Deep learning approaches to problems in speech recognition, computational chemistry, and natural language text processing

by

George Edward Dahl

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Deep learning approaches to problems in speech recognition, computational chemistry, and natural language text processing

George Edward Dahl
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2015

The deep learning approach to machine learning emphasizes high-capacity, scalable models that learn distributed representations of their input. This dissertation demonstrates the efficacy and generality of this approach in a series of diverse case studies in speech recognition, computational chemistry, and natural language processing. Throughout these studies, I extend and modify the neural network models as needed to be more effective for each task.

In the area of speech recognition, I develop a more accurate acoustic model using a deep neural network. This model, which uses rectified linear units and dropout, improves word error rates on a 50 hour broadcast news task. A similar neural network results in a model for molecular activity prediction substantially more effective than production systems used in the pharmaceutical industry. Even though training assays in drug discovery are not typically very large, it is still possible to train very large models by leveraging data from multiple assays in the same model and by using effective regularization schemes. In the area of natural language processing, I first describe a new restricted Boltzmann machine training algorithm suitable for text data. Then, I introduce a new neural network generative model of parsed sentences capable of generating reasonable samples and demonstrate a performance advantage for deeper variants of the model.

Acknowledgements

I am forever grateful to everyone who has made my time in graduate school at the University of Toronto the fulfilling and intensely meaningful experience it has been. I struggle to describe my mentor Geoff Hinton's supervision without descending into a morass of superlatives so effusive it strains believability, so I will be brief. Of course he is brilliant. Of course his piercing insight, intuitive clarity, droll wit, and laudable humility make being his student a delight. But even more exceptional are his integrity and his unwavering devotion to his students.

One of the great joys from my time in graduate school has been all of my office mates, past and present. I have been fortunate enough to share an office with both of the legendary Mnih brothers: Volodymyr and Andriy. I thank them for their humor and for being such inspiring role models. Thanks also to Navdeep Jaitly for "solving it" and embracing the office lingo; to Nitish Srivastava for putting up with my rants and dubious mentoring attempts (there is no escape when you share both an apartment and a workspace); to James Martens for speaking his mind; to Shenlong Wang for his good nature; and to James Bergstra for his productive influence.

Although I am grateful to all my research collaborators, the ones who worked with me on the projects described in this dissertation deserve specific commendations. Senpai Ryan Adams has been an excellent mentor and superb collaborator, not only on the word representation RBM project. In addition, I would like to acknowledge Hugo Larochelle's valuable research contributions and friendship. I am gratefully indebted to Navdeep Jaitly and Ruslan Salakhutdinov for work on the QSAR project (along with Chris Jordan-Squire's help during the Merck contest) and to Tara Sainath as well for being so generous with her time when I was struggling with Attila and also for running crucial full-sequence training experiments.

I also offer my heartfelt thanks to all of the machine learning graduate students and post-docs who overlapped with me at Toronto. Without a doubt, they made the machine learning group an enjoyable and stimulating research environment: Ryan Adams, Iain Murray, Hugo Larochelle, Marc'Aurelio Ranzato, Alex Graves, Tanya Schmah, Roger Grosse, Graham Taylor, Ruslan Salakhutdinov, Vinod Nair, Ilya Sutskever, Alex Krizhevsky, Tijmen Tieleman, Charlie Tang, Abdel-rahman Mohamed, Chris Maddison, Jasper Snoek, Maks Volkovs, Nikola Karamanov, Laurent Charlin, Danny Tarlow, Kevin Swersky, Jake Snell, Ryan Kiros, and everyone I didn't list (who nonetheless have my gratitude).

I would like to thank Ilya Sutskever in particular for his indomitable spirit, his inexhaustible optimism, and for all the late night research discussions in our apartment.

I would be remiss if I did not also thank Luna Keshwah for her great job keeping things running smoothly in the AI group and for all her help. My work would not have been possible without Relu Patrascu's excellent computing support. I am forever indebted to him for helping me keep my sanity when frustrating computing problems arose.

