in Fig. 4.1. CVGs combine the advantages of standard probabilistic context free grammars (PCFGs) with those of recursive neural networks (RNNs). The former can capture the discrete categorization of phrases into NP or PP while the latter can capture fine-grained syntactic and compositional-semantic information on phrases and words. This information can help in cases where syntactic ambiguity can only be resolved with semantic information, such as in the PP attachment of the two sentences: *They ate udon with forks.* vs. *They ate udon*

with chicken.

The RNN-based parser of Sec. 3.1 used the same (tied) weights at all nodes to compute the vector representing a constituent. This requires the composition function to be extremely powerful, since it has to combine phrases with different syntactic head words, and it is hard to optimize since the parameters form a very deep neural network. I generalize the fully tied RNN to one with syntactically untied weights. The weights at each node are conditionally dependent on the categories of the child constituents. This allows different composition functions when combining different types of phrases and is shown to result in a large improvement in parsing accuracy.

My compositional distributed representation allows a CVG parser to make accurate parsing decisions and capture similarities between phrases and sentences. Any PCFG-based parser can be improved with an RNN. I use a simplified version of the Stanford Parser (Klein and Manning, 2003a) as the base PCFG and improve its accuracy from 86.56 to 90.44% labeled F1 on all sentences of the WSJ section 23.

4.1.1 Compositional Vector Grammars

This section introduces Compositional Vector Grammars (CVGs), a model to jointly find syntactic structure and capture compositional semantic information.

CVGs build on two observations. Firstly, that a lot of the structure and regularity in languages can be captured by well-designed syntactic patterns. Hence, the CVG builds on top of a standard PCFG parser. However, many parsing decisions show fine-grained semantic factors at work. Therefore I combine syntactic and semantic information by giving the parser access to rich syntactico-semantic information in the form of distributional word vectors and compute compositional semantic vector representations for longer phrases (Costa et al., 2003; Menchetti et al., 2005; Socher et al., 2011b). The CVG model merges ideas from both generative models that assume discrete syntactic categories and discriminative models that are trained using continuous vectors.

Word Vector Representations

I use the same word vectors as described in detail in Sec. 2.3.

For ease of exposition, I define again the matrix with all the word embeddings as L. As before, the resulting L matrix is used as follows: Assume I am given a sentence as an ordered list of m words. Each word w has an index $[w] = i$ into the columns of the embedding matrix. This index is used to retrieve the word's vector representation a_w using a simple multiplication with a binary vector e, which is zero everywhere, except at the ith index. So $a_w = Le_i \in \mathbb{R}^n$. Henceforth, after mapping each word to its vector, I represent a sentence S as an ordered list of (word,vector) pairs: $x = ((w_1, a_{w_1}), \ldots, (w_m, a_{w_m}))$.

Now that I have discrete and continuous representations for all words, I can continue with the approach for computing tree structures and vectors for nonterminal nodes.

Max-Margin Training Objective for CVGs

The goal of supervised parsing is to learn a function $g : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ is the set of sentences and $\mathcal{Y}$ is the set of all possible labeled binary parse trees. The set of all possible trees for a given sentence x_i is defined as $Y(x_i)$ and the correct tree for a sentence is y_i.

The structured margin loss I will now describe now is very similar to that of Sec. 3.1.2. There are two main differences. The first one is that I am using a CVG here which is not only based on an RNN. The second difference is described after Eq. 4.4 just below.

I first define a structured margin loss $\Delta(y_i, \hat{y})$ for predicting a tree $\hat{y}$ for a given correct tree. The loss increases the more incorrect the proposed parse tree is (Goodman, 1998). The discrepancy between trees is measured by counting the number of nodes $N(y)$ with an incorrect span (or label) in the proposed tree:

$$\Delta(y_i, \hat{y}) = \sum_{d \in N(\hat{y})} \kappa \mathbf{1}\{d \notin N(y_i)\}. \tag{4.1}$$

I set $\kappa = 0.1$ in all experiments. For a given set of training instances (x_i, y_i), I search for the function g_θ, parameterized by θ, with the smallest expected loss on a new sentence. It has the following form:

$$g_\theta(x) = \arg\max_{\hat{y} \in Y(x)} s(\texttt{CVG}(\theta, x, \hat{y})), \tag{4.2}$$

where the tree is found by the Compositional Vector Grammar (CVG) introduced below and then scored via the function s. The higher the score of a tree the more confident the algorithm is that its structure is correct. This max-margin, structure-prediction objective (Taskar et al., 2004; Ratliff et al., 2007; Socher et al., 2011b) trains the CVG so that the highest scoring tree will be the correct tree: $g_\theta(x_i) = y_i$ and its score will be larger up to a margin to other possible trees $\hat{y} \in \mathcal{Y}(x_i)$:

$$s(\texttt{CVG}(\theta, x_i, y_i)) \geq s(\texttt{CVG}(\theta, x_i, \hat{y})) + \Delta(y_i, \hat{y}) \tag{4.3}$$

This leads to the regularized risk function for m training examples:

$$\begin{aligned} J(\theta) &= \frac{1}{m}\sum_{i=1}^{m} r_i(\theta) + \frac{\lambda}{2}\|\theta\|_2^2, \text{ where} \\ r_i(\theta) &= \max_{\hat{y} \in Y(x_i)} \big(s(\texttt{CVG}(x_i, \hat{y})) + \Delta(y_i, \hat{y})\big) - s(\texttt{CVG}(x_i, y_i)) \end{aligned} \tag{4.4}$$

Intuitively, to minimize this objective, the score of the correct tree y_i is increased and the score of the highest scoring incorrect tree $\hat{y}$ is decreased.

Note the similarity to the max-margin loss of Eq. 3.5 in Sec. 3.1.2. The equation here is a special case since it assumes there is only a single correct tree instead of an equivalent set of correct trees.

Scoring Trees with CVGs

For ease of exposition, I first summarize how to score an existing fully labeled tree with a standard RNN and then contrast that process in detail to that of the CVG. For more details on the standard RNN see Sec. 3.1.2. The subsequent section will then describe a bottom-up beam search and its approximation for finding the optimal

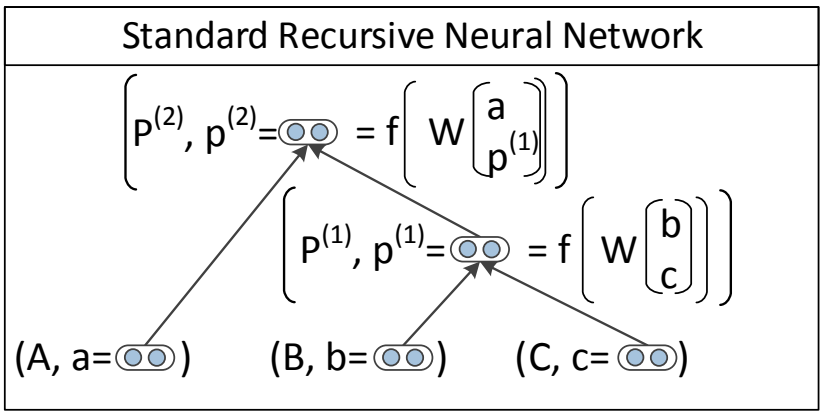


Figure 4.2: An example tree with a simple Recursive Neural Network: The same weight matrix is replicated and used to compute all non-terminal node representations.

tree.

Assume, for now, I am given a labeled parse tree as shown in Fig. 4.2. I define the word representations as (vector, POS) pairs: $((a, A), (b, B), (c, C))$, where the vectors are defined as in sections 4.1.1 and 2.3 and the POS tags come from a PCFG.

The standard RNN as defined in Sec. 3.1.2 and in Fig. 4.2 essentially ignores all POS tags and syntactic categories and each non-terminal node is associated with the same neural network (i.e., the weights across nodes are fully tied). In summary, it computes the non-terminals via the following equations:

$$p^{(1)} = f\left(W \begin{bmatrix} b \\ c \end{bmatrix}\right), \; p^{(2)} = f\left(W \begin{bmatrix} a \\ p^1 \end{bmatrix}\right)$$

The parent scores for all nodes i are computed via:

$$s(p^{(i)}) = v^T p^{(i)}.$$

The standard RNN requires a single composition function to capture all types of compositions: adjectives and nouns, verbs and nouns, adverbs and adjectives, etc.

Even though this function is a powerful one, I found a single neural network weight matrix cannot fully capture the richness of compositionality. Several extensions are possible: A two-layered RNN would provide more expressive power, however, it is much harder to train because the resulting neural network becomes very deep and suffers from vanishing gradient problems.

Based on the above considerations, I propose the Compositional Vector Grammar (CVG) that conditions the composition function at each node on discrete syntactic categories extracted from a PCFG. Hence, CVGs combine discrete, syntactic rule probabilities and continuous vector compositions. The idea is that the syntactic categories of the children determine what composition function to use for computing the vector of their parents. While not perfect, a dedicated composition function for each rule RHS can well capture common composition processes such as adjective or adverb modification versus noun or clausal complementation. For instance, it could learn that an NP should be similar to its head noun and little influenced by a determiner, whereas in an adjective modification both words considerably determine the meaning of a phrase. The original RNN is parameterized by a single weight matrix W. In contrast, the CVG uses a syntactically untied RNN (SU-RNN) which has a set of such weights. The size of this set depends on the number of sibling category combinations in the PCFG.

Fig. 4.3 shows an example SU-RNN that computes parent vectors with syntactically untied weights. The CVG computes the first parent vector via the SU-RNN:

$$p^{(1)} = f\left(W^{(B,C)}\begin{bmatrix} b \\ c \end{bmatrix}\right),$$

where $W^{(B,C)} \in \mathbb{R}^{n\times 2n}$ is now a matrix that depends on the categories of the two children. In this bottom up procedure, the score for each node consists of summing two elements: First, a single linear unit that scores the parent vector and second, the log probability of the PCFG for the rule that combines these two children:

$$s\left(p^{(1)}\right) = \left(v^{(B,C)}\right)^T p^{(1)} + \log P(P_1 \rightarrow B\ \ C), \tag{4.5}$$

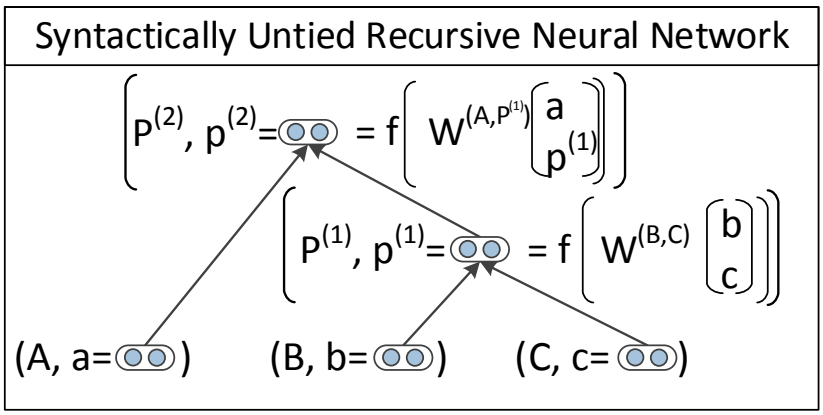


Figure 4.3: Example of a syntactically untied RNN in which the function to compute a parent vector depends on the syntactic categories of its children which I assume are given for now.

where $P(P_1 \rightarrow B\ C)$ comes from the PCFG. This can be interpreted as the log probability of a discrete-continuous rule application with the following factorization:

$$\begin{aligned} &P((P_1, p_1) \rightarrow (B, b)(C, c)) \\ &= P(p_1 \rightarrow b\ \ c | P_1 \rightarrow B\ \ C) P(P_1 \rightarrow B\ \ C), \end{aligned} \tag{4.6}$$

Note, however, that due to the continuous nature of the word vectors, the probability of such a CVG rule application is not comparable to probabilities provided by a PCFG since the latter sum to 1 for all children.

Assuming that node p_1 has syntactic category P_1, I compute the second parent vector via:

$$p^{(2)} = f\left(W^{(A,P_1)} \begin{bmatrix} a \\ p^{(1)} \end{bmatrix}\right).$$

The score of the last parent in this trigram is computed via:

$$s\left(p^{(2)}\right) = \left(v^{(A,P_1)}\right)^T p^{(2)} + \log P(P_2 \rightarrow A\ \ P_1).$$

Parsing with CVGs

The above scores (Eq. 4.5) are used in the search for the correct tree for a sentence. The goodness of a tree is measured in terms of its score and the CVG score of a complete tree is the sum of the scores at each node:

$$s(\texttt{CVG}(\theta, x, \hat{y})) = \sum_{d \in N(\hat{y})} s\left(p^d\right). \tag{4.7}$$

The main objective function in Eq. 4.4 includes a maximization over all possible trees $\max_{\hat{y} \in Y(x)}$. Finding the global maximum, however, cannot be done efficiently for longer sentences nor can I use dynamic programming. This is due to the fact that the vectors break the independence assumptions of the base PCFG. A (category, vector) node representation is dependent on all the words in its span and hence to find the true global optimum, I would have to compute the scores for all binary trees. For a sentence of length n, there are Catalan(n) many possible binary trees which is very large even for moderately long sentences.

One could also use a bottom-up beam search (Manning and Schütze, 1999). In that case, one would keep a k-best list at every cell of the chart, possibly for each syntactic category. However, such a beam search inference procedure is still considerably slower than using only the simplified base PCFG, especially since it has a small state space. Since each probability look-up is cheap but computing SU-RNN scores requires a matrix product, I would like to reduce the number of SU-RNN score computations to only those trees that require semantic information. I note that labeled F1 of the Stanford PCFG parser on the test set is 86.17%. However, if one used an oracle to select the best tree from the top 200 trees that it produces, one could get an F1 of 95.46%.

I use this knowledge to speed up inference via two bottom-up passes through the parsing chart. During the first one, I use only the base PCFG to run CKY dynamic programming through the tree. The $k = 200$-best parses at the top cell of the chart are calculated using the efficient algorithm of Huang and Chiang (2005). Then, the second pass is a beam search with the full CVG model (including the more expensive

matrix multiplications of the SU-RNN). This beam search only considers phrases that appear in the top 200 parses. This is similar to a re-ranking setup but with one main difference: the SU-RNN rule score computation at each node still only has access to its child vectors, not the whole tree or other global features. This allows the second pass to be very fast. I use this setup in my experiments below.

Training SU-RNNs

The full CVG model is trained in two stages. First the base PCFG is trained and its top trees are cached and then used for training the SU-RNN conditioned on the PCFG. The SU-RNN is trained using the objective in Eq. 4.4 and the scores as exemplified by Eq. 4.7. For each sentence, I use the method described above to efficiently find an approximation for the optimal tree.

To minimize the objective I want to increase the scores of the correct tree's constituents and decrease the score of those in the highest scoring incorrect tree. Derivatives are computed via backpropagation through structure as described in Sec. 3.1.4. The derivative of tree i has to be taken with respect to all parameter matrices $W^{(AB)}$ that appear in it. The main difference between backpropagation in standard RNNs and SU-RNNs is that the derivatives at each node only add to the overall derivative of the specific matrix at that node. For more details on backpropagation through RNNs, see Sec. 3.1.4.

Subgradient and AdaGrad

The objective function is not differentiable due to the hinge loss. As described in Sec. 2.6, I use the subgradient method (Ratliff et al., 2007) which computes a gradient-like direction. Let $\theta = (X, W^{(\cdot\cdot)}, v^{(\cdot\cdot)}) \in \mathbb{R}^M$ be a vector of all M model parameters, where I denote $W^{(\cdot\cdot)}$ as the set of matrices that appear in the training set. The subgradient of Eq. 4.4 becomes:

$$\frac{\partial J}{\partial \theta} = \sum_i \frac{\partial s(x_i, \hat{y}_{\max})}{\partial \theta} - \frac{\partial s(x_i, y_i)}{\partial \theta} + \theta,$$

where $\hat{y}_{\max}$ is the tree with the highest score. To minimize the objective, I use the diagonal variant of AdaGrad (Duchi et al., 2011) with minibatches which is also described in Sec. 2.6. Adapting the learning rate for each parameter is particularly helpful in the SU-RNN since some W matrices appear in only a few training trees. This procedure found much better optima (by ≈3% labeled F1 on the dev set), and converged more quickly than L-BFGS which I had used in RNN training in previous sections. Training time is roughly 4 hours on a single machine.

Initialization of Weight Matrices

In the absence of any knowledge on how to combine two categories, my prior for combining two vectors is to average them instead of performing a completely random projection. Hence, I initialize the binary W matrices with:

$$W^{(\cdot\cdot)} = 0.5[I_{n\times n} I_{n\times n} 0_{n\times 1}] + \epsilon,$$

where I include the bias in the last column and the random variable is uniformly distributed: $\epsilon \sim \mathcal{U}[-0.001, 0.001]$. The first block is multiplied by the left child and the second by the right child:

$$\begin{aligned} W^{(AB)} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix} &= \left[W^{(A)} W^{(B)} bias\right] \begin{bmatrix} a \\ b \\ 1 \end{bmatrix} \\ &= W^{(A)} a + W^{(B)} b + bias, \end{aligned}$$

where, for notational convenience, I changed the notation for the *bias* vector which was b in previous chapters.

4.1.2 Experiments

I evaluate the CVG in two ways: First, by a standard parsing evaluation on Penn Treebank WSJ and then by analyzing the model errors in detail.

Cross-validating Hyperparameters

I used the first 20 files of WSJ section 22 to cross-validate several model and optimization choices. The base PCFG uses simplified categories of the Stanford PCFG Parser (Klein and Manning, 2003a). I decreased the state splitting of the PCFG grammar (which helps both by making it less sparse and by reducing the number of parameters in the SU-RNN) by adding the following options to training: '-noRightRec -dominatesV 0 -baseNP 0'. This reduces the number of states from 15,276 to 12,061 states and 602 POS tags. These include split categories, such as parent annotation categories like VP^S. Furthermore, I ignore all category splits for the SU-RNN weights, resulting in 66 unary and 882 binary child pairs. Hence, the SU-RNN has 66+882 transformation matrices and scoring vectors. Note that any PCFG, including latent annotation PCFGs (Matsuzaki et al., 2005) could be used. However, since the vectors will capture lexical and semantic information, even simple base PCFGs can be substantially improved. Since the computational complexity of PCFGs depends on the number of states, a base PCFG with fewer states is much faster.

Testing on the full WSJ section 22 dev set (1700 sentences) takes roughly 470 seconds with the simple base PCFG, 1320 seconds with my new CVG and 1600 seconds with the currently published Stanford factored parser. Hence, increased performance comes also with a speed improvement of approximately 20%.

I fix the same regularization of $\lambda = 10^{-4}$ for all parameters. The minibatch size was set to 20. I also cross-validated on AdaGrad's learning rate which was eventually set to $\alpha = 0.1$ and word vector size. The 25-dimensional vectors provided by Turian et al. (2010) provided the best performance and were faster than 50-,100- or 200-dimensional ones. I hypothesize that the larger word vector sizes, while capturing more semantic knowledge, result in too many SU-RNN matrix parameters to train and hence perform worse.

Results on WSJ

The best single model model obtains a labeled F1 on all sentences of 90.0. One can ensemble two SU-RNNs to obtain 90.9% F1 on all dev set sentences. This model

resulted in 90.4% on the final test set (WSJ section 23). Table 4.1 compares my results to the two Stanford parser variants (the unlexicalized PCFG (Klein and Manning, 2003a) and the factored parser (Klein and Manning, 2003b)) and other parsers that use richer state representations: the Berkeley parser (Petrov and Klein, 2007), Collins parser (Collins, 1997), SSN: a statistical neural network parser (Henderson, 2004), Factored PCFGs (Hall and Klein, 2012), Charniak-SelfTrain: the self-training approach of McClosky et al. (2006), which bootstraps and parses additional large corpora multiple times, Charniak-RS: the state of the art self-trained and discriminatively re-ranked Charniak-Johnson parser combining (Charniak, 2000; McClosky et al., 2006; Charniak and Johnson, 2005). See Kummerfeld et al. (2012) for more comparisons and the related work section for a brief introduction to these parsing approaches. I compare also to a standard RNN 'CVG (RNN)', to the proposed CVG with SU-RNNs and a model that uses two randomly initialized SU-RNNs in an ensemble.

Parser	dev (all)	test$\leq$ 40	test (all)
Stanford PCFG	85.8	86.2	85.5
Stanford Factored	87.4	87.2	86.6
Factored PCFGs	89.7	90.1	89.4
Collins			87.7
SSN (Henderson)			89.4
Berkeley Parser			90.1
CVG (RNN)	85.7	85.1	85.0
CVG (SU-RNN)			90.0
CVG (2$\times$SU-RNN)	91.2	91.1	90.4
Charniak-SelfTrain			91.0
Charniak-RS			92.1

Table 4.1: Comparison of parsers with richer state representations on the WSJ. The last line is the self-trained re-ranked Charniak parser.

Model Analysis

Analysis of Error Types. Table 4.2 shows a detailed comparison of different errors. In particular, it shows the average number of bracket errors per sentence for several

error types. I use the code provided by Kummerfeld et al. (2012) and compare to the previous version of the Stanford factored parser as well as to the Berkeley and Charniak-reranked-self-trained parsers (defined above). See Kummerfeld et al. (2012) for details and comparisons to other parsers. One of the largest sources of improved performance over the original Stanford factored parser is in the correct placement of PP phrases. When measuring only the F1 of parse nodes that include at least one PP child, the CVG improves the Stanford parser by 6.2% to an F1 of 77.54%. This is a 0.23 reduction in the average number of bracket errors per sentence. The 'Other' category includes VP, PRN and other attachments, appositives and internal structures of modifiers and QPs.

Error Type	Stanford	CVG	Berkeley	Char-RS
PP Attach	1.02	0.79	0.82	0.60
Clause Attach	0.64	0.43	0.50	0.38
Diff Label	0.40	0.29	0.29	0.31
Mod Attach	0.37	0.27	0.27	0.25
NP Attach	0.44	0.31	0.27	0.25
Co-ord	0.39	0.32	0.38	0.23
1-Word Span	0.48	0.31	0.28	0.20
Unary	0.35	0.22	0.24	0.14
NP Int	0.28	0.19	0.18	0.14
Other	0.62	0.41	0.41	0.50

Table 4.2: Detailed comparison of different parsers. Numbers are bracket errors per sentence, lower is better.

Analysis of Composition Matrices. An analysis of the norms of the binary matrices reveals that the model learns a soft vectorized notion of head words: Head words are given larger weights and importance when computing the parent vector: For the matrices combining siblings with categories VP:PP, VP:NP and VP:PRT, the weights in the part of the matrix which is multiplied with the VP child vector dominates. Similarly NPs dominate DTs. Fig. 4.5 shows example matrices. The two strong diagonals are due to the initialization described in Sec. 4.1.1. Note how the headword has larger weights in its block of the composition matrix; the VP block has the largest weights and adjective phrase matter more than simple determiners.

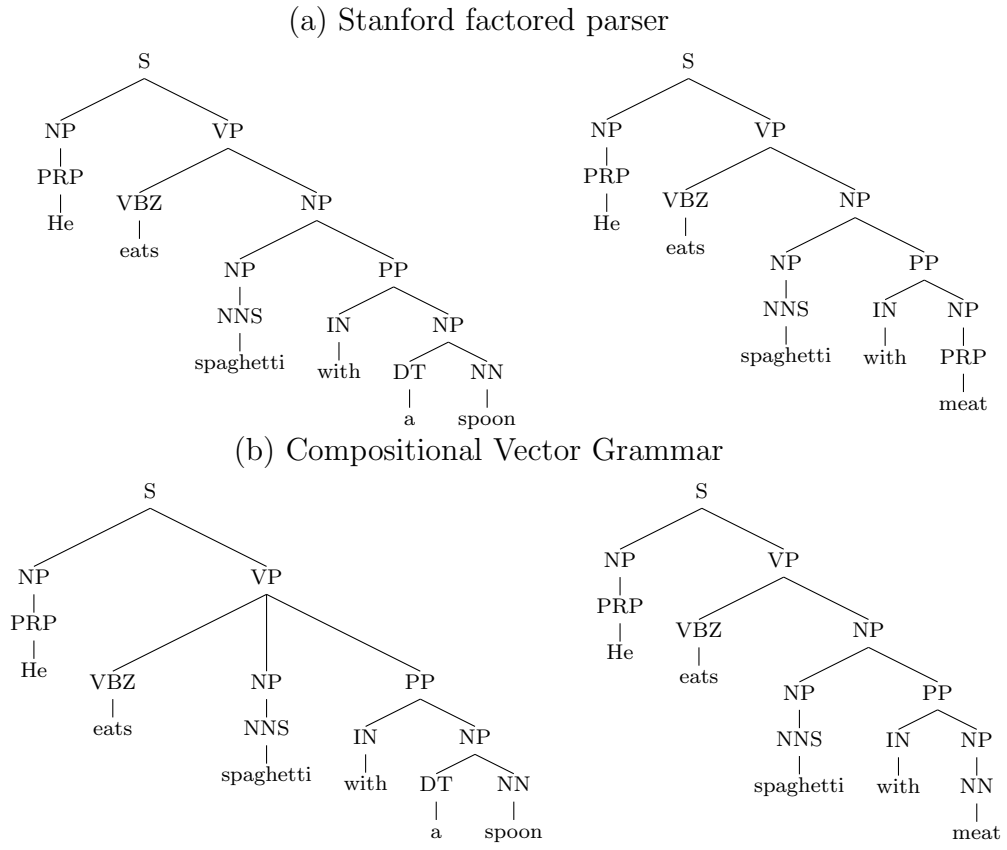
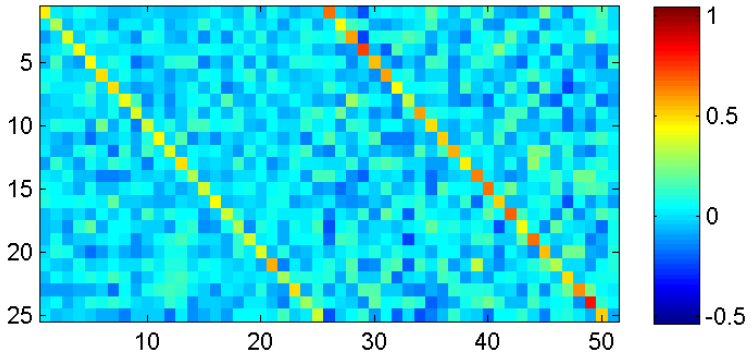
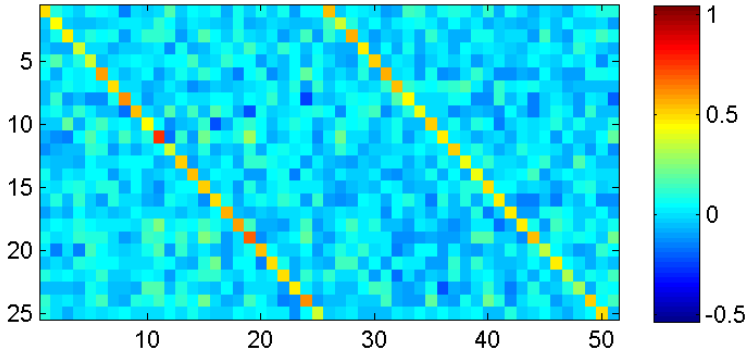


Figure 4.4: Test sentences of semantic transfer for PP attachments. The CVG was able to transfer semantic word knowledge from two related training sentences. In contrast, the Stanford parser could not distinguish the PP attachments based on the word semantics.

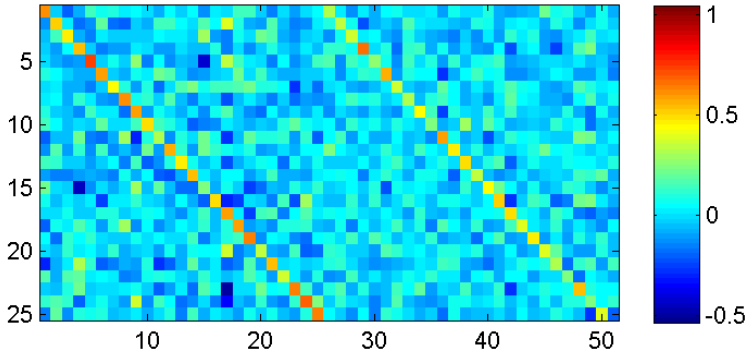
Semantic Transfer for PP Attachments. In this small model analysis, I use two pairs of sentences that the original Stanford parser and the CVG did not parse correctly after training on the WSJ. I then continue to train both parsers on two similar sentences and then analyze if the parsers correctly transferred the knowledge. The training sentences are *He eats spaghetti with a fork.* and *She eats spaghetti with pork.* The very similar test sentences are *He eats spaghetti with a spoon.* and *He eats spaghetti with meat.* Initially, both parsers incorrectly attach the PP to the verb in both test sentences. After training, the CVG parses both correctly, while the factored Stanford parser incorrectly attaches both PPs to *spaghetti*. The CVG's



DT-NP



ADJP-NP



VP-NP

Figure 4.5: Three binary composition matrices showing that head words dominate the composition. The model learns to not give determiners much importance, in contrast the adjective vectors have some important components. The two diagonals show clearly the two blocks that are multiplied with the left and right children, respectively. Note also that the NP here refers to the partial category "@NP" in the DT and ADJP modifier cases.

ability to transfer the correct PP attachments is due to the semantic word vector similarity between the words in the sentences. Fig. 4.4 shows the outputs of the two parsers.

4.1.3 Related Work

The CVG is inspired by two lines of research: Enriching PCFG parsers through more diverse sets of discrete states and recursive deep learning models that jointly learn classifiers and continuous feature representations for variable-sized inputs.

Improving Discrete Syntactic Representations

As mentioned in the introduction, there are several approaches to improving discrete representations for parsing. Klein and Manning (2003a) use manual feature engineering, while Petrov et al. (2006) use a learning algorithm that splits and merges the syntactic categories in order to maximize likelihood on the treebank. Their approach splits categories into several dozen subcategories. Another approach is lexicalized parsers (Collins, 2003; Charniak, 2000) that describe each category with a lexical item, usually the head word. More recently, Hall and Klein (2012) combine several such annotation schemes in a factored parser. I extend the above ideas from discrete representations to richer continuous ones. The CVG can be seen as factoring discrete and continuous parsing in one model. Another different approach to the above generative models is to learn discriminative parsers using many well designed features (Taskar et al., 2004; Finkel et al., 2008). I also borrow ideas from this line of research in that my parser combines the generative PCFG model with discriminatively learned RNNs.

Deep Learning and Recursive Deep Learning

Henderson (2003) was the first to show that neural networks can be successfully used for large scale parsing. He introduced a left-corner parser to estimate the probabilities of parsing decisions conditioned on the parsing history. The input to

Henderson's model consists of pairs of frequent words and their part-of-speech (POS) tags. Both the original parsing system and its probabilistic interpretation (Titov and Henderson, 2007) learn features that represent the *parsing history* and do not provide a principled linguistic representation like my phrase representations. Other related work includes Henderson (2004), who discriminatively trains a parser based on synchrony networks and Titov and Henderson (2006), who use an SVM to adapt a generative parser to different domains.

Costa et al. (2003) apply recursive neural networks to re-rank possible phrase attachments in an incremental parser. Their work is the first to show that RNNs can capture enough information to make correct parsing decisions, but they only test on a subset of 2000 sentences. Menchetti et al. (2005) use RNNs to re-rank different parses. For their results on full sentence parsing, they re-rank candidate trees created by the Collins parser (Collins, 2003). Similar to their work, I use the idea of letting discrete categories reduce the search space during inference. I compare to fully tied RNNs in which the same weights are used at every node. My syntactically untied RNNs outperform them by a significant margin. The idea of untying has also been successfully used in deep learning applied to vision (Le et al., 2012).

This section is based on several ideas of Socher et al. (2011b) which was the basis for Sec. 3.1. The main differences are (i) the dual representation of nodes as discrete categories and vectors, (ii) the combination with a PCFG, and (iii) the syntactic untying of weights based on child categories. I directly compare models with fully tied and untied weights. Another work that represents phrases with a dual discrete-continuous representation is by Kartsaklis et al. (2012).

The next section introduces a model that puts the idea of syntactic untying to the extreme: by giving every word and phrase its own composition function.

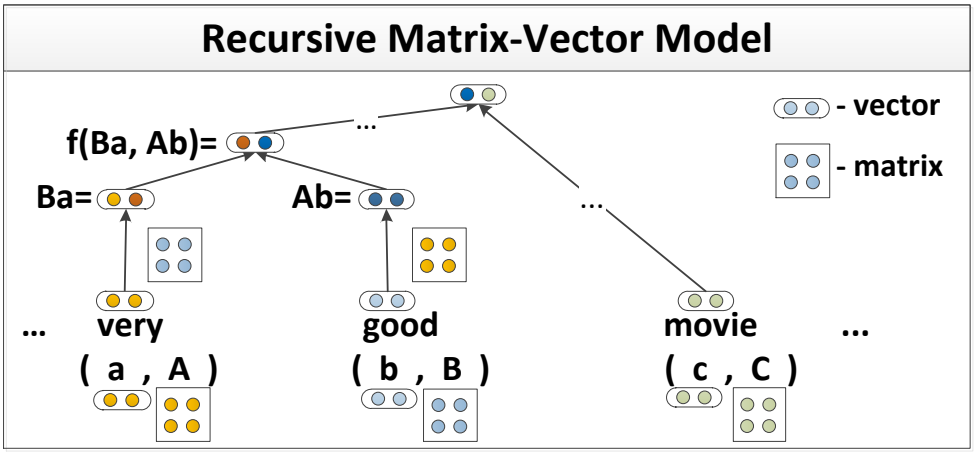


Figure 4.6: A recursive neural network which learns semantic vector representations of phrases in a tree structure. Each word and phrase is represented by a vector and a matrix, e.g., $very = (a, A)$. The matrix is applied to neighboring vectors. The same function is repeated to combine the phrase *very good* with *movie*.

4.2 Matrix Vector Recursive Neural Networks - For Relation Classification

The RNNs of previous sections tackle the problem of compositionality in vector spaces which has been investigates by several researchers (Mitchell and Lapata, 2010; Baroni and Zamparelli, 2010; Zanzotto et al., 2010; Yessenalina and Cardie, 2011). In this section, I revisit and further investigate the compositionality problem. I extend the above approaches with a more general and powerful model of semantic composition.

In particular, this section introduces a different recursive neural network model for semantic compositionality. Fig. 4.6 shows an illustration of the model in which each constituent (a word or longer phrase) has a matrix-vector (MV) representation. The vector captures the meaning of that constituent as in all previous models. The matrix captures how it modifies the meaning of the other word that it combines with. A representation for a longer phrase is again computed bottom-up by recursively combining the words according to the syntactic structure of a parse tree. Since the

model uses the MV representation with a neural network as the final merging function, I call this model a matrix-vector recursive neural network (MV-RNN).

I show that the ability to capture semantic compositionality in a syntactically plausible way translates into state of the art performance on various tasks. The first experiment of this section demonstrates that my model can learn fine-grained semantic compositionality. The task is to predict a sentiment distribution over movie reviews of adverb-adjective pairs such as *unbelievably sad* or *really awesome*. The MV-RNN is the only model that is able to properly negate sentiment when adjectives are combined with *not*. The MV-RNN outperforms previous state of the art models on full sentence sentiment prediction of movie reviews. The last experiment shows that the MV-RNN can also be used to find relationships between words using the learned phrase vectors. The relationship between words is recursively constructed and composed by words of arbitrary type in the variable length syntactic path between them. On the associated task of classifying relationships between nouns in arbitrary positions of a sentence the model outperforms all previous approaches on the SemEval-2010 Task 8 competition (Hendrickx et al., 2010). It outperforms all but one of the previous approaches without using any hand-designed semantic resources such as WordNet or FrameNet. By adding WordNet hypernyms, part-of-speech tags and named entity tags the MV-RNN outperforms the state of the art that uses significantly more resources.

4.2.1 MV-RNN: A Recursive Matrix-Vector Model

Apart from RNNs, the dominant approach for building representations of multi-word units from single word vector representations has been to form a linear combination of the single word representations, such as a sum or weighted average. This happens in information retrieval and in various text similarity functions based on lexical similarity. This can work well when the meaning of a text is literally "the sum of its parts", but fails when words function as operators that modify the meaning of another word: the meaning of "extremely strong" cannot be captured as the sum of word representations for "extremely" and "strong."

The models of previous sections provided a new way for moving beyond a linear combination, through use of a matrix W that multiplied the word vectors (a, b), and a nonlinearity function g (such as a sigmoid or tanh). Even though the nonlinearity allows to express a wider range of functions, it is almost certainly too much to expect a single fixed W matrix to be able to capture the meaning combination effects of all natural language operators. After all, inside the function g, I have the same linear transformation for all possible pairs of word vectors.

Recent work has started to capture the behavior of natural language operators inside semantic vector spaces by modeling them as matrices, which would allow a matrix for "extremely" to appropriately modify vectors for "smelly" or "strong" (Baroni and Zamparelli, 2010; Zanzotto et al., 2010). These approaches are along the right lines but so far have been restricted to capture linear functions of pairs of words whereas I would like nonlinear functions to compute compositional meaning representations for multi-word phrases or full sentences.

The MV-RNN combines the strengths of both of these ideas by (i) assigning a vector and a matrix to *every* word and (ii) learning an input-specific, nonlinear, compositional function for computing vector and matrix representations for multi-word sequences of any syntactic type. Assigning vector-matrix representations to all words instead of only to words of one part of speech category allows for greater flexibility which benefits performance. If a word lacks operator semantics, its matrix can be an identity matrix. However, if a word acts mainly as an operator, such as "extremely", its vector can become close to zero, while its matrix gains a clear operator meaning, here magnifying the meaning of the modified word in both positive and negative directions.

In this section I describe the initial word representations, the details of combining two words as well as the multi-word extensions. This is followed by an explanation of my training procedure.

Matrix-Vector Neural Word Representation

I represent a word as both a continuous vector (see Sec. 2.3) and a matrix of parameters. In this experiment, I initialize all word vectors $x \in \mathbb{R}^n$ with pre-trained

50-dimensional word vectors from the unsupervised model of Collobert and Weston (2008). Using Wikipedia text, their model learns word vectors by predicting how likely it is for each word to occur in its context. Similar to other local co-occurrence based vector space models, the resulting word vectors capture syntactic and semantic information. Every word is also associated with a matrix X. In all experiments, I initialize matrices as $X = I + \epsilon$, i.e., the identity plus a small amount of Gaussian noise. If the vectors have dimensionality n, then each word's matrix has dimensionality $X \in \mathbb{R}^{n \times n}$. While the initialization is random, the vectors and matrices will subsequently be modified to enable a sequence of words to compose a vector that can predict a distribution over semantic labels. Henceforth, I represent any phrase or sentence of length m as an ordered list of vector-matrix pairs $((a, A), \ldots, (m, M))$, where each pair is retrieved based on the word at that position.

Composition Models for Two Words

I first review composition functions for two words. In order to compute a parent vector p from two consecutive words and their respective vectors a and b, Mitchell and Lapata (2010) give as their most general function: $p = f(a, b, R, K)$,where R is the a-priori known syntactic relation and K is background knowledge.

There are many possible functions f. For my models, there is a constraint on p which is that it has the same dimensionality as each of the input vectors. This way, I can compare p easily with its children and p can be the input to a composition with another word. The latter is a requirement that will become clear in the next section. This excludes tensor products which were outperformed by simpler weighted addition and multiplication methods in (Mitchell and Lapata, 2010).

I will explore methods that do not require any manually designed semantic resources as background knowledge K. No explicit knowledge about the type of relation R is used. Instead I want the model to capture this implicitly via the learned matrices. I propose the following combination function which is input dependent:

$$p = f_{A,B}(a, b) = f(Ba, Ab) = g\left(W \begin{bmatrix} Ba \\ Ab \end{bmatrix}\right), \tag{4.8}$$

where A, B are matrices for single words, the global $W \in \mathbb{R}^{n \times 2n}$ is a matrix that maps both transformed words back into the same n-dimensional space. The element-wise function g could be simply the identity function but I use instead a nonlinearity such as the sigmoid or hyperbolic tangent tanh. Such a nonlinearity will allow us to approximate a wider range of functions beyond purely linear functions. I can also add a bias term before applying g but omit this for clarity. Rewriting the two transformed vectors as one vector z, I get $p = g(Wz)$ which is a single layer neural network. In this model, the word matrices can capture compositional effects specific to each word, whereas W captures a general composition function.

This function builds upon and generalizes several recent models in the literature. The most related work is that of Mitchell and Lapata (2010) and Zanzotto et al. (2010) who introduced and explored the composition function $p = Ba + Ab$ for word pairs. This model is a special case of Eq. 4.8 when I set $W = [II]$ (i.e. two concatenated identity matrices) and $g(x) = x$ (the identity function). Baroni and Zamparelli (2010) computed the parent vector of adjective-noun pairs by $p = Ab$, where A is an adjective matrix and b is a vector for a noun. This cannot capture nouns modifying other nouns, e.g., *disk drive*. This model too is a special case of the above model with $B = 0_{n \times n}$. Lastly, the RNN composition function of models in chapter 3 is also a special case with both A and B set to the identity matrix. I will compare to these special cases in my experiments.

Recursive Compositions of Multiple Words and Phrases

This section describes how I extend a word-pair matrix-vector-based compositional model to learn vectors and matrices for longer sequences of words. The main idea is to apply the same function f to pairs of constituents in a parse tree. For this to work, I need to take as input a binary parse tree of a phrase or sentence and also compute matrices at each nonterminal parent node. The function f can be readily used for phrase vectors since it is recursively compatible (p has the same dimensionality as its

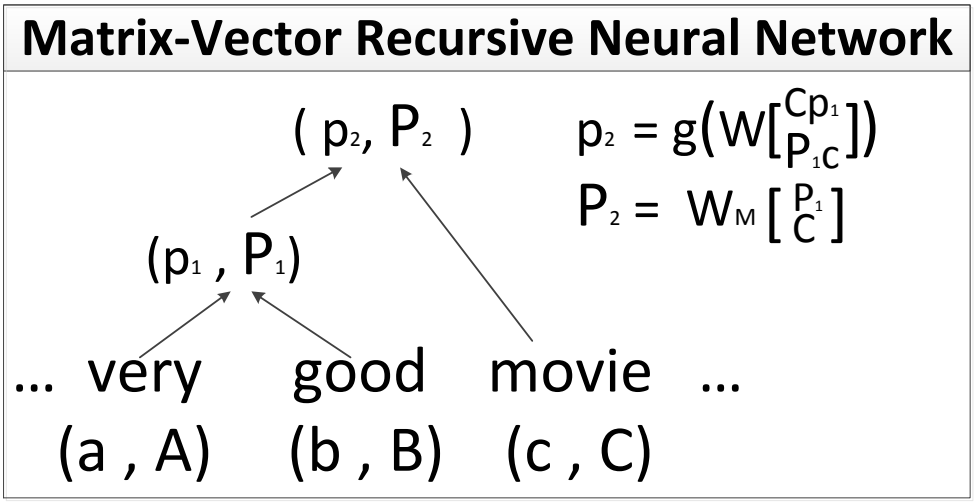


Figure 4.7: Example of how the MV-RNN merges a phrase with another word at a nonterminal node of a parse tree.

children). For computing nonterminal phrase matrices, I define the function

$$P = f_M(A, B) = W_M \begin{bmatrix} A \\ B \end{bmatrix}, \tag{4.9}$$

where $W_M \in \mathbb{R}^{n \times 2n}$, so $P \in \mathbb{R}^{n \times n}$ just like each input matrix.

After two words form a constituent in the parse tree, this constituent can now be merged with another one by applying the same functions f and f_M. For instance, to compute the vectors and matrices depicted in Fig. 4.7, I first merge words a and b and their matrices: $p_1 = f(Ba, Ab), P_1 = f_M(A, B)$. The resulting vector-matrix pair (p_1, P_1) can now be used to compute the full phrase when combining it with word c and computing $p_2 = f(Cp_1, P_1c), P_2 = f_M(P_1, C)$. The model computes vectors and matrices in a bottom-up fashion, applying the functions f, f_M to its own previous output (i.e. recursively) until it reaches the top node of the tree which represents the entire sentence.

For experiments with longer sequences I will compare to standard RNNs and the

special case of the MV-RNN that computes the parent by $p = Ab + Ba$, which I name the *linear Matrix-Vector Recursion* model (linear MVR). Previously, this model had not been trained for multi-word sequences. Sec. 4.2.5 talks about alternatives for compositionality.

Objective Functions for Training

One of the advantages of RNN-based models is that each node of a tree has associated with it a distributed vector representation (the parent vector p) which can also be seen as features describing that phrase. I train these representations by adding on top of each parent node a simple softmax classifier to predict a class distribution over e.g., sentiment or relationship classes: $d(p) = \text{softmax}(W^{label}p)$. For more details see section 3.2.1.

For the applications below (excluding logic), the corresponding error function $E(s, t, \theta)$ that I minimize for a sentence s and its tree t is the sum of cross-entropy errors at all nodes as described in Sec. 3.1.4 and again in Eq. 3.23 in the RAE section.

The only other methods that use this type of objective function are the other models in chapter 3 which also combine it with either a score or reconstruction error. Hence, for comparisons to other related work, I need to merge variations of computing the parent vector p with this classifier. The main difference is that the MV-RNN has more flexibility since it has an input-specific recursive function $f_{A,B}$ to compute each parent. In the following applications, I will use the softmax classifier to predict both sentiment distributions and noun-noun relationships.

Learning

Let $\theta = (W, W_M, W^{label}, L, L_M)$ be my model parameters and λ a vector with regularization hyperparameters for all model parameters. L and L_M are the sets of all word vectors and word matrices. The gradient of the overall objective function J becomes:

$$\frac{\partial J}{\partial \theta} = \frac{1}{N} \sum_{(x,t)} \frac{\partial E(x, t; \theta)}{\partial \theta} + \lambda\theta. \tag{4.10}$$

To compute this gradient, I first compute all tree nodes (p_i, P_i) from the bottom-up and then take derivatives of the softmax classifiers at each node in the tree from the top down. Derivatives are computed efficiently via backpropagation through structure as described in Sec. 3.1.4. Even though the objective is not convex, I found that L-BFGS run over the complete training data (batch mode) minimizes the objective well in practice and convergence is smooth.

Discussion: Evaluation and Generality

Evaluation of compositional vector spaces is a complex task. Most related work compares similarity judgments of unsupervised models to those of human judgments and aims at high correlation. These evaluations can give important insights. However, even with good correlation the question remains how these models would perform on downstream NLP tasks such as sentiment detection. I experimented with unsupervised learning of general vector-matrix representations by having the MV-RNN predict words in their correct context. Initializing the models with these general representations, did not improve the performance on the tasks I consider. For sentiment analysis, this is not surprising since antonyms often get similar vectors during unsupervised learning from co-occurrences due to high similarity of local syntactic contexts. In my experiments, the high prediction performance came from supervised learning of meaning representations using labeled data. While these representations are task-specific, they could be used across tasks in a multi-task learning setup. However, in order to fairly compare to related work, I use only the supervised data of each task. Before I describe my full-scale experiments, I analyze the model's expressive powers.

4.2.2 Model Analysis

This section analyzes the model with two proof-of-concept studies. First, I examine its ability to learn operator semantics for adverb-adjective pairs. If a model cannot correctly capture how an adverb operates on the meaning of adjectives, then there's little chance it can learn operators for more complex relationships. The second study

Method	Avg KL
$p = 0.5(a + b)$	0.189
$p = a \otimes b$	0.104
$p = [a; b]$	0.101
$p = Ab$	0.103
RNN	0.093
Linear MVR	0.093
MV-RNN	**0.091**

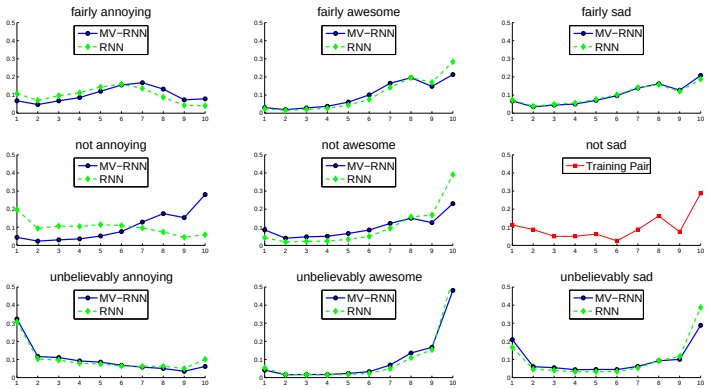


Figure 4.8: **Left**: Average KL-divergence for predicting sentiment distributions of unseen adverb-adjective pairs of the test set. See text for p descriptions. Lower is better. The main difference in the KL divergence comes from the few negation pairs in the test set. **Right**: Predicting sentiment distributions (over 1-10 stars on the x-axis) of adverb-adjective pairs. Each row has the same adverb and each column the same adjective. Many predictions are similar between the two models. The RNN and linear MVR are not able to modify the sentiment correctly: *not awesome* is more positive than *fairly awesome* and *not annoying* has a similar shape as *unbelievably annoying*. Predictions of the linear MVR model are almost identical to the standard RNN.

analyzes whether the MV-RNN can learn simple boolean operators of propositional logic such as conjunctives or negation from truth values. Again, if a model did not have this ability, then there's little chance it could learn these frequently occurring phenomena from the noisy language of real texts such as movie reviews.

Predicting Sentiment Distributions of Adverb-Adjective Pairs

The first study considers the prediction of fine-grained sentiment distributions of adverb-adjective pairs and analyzes different possibilities for computing the parent vector p. The results show that the MV-RNN operators are powerful enough to capture the operational meanings of various types of adverbs. For example, *very* is an intensifier, *pretty* is an attenuator, and *not* can negate or strongly attenuate the positivity of an adjective. For instance *not great* is still *pretty good* and not *terrible*;

see Potts (2010) for details.

I use a publicly available IMDB dataset of extracted adverb-adjective pairs from movie reviews.[1] The dataset provides the distribution over star ratings: Each consecutive word pair appears a certain number of times in reviews that have also associated with them an overall rating of the movie. After normalizing by the total number of occurrences, one gets a multinomial distribution over ratings. Only word pairs that appear at least 50 times are kept. Of the remaining pairs, I use 4211 randomly sampled ones for training and a separate set of 1804 for testing. I never give the algorithm sentiment distributions for single words, and, while single words overlap between training and testing, the test set consists of never before seen word pairs.

The softmax classifier is trained to minimize the cross entropy error. Hence, an evaluation in terms of KL-divergence is the most reasonable choice. It is defined as $KL(g||p) = \sum_i g_i \log(g_i/p_i)$, where g is the gold distribution and p is the predicted one.

I compare to several baselines and ablations of the MV-RNN model. An (adverb,adjective) pair is described by its vectors (a, b) and matrices (A, B).

1 $p = 0.5(a + b)$, vector average

2. $p = a \otimes b$, element-wise vector multiplication

3. $p = [a; b]$, vector concatenation

4. $p = Ab$, similar to Baroni and Lenci (2010)

5. $p = g(W[a; b])$, RNN, similar to Socher et al.

6. $p = Ab + Ba$, Linear MVR, similar to Mitchell and Lapata (2010); Zanzotto et al. (2010)

7. $p = g(W[Ba; Ab])$, MV-RNN

The final distribution is always predicted by a softmax classifier whose inputs p vary for each of the models.

I cross-validated all models over regularization parameters for word vectors, the softmax classifier, the RNN parameter W and the word operators $(10^{-4}, 10^{-3})$ and word vector sizes $(n = 6, 8, 10, 12, 15, 20)$. All models performed best at vector sizes of below 12. Hence, it is the model's power and not the number of parameters that

[1] `http://compprag.christopherpotts.net/reviews.html`

determines the performance. The table in Fig. 4.8 shows the average KL-divergence on the test set. It shows that the idea of matrix-vector representations for all words and having a nonlinearity are both important. The MV-RNN which combines these two ideas is best able to learn the various compositional effects. The main difference in KL divergence comes from the few negation cases in the test set. Fig. 4.8 shows examples of predicted distributions. Many of the predictions are accurate and similar between the top models. However, only the MV-RNN has enough expressive power to allow negation to completely shift the sentiment with respect to an adjective. A negated adjective carrying negative sentiment becomes slightly positive, whereas *not awesome* is correctly attenuated. All three top models correctly capture the U-shape of *unbelievably sad*. This pair peaks at both the negative and positive spectrum because it is ambiguous. When referring to the performance of actors, it is very negative, but, when talking about the plot, many people enjoy sad and thought-provoking movies. The $p = Ab$ model does not perform well because it cannot model the fact that for an adjective like "sad," the operator of "unbelievably" behaves differently.