Throughout my doctoral studies I have been fortunate to have an outstanding committee and I would like to thank my committee members Rich Zemel and Brendan Frey for their inspiration and guidance.

Thank you to Matt and Margaret for everything.

Finally, I want to thank my parents for giving me every advantage a child can be given, for the love of learning they fostered, and for their unconditional support and encouragement (and also my father's indispensable editorial help).

Contents

List of Tables

List of Figures

Introduction

If computer science is the systematic characterization of the computations that we can perform efficiently, what, then, is machine learning? To solve a problem with a computer, one first designs an appropriately efficient algorithm that solves the problem and then implements that algorithm in hardware or software. If we cannot specify the algorithm, then we cannot solve the problem with direct programming. Machine learning extends what we can do with computers by letting us solve problems even when we are unable to manually design an algorithm to solve them. Using examples of correct behavior, we can specify an algorithm non-constructively. Thus a machine learning algorithm is a meta-algorithm for creating algorithms from data that defines what they should produce. Meta-algorithms give us a powerful new way of interacting with computers by telling them *what* they should compute instead of *how* they should compute it.

Extending our ability to solve problems with computers is reason enough to study machine learning, but it is far from the only reason. Just as studying learning helps us understand what we can practically compute, we must also study computation in order to inform our understanding of learning. Machine learning as a scientific discipline examines the computational basis of learning; thus it is essential even if we are only interested in how humans and animals learn. These two central motivations support each other. Trying to solve problems using computational models of learning sheds light on our understanding of the brain, and, by the same token, what we learn about the brain can serve as inspiration for designing learning machines.

Studying machine learning has scientific value both as a way to understand computation and as a way to understand learning, but for science to matter it must have a positive impact on the world. By always maintaining a connection to important practical problems, one increases the chance that their research on machine learning will have such a positive impact. One great property of the field of machine learning is that its methods can help us solve many specific problems of practical and commercial interest. However, if as researchers our only concern is making scientific progress, should we still labor over specific applications of our more general methods? Obviously we need to try new methods on actual problems to make sure that they work, but perhaps we can start with a new method and then look for problems it can attack. Alternatively, we can start with a problem and then do whatever it takes to solve it; sometimes solving a problem will require new methods and sometimes not, but in either case good research will help us learn the limitations and advantages of existing methods as well as what aspects of the problem are important. In the body of this dissertation, I use connectionist techniques to attack problems in three major machine learning application areas: speech recognition, computational chemistry, and natural language text processing (NLP). My goals are to test the limits of models based on neural networks and Boltzmann machines on problems in these areas, to determine how well successful recipes generalize across problems, and to extend the methods as needed to make them effective by

introducing new techniques where appropriate. In particular, I focus much of my attention on applying deep models to problems I view as sufficiently difficult and realistic to be of practical interest.

This dissertation makes several specific research contributions to computer science spread over the three machine learning application areas mentioned above. In some cases, these contributions have already had a substantial impact. In the first application area, automatic speech recognition, the contribution is an improvement to the standard deep neural net acoustic modeling approach popularized in Dahl et al. [2012a] and Hinton et al. [2012a]. This improvement has already made its way into most of the commercial and academic state of the art speech recognition pipelines and has enabled more accurate voice search and speech-to-speech translation. The next contribution pertains to computational chemistry: a deep neural network model for making predictions about the molecular activity of potential drug ingredients. Initial versions of this model were developed to win a virtual screening drug discovery competition sponsored by Merck. The model has already improved the computer-aided drug discovery tools Merck uses in production, potentially making it easier to develop new medicines. The remaining two contributions pertain to natural language text processing, the third domain considered. The first is a new restricted Boltzmann machine (RBM) training algorithm that makes training RBMs on word observations drawn from a large vocabulary feasible. This contribution has the potential to make deep models based on RBMs practical on natural language text data. The second contribution is a new deep neural network generative model of parsed text. This research enhances our understanding of the challenges in creating statistical models of text and may have an impact on natural language generation applications in the future. Although they cover three diverse and varied application areas, each one of these contributions tests and extends recent advances in deep learning.