Logic- and Vector-based Compositionality

Another natural question is whether the MV-RNN can, in general, capture some of the simple boolean logic that is sometimes found in language. In other words, can it learn some of the propositional logic operators such as *and, or, not* in terms of vectors and matrices from a few examples. Answering this question can also be seen as a first step towards bridging the gap between logic-based, formal semantics (Montague, 1974) and vector space models.

The logic-based view of language accounts nicely for compositionality by directly mapping syntactic constituents to lambda calculus expressions. At the word level, the focus is on function words, and nouns and adjectives are often defined only in terms of the sets of entities they denote in the world. Most words are treated as atomic symbols with no relation to each other. There have been many attempts at automatically parsing natural language to a logical form using recursive compositional rules.

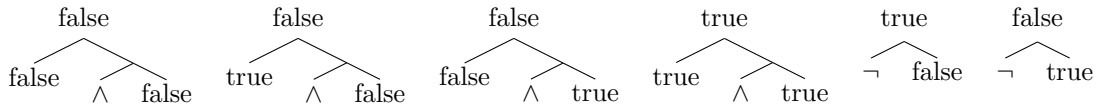


Figure 4.9: Training trees for the MV-RNN to learn propositional operators. The model learns vectors and operators for $\wedge$ (*and*) and $\neg$ (negation). The model outputs the exact representations of *false* and *true* respectively at the top node. Hence, the operators can be combined recursively an arbitrary number of times for more complex logical functions.

Conversely, vector space models have the attractive property that they can automatically extract knowledge from large corpora without supervision. Unlike logic-based approaches, these models allow us to make fine-grained statements about the semantic similarity of words which correlate well with human judgments (Griffiths et al., 2007). Logic-based approaches are often seen as orthogonal to distributional vector-based approaches. However, Garrette et al. (2011) recently introduced a combination of a vector space model inside a Markov Logic Network. Another interesting new line of work is that of Bowman et al. (2014) who show that recursive neural models (such as the one of Sec. 4.3) can be used for reasoning with quantifiers.

One open question is whether vector-based models can learn some of the simple logic encountered in language such as negation or conjunctives. To this end, I illustrate in a simple example that my MV-RNN model and its learned word matrices (operators) have the ability to learn propositional logic operators such as $\wedge, \vee, \neg$ (and, or, not). This is a necessary (though not sufficient) condition for the ability to pick up these phenomena in real datasets and tasks such as sentiment detection which I focus on in the subsequent sections.

My setup is as follows. I train on 6 strictly right-branching trees as in Fig. 4.9. I consider the 1-dimensional case and fix the representation for *true* to $(t = 1, T = 1)$ and *false* to $(f = 0, F = 1)$. Fixing the operators to the 1×1 identity matrix 1 is essentially ignoring them. The objective is then to create a perfect reconstruction of (t, T) or (f, F) (depending on the formula), which I achieve by the least squares error between the top vector's representation and the corresponding truth value, e.g. for $\neg false$: $\min ||p_{top} - t||^2 + ||P_{top} - T||^2$.

As my function g (see Eq. 4.8), I use a linear threshold unit: $g(x) = \max(\min(x, 1), 0)$. Giving the derivatives computed for the objective function for the examples in Fig. 4.9 to a standard L-BFGS optimizer quickly yields a training error of 0. Hence, the output of these 6 examples has exactly one of the truth representations, making it recursively compatible with further combinations of operators. Thus, I can combine these operators to construct any propositional logic function of any number of inputs (including xor). Hence, this MV-RNN is complete in terms of propositional logic.

S.	C.	Review sentence
1	$\surd$	The film is bright and flashy in all the right ways.
0	$\surd$	Not always too whimsical for its own good this strange hybrid of crime thriller, quirky character study, third-rate romance and female empowerment fantasy never really finds the tonal or thematic glue it needs.
0	$\surd$	Doesn't come close to justifying the hype that surrounded its debut at the Sundance film festival two years ago.
0	x	Director Hoffman, his writer and Kline's agent should serve detention.
1	x	A bodice-ripper for intellectuals.

Table 4.3: Hard movie review examples of positive (1) and negative (0) sentiment (S.) that of all methods only the MV-RNN predicted correctly (C: $\surd$) or could not classify as correct either (C: x).

4.2.3 Predicting Movie Review Ratings

In this section, I analyze the model's performance on full length sentences. I compare to previous state of the art methods on the same benchmark dataset of movie reviews as in Sec. 3.2. In this and the next experiment I use binarized trees from the Stanford Parser (Klein and Manning, 2003a). I use the exact same setup and parameters (regularization, word vector size, etc.) as the code published for the model in Sec. 3.2 which is available at `www.socher.org`.

Table 4.4 shows comparisons to the system of Nakagawa et al. (2010), a dependency tree based classification method that uses CRFs with hidden variables. The recursive autoencoder model of Sec. 3.2 obtained 77.7% accuracy. My new MV-RNN gives the highest performance, outperforming also the linear MVR (Sec. 4.2.1). However, the next Sec. 4.3 introduces another model that is even more accurate than all

Method	Acc.
Tree-CRF, Nakagawa et al. (2010)	77.3
RAE, Socher et al. (2011c)	77.7
Linear MVR	77.1
MV-RNN	**79.0**

Table 4.4: Accuracy of classification on full length movie review polarity (MR).

of those in the above table.

Table 4.3 shows several hard examples that only the MV-RNN was able to classify correctly. None of the methods correctly classified the last two examples which require more world knowledge.

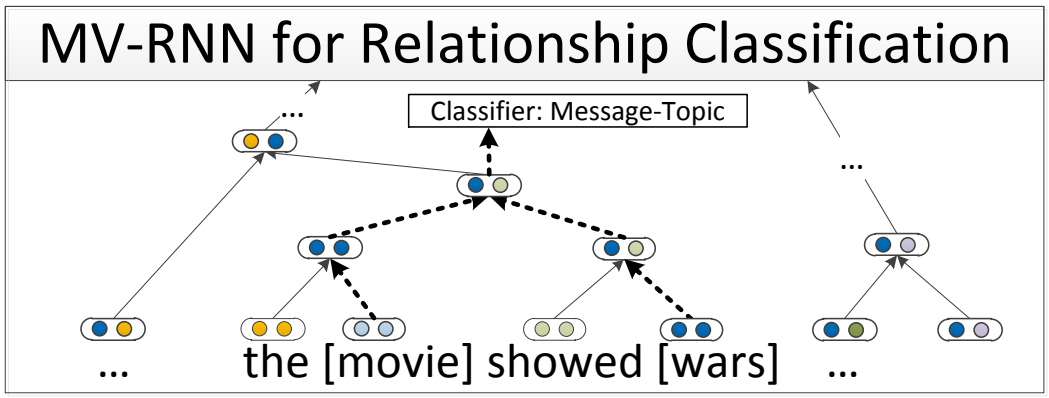


Figure 4.10: The MV-RNN learns vectors in the path connecting two words (dotted lines) to determine their semantic relationship. It takes into consideration a variable length sequence of various word types in that path.

4.2.4 Classification of Semantic Relationships

The previous task considered global classification of an entire phrase or sentence. In my last experiment I show that the MV-RNN can also learn how a syntactic context composes an aggregate meaning of the semantic relationships between words. In particular, the task is finding semantic relationships between pairs of nominals. For

Relationship	Sentence with labeled nouns for which to predict relationships
Cause-Effect(e2,e1)	Avian [influenza]$_{e1}$ is an infectious disease caused by type a strains of the influenza [virus]$_{e2}$.
Entity-Origin(e1,e2)	The [mother]$_{e1}$ left her native [land]$_{e2}$ about the same time and they were married in that city.
Message-Topic(e2,e1)	Roadside [attractions]$_{e1}$ are frequently advertised with [billboards]$_{e2}$ to attract tourists.
Product-Producer(e1,e2)	A child is told a [lie]$_{e1}$ for several years by their [parents]$_{e2}$ before he/she realizes that ...
Entity-Destination(e1,e2)	The accident has spread [oil]$_{e1}$ into the [ocean]$_{e2}$.
Member-Collection(e2,e1)	The siege started, with a [regiment]$_{e1}$ of lightly armored [swordsmen]$_{e2}$ ramming down the gate.
Instrument-Agency(e2,e1)	The core of the [analyzer]$_{e1}$ identifies the paths using the constraint propagation [method]$_{e2}$.
Component-Whole(e2,e1)	The size of a [tree]$_{e1}$ [crown]$_{e2}$ is strongly correlated with the growth of the tree.
Content-Container(e1,e2)	The hidden [camera]$_{e1}$, found by a security guard, was hidden in a business card-sized [leaflet box]$_{e2}$ placed at an unmanned ATM in Tokyo's Minato ward in early September.

Table 4.5: Examples of correct classifications of ordered, semantic relations between nouns by the MV-RNN. Note that the final classifier is a recursive, compositional function of all the words in the syntactic path between the bracketed words. The paths vary in length and the words vary in type.

instance, in the sentence "My [apartment]$_{e1}$ has a pretty large [kitchen]$_{e2}$.", I want to predict that the kitchen and apartment are in a *component-whole* relationship. Predicting such semantic relations is useful for information extraction and thesaurus construction applications. Many approaches use features for all words on the path between the two words of interest. I show that by building a single compositional semantics for the minimal constituent including both terms one can achieve a higher performance.

This task requires the ability to deal with sequences of words of arbitrary type and length in between the two nouns in question. Fig. 4.10 explains my method for classifying nominal relationships. I first find the path in the parse tree between the two words whose relation I want to classify. I then select the highest node of the path and classify the relationship using that node's vector as features. I apply the same type of MV-RNN model as in sentiment to the subtree spanned by the two words.

I use the dataset and evaluation framework of SemEval-2010 Task 8 (Hendrickx

et al., 2010). There are 9 ordered relationships (with two directions) and an undirected *other* class, resulting in 19 classes. Among the relationships are: message-topic, cause-effect, instrument-agency (etc. see Table 4.5 for list). A pair is counted as correct if the order of the words in the relationship is correct.

I use 50-dimensional, pre-trained word vectors (see Sec. 4.2.1) and a pooling size of 3. Table 4.6 lists results for several competing methods together with the resources and features used by each method. I compare to the systems of the competition which are described in Hendrickx et al. (2010) as well as the RNN and linear MVR. Most systems used a considerable amount of hand-designed semantic resources. In contrast to these methods, the MV-RNN only needs a parser for the tree structure and learns all semantics from unlabeled corpora and the training data. Only the SemEval training dataset is specific to this task, the remaining inputs and the training setup are the same as in previous sentiment experiments.

The best method on this dataset (Rink and Harabagiu, 2010) obtains 82.2% F1. In order to see whether my system can improve over this system, I added three features to the MV-RNN vector and trained another softmax classifier. The features and their performance increases were POS tags (+0.9); WordNet hypernyms (+1.3) and named entity tags (NER) of the two words (+0.6). Features were computed using the code of Ciaramita and Altun (2006).[2] With these features, the performance improved over the state of the art system. Table 4.5 shows random correct classification examples.

4.2.5 Related work

Distributional approaches have become omnipresent for the recognition of semantic similarity between words and the treatment of compositionality has seen much progress in recent years.

Semantic Word Vector Spaces The dominant approach in semantic vector spaces uses distributional similarities of single words. Often, co-occurrence statistics of a word and its context are used to describe each word (Pado and Lapata, 2007; Turney and Pantel, 2010; Baroni and Lenci, 2010), such as tf-idf. Variants of this idea use

[2] `sourceforge.net/projects/supersensetag/`

Classifier	Feature Sets	F1
SVM	POS, stemming, syntactic patterns	60.1
SVM	word pair, words in between	72.5
SVM	POS, WordNet, stemming, syntactic patterns	74.8
SVM	POS, WordNet, morphological features, thesauri, Google n-grams	77.6
MaxEnt	POS, WordNet, morphological features, noun compound system, thesauri, Google n-grams	77.6
SVM	POS, WordNet, prefixes and other morphological features, POS, dependency parse features, Levin classes, PropBank, FrameNet, NomLex-Plus, Google n-grams, paraphrases, TextRunner	82.2
RNN	-	74.8
Lin.MVR	-	73.0
MV-RNN	-	79.1
RNN	POS,WordNet,NER	77.6
Lin.MVR	POS,WordNet,NER	78.7
MV-RNN	POS,WordNet,NER	**82.4**

Table 4.6: Learning methods, their feature sets and F1 results for predicting semantic relations between nouns. The MV-RNN outperforms all but one method without any additional feature sets. By adding three such features, it obtains state of the art performance.

more complex frequencies such as how often a word appears in a certain syntactic context (Pado and Lapata, 2007; Erk and Padó, 2008). These representations have proven very effective in sense discrimination and disambiguation (Schütze, 1998), automatic thesaurus extraction (Lin, 1998; Curran, 2004) and selectional preferences (Erk and Padó, 2008) and cognitive modeling (Landauer and Dumais, 1997). However, distributional vectors often do not properly capture the differences in antonyms since those often have similar contexts. One possibility to remedy this is to use neural word vectors (Bengio et al., 2003). These vectors can be trained in an unsupervised fashion to capture distributional similarities (Collobert and Weston, 2008; Huang et al., 2012) but then also be fine-tuned and trained to specific tasks such as sentiment detection (Socher et al., 2011c). The models in this section can use purely supervised word representations learned entirely on the new corpus.

Compositionality in Vector Spaces Most of the compositionality algorithms and related datasets capture two word compositions. Mitchell and Lapata (2010) use e.g. two-word phrases and analyze similarities computed by vector addition, multiplication, convolution (Metcalfe, 1990) and others. They measured the similarity between word pairs such as compound nouns or verb-object pairs and compared these with human similarity judgments. Simple vector averaging or multiplication performed best, hence my comparisons to related baselines above. Some related models such as holographic reduced representations (Plate, 1995), quantum logic (Widdows, 2008), discrete-continuous models (Clark and Pulman, 2007) and the recent compositional matrix space model (Rudolph and Giesbrecht, 2010) have not been experimentally validated on larger corpora. Grefenstette and Sadrzadeh (2011) analyze subject-verb-object triplets and find a matrix-based categorical model to correlate well with human judgments. I compare to the recent line of work on supervised compositional models. In particular I will describe and experimentally compare my new RNTN model to RNNs (Socher et al., 2011c) and MV-RNNs (Socher et al., 2012b) both of which have been applied to bag of words sentiment corpora.

My model builds upon and generalizes the models of Mitchell and Lapata (2010); Baroni and Zamparelli (2010); Zanzotto et al. (2010); Socher et al. (2011c) (see Sec. 4.2.1). I compare to them in my experiments. Yessenalina and Cardie (2011) introduce a sentiment analysis model that describes words as matrices and composition as matrix multiplication. Since matrix multiplication is associative, this cannot capture different scopes of negation or syntactic differences. Their model, is a special case of my encoding model (when you ignore vectors, fix the tree to be strictly branching in one direction and use as the matrix composition function $P = AB$). Since my classifiers are trained on the vectors, I cannot compare to this approach directly. Grefenstette and Sadrzadeh (2011) learn matrices for verbs in a categorical model. The trained matrices improve correlation with human judgments on the task of identifying relatedness of subject-verb-object triplets. Another alternative would be to use CCG trees as a backbone for vector composition (K.M. Hermann, 2013).

This concludes the MV-RNN section. The MV-RNN combines attractive theoretical properties with good performance on large, noisy datasets. It generalizes several

models in the literature, can learn propositional logic, accurately predicts sentiment and can be used to classify semantic relationships between nouns in a sentence. However, it has a large number of parameters and there are no methods yet for pre-training the word matrices. For more discussion and a comparison between all models see the final conclusion chapter of this thesis. The next section will introduce the last and most powerful composition function.

4.3 Recursive Neural Tensor Layers - For Sentiment Analysis

Despite the large attention that compositionality in semantic vector spaces has received in the previous section and recent work (Mitchell and Lapata, 2010; Socher et al., 2010; Zanzotto et al., 2010; Yessenalina and Cardie, 2011; Socher et al., 2012b; Grefenstette et al., 2013), progress is held back by the current lack of large and labeled compositionality resources and even more powerful models to accurately capture the underlying phenomena presented in such data. To address this need, I introduce the Stanford Sentiment Treebank and a powerful Recursive Neural Tensor Network that can accurately predict the compositional semantic effects present in this new corpus.

The *Stanford Sentiment Treebank* is the first corpus with fully labeled parse trees that allows for a complete analysis of the compositional effects of sentiment in language. The corpus is based on the dataset introduced by Pang and Lee (2005) and used in Sec. 3.2 and 4.2. It was parsed with the Stanford parser (Klein and Manning, 2003a) and includes a total of 215,154 unique phrases from those parse trees, each annotated by 3 human judges on Amazon Mechanical Turk. This new dataset allows us to analyze the intricacies of sentiment and to capture complex linguistic phenomena with powerful algorithms. Fig. 4.11 shows one prediction examples with the model of this section which shows clear compositional structure. The granularity and size of this dataset will enable the community to train compositional models that are based on supervised and structured machine learning techniques. While there are several datasets with document and chunk labels available, there is a need to better

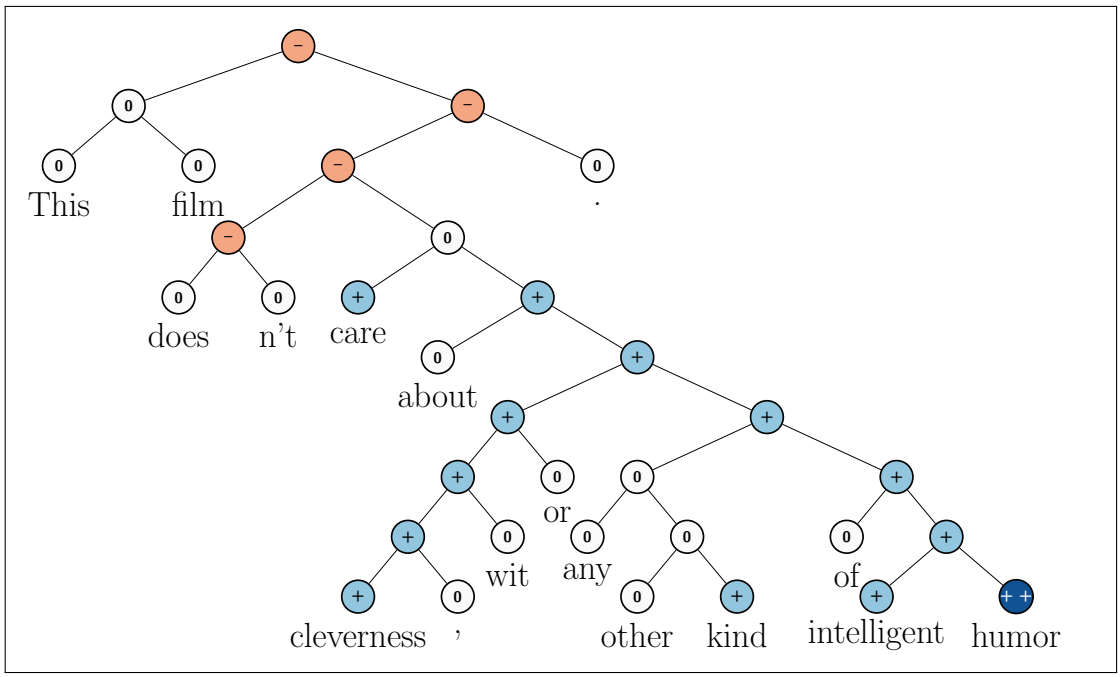


Figure 4.11: Example of the Recursive Neural Tensor Network accurately predicting 5 sentiment classes, very negative to very positive (– –, –, 0, +, + +), at every node of a parse tree and capturing the negation and its scope in this sentence.

capture sentiment from short comments, such as Twitter data, which provide less overall signal per document.

In order to capture the compositional effects with higher accuracy, I propose a new model called the Recursive Neural Tensor Network (RNTN). Recursive Neural Tensor Networks take as input phrases of any length. Like RNN models from previous sections, they represent a phrase through word vectors and a parse tree and then compute vectors for higher nodes in the tree using the same tensor-based composition function. I compare to several compositional models of previous sections, such as standard RNNs (Socher et al., 2011c), MV-RNNs (Socher et al., 2012b), and baselines such as neural networks that ignore word order, Naive Bayes (NB), bi-gram NB and SVM. All models get a significant boost when trained with the new dataset but the RNTN obtains the highest performance with 80.7% accuracy when predicting fine-grained sentiment for all nodes. Lastly, I use a test set of positive and negative

sentences and their respective negations to show that, unlike bag of words models, the RNTN accurately captures the sentiment change and scope of negation. RNTNs also learn that sentiment of phrases following the contrastive conjunction 'but' dominates.

The complete training and testing code, a live demo and the Stanford Sentiment Treebank dataset are available at `http://nlp.stanford.edu/sentiment`.

4.3.1 Stanford Sentiment Treebank

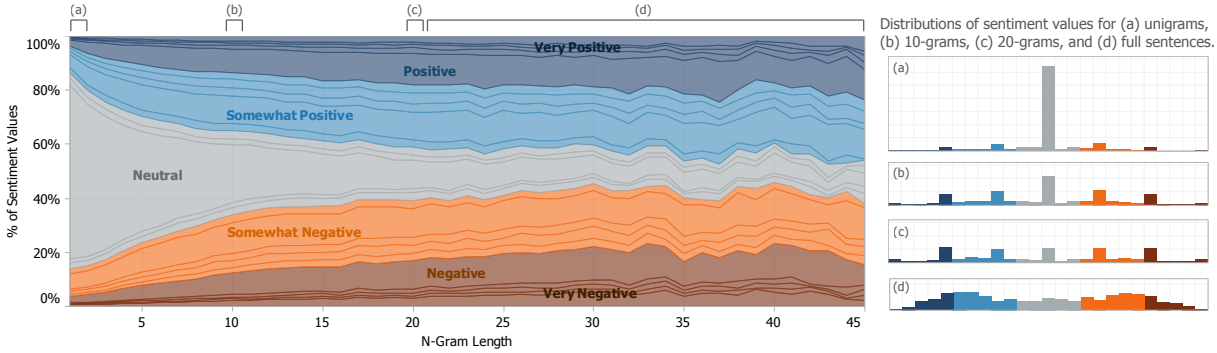


Figure 4.12: Normalized histogram of sentiment annotations at each n-gram length. Many shorter n-grams are neutral; longer phrases are well distributed. Few annotators used slider positions between ticks or the extreme values. Hence the two strongest labels and intermediate tick positions are merged into 5 classes.

Bag of words classifiers can work well in longer documents by relying on a few words with strong sentiment like 'awesome' or 'exhilarating.' However, sentiment accuracies even for binary positive/negative classification for single sentences has not exceeded 80% for several years. For the more difficult multiclass case including a neutral class, accuracy is often below 60% for short messages on Twitter (Wang et al., 2012). From a linguistic or cognitive standpoint, ignoring word order in the treatment of a semantic task is not plausible, and, as I will show, it cannot accurately classify hard examples of negation. Correctly predicting these hard cases is necessary to further improve performance.

In this section I will introduce and provide some analyses for the new *Sentiment Treebank* which includes labels for every syntactically plausible phrase in thousands

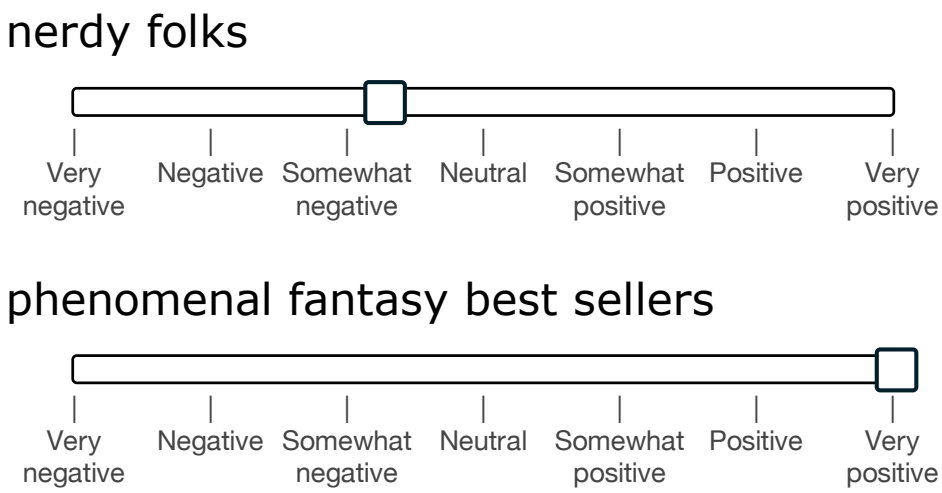


Figure 4.13: The labeling interface. Random phrases were shown and annotators had a slider for selecting the sentiment and its degree.

of sentences, allowing us to train and evaluate compositional models.

I consider the corpus of movie review excerpts from the `rottentomatoes.com` website originally collected and published by Pang and Lee (2005). The original dataset includes 10,662 sentences, half of which were considered positive and the other half negative. Each label is extracted from a longer movie review and reflects the writer's overall intention for this review. The normalized, lower-cased text is first used to recover, from the original website, the text with capitalization. Remaining HTML tags and sentences that are not in English are deleted. The Stanford Parser (Klein and Manning, 2003a) is used to parses all 10,662 sentences. In approximately 1,100 cases it splits the snippet into multiple sentences. I then used Amazon Mechanical Turk to label the resulting 215,154 phrases. Fig. 4.13 shows the interface annotators saw. The slider has 25 different values and is initially set to neutral. The phrases in each hit are randomly sampled from the set of all phrases in order to prevent labels being influenced by what follows.

Fig. 4.12 shows the normalized label distributions at each n-gram length. Starting at length 20, the majority are full sentences. One of the findings from labeling sentences based on *reader's perception* is that many of them could be considered neutral. I also notice that stronger sentiment often builds up in longer phrases and the majority of the shorter phrases are neutral. Another observation is that most annotators moved the slider to one of the five positions: negative, somewhat negative, neutral, positive or somewhat positive. The extreme values were rarely used and the slider was not often left in between the ticks. Hence, *even a 5-class classification into these categories captures the main variability of the labels.* I will name this *fine-grained sentiment* classification and my main experiment will be to recover these five labels for phrases of all lengths.

4.3.2 RNTN: Recursive Neural Tensor Networks

All the models in this thesis compute compositional vector representations for phrases of variable length and syntactic type. These representations are used as features to classify each phrase. Fig. 4.14 displays this approach with an example of phrase-labeled sentiment. When an n-gram is given to the compositional models, it is parsed into a binary tree and each leaf node, corresponding to a word, is represented as a vector as in previous sections. Recursive neural models will then compute parent vectors in a bottom up fashion using different types of compositionality functions g. The parent vectors are again given as features to a classifier. For ease of exposition, I will use the tri-gram in this figure to explain the new model.

As before, each word is represented as a d-dimensional vector. In this section, I simply initialize all word vectors by randomly sampling each value from a uniform distribution: $\mathcal{U}(-r, r)$, where $r = 0.0001$. All the word vectors are stacked in the word embedding matrix $L \in \mathbb{R}^{d \times |V|}$, where $|V|$ is the size of the vocabulary. Initially the word vectors will be random but the L matrix is seen as a parameter that is trained jointly with the model.

I can use the word vectors directly as parameters to optimize and as feature inputs to a softmax classifier. For classification into five classes, I compute the posterior

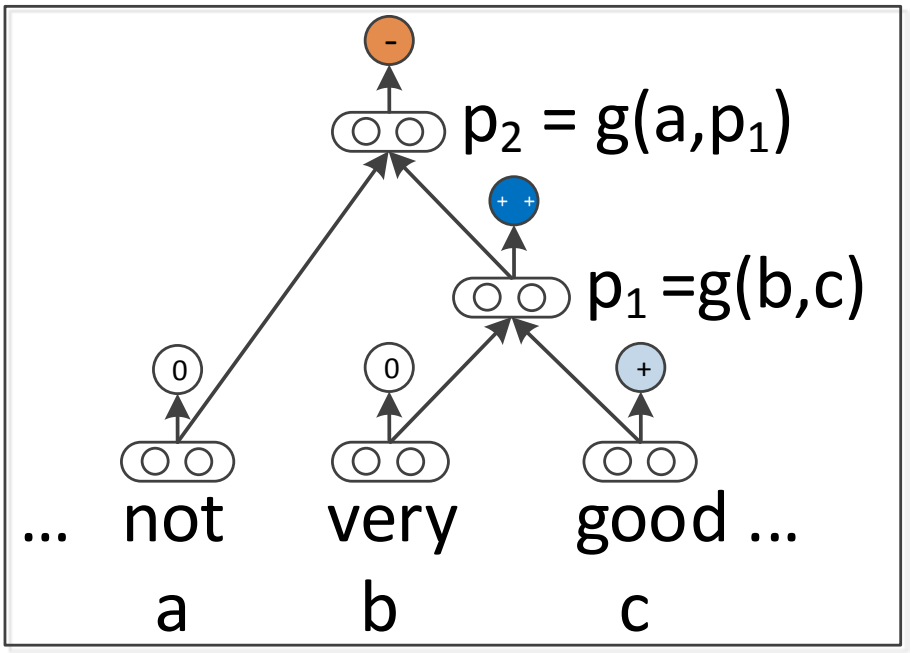


Figure 4.14: Approach of Recursive Neural Network models for sentiment: Compute parent vectors in a bottom up fashion using a compositionality function g and use node vectors as features for a classifier at that node. This function varies for the different models.

probability over labels given the word vector via:

$$y^a = \text{softmax}(W_s a), \tag{4.11}$$

where $W_s \in \mathbb{R}^{5\times d}$ is the sentiment classification matrix. For the given tri-gram, this is repeated for vectors b and c.

Two previous models in this thesis had been applied to the sentiment analysis task: RAEs of Sec. 3.2 and MV-RNNs of Sec. 4.2. One problem with the RAE sentiment model was that its unsupervised tree structures did not correspond to proper linguistic parses. Furthermore, I found that with the additional amount of training data, the

reconstruction loss at each node is not necessary to obtain high performance. The problem with MV-RNNs is that the number of parameters becomes very large and depends on the size of the vocabulary. While linguistically plausible, it would be cognitively more plausible if there was a single powerful composition function with a fixed number of parameters. The standard RNN is a good candidate for such a function. However, in the standard RNN, the input vectors only implicitly interact through the nonlinearity (squashing) function. A more direct, possibly multiplicative, interaction would allow the model to have greater interactions between the input vectors.

Motivated by these ideas I ask the question: Can a single, more powerful composition function perform better and compose aggregate meaning from smaller constituents more accurately than many input-specific ones? In order to answer this question, I propose a new model called the Recursive Neural Tensor Network (RNTN). The main idea is to use the same, tensor-based composition function for all nodes.

Fig. 4.15 shows a single tensor layer. I define the output of a tensor product $h \in \mathbb{R}^d$ via the following vectorized notation and the equivalent but more detailed notation for each slice $V^{[i]} \in \mathbb{R}^{d \times d}$:

$$h = \begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} b \\ c \end{bmatrix}; h_i = \begin{bmatrix} b \\ c \end{bmatrix}^T V^{[i]} \begin{bmatrix} b \\ c \end{bmatrix}.$$

where $V^{[1:d]} \in \mathbb{R}^{2d \times 2d \times d}$ is the tensor that defines multiple bilinear forms.
The RNTN uses this definition for computing p_1:

$$p_1 = f\left(\begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} b \\ c \end{bmatrix} + W \begin{bmatrix} b \\ c \end{bmatrix}\right),$$

where W is as defined in the previous models. The next parent vector p_2 in the tri-gram will be computed with the same weights:

$$p_2 = f\left(\begin{bmatrix} a \\ p_1 \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} a \\ p_1 \end{bmatrix} + W \begin{bmatrix} a \\ p_1 \end{bmatrix}\right).$$

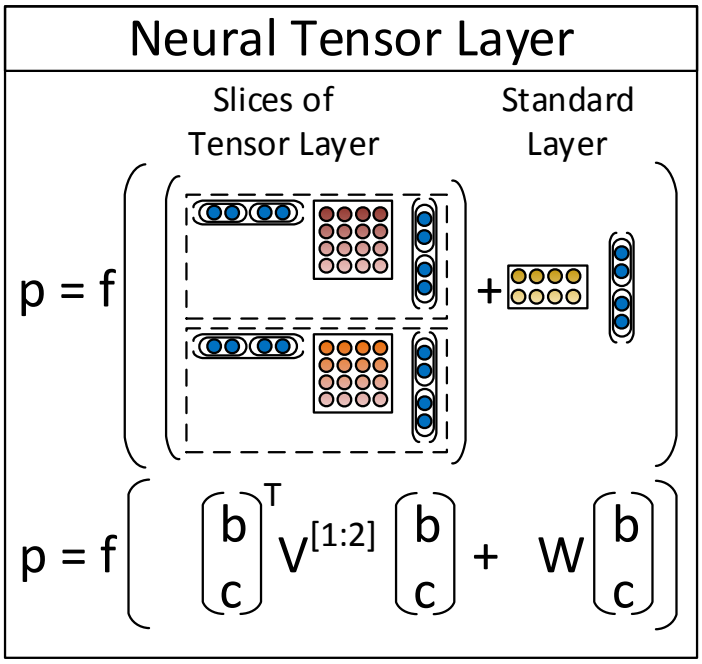


Figure 4.15: A single layer of the Recursive Neural Tensor Network. Each dashed box represents one of d-many slices and can capture a type of influence a child can have on its parent.

The main advantage over the previous RNN model, which is a special case of the RNTN when V is set to 0, is that the tensor can directly relate input vectors. Intuitively, I can interpret each slice of the tensor as capturing a specific type of composition.

An alternative to RNTNs would be to make the compositional function more powerful by adding a second neural network layer. However, initial experiments showed that it is hard to optimize this model and vector interactions are still more implicit than in the RNTN.

Tensor Backprop through Structure

I describe in this section how to train the RNTN model. For more intuition and an introduction to backprop through structure see Sec. 3.1.4. As mentioned above, each node has a softmax classifier trained on its vector representation to predict a given ground truth or target vector t. I assume the target distribution vector at each node has a 0-1 encoding. If there are C classes, then it has length C and a 1 at the correct label. All other entries are 0.

I want to maximize the probability of the correct prediction, or minimize the cross-entropy error between the predicted distribution $y^i \in \mathbb{R}^{C\times 1}$ at node i and the target distribution $t^i \in \mathbb{R}^{C\times 1}$ at that node. This is equivalent (up to a constant) to minimizing the KL-divergence between the two distributions. The error as a function of the RNTN parameters $\theta = (V, W, W_s, L)$ for a sentence is:

$$E(\theta) = \sum_i \sum_j t^i_j \log y^i_j + \lambda \|\theta\|^2 \tag{4.12}$$

The derivative for the weights of the softmax classifier are standard and simply sum up from each node's error. I define x^i to be the vector at node i (in the example trigram, the $x^i \in \mathbb{R}^{d\times 1}$ are (a, b, c, p_1, p_2)). I skip the standard derivative for W_s. Each node backpropagates its error through to the recursively used weights V, W. Let $\delta^{i,s} \in \mathbb{R}^{d\times 1}$ be the softmax error vector at node i:

$$\delta^{i,s} = \left(W_s^T (y^i - t^i)\right) \otimes f'(x^i),$$

where $\otimes$ is the Hadamard product between the two vectors and f' is the element-wise derivative of f which in the standard case of using $f = \tanh$ can be computed using only $f(x^i)$.

The remaining derivatives can only be computed in a top-down fashion from the top node through the tree and into the leaf nodes. The full derivative for V and W is the sum of the derivatives at each of the nodes. I define the complete incoming error messages for a node i as $\delta^{i,com}$. The top node, in my case p_2, only received errors from the top node's softmax. Hence, $\delta^{p_2,com} = \delta^{p_2,s}$ which I can use to obtain the standard

backprop derivative for W as described in Sec. 3.1.4. For the derivative of each slice $k = 1, \ldots, d$, I get:

$$\frac{\partial E^{p_2}}{\partial V^{[k]}} = \delta_k^{p_2,com} \begin{bmatrix} a \\ p_1 \end{bmatrix} \begin{bmatrix} a \\ p_1 \end{bmatrix}^T,$$

where $\delta_k^{p_2,com}$ is just the k'th element of this vector. Now, I can compute the error message for the two children of p_2:

$$\delta^{p_2,down} = \left(W^T \delta^{p_2,com} + S \right) \otimes f' \left(\begin{bmatrix} a \\ p_1 \end{bmatrix} \right),$$

where I define

$$S = \sum_{k=1}^{d} \delta_k^{p_2,com} \left(V^{[k]} + \left(V^{[k]} \right)^T \right) \begin{bmatrix} a \\ p_1 \end{bmatrix}$$

The children of p_2, will then each take half of this vector and add their own softmax error message for the complete δ. In particular, I have

$$\delta^{p_1,com} = \delta^{p_1,s} + \delta^{p_2,down}[d+1:2d],$$

where $\delta^{p_2,down}[d+1:2d]$ indicates that p_1 is the right child of p_2 and hence takes the 2nd half of the error, for the final word vector derivative for a, it will be $\delta^{p_2,down}[1:d]$.

The full derivative for slice $V^{[k]}$ for this trigram tree then is the sum at each node:

$$\frac{\partial E}{\partial V^{[k]}} = \frac{E^{p_2}}{\partial V^{[k]}} + \delta_k^{p_1,com} \begin{bmatrix} b \\ c \end{bmatrix} \begin{bmatrix} b \\ c \end{bmatrix}^T,$$

and similarly for W. For this nonconvex optimization I use AdaGrad (Duchi et al., 2011) which converges in less than 3 hours to a local optimum.

4.3.3 Experiments

I include two types of analyses. The first type includes several large quantitative evaluations on the test set. The second type focuses on two linguistic phenomena that are important in sentiment.

For all models, I use the dev set and cross-validate over regularization of the weights, word vector size as well as learning rate and minibatch size for AdaGrad. Optimal performance for all models was achieved at word vector sizes between 25 and 35 dimensions and batch sizes between 20 and 30. Performance decreased at larger or smaller vector and batch sizes. This indicates that the RNTN does not outperform the standard RNN due to simply having more parameters. The MV-RNN has orders of magnitudes more parameters than any other model due to the word matrices. The RNTN would usually achieve its best performance on the dev set after training for 3 - 5 hours. Initial experiments showed that the recursive models worked significantly worse (over 5% drop in accuracy) when no nonlinearity was used. I use $f = \tanh$ in all experiments.

I compare to commonly used methods that use bag of words features with Naive Bayes and SVMs, as well as Naive Bayes with bag of bigram features. I abbreviate these with NB, SVM and biNB. I also compare to a model that averages neural word vectors and ignores word order (VecAvg).

The sentences in the treebank were split into a train (8544), dev (1101) and test splits (2210) and these splits are made available with the data release. I also analyze performance on only positive and negative sentences, ignoring the neutral class. This filters about 20% of the data with the three sets having 6920/872/1821 sentences.

Fine-grained Sentiment For All Phrases

The main novel experiment and evaluation metric analyze the accuracy of fine-grained sentiment classification for all phrases. Fig. 4.12 showed that a fine grained classification into 5 classes is a reasonable approximation to capture most of the data variation.

Fig. 4.16 shows the result on this new corpus. The RNTN gets the highest performance, followed by the MV-RNN and RNN. The recursive models work very well on shorter phrases, where negation and composition are important, while bag of features baselines perform well only with longer sentences. The RNTN accuracy upper bounds other models at most n-gram lengths.

Table 4.7 (left) shows the overall accuracy numbers for fine grained prediction at

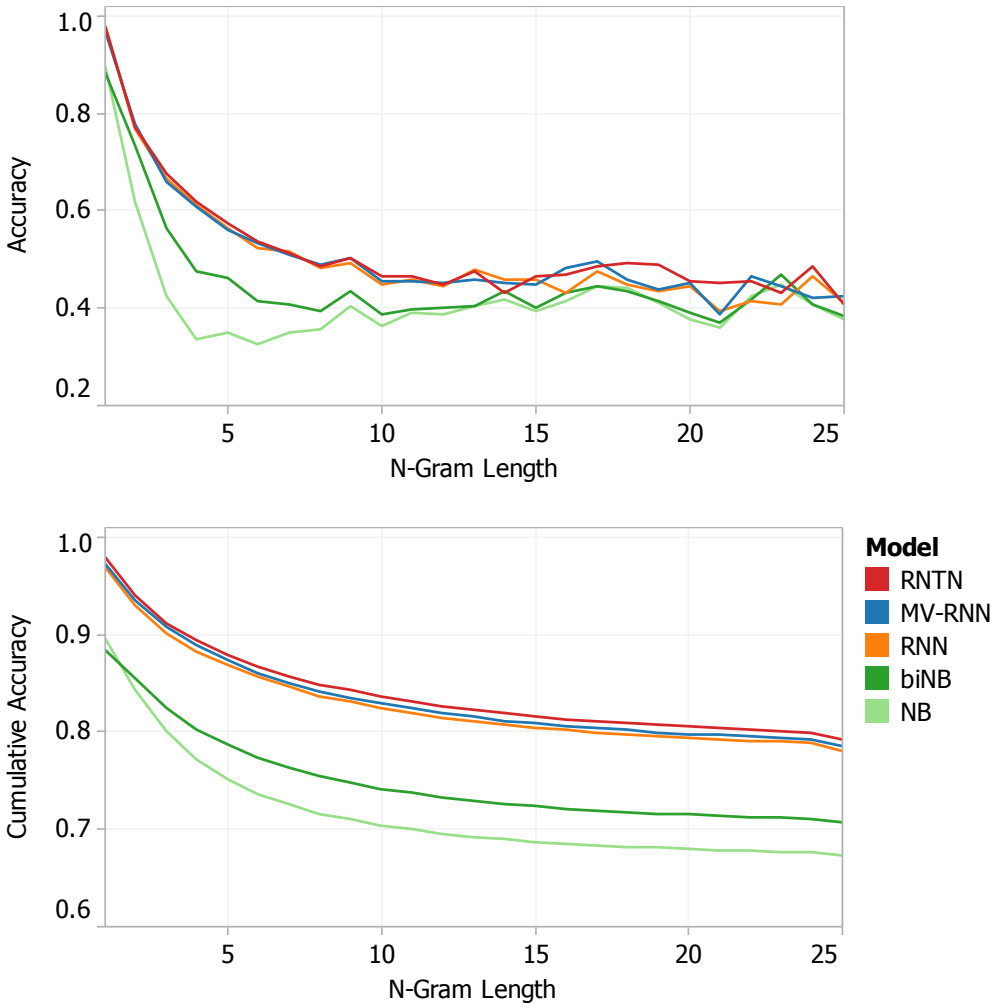


Figure 4.16: Accuracy curves for fine grained sentiment classification at each n-gram lengths. Top: Accuracy separately for each set of n-grams. Bottom: Cumulative accuracy of all $\leq n$-grams.

all phrase lengths and full sentences.

Full Sentence Binary Sentiment

This setup is comparable to previous work on the original rotten tomatoes dataset which only used full sentence labels and binary classification of positive/negative. Hence, these experiments show the improvement even baseline methods can achieve with the sentiment treebank. Table 4.7 shows results of this binary classification for

Model	Fine-grained		Positive/Negative	
	All	Root	All	Root
NB	67.2	41.0	82.6	81.8
SVM	64.3	40.7	84.6	79.4
BiNB	71.0	41.9	82.7	83.1
VecAvg	73.3	32.7	85.1	80.1
RNN	79.0	43.2	86.1	82.4
MV-RNN	78.7	44.4	86.8	82.9
RNTN	**80.7**	**45.7**	**87.6**	**85.4**

Table 4.7: Accuracy for fine grained (5-class) and binary predictions at the sentence level (root) and for all nodes.

both all phrases and for only full sentences. The previous state of the art was below 80% (Socher et al., 2012b). With the coarse bag of words annotation for training, many of the more complex phenomena could not be captured, even by more powerful models. The combination of the new sentiment treebank and the RNTN pushes the state of the art on short phrases up to 85.4%.

Model Analysis: Contrastive Conjunction

In this section, I use a subset of the test set which includes only sentences with an 'X *but* Y' structure: A phrase X being followed by *but* which is followed by a phrase Y. The conjunction is interpreted as an argument for the second conjunct, with the first functioning concessively (Lakoff, 1971; Blakemore, 1989; Merin, 1999). Fig. 4.17 contains an example. I analyze a strict setting, where X and Y are phrases of different sentiment (including neutral). The example is counted as correct, if the classifications for both phrases X and Y are correct. Furthermore, the lowest node that dominates both of the word *but* and the node that spans Y also have to have the same correct sentiment. For the resulting 131 cases, the RNTN obtains an accuracy of 41% compared to MV-RNN (37), RNN (36) and biNB (27).

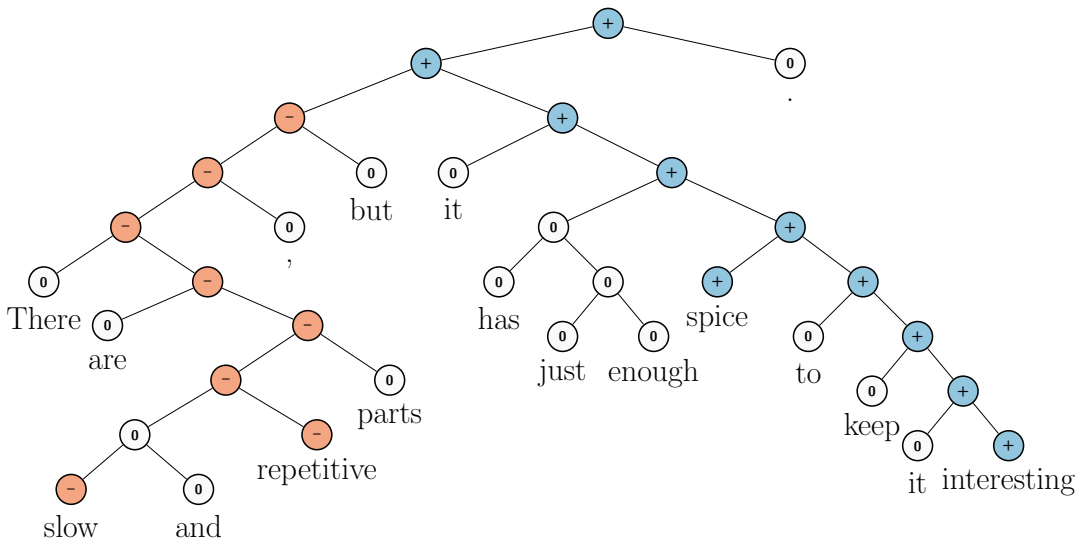


Figure 4.17: Example of correct prediction for contrastive conjunction X *but* Y.

Model Analysis: High Level Negation

I investigate two types of negation. For each type, I use a separate dataset for evaluation.

Set 1: Negating Positive Sentences. The first set contains positive sentences and their negation. In this set, the negation changes the overall sentiment of a sentence from positive to negative. Hence, I compute accuracy in terms of correct sentiment reversal from positive to negative. Fig. 4.19 shows two examples of positive negation the RNTN correctly classified, even if negation is less obvious in the case of 'least'. Table 4.8 (left) gives the accuracies over 21 positive sentences and their negation for all models. The RNTN has the highest reversal accuracy, showing its ability to structurally learn negation of positive sentences. But what if the model simply makes phrases very negative when negation is in the sentence? The next experiments show that the model captures more than such a simplistic negation rule.

Set 2: Negating Negative Sentences. The second set contains negative sentences and their negation. When negative sentences are negated, the sentiment treebank shows that overall sentiment should become *less negative*, but not necessarily

Model	Accuracy	
	Negated Positive	Negated Negative
biNB	19.0	27.3
RNN	33.3	45.5
MV-RNN	52.4	54.6
RNTN	**71.4**	**81.8**

Table 4.8: Accuracy of negation detection. Negated positive is measured as correct sentiment inversions. Negated negative is measured as increases in positive activations.

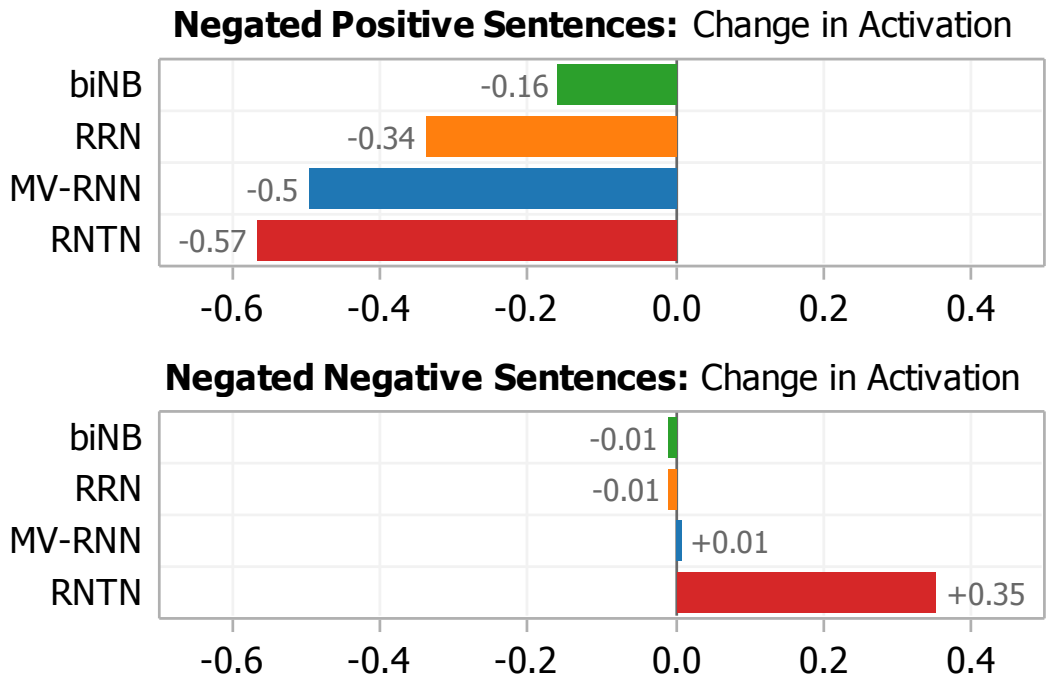


Figure 4.18: Change in activations for negations. Only the RNTN correctly captures both types. It decreases positive sentiment more when it is negated and learns that negating negative phrases (such as *not terrible*) should increase neutral and positive activations.

positive. For instance, 'The movie was terrible' is negative but the 'The movie was not terrible' says only that it was less bad than a terrible one, not that it was good (Horn, 1989; Israel, 2001). Hence, I evaluate accuracy in terms of how often each model was

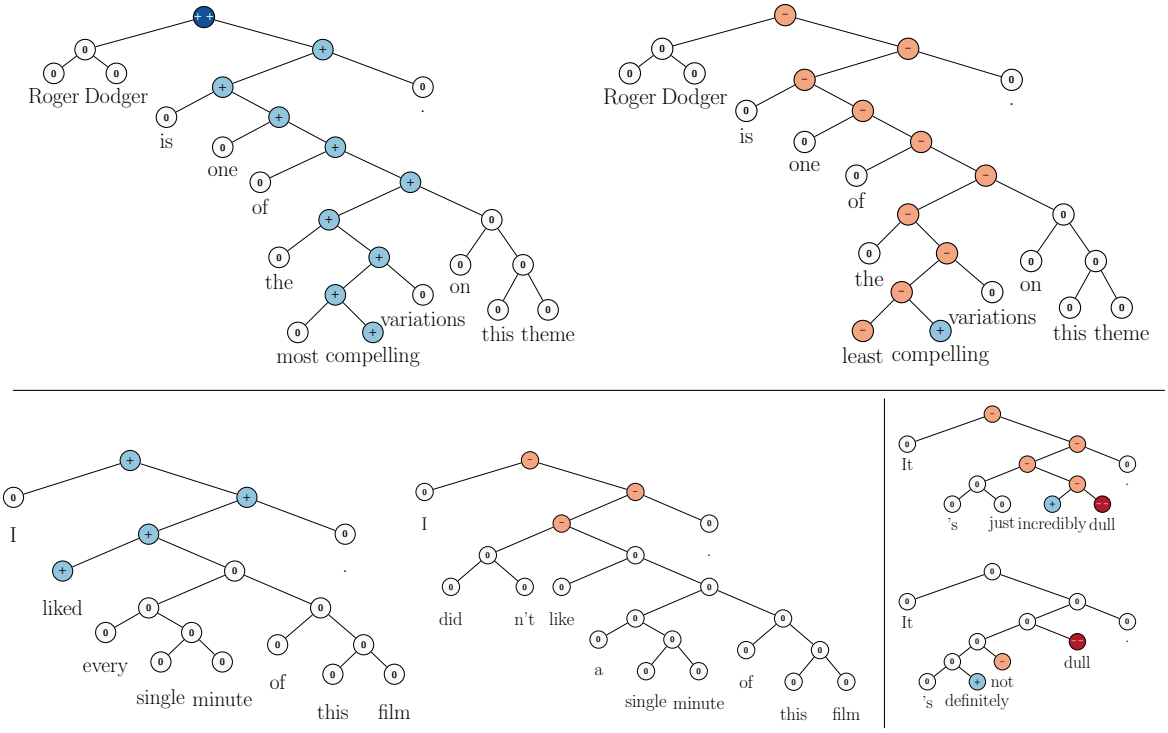


Figure 4.19: RNTN prediction of positive and negative (bottom right) sentences and their negation.

able to increase non-negative activation in the sentiment of the sentence. Table 4.8 (right) shows the accuracy. In over 81% of cases, the RNTN correctly increases the positive activations. Fig. 4.19 (bottom right) shows a typical case in which sentiment was made more positive by switching the main class from negative to neutral even though both *not* and *dull* were negative. Fig. 4.18 shows the changes in activation for both sets. Negative values indicate a decrease in average positive activation (for set 1) and positive values mean an increase in average positive activation (set 2). The RNTN has the largest shifts in the correct directions. Therefore I can conclude that the RNTN is best able to identify the effect of negations upon both positive and negative sentiment sentences.

n	Most positive n-grams	Most negative n-grams
1	engaging; best; powerful; love; beautiful	bad; dull; boring; fails; worst; stupid; painfully
2	excellent performances; A masterpiece; masterful film; wonderful movie; marvelous performances	worst movie; very bad; shapeless mess; worst thing; instantly forgettable; complete failure
3	an amazing performance; wonderful all-ages triumph; a wonderful movie; most visually stunning	for worst movie; A lousy movie; a complete failure; most painfully marginal; very bad sign
5	nicely acted and beautifully shot; gorgeous imagery, effective performances; the best of the year; a terrific American sports movie; refreshingly honest and ultimately touching	silliest and most incoherent movie; completely crass and forgettable movie; just another bad movie. A cumbersome and cliche-ridden movie; a humorless, disjointed mess
8	one of the best films of the year; A love for films shines through each frame; created a masterful piece of artistry right here; A masterful film from a master filmmaker,	A trashy, exploitative, thoroughly unpleasant experience ; this sloppy drama is an empty vessel.; quickly drags on becoming boring and predictable.; be the worst special-effects creation of the year

Table 4.9: Examples of n-grams for which the RNTN predicted the most positive and most negative responses.

Model Analysis: Most Positive and Negative Phrases

I queried the model for its predictions on what the most positive or negative n-grams are, measured as the highest activation of the most negative and most positive classes. Table 4.9 shows some phrases from the dev set which the RNTN selected for their strongest sentiment. Fig. 4.20 shows that the RNTN selects more strongly positive phrases at most n-gram lengths compared to other models.

4.3.4 Related Work

This work is connected to five different areas of NLP research, each with their own large amount of related work.

For related work on **semantic vector spaces** and **compositionality in vector**

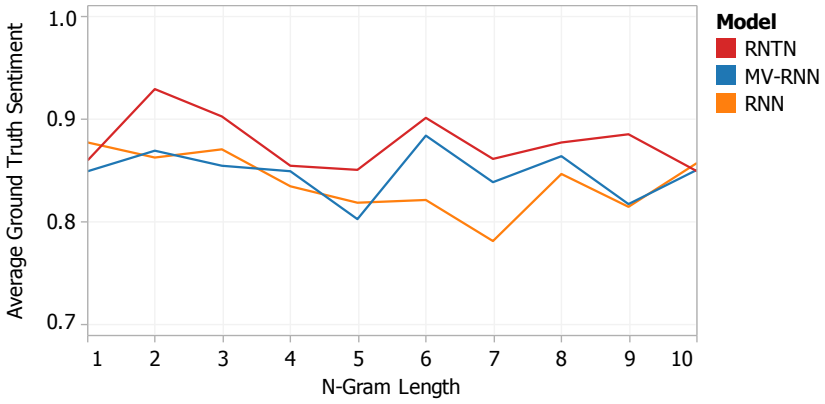


Figure 4.20: Average ground truth sentiment of top 10 most positive n-grams at various n. The RNTN correctly picks the more negative and positive examples.

spaces see Sec. 4.2.5.

Logical Form. A related field that tackles compositionality from a very different angle is that of trying to map sentences to logical form (Zettlemoyer and Collins, 2005). While these models are highly interesting and work well in closed domains and on discrete sets, they could only capture sentiment distributions using separate mechanisms beyond the currently used logical forms.

Deep Learning. Apart from the above mentioned work on RNNs, several compositionality ideas related to neural networks have been discussed by Bottou (2011) and Hinton (1990). The idea to relate inputs through three way interactions, parameterized by a tensor have been proposed for relation classification (Sutskever et al., 2009; Jenatton et al., 2012), extending Restricted Boltzmann machines (Ranzato and Hinton, 2010) and as a special layer for speech recognition (Yu et al., 2012).

Sentiment Analysis. Apart from the above-mentioned work, most approaches in sentiment analysis use bag of words representations (Pang and Lee, 2008). Snyder and Barzilay (2007) analyzed larger reviews in more detail by analyzing the sentiment of multiple aspects of restaurants, such as food or atmosphere. Several works

have explored sentiment compositionality through careful engineering of features or polarity shifting rules on syntactic structures (Polanyi and Zaenen, 2006; Nakagawa et al., 2010). For more related prior work on sentiment analysis, see Sec. 3.2.4. Since the publication of the Stanford Sentiment Treebank, many researchers have used this dataset and improved performance (Le and Mikolov., 2014; Kiritchenko et al., 2014; Dong et al., 2014).

This section introduced Recursive Neural Tensor Networks and the Stanford Sentiment Treebank. The combination of new model and data results in a system for single sentence sentiment detection that pushes state of the art by 5.4% for positive/negative sentence classification. Apart from this standard setting, the dataset also poses important new challenges and allows for new evaluation metrics. For instance, the RNTN obtains 80.7% accuracy on fine-grained sentiment prediction across all phrases and captures negation of different sentiments and scope more accurately than previous models.

4.4 Conclusion

This concludes the exploration of composition functions. For linguistically well motivated problems with very large training data (such as adverb-adjective occurrence prediction) the MV-RNN is suitable. When less data is available, one can use well motivated untying schemes as in the SU-RNN. However, from my experience until now, the composition most likely to be successful on a variety of tasks is the RNTN.

The next chapter will illustrate that not only can the objective and composition functions be tuned for specific tasks and problems but also the tree structure itself.

Chapter 5

Compositional Tree Structures Variants

Up until this chapter all RNN models of this thesis were based on constituency trees and input-specific trees. This chapter shows that these are not mandatory constraints but that various other tree structures can also be used. The first section explores syntactically untied RNNs on dependency trees for image-sentence search. The second section shows that for classifying 3d images, RNNs need not have an input-specific tree structure but instead can use multiple, fixed-tree RNNs. This model also illustrates interesting connections to widely used convolutional neural networks.

5.1 Dependency Tree RNNs - For Sentence-Image Mapping

In this section, I introduce a model, illustrated in Fig. 5.1, which learns to map sentences and images into a common embedding space in order to be able to retrieve one from the other. I assume word and image representations are first learned in their respective single modalities but finally mapped into a jointly learned multimodal embedding space.

Similar to previous sections, the model for mapping sentences into multimodal

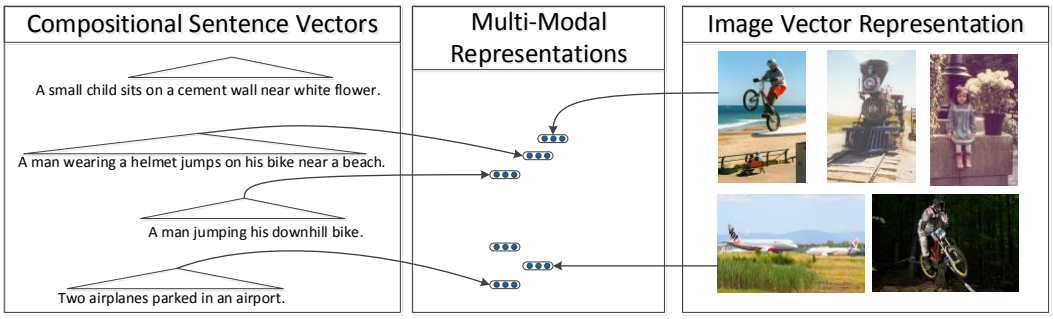


Figure 5.1: The DT-RNN learns vector representations for sentences based on their dependency trees. I learn to map the outputs of convolutional neural networks applied to images into the same space and can then compare both sentences and images. This allows us to query images with a sentence and give sentence descriptions to images.

space is based on RNNs. However, unlike RNN models of previous sections, which are based on constituency trees (CT-RNNs), the model in this section computes compositional vector representations inside dependency trees (de Marneffe et al., 2006), a formalism I will illustrate below by means of an example. The compositional vectors computed by this new dependency tree RNN (DT-RNN) better capture the meaning of sentences, where I define meaning in terms of similarity to a "visual representation" of the textual description. DT-RNN induced vector representations of sentences are more robust to changes in the syntactic structure or word order than related models such as CT-RNNs or Recurrent Neural Networks since they naturally focus on a sentence's action and its agents.

I evaluate and compare DT-RNN induced representations on their ability to use a sentence such as *"A man wearing a helmet jumps on his bike near a beach."* to find images that show such a scene. The goal is to learn sentence representations that capture the visual scene being described and to find appropriate images in the learned, multi-modal sentence-image space. Conversely, when given a query image, I would like to find a description that goes beyond a single label by providing a correct sentence describing it, a task that has recently garnered a lot of attention (Farhadi et al., 2010; Ordonez et al., 2011; Kuznetsova et al., 2012). I use the dataset introduced by Rashtchian et al. (2010) which consists of 1000 images, each with 5

descriptions. On all tasks, my model outperforms baselines and related models.

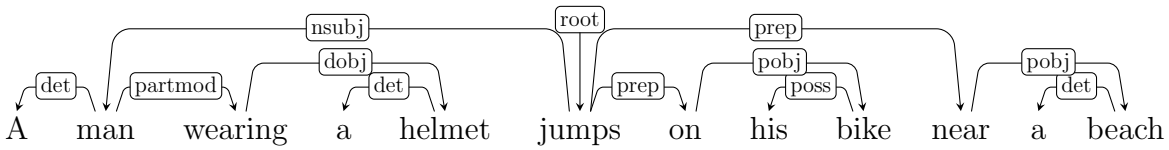


Figure 5.2: Example of a full dependency tree for a longer sentence. The DT-RNN will compute vector representations at every word that represents that word and an arbitrary number of child nodes. The final representation is computed at the *root* node, here at the verb *jumps*. Note that more important activity and object words are higher up in this tree structure.

5.1.1 Dependency-Tree Recursive Neural Networks

I first focus on the DT-RNN model that - like all previous RNN models of this thesis - computes compositional vector representations for phrases and sentences of variable length and syntactic type. In subsequent section 5.1.3 the resulting vectors will then become multimodal features by mapping images that show what the sentence describes to the same space and learning both the image and sentence mapping jointly.

The most common way of building representations for longer phrases from single word vectors is to simply linearly average the word vectors. While this bag-of-words approach can yield reasonable performance in some tasks, it gives all the words the same weight and cannot distinguish important differences in simple visual descriptions such as *The bike crashed into the standing car.* vs. *The car crashed into the standing bike.*.

The RNN models of previous chapters provided a way of combining word vectors for longer phrases that moved beyond simple averaging. They combine vectors with an RNN in binary constituency trees which have potentially many hidden layers. While the induced vector representations work very well on many tasks, they also inevitably capture a lot of syntactic structure of the sentence. However, the task of finding images from sentence descriptions requires us to be more invariant to syntactic differences. One such example are active-passive constructions. In some formalisms

(de Marneffe et al., 2006), the "by" in a passive construction can be collapsed and the semantic relationship of the following word will become "agent". For instance, *The mother hugged her child.* and *The child was hugged by its mother.* should map to roughly the same visual space. Other recursive and recurrent neural networks do not exhibit this behavior and even bag of words representations would be influenced by the words *was* and *by*. The model I describe below focuses more on recognizing actions and agents and has the potential to learn representations that are invariant to active-passive differences.

DT-RNN Inputs: Word Vectors and Dependency Trees

In order for the DT-RNN to compute a vector representation for an ordered list of m words (a phrase or sentence), I map the single words to a vector space and then parse the sentence.

As outlined in section 2.3, I map each word to a d-dimensional vector. I initialize these word vectors with the unsupervised model of Huang et al. (2012) which can learn single word vector representations from both local and global contexts. I use $d = 50$ in all experiments. The word embedding matrix X is used by finding the column index i of each word: $[w] = i$ and retrieving the corresponding column x_w from X. Henceforth, I represent an input sentence s as an ordered list of (word,vector) pairs: $s = ((w_1, x_{w_1}), \ldots, (w_m, x_{w_m}))$.

Next, the sequence of words $(w_1, \ldots, w_m)$ is parsed by the dependency parser of de Marneffe et al. (2006). Fig. 5.1 shows an example tree. I can represent a dependency tree d of a sentence s as an ordered list of (child,parent) indices: $d(s) = \{(i, j)\}$, where every child word in the sequence $i = 1, \ldots, m$ is present and has any word $j \in \{1, \ldots, m\} \cup \{0\}$ as its parent. The root word has as its parent 0 and we notice that the same word can be a parent between zero and m number of times. Without loss of generality, I assume that these indices form a tree structure. To summarize, the input to the DT-RNN for each sentence is the pair (s, d): the words and their vectors and the dependency tree.

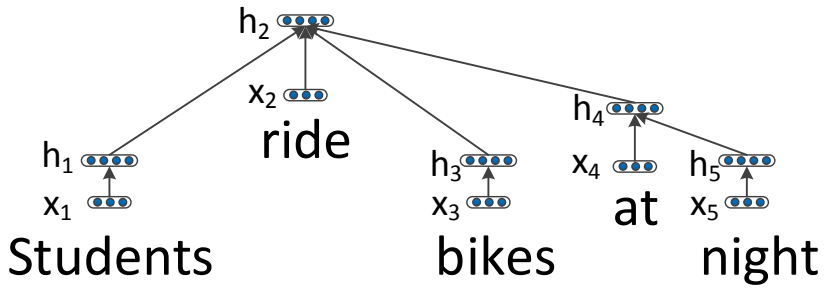


Figure 5.3: Example of a DT-RNN tree structure for computing a sentence representation in a bottom up fashion.

Forward Propagation in DT-RNNs

Given these two inputs, I now illustrate how the DT-RNN computes parent vectors. I will use the following sentence as a running example: $Students_1$ $ride_2$ $bikes_3$ at_4 $night_5$. Fig. 5.3 shows its tree and computed vector representations. The dependency tree for this sentence can be summarized by the following set of (child, parent) edges: $d = \{(1,2),(2,0),(3,2),(4,2),(5,4)\}$.

The DT-RNN model will compute parent vectors at each word that include all the dependent (children) nodes in a bottom up fashion using a compositionality function g_θ which is parameterized by all the model parameters θ. To this end, the algorithm searches for nodes in a tree that have either (i) no children or (ii) whose children have already been computed and then computes the corresponding vector.

In my example, the words x_1, x_3, x_5 are leaf nodes and hence, I can compute their corresponding hidden nodes via:

$$h_c = g_\theta(x_c) = f(W_v x_c) \qquad \text{for} \quad c = 1, 3, 5, \tag{5.1}$$

where I compute the hidden vector at position c via my general composition function g_θ. In the case of leaf nodes, this composition function becomes simply a linear layer, parameterized by $W_v \in \mathbb{R}^{n \times d}$, followed by a nonlinearity. I cross-validate over using no nonlinearity ($f = \text{id}$), tanh, sigmoid or rectified linear units ($f = \max(0, x)$), but

generally find tanh to perform best.

The final sentence representation I want to compute is at h_2, however, since I still do not have h_4, I compute that one next:

$$h_4 = g_\theta(x_4, h_5) = f(W_v x_4 + W_{r1} h_5), \tag{5.2}$$

where I use the same W_v as before to map the word vector into hidden space but I now also have a linear layer that takes as input h_5, the only child of the fourth node. The matrix $W_{r1} \in \mathbb{R}^{n \times n}$ is used because node 5 is the first child node on the right side of node 4. Generally, I have multiple matrices for composing with hidden child vectors from the right and left sides: $W_{r\cdot} = (W_{r1}, \ldots, W_{rk_r})$ and $W_{l\cdot} = (W_{l1}, \ldots, W_{lk_l})$. The number of needed matrices is determined by the data by simply finding the maximum numbers of left k_l and right k_r children any node has. If at test time a child appeared at an even large distance (this does not happen in my test set), the corresponding matrix would be the identity matrix.

Now that all children of h_2 have their hidden vectors, I can compute the final sentence representation via:

$$h_2 = g_\theta(x_2, h_1, h_3, h_4) = f(W_v x_2 + W_{l1} h_1 + W_{r1} h_3 + W_{r2} h_4). \tag{5.3}$$

Notice that the children are multiplied by matrices that depend on their location relative to the current node.

Another modification that improves the mean rank by approximately 6 in image search on the dev set is to weight nodes by the number of words underneath them and normalize by the sum of words under all children. This encourages the intuitive desideratum that nodes describing longer phrases are more important. Let $\ell(i)$ be the number of leaf nodes (words) under node i and $C(i, y)$ be the set of child nodes of node i in dependency tree y. The final composition function for a node vector h_i becomes:

$$h_i = f\left(\frac{1}{\ell(i)}\left(W_v x_i + \sum_{j \in C(i)} \ell(j) W_{\text{pos}(i,j)} h_j\right)\right), \tag{5.4}$$

where by definition $\ell(i) = 1 + \sum_{j \in C(i)} \ell(j)$ and $\text{pos}(i,j)$ is the relative position of child j with respect to node i, e.g. $l1$ or $r2$ in Eq. 5.3.

Semantic Dependency Tree RNNs

An alternative is to condition the weight matrices on the semantic relations given by the dependency parser. I use the collapsed tree formalism of the Stanford dependency parser (de Marneffe et al., 2006). With such a semantic untying of the weights, the DT-RNN makes better use of the dependency formalism and could give active-passive reversals similar semantic vector representation. The equation for this semantic DT-RNN (**SDT-RNN**) is the same as the one above except that the matrices $W_{\text{pos}(i,j)}$ are replaced with matrices based on the dependency relationship. There are a total of 141 unique such relationships in the dataset. However, most are very rare. For examples of semantic relationships, see Fig. 5.1 and the model analysis section 5.1.4.

This forward propagation can be used for computing compositional vectors and in Sec. 5.1.3 I will explain the objective function in which these are trained.

Comparison to Previous RNN Models

The DT-RNN has several important differences to the other RNN models of this thesis which are based on constituency trees (CT-RNNs) and use the standard composition function to compute a hidden parent vector h from exactly two child vectors (c_1, c_2) in a binary tree: $h = f\left(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}\right)$. This can be rewritten to show the similarity to the DT-RNN as $h = f(W_{l1}c_1 + W_{r1}c_2)$. However, there are several important differences.

Note first that in previous RNN models the parent vectors were of the same dimensionality to be recursively compatible and be used as input to the next composition. In contrast, my new model first maps single words into a hidden space and then parent nodes are composed from these hidden vectors. This allows a higher capacity representation which is especially helpful for nodes that have many children.

Secondly, the DT-RNN allows for n-ary nodes in the tree. This is an improvement that is possible even for constituency tree CT-RNNs but it has not been explored in

previous models.

Third, due to computing parent nodes in constituency trees, previous models had the problem that words that are merged last in the tree have a larger weight or importance in the final sentence representation. This can be problematic since these are often simple non-content words, such as a leading 'But,'. While such single words can be important for tasks such as sentiment analysis, I argue that for describing visual scenes the DT-RNN captures the more important effects: The dependency tree structures push the central content words such as the main action or verb and its subject and object to be merged last and hence, by construction, the final sentence representation is more robust to less important adjectival modifiers, word order changes, etc.

Fourth, I allow some untying of weights depending on either how far away a constituent is from the current word or what its semantic relationship is.

Now that I can compute compositional vector representations for sentences, the next section describes how I represent images.

5.1.2 Learning Image Representations with Neural Networks

The image features that I use in my experiments are extracted from a deep convolutional neural network, replicated from the one described in Le et al. (2012). The network was trained using both unlabeled data (random web images) and labeled data to classify 22,000 categories in ImageNet (Deng et al., 2009). I then used the features at the last layer, before the classifier, as the feature representation in my experiments. The dimension of the feature vector of the last layer is 4,096. The details of the model and its training procedures are as follows.

The architecture of the network can be seen in Figure 5.4. The network takes 200x200 pixel images as inputs and has 9 layers. The layers consist of three sequences of filtering, pooling and local contrast normalization (Jarrett et al., 2009). The pooling function is L2 pooling of the previous layer (taking the square of the filtering units, summing them up in a small area in the image, and taking the square-root). The local contrast normalization takes inputs in a small area of the lower layer,

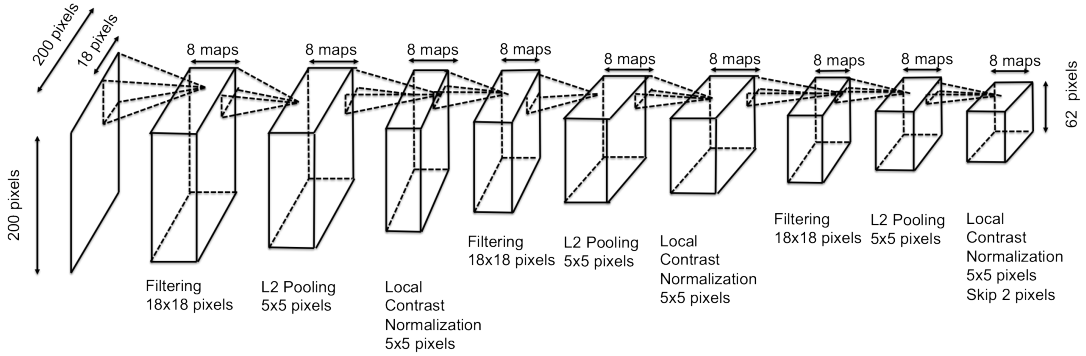


Figure 5.4: The architecture of the visual model. This model has 3 sequences of filtering, pooling and local contrast normalization layers. The learnable parameters are the filtering layer. The filters are not shared, i.e., the network is nonconvolutional.

subtracts the mean and divides by the standard deviation.

The network was first trained using an unsupervised objective: trying to reconstruct the input while keeping the neurons sparse. In this phase, the network was trained on 20 million images randomly sampled from the web. Quoc Le resized a given image so that its short dimension has 200 pixels. He then cropped a fixed size 200x200 pixel image right at the center of the resized image. This means a fraction of the long dimension of the image may be discarded.

After unsupervised training, Quoc Le used ImageNet (Deng et al., 2009) to adjust the features in the entire network. The ImageNet dataset has 22,000 categories and 14 million images. The number of images in each category is equal across categories. The 22,000 categories are extracted from WordNet.

To speed up the supervised training of this network, Quoc Le made a simple modification to the algorithm described in Le et al. (2012): adding a "bottleneck" layer in between the last layer and the classifier to reduce the number of connections. Quoc Le added one "bottleneck" layer which has 4,096 units in between the last layer of the network and the softmax layer. This newly-added layer is fully connected to the previous layer and has a linear activation function. The total number of connections of this network is approximately 1.36 billion.

The network was trained again using the supervised objective of classifying the 22,000 classes in ImageNet. Most features in the networks are local, which allows

1. A woman and her dog watch the cameraman in their living with wooden floors.
2. A woman sitting on the couch while a black faced dog runs across the floor.
3. A woman wearing a backpack sits on a couch while a small dog runs on the hardwood floor next to her.
4. A women sitting on a sofa while a small Jack Russell walks towards the camera.
5. White and black small dog walks toward the camera while woman sits on couch, desk and computer seen in the background as well as a pillow, teddy bear and moggie toy on the wood floor.

1. A man in a cowboy hat check approaches a small red sports car.
2. The back and left side of a red Ferrari and two men admiring it.
3. The sporty car is admired by passer by.
4. Two men next to a red sports car in a parking lot.
5. Two men stand beside a red sports car.

Figure 5.5: Examples from the dataset of images and their sentence descriptions Rashtchian et al. (2010). Sentence length varies greatly and different objects can be mentioned first. Hence, models have to be invariant to word ordering.

model parallelism. Data parallelism by asynchronous SGD was also employed as in Le et al. (2012). The entire training, both unsupervised and supervised, took 8 days on a large cluster of machines. This network achieves 18.3% precision@1 on the full ImageNet dataset (Release Fall 2011).

We will use the features at the bottleneck layer as the feature vector z of an image. Each scaled and cropped image is presented to my network. The network then performs a feedforward computation to compute the values of the bottleneck layer. This means that every image is represented by a fixed length vector of 4,096 dimensions. Note that during training, no aligned sentence-image data was used and the ImageNet classes do not fully intersect with the words used in the image-sentence dataset.

5.1.3 Multimodal Mappings

The previous two sections described how I can map sentences into a $d = 50$-dimensional space and how to extract high quality image feature vectors of 4096 dimensions. I now define my final multimodal objective function for learning joint image-sentence representations with these models. My training set consists of N images and their feature vectors z_i and each image has 5 sentence descriptions $s_{i1}, \ldots, s_{i5}$ for which I use the DT-RNN to compute vector representations. See Fig. 5.5 for examples from

the dataset. For training, I use a max-margin objective function which intuitively trains pairs of correct image and sentence vectors to have high inner products and incorrect pairs to have low inner products. Let $v_i = W_I z_i$ be the mapped image vector and $y_{ij} = DTRNN_\theta(s_{ij})$ the composed sentence vector. I define $\mathcal{S}$ to be the set of all sentence indices and $\mathcal{S}(i)$ the set of sentence indices corresponding to image i. Similarly, $\mathcal{I}$ is the set of all image indices and $\mathcal{I}(j)$ is the image index of sentence j. The set $\mathcal{P}$ is the set of all correct image-sentence training pairs (i, j). The ranking cost function to minimize is then:

$$\begin{aligned} J(W_I, \theta) = & \sum_{(i,j)\in\mathcal{P}} \sum_{c\in\mathcal{S}\setminus\mathcal{S}(i)} \max(0, \Delta - v_i^T y_j + v_i^T y_c) \\ & + \sum_{(i,j)\in\mathcal{P}} \sum_{c\in\mathcal{I}\setminus\mathcal{I}(j)} \max(0, \Delta - v_i^T y_j + v_c^T y_j), \end{aligned} \tag{5.5}$$

where θ are the language composition matrices, and both second sums are over other sentences coming from different images and vice versa. The hyperparameter Δ is the margin. The margin is found via cross validation on the dev set and usually around 1.

The final objective also includes the regularization term $\lambda\left(\|\theta\|_2^2 + \|W_I\|_F\right)$. Both the visual model and the word vector learning require a very large amount of training data and both have a huge number of parameters. Hence, to prevent overfitting, I assume their weights are fixed and only train the DT-RNN parameters W_I. If larger training corpora become available in the future, training both jointly becomes feasible and would present a very promising direction. I use a modified version of AdaGrad (Duchi et al., 2011) for optimization of both W_I and the DT-RNN as well as the other baselines (except kCCA). I modify it by resetting all squared gradient sums to 1 every 5 epochs. With both images and sentences in the same multimodal space, I can easily query the model for similar images or sentences by finding the nearest neighbors in terms of negative inner products.

An alternative objective function is based on the squared loss

$$J(W_I, \theta) = \sum_{(i,j)\in\mathcal{P}} \|v_i - y_j\|_2^2. \tag{5.6}$$

This requires an alternating minimization scheme that first trains only W_I, then fixes W_I and trains the DT-RNN weights θ and then repeats this several times. I find that the performance with this objective function (paired with finding similar images using Euclidean distances) is worse for all models than the margin loss of Eq. 5.5. In addition kCCA should also be used with inner products in the multimodal space.

5.1.4 Experiments

I use the dataset of Rashtchian et al. (2010) which consists of 1000 images, each with 5 sentences. See Fig. 5.5 for examples.

I evaluate and compare the DT-RNN in three different experiments. First, I analyze how well the sentence vectors capture similarity in visual meaning. Then I analyze *Image Search with Query Sentences*: to query each model with a sentence in order to find an image showing that sentence's visual 'meaning.' The last experiment *Describing Images by Finding Suitable Sentences* does the reverse search where I query the model with an image and try to find the closest textual description in the embedding space.

In my comparison to other methods I focus on those models that can also compute fixed, continuous vectors for sentences. In particular, I compare to the RNN model on constituency trees (as described in chapter 3), a standard recurrent neural network; a simple bag-of-words baseline which averages the words. All models use the word vectors provided by Huang et al. (2012) and do not update them as discussed above. Models are trained with their corresponding gradients and backpropagation techniques. A standard recurrent model is used where the hidden vector at word index t is computed from the hidden vector at the previous time step and the current word vector: $h_t = f(W_h h_{t-1} + W_x x_t)$. During training, I take the last hidden vector of the sentence chain and propagate the error into that. It is also this vector that is used to represent the sentence.

Other possible comparisons are to the very different models mentioned in the related work section. These models use a lot more task-specific engineering, such as running object detectors with bounding boxes, attribute classifiers, scene classifiers,

CRFs for composing the sentences, etc. Another line of work uses large sentence-image aligned resources (Kuznetsova et al., 2012), whereas I focus on easily obtainable training data of each modality separately and a rather small multimodal corpus.

In the experiments I split the data into 800 training, 100 development and 100 test images. Since there are 5 sentences describing each image, I have 4000 training sentences and 500 testing sentences. The dataset has 3020 unique words, half of which only appear once. Hence, the unsupervised, pretrained semantic word vector representations are crucial. Word vectors are not fine-tuned during training. Hence, the main parameters are the DT-RNN's $W_{l\cdot}, W_{r\cdot}$ or the semantic matrices of which there are 141 and the image mapping W_I. For both DT-RNNs the weight matrices are initialized to block identity matrices plus Gaussian noise. Word vectors and hidden vectors are set o length 50. Using the development split, I found $\lambda = 0.08$ and the learning rate of AdaGrad to 0.0001. The best model uses a margin of $\Delta = 3$.

Inspired by Socher and Fei-Fei (2010) and Hodosh et al. (2013) I also compare to kernelized Canonical Correlation Analysis (kCCA). I use the average of word vectors for describing sentences and the same powerful image vectors as before. I use the code of Socher and Fei-Fei (2010). Technically, one could combine the recently introduced deep CCA (Andrew et al., 2013) and train the recursive neural network architectures with the CCA objective. I leave this to future work. With linear kernels, kCCA does well for image search but is worse for sentence self similarity and describing images with sentences close-by in embedding space. All other models are trained by replacing the DT-RNN function in Eq. 5.5.

Similarity of Sentences Describing the Same Image

In this experiment, I first map all 500 sentences from the test set into the multi-modal space. Then for each sentence, I find the nearest neighbor sentences in terms of inner products. I then sort these neighbors and record the rank or position of the nearest sentence *that describes the same image*. If all the images were very unique and the visual descriptions close-paraphrases and consistent, I would expect a very low rank. However, usually a handful of images are quite similar (for instance, there are various images of airplanes flying, parking, taxiing or waiting on the runway) and sentence

Sentences Similarity for Image

Model	Mean Rank
Random	101.1
BoW	11.8
CT-RNN	15.8
Recurrent NN	18.5
kCCA	10.7
DT-RNN	11.1
SDT-RNN	**10.5**

Image Search

Model	Mean Rank
Random	52.1
BoW	14.6
CT-RNN	16.1
Recurrent NN	19.2
kCCA	15.9
DT-RNN	13.6
SDT-RNN	**12.5**

Describing Images

Model	Mean Rank
Random	92.1
BoW	21.1
CT-RNN	23.9
Recurrent NN	27.1
kCCA	18.0
DT-RNN	19.2
SDT-RNN	**16.9**

Table 5.1: **Top Left:** Comparison of methods for sentence similarity judgments. Lower numbers are better since they indicate that sentences describing the same image rank more highly (are closer). The ranks are out of the 500 sentences in the test set. **Top Right:** Comparison of methods for image search with query sentences. Shown is the average rank of the single correct image that is being described. **Bottom:** Average rank of a correct sentence description for a query image.

descriptions can vary greatly in detail and specificity for the same image.

Table 5.1 (top left) shows the results. Averaging the high quality word vectors already captures a lot of similarity. The chain structure of a standard recurrent neural net performs worst since its representation is dominated by the last words in the sequence which may not be as important as earlier words.

Image Search with Query Sentences

This experiment evaluates how well I can find images that display the visual meaning of a given sentence. I first map a query sentence into the vector space and then find images in the same space using simple inner products. As shown in Table 5.1 (top right), the new DT-RNN outperforms all other models.

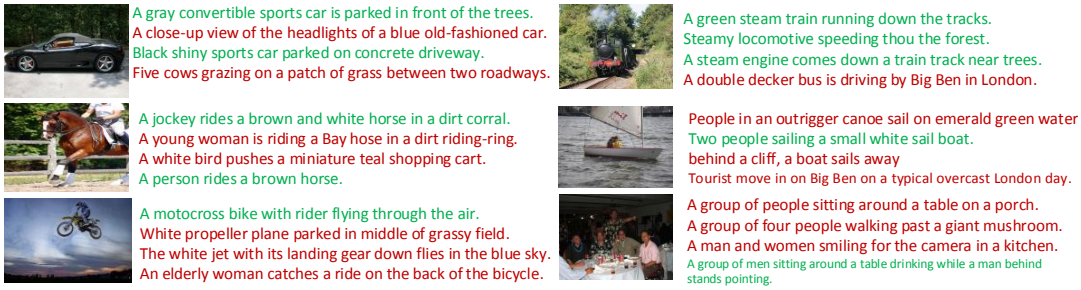


Figure 5.6: Images and their sentence descriptions assigned by the DT-RNN. Green sentences are correct, red ones are close to the image vector but incorrect.

Describing Images by Finding Suitable Sentences

Lastly, I repeat the above experiments but with roles reversed. For an image, I search for suitable textual descriptions again simply by finding close-by sentence vectors in the multi-modal embedding space. Table 5.1 (bottom) shows that the DT-RNN again outperforms related models. Fig. 5.6 shows sentences that were found by using the image vector for search. The average ranking of 25.3 for a correct sentence description is out of 500 possible sentences. A random assignment would give an average ranking of 100.

Analysis: Squared Error Loss vs. Margin Loss

I analyze the influence of the multimodal loss function on the performance. In addition, I compare using Euclidean distances instead of inner products. Table 5.2 shows that performance is worse for all models in this setting.

Analysis: Recall at n vs Mean Rank

Hodosh et al. (2013) and other related work use recall at n as an evaluation measure. Recall at n captures how often one of the top n closest vectors were a correct image or sentence and gives a good intuition of how a model would perform in a ranking task that presents n such results to a user. Below, I compare three commonly used and high performing models: bag of words, kCCA and my SDT-RNN on this different

Image Search		*Describing Images*	
Model	mRank	Model	mRank
BoW	24.7	BoW	30.7
CT-RNN	22.2	CT-RNN	29.4
Recurrent NN	28.4	Recurrent NN	31.4
kCCA	13.7	kCCA	38.0
DT-RNN	13.3	DT-RNN	26.8
SDT-RNN	15.8	SDT-RNN	37.5

Table 5.2: Results of multimodal ranking when models are trained with a squared error loss and using Euclidean distance in the multimodal space. Better performance is reached for all models when trained in a max-margin loss and using inner products as in the previous table.

Image Search				
Model	mRank $\triangle$	R@1 $\bigtriangledown$	R@5 $\bigtriangledown$	R@10 $\bigtriangledown$
BoW	14.6	15.8	42.2	60.0
kCCA	15.9	**16.4**	41.4	58.0
SDT-RNN	**12.5**	**16.4**	**46.6**	**65.6**
Describing Images				
BoW	21.1	19.0	38.0	57.0
kCCA	18.0	21.0	**47.0**	61.0
SDT-RNN	**16.9**	**23.0**	45.0	**63.0**

Table 5.3: Evaluation comparison between mean rank of the closest correct image or sentence (lower is better $\triangle$) with recall at different thresholds (higher is better, $\bigtriangledown$). With one exception (R@5, bottom table), the SDT-RNN outperforms the other two models and all other models I did not include here.

metric. Table 5.3 shows that the measures do correlate well and the SDT-RNN also performs best on the multimodal ranking tasks when evaluated with this measure.

Error Analysis

In order to understand the main problems with the composed sentence vectors, I analyze the sentences that have the worst nearest neighbor rank between each other. I find that the main failure mode of the SDT-RNN occurs when a sentence that should describe the same image does not use a verb but the other sentences of that image do include a verb. For example, the following sentence pair has vectors that are very

far apart from each other even though they are supposed to describe the same image:

1. A blue and yellow airplane flying straight down while emitting white smoke
2. Airplane in dive position

Generally, as long as both sentences either have a verb or do not, the SDT-RNN is more robust to different sentence lengths than bag of words representations.

Model Analysis: Semantic Composition Matrices

The best model uses composition matrices based on semantic relationships from the dependency parser. I give some insights into what the model learns by listing the composition matrices with the largest Frobenius norms. Intuitively, these matrices have learned larger weights that are being multiplied with the child vector in the tree and hence that child will have more weight in the final composed parent vector. In decreasing order of Frobenius norm, the relationship matrices are: nominal subject, possession modifier (e.g. their), passive auxiliary, preposition *at*, preposition *in front of*, passive auxiliary, passive nominal subject, object of preposition, preposition *in* and preposition *on*.

The model learns that nouns are very important as well as their spatial prepositions and adjectives.

5.1.5 Related Work

The presented model is connected to several areas of NLP and vision research, each with a large amount of related work.

Semantic Vector Spaces and Their Compositionality. For related work in this area, see Sec. 4.2.5. In this section, I compared to supervised compositional models that can learn task-specific vector representations such as constituency tree recursive neural networks (see chapter 3), chain structured recurrent neural networks and other baselines.

Multimodal Embeddings. Multimodal embedding methods project data from multiple sources such as sound and video (Ngiam et al., 2011) or images and text.

Socher and Fei-Fei (2010) project words and image regions into a common space using kernelized canonical correlation analysis to obtain state of the art performance in annotation and segmentation. Similar to my work, they use unsupervised large text corpora to learn semantic word representations. Among other recent work is that by Srivastava and Salakhutdinov (2012) who developed multimodal Deep Boltzmann Machines. Similar to their work, I use techniques from the broad field of deep learning to represent images and words.

Recently, single word vector embeddings have been used for zero shot learning (Socher et al., 2013c). Mapping images to word vectors enabled this system to classify images as depicting objects such as "cat" without seeing any examples of this class. Related work has also been presented at NIPS (Socher et al., 2013b; Frome et al., 2013). The model in this section moves zero-shot learning beyond single categories per image and extends it to unseen phrases and full length sentences, making use of similar ideas of semantic spaces grounded in visual knowledge.

Detailed Image Annotation. Interactions between images and texts is a growing research field. Early work in this area includes generating single words or fixed phrases from images (Duygulu et al., 2002; Barnard et al., 2003) or using contextual information to improve recognition (Gupta and Davis, 2008; Torralba et al., 2010).

Apart from a large body of work on single object image classification (Le et al., 2012), there is also work on attribute classification and other mid-level elements (Kumar et al., 2009), some of which I hope to capture with my approach as well.

My work is close in spirit to recent work on describing images with more detailed, longer textual descriptions. In particular, Yao et al. (2010) describe images using hierarchical knowledge and humans in the loop. In contrast, my work does not require human interactions. Farhadi et al. (2010) and Kulkarni et al. (2011), on the other hand, use a more automatic method to parse images. For instance, the former approach uses a single triple of objects estimated for an image to retrieve sentences from a collection written to describe similar images. It forms representations to describe 1 object, 1 action, and 1 scene. Kulkarni et al. (2011) extends their method to describe an image with multiple objects. None of these approaches have used a

compositional sentence vector representation and they require specific language generation techniques and sophisticated inference methods. Since my model is based on neural networks inference it is fast and simple. Kuznetsova et al. (2012) use a very large parallel corpus to connect images and sentences. Feng and Lapata (2013) use a large dataset of captioned images and experiments with both extractive (search) and abstractive (generation) models.

Most related is the very recent work of Hodosh et al. (2013). They too evaluate using a ranking measure. In my experiments, I compare to kernelized Canonical Correlation Analysis which is the main technique in their experiments. Karpathy et al. (2014) also use CNNs and deep learning but embed dependency tree fragments and use a max-margin objective to find corresponding bounding boxes in the image.

The next section is the last section to introduce another type of RNN model: One that is based on fixed, input-independent tree structures.

5.2 Multiple Fixed Structure Trees - For 3d Object Recognition

All RNN models so far in this thesis assumed that tree structures are input dependent and that there is only a single forward tree. Unfortunately, the input dependence and different tree structures prevent the models from being implemented in a very efficient manner and (in the language case) require a parser. Taking inspiration from convolutional neural networks (LeCun et al., 1998), I extend the RNN family by allowing multiple RNNs, each acting essentially as a filter to capture certain phenomena.

In this section, I use 3d object recognition as a motivating application to develop a fixed tree RNN. Object recognition is one of the hardest problems in computer vision and important for making robots useful in home environments. New sensing technology, such as the Kinect, that can record high quality RGB and depth images (RGB-D) has now become affordable and could be combined with standard vision systems in household robots. The depth modality provides useful extra information to the complex problem of general object detection since depth information is invariant

to lighting or color variations, provides geometrical cues and allows better separation from the background. Most recent methods for object recognition with RGB-D images use hand-designed features such as SIFT for 2d images (Lai et al., 2011), Spin Images (Johnson, 1997) for 3D point clouds, or specific color, shape and geometry features (Koppula et al., 2011). In this section, I introduce a convolutional-recursive deep learning model for object recognition that can learn from raw RGB-D (color and depth) images. Fig. 5.7 outlines the approach.

My model starts with raw RGB and depth images and first separately extracts features from them. Each modality is first given to a single convolutional neural net layer (CNN, LeCun and Bengio (1995)) which provides useful translational invariance of low level features such as edges and allows parts of an object to be deformable to some extent. The pooled filter responses are then given to an RNN which can learn compositional features and part interactions.

My previous work on RNNs in natural language processing and computer vision (as described in Sec. 3.1) (i) used a different tree structure for each input, (ii) employed a single RNN with one set of weights, (iii) restricted tree structures to be strictly binary, and (iv) trained the RNN with backpropagation through structure (see Sec. 3.1.4 for details). I expand and investigate the space of possible RNN-based architectures in these four dimensions by using fixed tree structures across all images, using multiple RNNs on the same input and allowing n-ary trees. I show that because of the CNN layer, fixing the tree structure does not hurt performance and it allows us to speed up recognition. Similar to other work in deep learning (Coates and Ng, 2011; Le et al., 2012), I show that performance of RNN models can improve with an increasing number of features. The hierarchically composed RNN features of each modality are concatenated and given to a joint softmax classifier.

I further demonstrate that RNNs with random weights can also produce high quality features. So far, random weights have only been shown to work for convolutional neural networks (Jarrett et al., 2009; Saxe et al., 2011). Because the supervised training reduces to optimizing the weights of the final softmax classifier, a large set of RNN architectures can quickly be explored. By combining the above ideas, I obtain a state of the art system for classifying 3D objects which is extremely fast to train and

highly parallelizable at test time. The main bottleneck of my method is the single CNN layer. However, fast GPU implementations for CNNs exist (Ciresan et al., 2011; Farabet et al., 2010; Krizhevsky et al., 2012).

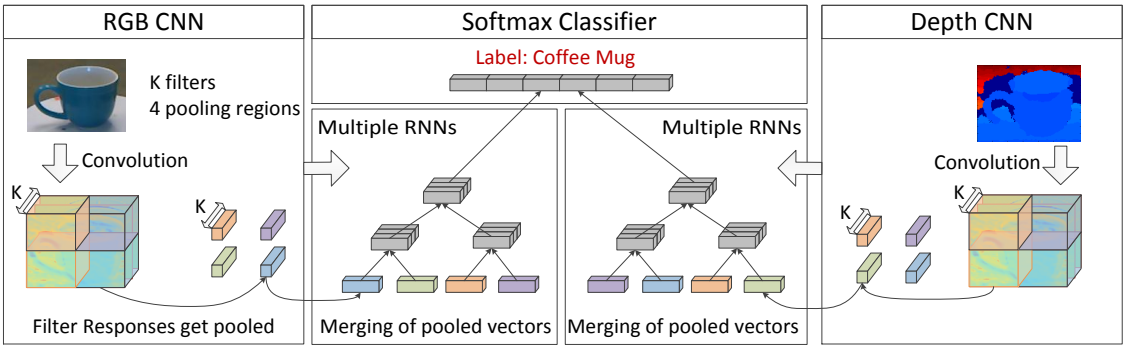


Figure 5.7: An overview of the model: A single CNN layer extracts low level features from RGB and depth images. Both representations are given as input to a set of RNNs with random weights. Each of the many RNNs then recursively maps the features into a lower dimensional space. The concatenation of all the resulting vectors forms the final feature vector for a softmax classifier.

I first briefly describe the unsupervised learning of filter weights and their convolution to obtain low level features. Next I give details of how multiple random RNNs can be used to obtain high level features of the entire image. In my experiments I show quantitative comparisons of different models, analyze model ablations and describe my state of the art results on the RGB-D dataset of Lai et al. (2011).

5.2.1 Convolutional-Recursive Neural Networks

I focus on the RGB-D dataset of Lai et al. (2011) which consists of 51 object classes. Each class has about 5 instances and each instance has roughly 600 images from different viewpoints. All images are resized to $d_I \times d_I = 148 \times 148$ pixels. More details on the dataset and pre-processing are in the experiments section.

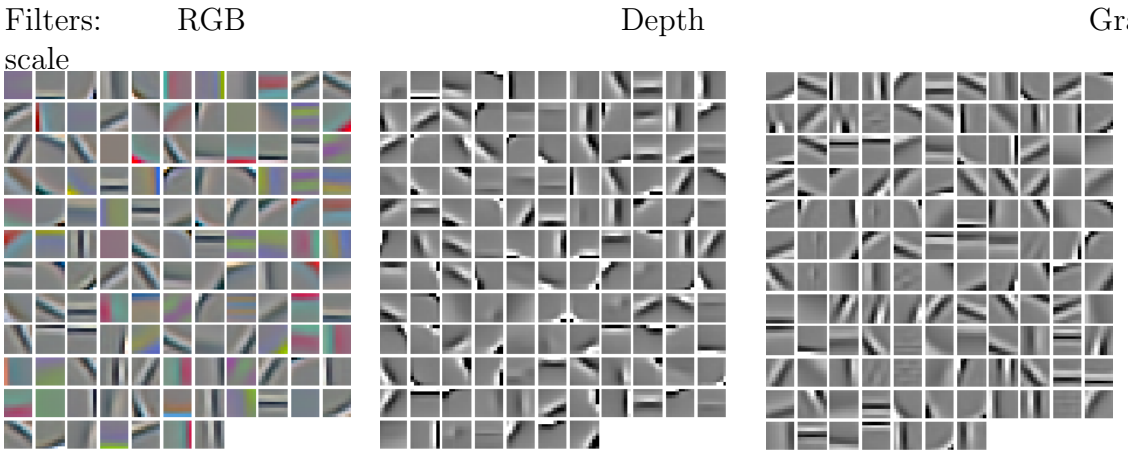


Figure 5.8: Visualization of the k-means filters used in the CNN layer after unsupervised pre-training: (**left**) Standard RGB filters (best viewed in color) capture edges and colors. When the method is applied to depth images (**center**) the resulting filters have sharper edges which arise due to the strong discontinuities at object boundaries. The same is true, though to a lesser extent, when compared to filters trained on gray scale versions of the color images (**right**).

Unsupervised Pre-training of CNN Filters

I follow the procedure described by Coates and Ng (2011). First, random patches are extracted into two sets, one for each modality (RGB and depth). Each set of patches is then normalized and whitened. The pre-processed patches are clustered by simply running k-means. Fig. 5.8 shows the resulting filters for both modalities. They capture standard edge and color features. One interesting result when applying this method to the depth channel is that the edges are much sharper. This is due to the large discontinuities between object boundaries and the background. While the depth channel is often quite noisy most of the features are still smooth.

A Single CNN Layer

To generate features for the RNN layer, a CNN architecture is chosen for its translational invariance properties. The main idea of CNNs is to convolve filters over the input image in order to extract features. My single layer CNN is similar to the one

proposed by Jarrett et al. (2009) and consists of a convolution, followed by rectification and local contrast normalization (LCN). LCN was inspired by computational neuroscience and is used to contrast features within a feature map, as well as across feature maps at the same spatial location (Jarrett et al., 2009; Pinto et al., 2008; Le et al., 2012).

I convolve each image of size (height and width) d_I with K square filters of size d_P, resulting in K filter responses, each of dimensionality $d_I - d_P + 1$. I then average pool them with square regions of size $d_l = 10$ and a stride size of $s = 5$, to obtain a pooled response with width and height equal to $r = (d_I - d_l)/s + 1 = 27$. So the output X of the CNN layer applied to one image is a $K \times r \times r$ dimensional 3D matrix. I apply this same procedure to both color and depth images separately.

Fixed-Tree Recursive Neural Networks

The idea of RNNs in this thesis is to learn hierarchical feature representations by applying the same neural network recursively in a tree structure. In the case of this section, the leaf nodes of the tree are K-dimensional vectors (the result of the CNN pooling over an image patch repeated for all K filters) and there are r^2 many of them.

In all previous sections, the RNN tree structures depended on the input. While this allows for more flexibility, I found that for the task of object classification and in conjunction with a CNN layer it was not necessary for obtaining high performance. Furthermore, the search over optimal trees slows down the method considerably as one can not easily parallelize the search or make use of parallelization of large matrix products. The latter could benefit immensely from new multicore hardware such as GPUs. In this section, I focus on fixed-trees which I design to be balanced. Previous work also only combined pairs of vectors. I generalize my RNN architecture to allow each layer to merge blocks of adjacent vectors instead of only pairs.

I start with a 3D matrix $X \in \mathbb{R}^{K \times r \times r}$ for each image (the columns are K-dimensional). I define a block to be a list of adjacent column vectors which are merged into a parent vector $p \in \mathbb{R}^K$. In the following I use only square blocks for convenience. Blocks are of size $b \times b \times K$. For instance, if I merge vectors in a block with $b = 3$, I get a total size $3 \times 3 \times 128$ and a resulting list of vectors $(x_1, \vdots, x_9)$. In

general, I have b^2 many vectors in each block. The neural network for computing the parent vector is

$$p = f\left(W \begin{bmatrix} x_1 \\ \dots \\ x_{b^2} \end{bmatrix}\right), \tag{5.7}$$

where the parameter matrix $W \in \mathbb{R}^{K \times b^2 K}$, f is a nonlinearity such as tanh. I omit the bias term which turns out to have no effect in the experiments below. Eq. 5.7 will be applied to all blocks of vectors in X until I have a new set of vectors which are now one layer above the leaf nodes. Generally, there will be $(r/b)^2$ many parent vectors p, forming a new matrix P_1. The vectors in P_1 will again be merged in form blocks just as those in matrix X using Eq. 5.7 with the same tied weights resulting in matrix P_2. This procedure continues until only one parent vector remains.

Fig. 5.9 shows an example of a pooled CNN output of size $4 \times 4 \times K$ and a RNN tree structure with blocks of 4 children.

The model so far has been unsupervised. However, my original task is to classify each block into one of many object categories. Therefore, I use the top vector P_{top} as the feature vector to a softmax classifier. In order to minimize the cross entropy error of the softmax, I could backpropagate through the recursive neural network (see Sec. 3.1.4) and convolutional layers (LeCun and Bengio, 1995). In practice, this is very slow and I will discuss alternatives in the next section.

Multiple Random RNNs

Previous work used only a single RNN. I can actually use the 3D matrix X as input to a number of RNNs. Each of N RNNs will output a K-dimensional vector. After I forward propagate through all the RNNs, I concatenate their outputs to a NK-dimensional vector which is then given to the softmax classifier.

Instead of taking derivatives of the W matrices of the RNNs which would require backprop through structure as described in Sec. 3.1.4 and used in all previous sections, I found that even RNNs with random weights produce high quality feature vectors. Similar results have been found for random weights in the closely related CNNs (Saxe

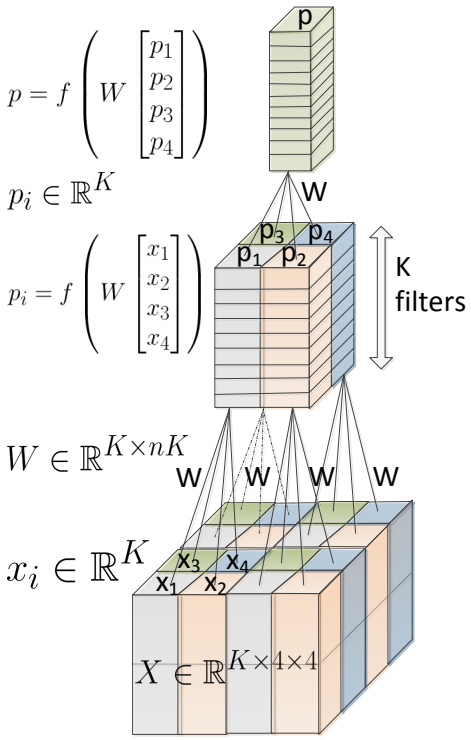


Figure 5.9: Recursive Neural Network applied to blocks: At each node, the same neural network is used to compute the parent vector of a set of child vectors. The original input matrix is the output of a pooled convolution.

et al., 2011).

5.2.2 Experiments

All experiments of this section are carried out on the recent RGB-D dataset of Lai et al. (2011). There are 51 different classes of household objects and 300 instances of these classes. Each object instance is imaged from 3 different angles resulting in roughly 600 images per instance. The dataset consists of a total of 207,920 RGB-D images. I subsample every 5th frame of the 600 images resulting in a total of 120 images per instance.

In this experiment, I focus on the problem of category recognition and I use the same setup as Lai et al. (2011) and the 10 random splits they provide. All development

is carried out on a separate split and model ablations are run on one of the 10 splits. For each split's test set I sample one object from each class resulting in 51 test objects, each with about 120 independently classified images. This leaves about 34,000 images for training my model. Before the images are given to the CNN they are resized to be $d_I = 148$.

Unsupervised pre-training for CNN filters is performed for all experiments by using k-means on 500,000 image patches randomly sampled from each split's training set. Before unsupervised pre-training, the $9 \times 9 \times 3$ patches for RGB and 9×9 patches for depth are individually normalized by subtracting the mean and divided by the standard deviation of its elements. In addition, ZCA whitening is performed to de-correlate pixels and get rid of redundant features in raw images (Hyvärinen and Oja, 2000). A valid convolution is performed with filter bank size $K = 128$ and filter width and height of 9. Average pooling is then performed with pooling regions of size 10 and stride size 5 to produce a 3D matrix of size $27 \times 27 \times 128$ for each image.

Each RNN has non-overlapping child sizes of 3×3 applied spatially. This leads to the following matrices at each depth of the tree: $X \in \mathbb{R}^{128\times27\times27}$ to $P_1 \in \mathbb{R}^{128\times9\times9}$ to $P_1 \in \mathbb{R}^{128\times3\times3}$ to finally $P_3 \in \mathbb{R}^{128}$.

This is repeated for a depth of 3 which produces a 128 dimensional feature vector at the top of each RNN. I use 128 randomly initialized RNNs in both modalities. The combination of RGB and depth is done by concatenating the final features which have $2 \times 128^2 = 32,768$ dimensions.

Comparison to Other Methods

In this section I compare my model to related models in the literature. Table 5.4 lists the main accuracy numbers and compares to the published results in Lai et al. (2011). Recent work by Bo et al. (2011) investigates multiple kernel descriptors on top of various features, including 3D shape, physical size of the object, depth edges, gradients, kernel PCA, local binary patterns,etc. In contrast, all my features are learned in an unsupervised way from the raw color and depth images. I outperform all but one method (Bo et al., 2012) which makes additional use of surface normals and gray scale as additional inputs on top of RGB and depth.

Classifier	Extra Features for 3D;RGB	3D	RGB	Both
Linear SVM (Lai et al., 2011)	Spin Images, efficient match kernel (EMK), random Fourier sets, width, depth, height; SIFT, EMK, texton histogram, color histogram	53.1±1.7	74.3±3.3	81.9±2.8
Kernel SVM (Lai et al., 2011)	same as above	64.7±2.2	74.5±3.1	83.9±3.5
Random Forest (Lai et al., 2011)	same as above	66.8±2.5	74.7±3.6	79.6±4.0
SVM (Bo et al., 2011)	3D shape, physical size of the object, depth edges, gradients, kernel PCA, local binary patterns,multiple depth kernels	78.8±2.7	77.7±1.9	86.2±2.1
CKM (Blum et al., 2012)	SURF interest points	–	–	86.4±2.3
SP+HMP (Bo et al., 2012)	surface normals	81.2±2.3	82.4±3.1	87.5±2.9
CNN-RNN	–	78.9±3.8	80.8±4.2	86.8±3.3

Table 5.4: Comparison to multiple related approaches. While other approaches use significantly more information (including the actual physical size of the object), my fully learned features and model perform competitively.

Model Analysis

I analyze my model through several ablations. I picked one of the splits as my development fold and I focus on depth data only. Most results carry over to the RGB case.

Two Layer CNN vs. CNN-RNN. Fig. 5.10 (left) shows a comparison of two different pre-training options for the first layer CNN filter. The two layer CNN does not benefit from filters initialized with k-means. The table also shows that the combination of a single layer CNN with multiple random RNNs outperforms a 2 layer CNN architecture.

Number of random RNNs: Fig. 5.10 (center) shows that increasing the number of random RNNs improves performance.

RGB & depth combinations and features: Fig. 5.10 (right) shows that combining RGB and depth features from RNNs improves performance. The two modalities complement each other and produce features that are independent enough so

that the classifier can benefit from their combination.

Global autoencoder on voxels (leaf nodes only, no RAE). In this experiment I investigate whether the recursive structure learns better features than simply using a single layer of features on raw pixels. Many methods such as those of Coates and Ng (2011) show remarkable results with a single very wide layer. The global autoencoder achieves only 61.1%, (it is overfitting at 93.3% training accuracy and I cross validated over the number of hidden units and sparsity parameters). This shows that even random recursive neural nets can clearly capture more of the underlying class structure in its learned vector representations than a single layer autoencoder.

Filters	2nd Layer	Acc.
Jarrett et al. (2009)	CNN	77.66
k-means (Coates and Ng, 2011)	CNN	78.65
k-means (Coates and Ng, 2011)	RNN	80.15

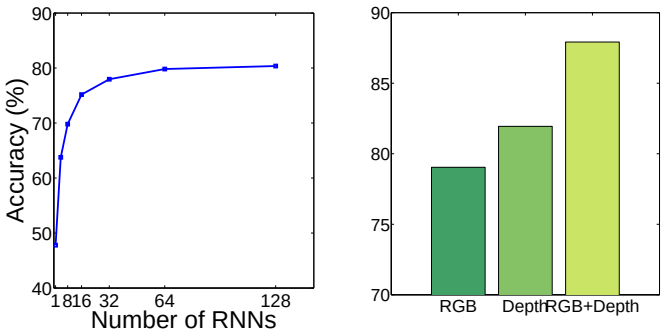


Figure 5.10: Model analysis on the development split (left and center use rgb only). **Left:** Comparison of variants for filters of the first CNN layer and of choices of the second layer. The best performance is achieved with filters trained by k-means and when RNNs are used on top of the first CNN layer. **Center:** Increasing the number of random RNNs improves performance. **Right:** Combining both modalities improves performance to 88% on the development split.

Error Analysis

Fig. 5.11 shows my confusion matrix across all 51 classes. Most model confusions are very reasonable showing that recursive deep learning methods on raw pixels and point clouds can give interpretable results.

Fig. 5.12 shows 4 pairs of often confused classes. Both garlic and mushrooms have very similar appearances and colors. Water bottles and shampoo bottles in particular are problematic because the IR sensors do not properly reflect from see through surfaces.

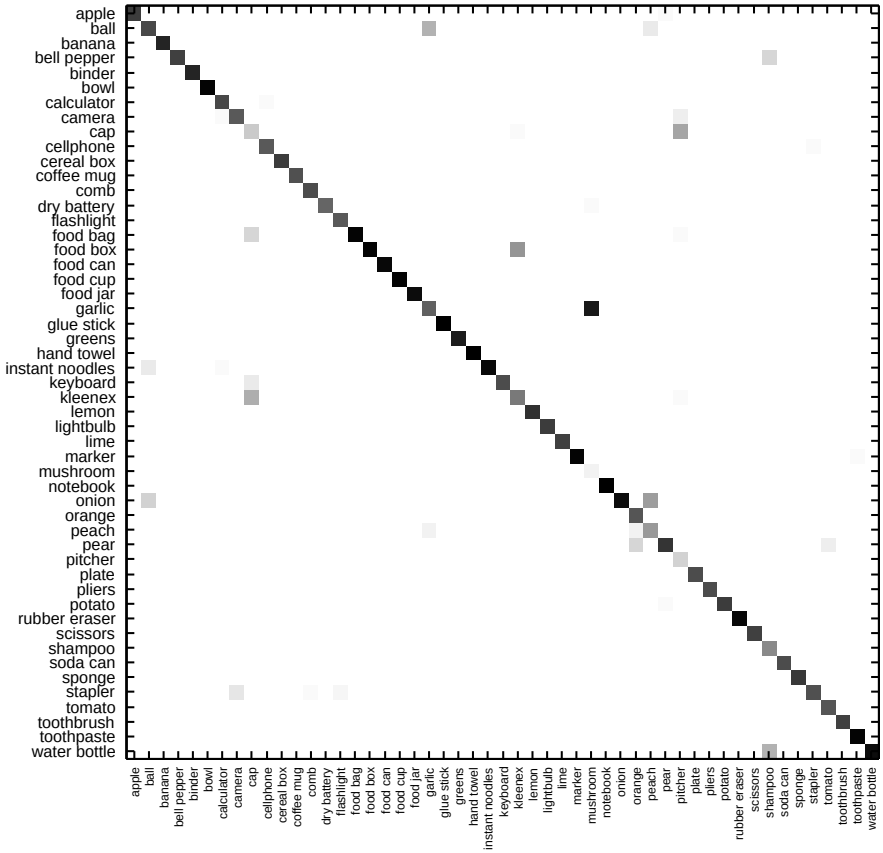


Figure 5.11: Confusion Matrix of my full RAE model on images and 3D data. The ground truth labels are on the y-axis and the predicted labels on the x-axis. The largest mistakes are between (a) garlic and mushroom (b) marker and toothpaste and (c) water bottle and shampoo.

5.2.3 Related Work

There has been great interest in object recognition and scene understanding using RGB-D data. Silberman and Fergus (2011) have published a 3D dataset for full scene understanding. Koppula et al. (2011) also recently provided a new dataset for indoor scene segmentation. The dataset I am using was introduced by Lai et al. (2011).

Figure 5.12: Examples of confused classes: Shampoo bottle and water bottle, mushrooms labeled as garlic, pitchers classified as caps due to shape and color similarity, white caps classified as kleenex boxes at certain angles.

The most common approach today for standard object recognition is to use well-designed features based on orientation histograms such as SIFT, SURF (Bay et al., 2008) or textons and give them as input to a classifier such as a random forest. Despite their success, they have several shortcomings such as being only applicable to one modality (grey scale images in the case of SIFT), not adapting easily to new modalities such as RGB-D or to varying image domains. There have been some attempts to modify these features to colored images via color histograms (Abdel-Hakim and Farag, 2006) or simply extending SIFT to the depth channel (Lai et al., 2011). More advanced methods that generalize these ideas and can combine several important RGB-D image characteristics such as size, 3D shape and depth edges are kernel descriptors (Bo et al., 2011).

Another solution to the above mentioned problems is to employ unsupervised feature learning methods (Hinton and Salakhutdinov, 2006; Bengio, 2009; Ranzato et al., 2007) (among many others) which have made large improvements in object recognition. While many deep learning methods exist for learning features from images, no neural network architectures have yet been investigated for 3D point cloud data. Very recently, Blum et al. (2012) introduced convolutional k-means descriptors

(CKM) for RGB-D data. They use SURF interest points and learn features using k-means similar to Coates and Ng (2011). Their work is similar to ours in that they also learn features in an unsupervised way. One of the main advantages of my algorithm is speed. Unfortunately, they do not provide running times for their algorithm but due to the necessity to find interest points and running convolution with hundreds of filters, it is likely slower than the RAE proposed here.

Another related line of work is about spatial pyramids in object classification, in particular the pyramid matching kernel (Grauman and Darrell, 2005). The similarity is mostly in that my model also learns a hierarchical image representation that can be used to classify objects.

Recent work by Bo et al. (2012) uses sparse coding to learn dictionaries from 8 different channels including grayscale intensity, RGB, depth scalars, and surface normals. Features are then used in a hierarchal matching pursuit algorithm which consists of two layers, each with three modules: batch orthogonal matching pursuit, pyramid max pooling, and contrast normalization. This results in a feature vector size of 188,300 dimensions which is used for classification.

Lastly, I compared to the related RNN models of this thesis.

5.3 Conclusion

This concludes the chapter on tree structure variants for RNNs. Technically, RNNs can be used on any acyclic directed graph but I have focused here on the more common subtype of trees. Dependency trees are particularly well suited for many downstream semantic tasks and can be computed very efficiently. The DT-RNNs can also deal with multiple children instead of only working on binary trees. Having the same tree structure for multiple RNNs has worked well for 3d object classification. Recent success with convolutional neural networks (Kalchbrenner et al., 2014) for language problems suggest that this route is also potentially promising for sentence processing.

Chapter 6

Conclusions

In this dissertation I introduced a new family of recursive deep learning algorithms for accurate, large scale natural language processing. Chapter 3 introduced various objective functions. The choice of objective function is the first of three explored axes of variation for this model family. As in most machine learning models, an objective, or cost function is optimized in order to improve a certain prediction on the training data. The proposed objective functions solved general tasks such as structure prediction, structured classification or structure compression (with unfolding recursive auto encoders). The specific tasks that these models obtained state of the art performance on included scene image parsing, sentiment analysis and paraphrase detection. After additional analysis, I believe that the Euclidean-distance-based reconstruction objectives act largely as a regularizer and other supervised objective functions that guide the learning process are very important. However, this objective function can be very usefully employed when vectors that are not the same as the inputs are reconstructed. This is the case for machine translation (Li et al., 2013).

Chapter 4 turned to the second major design choice for RNNs: the composition function. There are different motivations for the various choices. The matrix vector RNN (MV-RNN) has the most parameters and is largely linguistically motivated, giving each word the power to modify the meaning of its neighboring words. It requires a large dataset and learning is slower because each parent computation requires 3 matrix-vector products. Since there is no good way to initialize word matrices in

an unsupervised way this model will require more future work. Another variant of the idea to use different compositions depending on the words being combined is the syntactically untied RNN (SU-RNN). In that model the composition function depends on the syntactic category of the children being combined. This can be seen as a group prior over the composition function, essentially tying the composition functions of multiple words in the MV-RNN model. Both of these are linguistically motivated models. The last model, the recursive neural tensor network (RNTN) on the other hand is based on the assumption that there can be a single, albeit very powerful, composition function. The idea of the tensor is to allow multiple, mediated multiplicative interactions between word vectors. This has the huge advantage of being able to use unsupervised learned word vectors. Each slice has the ability to pick up on different aspects of the word vector composition. I believe that this model is the most promising model put forward in my thesis.

Finally, chapter 5 explored different types of tree structures. All but one of the models in this thesis are based on constituency trees. However, the dependency tree model has the main advantage of capturing more semantic structure and being less dependent on the syntactic form of a sentence. The dependency tree model (DT-RNN) is also not restricted to only binary trees nor is it forcing the hidden nonterminal layers to be of the same size as the word vectors. These are small but useful improvements over previous models. While the RNN for scene image parsing and all the language RNNs have input-specific (parse) tree structures, I show in Sec. 5.2 that using the same tree structure for every input can also work well in RNNs. Similar to having multiple filters in a CNN, one can also have multiple RNN "filters." Similar ideas have shown recent improvements in convolutional architectures (Kalchbrenner et al., 2014). However, the main drawback is that a detailed understanding of how each word influences the meaning of the phrase is lost and more semantic analyses such as those historically performed by lambda calculus based approaches cannot intuitively be solved in CNN or fixed tree frameworks. I will discuss this in the outlook below. In summary, I believe that for many natural language tasks, dependency trees provide the most suitable structures for semantic understanding with RNN models. I believe that the most promising future model is to combine the ideas of the RNTN

the use of dependency tree structures.

Table 6.1 summarizes the composition functions and training objective functions for all RNN models of this thesis. Table 6.2 gives an overview of RNN model properties such as whether they are used for parsing (finding the tree structure); whether they use trees that are input dependent; have only been explored with binary trees; are supervised, semi-supervised or unsupervised; and how many parameters they have. The variables that determine the total number of parameters are: n - the dimensionality of word and phrase vectors (usually around 50), V - the size of the vocabulary (usually around 100,000), C - the number of classes (2-50), S - the number of syntactic category pairs (882 in the case of the SU-RNN), r - the rank of all the word matrices (only applicable for the MV-RNN, set to around 3 in the experiments), D - the set of all syntactic dependency relationships (around 42), R - the number of random RNNs (only applicable for the CRNN which used multiple random RNNs to extract features). Lastly, Table 6.3 lists the tasks each model has been applied to as well as advantages and disadvantages of all the models.

One major disadvantage that all the RNN models for NLP share is that they require parsing which can be slow and is not always accurate. I have not extensively analyzed how much parsing errors affect the overall performance. However, the models are surprisingly robust and accurate even when using trees from older parsers with a labeled F1 several per cent below 90%. Another disadvantage caused by input-specific tree structures is that the model is not easily parallelizable on current graphics cards. A potential avenue to remedy this is to use the same tree structure for all sentences of the same length as described in the previous paragraph.

One can also argue that a single vector may be insufficient to store all the information of multiple word vectors. However, if the nonterminal vectors can be of a higher dimensionality than the word vectors, this can easily be fixed. A more fundamental question is whether meaning should be represented in terms of vectors in $\mathbb{R}^n$ at all. There are many apparent advantages for vector space semantics: great empirical performance across multiple different tasks, the ability to capture similarity judgments that correlate well with those of humans, sharing of statistical strength because similar words can lead to similar predictions (which is an effect that cannot

Model	Composition $p(a,b) = f(\cdot)$	Objective
RNN Sec. 3.1	$W \begin{bmatrix} a \\ b \end{bmatrix}$	max-margin structure prediction with linear scores
RAE Sec. 3.2	$W \begin{bmatrix} a \\ b \end{bmatrix}$	tree construction with reconstruction of child nodes + cross-entropy error for classification
URAE Sec. 3.3	$W \begin{bmatrix} a \\ b \end{bmatrix}$	reconstruction of all leafs below node p
SU-RNN Sec. 4.1	$W^{syn(a,b)} \begin{bmatrix} a \\ b \end{bmatrix}$	max-margin structure prediction with linear scores
MV-RNN Sec. 4.2	$W \begin{bmatrix} Ba \\ Ab \end{bmatrix}$	cross-entropy error for classification
RNTN Sec. 4.3	$W \begin{bmatrix} a \\ b \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} a \\ b \end{bmatrix}$	cross-entropy error for classification
DT-RNN Sec. 5.1	$\frac{1}{\ell(i)} \left(W_v x_i + \sum_{j \in C(i)} \ell(j) W_{\text{pos}(i,j)} h_j \right)$	inner product with image vector
CRNN Sec. 5.2	$W \begin{bmatrix} x_1 \\ \dots \\ x_{b^2} \end{bmatrix}$	cross-entropy error for classification

Table 6.1: Comparison of all RNNs in this thesis: composition and objective function. Note that both pieces are independent and all composition functions could be used for all objective functions. Several of these combinations have been explored but there is room for a more complete comparison across different tasks.

be achieved with simple discrete word counts).

The models have not yet been shown to be able to capture complex reasoning patterns that would require first-order logic over a set of entities. Despite the lack of such a general result, first steps in the direction of capturing logical semantics have been taken by Bowman et al. (2014). This is one of the most promising future directions that I will outline below.

Model	Structure prediction	Input dependent tree	Binary trees	Super-vised	# parameters
RNN Sec. 3.1	Yes	Yes	Yes	Yes	$n + 2n^2 + nV$
RAE Sec. 3.2	Yes	Yes	Yes	Semi	$nC+4n^2+nV$
URAE Sec. 3.3	No	Yes	Yes	No	$4n^2$
SU-RNN Sec. 4.1	Yes	Yes	Yes	Yes	$Sn + 2Sn^2$
MV-RNN Sec. 4.2	No	Yes	Yes	Yes	$2n^2 + rnV + nC$
RNTN Sec. 4.3	No	Yes	Yes	Yes	$4n^3+nC+nV$
DT-RNN Sec. 5.1	No	Yes	No	Yes	$(D + 1)n^2$
CRNN Sec. 5.2	No	No	No	Yes	RnC

Table 6.2: Comparison of all RNNs in this thesis: Structure prediction (whether this model was used for parsing), input dependent tree (whether the tree structure is input dependent or the same for every input), binary trees (whether I only experimented with binary versions of this model), supervised (whether the model was trained with labeled training data or unsupervised), number of trainable parameters (see text for details).

Apart from the unfolding recursive autoencoder, which can encode phrases of length up to 7 with high accuracy, there is still no perfect unsupervised objective function for RNNs that could learn compositional and noncompositional meaning of longer phrases. This is another contentious direction for potential future research since the meaning of a sentence can arguably be very broad. A sentence can have a multitude of potential meanings anywhere from a question (in which case its meaning is closely related to an answer) to a simple statement expressing a sentiment about a movie or a call to action. It is still unclear whether a single vector can express these various relationships between language and its varied social uses and functions.

One long-term goal is to develop a general model of natural language which jointly captures both the continuous, fuzzy and discrete, logical nature of language and which can connect to a set of facts. Capturing the fuzzy nature of language is required for

understanding vague sentiment descriptions like "The movie started out funny but I wasn't really happy with most of the rest." The latter is necessary for semantic parsing to retrieve information from knowledge bases. Solving such a task requires understanding precise, logical language and connecting it to specific entities as in the following question: "Which companies did Google acquire in 2013?"

In this thesis, I showed that recursive deep learning models can solve multiple language tasks involving word and sentence-level predictions of both continuous and discrete nature. However, in order to provide a general model for the complete fuzzy to logical language spectrum, there are two crucial pieces missing which cannot yet be captured by my deep learning models. The first is inter-sentence information flow for understanding discourse structure or solving concrete tasks such as coreference and anaphora resolution. Coreference resolution attempts to find all mentions that refer to the same real world entity while anaphora resolution tries to find the correct antecedents of pronouns in previous sentences. One of the many possible solutions towards this goal is to apply recursive or recurrent techniques to compute paragraph or document level representations. The second challenge for deep models is first order logical reasoning, which may be required for retrieving the right information from knowledge bases using natural language questions. Question answering is a real task and a great way to verify models of grounding semantics in world knowledge. I hope that the models in this thesis can be extended to eventually jointly model language, images and knowledge bases in one coherent semantic framework.

Model	Tasks & Inputs	Pros/Cons
RNN Sec. 3.1	image and sentence parsing from unlabeled parse trees or segmented images	**pro**: simple, fast with greedy search; **con**: not powerful enough for longer sentence parsing
RAE Sec. 3.2	sentiment analysis from sentences with global label	**pro**: no parser required, simple; **con**: nonstandard trees, not very powerful
URAE Sec. 3.3	paraphrase detection from unlabeled sentences and paraphrase pairs	**pro**: unsupervised, better reconstruction ability; **con**: unclear if reconstruction captures the right linguistic phenomena
SU-RNN Sec. 4.1	sentence parsing from labeled parses	**pro**: very powerful, minimal linguistic knowledge + RNN gives good parser; **con**: requires linguistic knowldege, constituency parsing very slow compared to dependency parsing
MV-RNN Sec. 4.2	adverb-adjective prediction, sentiment analysis, simple logic, relationship classification from adv-adj distributions, the logical outputs, labeled words and sentences with global label, respectively	**pro**: very powerful model, great performance on variety of tasks; **con**: too many parameters for most datasets
RNTN Sec. 4.3	sentiment analysis from sentences with all subphrases labeled	**pro**: best performance on sentiment, very powerful model, allows immediate multiplicative interactions without many hidden layers; **con**: not much, still requires parser
DT-RNN Sec. 5.1	sentence-image search from matched image sentence pairs	**pro**: less dependent on surface form, more focus on semantics, fast; **con**: not much, requires Stanford typed dependencies for best performance
CRNN Sec. 5.2	3d object classification from labeled RGBD images	**pro**: very fast to train, simple, no backprop through structure due to multiple random RNNs; **con**: not much, more exploration needed

Table 6.3: Comparison of all RNN models. The first column describes the tasks that each model has been applied to in this thesis (several of these models have since been applied to numerous other tasks by other researchers), the second columns lists advantages and disadvantages.

Bibliography

A. E. Abdel-Hakim and A. A. Farag. 2006. CSIFT: A SIFT descriptor with color invariant characteristics. In *CVPR*.

G. Andrew, R. Arora, K. Livescu, and J. Bilmes. 2013. Deep canonical correlation analysis. In *ICML*. Atlanta, Georgia.

K. Barnard, P. Duygulu, N. de Freitas, D. Forsyth, D. Blei, and M. Jordan. 2003. Matching words and pictures. *JMLR*.

M. Baroni and A. Lenci. 2010. Distributional memory: A general framework for corpus-based semantics. *Computational Linguistics*, 36(4):673–721.

M. Baroni and Roberto Zamparelli. 2010. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *EMNLP*.

R. Barzilay and L. Lee. 2003. Learning to paraphrase: an unsupervised approach using multiple-sequence alignment. In *NAACL*.

H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. 2008. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3).

P. Beineke, T. Hastie, C. D. Manning, and S. Vaithyanathan. 2004. Exploring sentiment summarization. In *AAAI Spring Symposium on Exploring Attitude and Affect in Text: Theories and Applications*.

Y. Bengio. 2009. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*.

Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. 2003. A neural probabilistic language model. *JMLR*, 3:1137–1155.

Y. Bengio, J. Louradour, Collobert R, and J. Weston. 2009. Curriculum learning. In *ICML*.

D. Blakemore. 1989. Denial and contrast: A relevance theoretic analysis of 'but'. *Linguistics and Philosophy*, 12:15–37.

D.M. Blei, A.Y. Ng, and M.I. Jordan. 2003. Latent Dirichlet allocation. *JMLR*, 3:993–1022.

M. Blum, J. T. Springenberg, J. Wuelfing, and M. Riedmiller. 2012. A Learned Feature Descriptor for Object Recognition in RGB-D Data. In *IEEE International Conference on Robotics and Automation (ICRA)*.

L. Bo, X. Ren, and D. Fox. 2011. Depth kernel descriptors for object recognition. In *IROS*.

L. Bo, X. Ren, and D. Fox. 2012. Unsupervised Feature Learning for RGB-D Based Object Recognition. In *ISER*.

L. Bottou. 2011. From machine learning to machine reasoning. *CoRR*, abs/1102.1808.

S. R. Bowman, C. Potts, and C. D. Manning. 2014. Recursive neural networks for learning logical semantics. *CoRR*, abs/1406.1827.

P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. Della Pietra, and J. C. Lai. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18.

A. E. Bryson, W. F. Denham, and S. E. Dreyfus. 1963. Optimal programming problems with inequality constraints I: necessary conditions for extremal solutions. *AIAA Journal*, 1:2544–2550.

C. Callison-Burch. 2008. Syntactic constraints on paraphrases extracted from parallel corpora. In *EMNLP*, pages 196–205.

E. Charniak. 2000. A maximum-entropy-inspired parser. In *ACL*, pages 132–139.

E. Charniak and M. Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *ACL*.

Y. Choi and C. Cardie. 2008. Learning with compositional semantics as structural inference for subsentential sentiment analysis. In *EMNLP*.

M. Ciaramita and Y. Altun. 2006. Broad-coverage sense disambiguation and information extraction with a supersense sequence tagger. In *EMNLP*.

D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. 2011. Flexible, high performance convolutional neural networks for image classification. In *IJCAI*.

S. Clark and S. Pulman. 2007. Combining symbolic and distributional models of meaning. In *AAAI Spring Symposium on Quantum Interaction*, pages 52–55.

P. Clough, R. Gaizauskas, S. S. L. Piao, and Y. Wilks. 2002. METER: MEasuring TExt Reuse. In *ACL*.

A. Coates and A. Ng. 2011. The Importance of Encoding Versus Training with Sparse Coding and Vector Quantization. In *ICML*.

M. Collins. 1997. Three generative, lexicalised models for statistical parsing. In *ACL*.

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

R. Collobert and J. Weston. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML*, pages 160–167.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *JMLR*, 12:2493–2537.

D. Comaniciu and P. Meer. 2002. Mean shift: a robust approach toward feature space analysis. *IEEE PAMI*, 24(5):603–619.

F. Costa, P. Frasconi, V. Lombardo, and G. Soda. 2003. Towards incremental parsing of natural language using recursive neural networks. *Applied Intelligence*.

J. Curran. 2004. *From Distributional to Semantic Similarity*. Ph.D. thesis, University of Edinburgh.

G. E. Dahl, M. A. Ranzato, A. Mohamed, and G. E. Hinton. 2010. Phone recognition with the mean-covariance restricted Boltzmann machine. In *NIPS*.

D. Das and N. A. Smith. 2009. Paraphrase identification as probabilistic quasi-synchronous recognition. In *ACL-IJCNLP*.

S. Das and M. Chen. 2001. Yahoo! for Amazon: Extracting market sentiment from stock message boards. In *Asia Pacific Finance Association Annual Conference (APFA)*.

K. Dave, S. Lawrence, and D. M. Pennock. 2003. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *WWW*, pages 519–528.

M. de Marneffe, B. MacCartney, and C. D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *LREC*.

S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*.

X. Ding, B. Liu, and P. S. Yu. 2008. A holistic lexicon-based approach to opinion mining. In *Conference on Web Search and Web Data Mining (WSDM)*.

B. Dolan, C. Quirk, and C. Brockett. 2004. Unsupervised construction of large paraphrase corpora: exploiting massively parallel news sources. In *COLING*.

L. Dong, F. Wei, M. Zhou, and K. Xu. 2014. Adaptive multi-compositionality for recursive neural models with applications to sentiment analysis. In *AAAI*.

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

P. Duygulu, K. Barnard, N. de Freitas, and D. Forsyth. 2002. Object recognition as machine translation. In *ECCV*.

J. L. Elman. 1991. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7(2-3):195–225.

D. Erhan, A. Courville, Y. Bengio, and P. Vincent. 2010. Why does unsupervised pre-training help deep learning? *JMLR*, 11.

K. Erk and S. Padó. 2008. A structured vector space model for word meaning in context. In *EMNLP*.

A. Esuli and F. Sebastiani. 2007. Pageranking wordnet synsets: An application to opinion mining. In *ACL*.

C. Farabet, C. Couprie, L. Najman, and Y. LeCun. 2012. Scene parsing with multiscale feature learning, purity trees, and optimal covers. In *ICML*.

C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello. 2010. Hardware accelerated convolutional neural networks for synthetic vision systems. In *Proc. International Symposium on Circuits and Systems (ISCAS'10)*.

A. Farhadi, M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth. 2010. Every picture tells a story: Generating sentences from images. In *ECCV*.

Y. Feng and M. Lapata. 2013. Automatic caption generation for news images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35.

S. Fernando and M. Stevenson. 2008. A semantic similarity approach to paraphrase detection. *11th Annual Research Colloquium of the UK Special Interest Group for Computational Linguistics*.

J. R. Finkel, A. Kleeman, and C. D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *ACL*, pages 959–967.

J.R. Firth. 1957. A synopsis of linguistic theory 1930-1955. *Studies in linguistic analysis*, pages 1–32.

G. Frege. 1892. Über Sinn und Bedeutung. In *Zeitschrift für Philosophie und philosophische Kritik, 100*.

A. Frome, G. Corrado, J. Shlens, S. Bengio, J. Dean, M. Ranzato, and T. Mikolov. 2013. Devise: A deep visual-semantic embedding model. In *NIPS*.

D. Garrette, K. Erk, and R. Mooney. 2011. Integrating Logical Representations with Probabilistic Information using Markov Logic. In *International Conference on Computational Semantics*.

D. Gildea and M. Palmer. 2002. The necessity of parsing for predicate argument recognition. In *ACL*, pages 239–246.

C. Goller and A. Küchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *International Conference on Neural Networks*.

J. Goodman. 1998. *Parsing Inside-Out*. Ph.D. thesis, MIT.

S. Gould, R. Fulton, and D. Koller. 2009. Decomposing a Scene into Geometric and Semantically Consistent Regions. In *ICCV*.

K. Grauman and T. Darrell. 2005. The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features. In *ICCV*.

E. Grefenstette, G. Dinu, Y.-Z. Zhang, M. Sadrzadeh, and M. Baroni. 2013. Multi-step regression learning for compositional distributional semantics. In *IWCS*.

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

G. Grefenstette, Y. Qu, J. G. Shanahan, and D. A. Evans. 2004. Coupling niche browsers and affect analysis for an opinion mining application. In *Recherche d'Information Assistée par Ordinateur (RIAO)*.

T. L. Griffiths, J. B. Tenenbaum, and M. Steyvers. 2007. Topics in semantic representation. *Psychological Review*, 114.

A. Gupta and L. S. Davis. 2008. Beyond nouns: Exploiting prepositions and comparative adjectives for learning visual classifiers. In *ECCV*.

D. Hall and D. Klein. 2012. Training factored pcfgs with expectation propagation. In *EMNLP*.

J. Henderson. 2003. Neural network probability estimation for broad coverage parsing. In *EACL*.

J. Henderson. 2004. Discriminative training of a neural network statistical parser. In *ACL*.

I. Hendrickx, S.N. Kim, Z. Kozareva, P. Nakov, D. Ó Séaghdha, S. Padó, M. Pennacchiotti, L. Romano, and S. Szpakowicz. 2010. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *5^{th} International Workshop on Semantic Evaluation*.

G. E. Hinton. 1990. Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence*, 46(1-2).

G. E. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.*, 29(6):82–97.

G. E. Hinton and R. R. Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

M. Hodosh, P. Young, and J. Hockenmaier. 2013. Framing image description as a ranking task: Data, models and evaluation metrics. *JAIR*, 47:853–899.

D. Hoiem, A.A. Efros, and M. Hebert. 2006. Putting Objects in Perspective. *CVPR*.

L. R. Horn. 1989. *A natural history of negation*, volume 960. University of Chicago Press Chicago.

E. H. Huang, R. Socher, C. D. Manning, and A. Y. Ng. 2012. Improving Word Representations via Global Context and Multiple Word Prototypes. In *ACL*.

L. Huang and D. Chiang. 2005. Better k-best parsing. In *9th International Workshop on Parsing Technologies (IWPT 2005)*.

A. Hyvärinen and E. Oja. 2000. Independent component analysis: algorithms and applications. *Neural Networks*, 13.

D. Ikeda, H. Takamura, L. Ratinov, and M. Okumura. 2008. Learning to shift the polarity of words for sentiment classification. In *IJCNLP*.

A. Islam and D. Inkpen. 2007. Semantic Similarity of Short Texts. In *International Conference on Recent Advances inNatural Language Processing (RANLP 2007)*.

M. Israel. 2001. Minimizers, maximizers, and the rhetoric of scalar reasoning. *Journal of Semantics*, 18(4):297–331.

K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. 2009. What is the best multi-stage architecture for object recognition? In *ICCV*.

R. Jenatton, N. Le Roux, A. Bordes, and G. Obozinski. 2012. A latent factor model for highly multi-relational data. In *NIPS*.

A. Johnson. 1997. *Spin-Images: A Representation for 3-D Surface Matching*. Ph.D. thesis, Robotics Institute, Carnegie Mellon University.

N. Kalchbrenner, E. Grefenstette, and P. Blunsom. 2014. A convolutional neural network for modelling sentences. In *ACL*.

A. Karpathy, A. Joulin, and L. Fei-Fei. 2014. Deep fragment embeddings for bidirectional image sentence mapping. Technical report, Stanford University.

D. Kartsaklis, M. Sadrzadeh, and S. Pulman. 2012. A unified sentence space for categorical distributional-compositional semantics: Theory and experiments. *Conference on Computational Linguistics (COLING).*

S. Kim and E. Hovy. 2007. Crystal: Analyzing predictive opinions on the web. In *EMNLP-CoNLL.*

S. Kiritchenko, X. Zhu, and S. M. Mohammad. 2014. Sentiment analysis of short informal texts. *JAIR.*

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

D. Klein and C.D. Manning. 2003b. Fast exact inference with a factored model for natural language parsing. In *NIPS.*

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

H.S. Koppula, A. Anand, T. Joachims, and A. Saxena. 2011. Semantic labeling of 3D point clouds for indoor scenes. In *NIPS.*

Z. Kozareva and A. Montoyo. 2006. Paraphrase Identification on the Basis of Supervised Machine Learning Techniques. In *Advances in Natural Language Processing, 5^{th} International Conference on NLP, FinTAL.*

A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS.*

G. Kulkarni, V. Premraj, S. Dhar, S. Li, Y. Choi, A. C. Berg, and T. L. Berg. 2011. Baby talk: Understanding and generating image descriptions. In *CVPR.*

N. Kumar, A. C. Berg, P. N. Belhumeur, , and S. K. Nayar. 2009. Attribute and simile classifiers for face verification. In *ICCV.*

J. K. Kummerfeld, D. Hall, J. R. Curran, and D. Klein. 2012. Parser showdown at the wall street corral: An empirical investigation of error types in parser output. In *EMNLP*.

P. Kuznetsova, V. Ordonez, A. C. Berg, T. L. Berg, and Yejin Choi. 2012. Collective generation of natural image descriptions. In *ACL*.

K. Lai, L. Bo, X. Ren, and D. Fox. 2011. A Large-Scale Hierarchical Multi-View RGB-D Object Dataset. In *IEEE International Conference on on Robotics and Automation*.

R. Lakoff. 1971. If's, and's, and but's about conjunction. In Charles J. Fillmore and D. Terence Langendoen, editors, *Studies in Linguistic Semantics*, pages 114–149. Holt, Rinehart, and Winston, New York.

Thomas K. Landauer and Susan T. Dumais. 1997. A solution to Plato's problem: the Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review*, 104(2):211–240.

H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. 2009. Exploring strategies for training deep neural networks. *JMLR*, 10.

Q.V. Le and T. Mikolov. 2014. Distributed representations of sentences and documents. In *ICML*.

Q.V. Le, M.A. Ranzato, R. Monga, M. Devin, K. Chen, G.S. Corrado, J. Dean, and A.Y. Ng. 2012. Building high-level features using large scale unsupervised learning. In *ICML*.

Y. LeCun and Y. Bengio. 1995. Convolutional networks for images, speech, and time-series. *The Handbook of Brain Theory and Neural Networks*.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *IEEE*, 86(11):2278–2324.

H. Lee, A. Battle, R. Raina, and Andrew Y. Ng. 2007. Efficient sparse coding algorithms. In *NIPS*.

H. Lee, R. Grosse, R. Ranganath, and A. Ng. 2009. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML*.

L-J. Li, R. Socher, and L. Fei-Fei. 2009. Towards total scene understanding:classification, annotation and segmentation in an automatic framework. In *CVPR*.

P. Li, Y. Liu, and M. Sun. 2013. Recursive autoencoders for ITG-based translation. In *EMNLP*.

D. Lin. 1998. Automatic retrieval and clustering of similar words. In *COLING-ACL*, pages 768–774.

M. Luong, R. Socher, and C. D. Manning. 2013. Better word representations with recursive neural networks for morphology. In *CoNLL*.

A. L. Maas, A. Y. Ng, and C. Potts. 2011. Multi-Dimensional Sentiment Analysis with Learned Representations. *Technical Report*.

C. D. Manning and H. Schütze. 1999. *Foundations of Statistical Natural Language Processing*. The MIT Press.

Y. Mao and G. Lebanon. 2007. Isotonic Conditional Random Fields and Local Sentiment Flow. In *NIPS*.

E. Marsi and E. Krahmer. 2005. Explorations in sentence fusion. In *European Workshop on Natural Language Generation*.

T. Matsuzaki, Y. Miyao, and J. Tsujii. 2005. Probabilistic cfg with latent annotations. In *ACL*.

D. McClosky, E. Charniak, and M. Johnson. 2006. Effective self-training for parsing. In *NAACL*.

S. Menchetti, F. Costa, P. Frasconi, and M. Pontil. 2005. Wide coverage natural language processing using kernel methods and neural networks for structured data. *Pattern Recognition Letters*, 26(12):1896–1906.

A. Merin. 1999. Information, relevance, and social decisionmaking: Some principles and results of decision-theoretic semantics. In Lawrence S. Moss, Jonathan Ginzburg, and Maarten de Rijke, editors, *Logic, Language, and Information*, volume 2. CSLI, Stanford, CA.

E. J. Metcalfe. 1990. A compositive holographic associative recall model. *Psychological Review*, 88:627–661.

R. Mihalcea, C. Corley, and C. Strapparava. 2006. Corpus-based and Knowledge-based Measures of Text Semantic Similarity. In *21st National Conference on Artificial Intelligence - Volume 1*.

T. Mikolov, W. Yih, and G. Zweig. 2013. Linguistic regularities in continuous space-word representations. In *HLT-NAACL*.

T. Mikolov and G. Zweig. 2012. Context dependent recurrent neural network language model. In *SLT*, pages 234–239. IEEE.

P. Mirowski, M. Ranzato, and Y. LeCun. 2010. Dynamic auto-encoders for semantic indexing. In *NIPS 2010 Workshop on Deep Learning*.

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

R. Montague. 1974. English as a formal language. *Linguaggi nella Societa e nella Tecnica*, pages 189–224.

T. Nakagawa, K. Inui, and S. Kurohashi. 2010. Dependency tree-based sentiment classification using CRFs with hidden variables. In *NAACL*, HLT.

J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A.Y. Ng. 2011. Multimodal deep learning. In *ICML*.

A. Oliva and A. Torralba. 2001a. Modeling the shape of the scene: a holistic representation of the spatial envelope. *IJCV*, 42.

A. Oliva and A. Torralba. 2001b. Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope. *IJCV*, 42.

V. Ordonez, G. Kulkarni, and T. L. Berg. 2011. Im2text: Describing images using 1 million captioned photographs. In *NIPS*.

S. Pado and M. Lapata. 2007. Dependency-based construction of semantic space models. *Computational Linguistics*, 33(2):161–199.

B. Pang and L. Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *ACL*.

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

B. Pang and L. Lee. 2008. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1-2):1–135.

B. Pang, L. Lee, and S. Vaithyanathan. 2002. Thumbs up? Sentiment classification using machine learning techniques. In *EMNLP*.

J. W. Pennebaker, R.J. Booth, and M. E. Francis. 2007. Linguistic inquiry and word count: Liwc2007 operator?s manual. University of Texas.

J. Pennington, R. Socher, and C. D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

S. Petrov, L. Barrett, R. Thibaux, and D. Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *ACL*, pages 433–440.

S. Petrov and D. Klein. 2007. Improved inference for unlexicalized parsing. In *NAACL*.

N. Pinto, D. D. Cox, and J. J. DiCarlo. 2008. Why is real-world visual object recognition hard? *PLoS Computational Biology*.

T. A. Plate. 1995. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623–641.

L. Polanyi and A. Zaenen. 2006. Contextual valence shifters. *Computing Attitude and Affect in Text: Theory and Applications*.

J. B. Pollack. 1990. Recursive distributed representations. *Artificial Intelligence*, 46.

C. Potts. 2010. On the negativity of negation. In David Lutz and Nan Li, editors, *Semantics and Linguistic Theory 20*. CLC Publications, Ithaca, NY.

L. Qiu, M. Kan, and T. Chua. 2006. Paraphrase recognition via dissimilarity significance classification. In *EMNLP*.

A. Rabinovich, A. Vedaldi, C. Galleguillos, E. Wiewiora, and S. Belongie. 2007. Objects in context. In *ICCV*.

M. Ranzato and A. Krizhevsky G. E. Hinton. 2010. Factored 3-Way Restricted Boltzmann Machines For Modeling Natural Images. *AISTATS*.

M. Ranzato, F. J. Huang, Y. Boureau, and Y. LeCun. 2007. Unsupervised learning of invariant feature hierarchies with applications to object recognition. *CVPR*, 0:1–8.

C. Rashtchian, P. Young, M. Hodosh, and J. Hockenmaier. 2010. Collecting image annotations using Amazon's Mechanical Turk. In *Workshop on Creating Speech and Language Data with Amazon's MTurk*.

N. Ratliff, J. A. Bagnell, and M. Zinkevich. 2007. (Online) subgradient methods for structured prediction. In *AIStats*.

B. Rink and S. Harabagiu. 2010. UTD: Classifying semantic relations by combining lexical and semantic resources. In *5th International Workshop on Semantic Evaluation*.

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

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Learning representations by back-propagating errors. *Nature*.

V. Rus, P. M. McCarthy, M. C. Lintean, D. S. McNamara, and A. C. Graesser. 2008. Paraphrase identification with lexico-syntactic graph subsumption. In *FLAIRS Conference*.

A. Saxe, P.W. Koh, Z. Chen, M. Bhand, B. Suresh, and A.Y. Ng. 2011. On random weights and unsupervised feature learning. In *ICML*.

C. Schmid. 2006. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*.

H. Schütze. 1998. Automatic word sense discrimination. *Computational Linguistics*, 24:97–124.

J. Shotton, J. Winn, C. Rother, and A. Criminisi. 2006. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *ECCV*.

N. Silberman and R. Fergus. 2011. Indoor scene segmentation using a structured light sensor. In *International Conference on Computer Vision - Workshop on 3D Representation and Recognition*.

N. A. Smith and J. Eisner. 2005. Contrastive estimation: Training log-linear models on unlabeled data. In *ACL*. Association for Computational Linguistics, Stroudsburg, PA, USA.

B. Snyder and R. Barzilay. 2007. Multiple aspect ranking using the Good Grief algorithm. In *HLT-NAACL*, pages 300–307.

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

R. Socher and L. Fei-Fei. 2010. Connecting modalities: Semi-supervised segmentation and annotation of images using unaligned text corpora. In *CVPR*.

R. Socher, M. Ganjoo, C. D. Manning, and A. Y. Ng. 2013b. Zero-Shot Learning Through Cross-Modal Transfer. In *NIPS*.

R. Socher, M. Ganjoo, H. Sridhar, O. Bastani, and A. Y. Ng. C. D. Manning and. 2013c. Zero-shot learning through cross-modal transfer. In *International Conference on Learning Representations (ICLR, Workshop Track)*.

R. Socher, E. H. Huang, J. Pennington, A. Y. Ng, and C. D. Manning. 2011a. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. In *NIPS*.

R. Socher, B. Huval, B. Bhat, C. D. Manning, and A. Y. Ng. 2012a. Convolutional-Recursive Deep Learning for 3D Object Classification. In *NIPS*.

R. Socher, B. Huval, C. D. Manning, and A. Y. Ng. 2012b. Semantic Compositionality Through Recursive Matrix-Vector Spaces. In *EMNLP*.

R. Socher, A. Karpathy, Q. V. Le, C. D. Manning, and A. Y. Ng. 2014. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*.

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

R. Socher, C. D. Manning, and A. Y. Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*.

R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning. 2011c. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *EMNLP*.

R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, and C. Potts. 2013d. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

N. Srivastava and R. Salakhutdinov. 2012. Multimodal learning with deep boltzmann machines. In *NIPS*.

P. J. Stone. 1966. *The General Inquirer: A Computer Approach to Content Analysis*. The MIT Press.

I. Sutskever, R. Salakhutdinov, and J. B. Tenenbaum. 2009. Modelling relational data using Bayesian clustered tensor factorization. In *NIPS*.

B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. 2004. Max-margin parsing. In *EMNLP*.

J. Tighe and S. Lazebnik. 2010. Superparsing: scalable nonparametric image parsing with superpixels. In *ECCV*.

I. Titov and J. Henderson. 2006. Porting statistical parsers with data-defined kernels. In *CoNLL-X*.

I. Titov and J. Henderson. 2007. Constituent parsing with incremental sigmoid belief networks. In *ACL*.

A. Torralba, K. P. Murphy, and W. T. Freeman. 2010. Using the forest to see the trees: exploiting context for visual object detection and localization. *Communications of the ACM*.

J. Turian, L. Ratinov, and Y. Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *ACL*.

P. Turney. 2002. Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews. In *ACL*, pages 417–424.

P. D. Turney and P. Pantel. 2010. From frequency to meaning: Vector space models of semantics. *JAIR*, 37:141–188.

L. Velikovich, S. Blair-Goldensohn, K. Hannan, and R. McDonald. 2010. The viability of web-derived polarity lexicons. In *NAACL*, HLT.

P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *ICML*.

T. Voegtlin and P. Dominey. 2005. Linear Recursive Distributed Representations. *Neural Networks*, 18(7).

S. Wan, M. Dras, R. Dale, and C. Paris. 2006. Using dependency-based features to take the "para-farce" out of paraphrase. In *Australasian Language Technology Workshop 2006*.

H. Wang, D. Can, A. Kazemzadeh, F. Bar, and S. Narayanan. 2012. A system for real-time twitter sentiment analysis of 2012 u.s. presidential election cycle. In *ACL 2012 System Demonstrations*.

P. J. Werbos. 1974. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D. thesis, Harvard University.

D. Widdows. 2008. Semantic vector products: Some initial investigations. In *Second AAAI Symposium on Quantum Interaction*.

J. Wiebe, T. Wilson, and Claire Cardie. 2005. Annotating expressions of opinions and emotions in language. *Language Resources and Evaluation*, 39.

T. Wilson, J. Wiebe, and P. Hoffmann. 2005. Recognizing contextual polarity in phrase-level sentiment analysis. In *HLT/EMNLP*.

B. Yao, X. Yang, L. Lin, M. W. Lee, and S.-C. Zhu. 2010. I2t:image parsing to text description. *IEEE Xplore*.

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

D. Yu, L. Deng, and F. Seide. 2012. Large vocabulary speech recognition using deep tensor neural networks. In *INTERSPEECH*.

H. Yu and V. Hatzivassiloglou. 2003. Towards answering opinion questions: Separating facts from opinions and identifying the polarity of opinion sentences. In *EMNLP*.

F.M. Zanzotto, I. Korkontzelos, F. Fallucchi, and S. Manandhar. 2010. Estimating linear models for compositional distributional semantics. In *COLING*.

L. Zettlemoyer and M. Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*.

Y. Zhang and J. Patrick. 2005. Paraphrase identification by text canonicalization. In *Australasian Language Technology Workshop 2005*.

F. Zhu and X. Zhang. 2006. The influence of online consumer reviews on the demand for experience goods: The case of video games. In *International Conference on Information Systems (ICIS)*.