This dissertation consists of an initial methodological prolegomenon (chapter 1) followed by several case studies grouped by application area. Chapter 1 describes my personal approach to machine learning and my conception of deep learning and includes a brief history of important deep learning advances of particular relevance to the later chapters. In order to set the stage for the case studies in chapters 2, 3, and 4, the chapter also includes important definitions and algorithmic background material.

The case studies begin with speech recognition and chapter 2, which introduces the basic supervised deep learning recipe that is used throughout this dissertation in the context of acoustic modeling. The chapter contains material previously published in "Improving deep neural networks for LVCSR using rectified linear units and dropout" [Dahl et al., 2013]. Chapter 3 discusses the application of similar deep learning techniques to computer-aided drug discovery. The work described in chapter 3 has previously appeared in a technical report [Dahl et al., 2014]; it has not yet been formally published. Chapter 4 presents two projects at the intersection of deep learning and natural language processing. Section 4.2 introduces a technique to efficiently train restricted Boltzmann machines (RBMs) on text that makes RBMs practical as building blocks for deep models of text. That work has previously appeared in "Training Restricted Boltzmann Machines on Word Observations" [Dahl et al., 2012b]. The remainder of chapter 4 presents a deep generative model of parsed sentences and explores some of the challenges to doing effective deep learning on natural language text.

Chapter 1

The deep learning approach and modern connectionism

Every approach to machine learning must ultimately grapple with the fundamental impossibility of truly general purpose learning. The so called no free lunch theorem for supervised machine learning [Wolpert, 1996] states that there is no learning algorithm that outperforms any given algorithm over all possible functions to be learned. By eliminating any possibility of a single, universally effective learning algorithm, the theorem leaves us in a difficult position. One way to recover some approximate notion of general purpose learning from the wreckage left by the no free lunch theorem is to drastically restrict the set of problems one would try to solve with machine learning. If we only consider learning tasks that are not typical of random functions and have a structure that allows a much simpler and more concise description than just a massive table of function values, we might be able to find a small set of learning algorithms that work well on a large number of these problems. In this regard, the example of the human brain offers an element of hope. Although not capable of learning typical random functions, the human brain is a powerful learning machine that works well on a wide array of problems. If we restrict our attention to problems that we believe an intelligent agent should be capable of solving, then perhaps the functions to be learned will have enough structure to allow a small number of different learning algorithms to perform well on all of them. Bengio and LeCun [2007] have taken this approach. They informally define the *AI-set* as exactly the restricted set of problems we expect intelligent agents to solve. Bengio and LeCun [2007] list visual and auditory perception, planning and control problems, and natural language processing as examples of AI-set tasks. The AI-set also includes problems that humans might not routinely solve, but that we could expect intelligent agents would be able to solve.

Although I cannot state precisely what problems should be in the AI-set, I do know that they will be problems with rich and complicated statistical structure. A statistical learning problem can be difficult for a variety of reasons. In a noise-dominant problem, for example, the central difficulty is distinguishing a faint signal in the data from much stronger noise. In a structure-dominant problem, on the other hand, the central learning difficulty is modeling the complexity of the correlations present in the data, while at the same time making efficient use of scarce training data and computational resources. An example of a noise-dominant problem would be determining whether a treatment tried only on a few dozen patients is effective or not. A very simple discrete distribution might suffice for describing the disease status of patients in the treatment and control groups, but difficulties arise when we need to detect small

treatment effects using very few samples, even in the presence of natural variations in patient outcomes unrelated to the treatment. Visual scene understanding is an example of a structure-dominant problem because, even when observations are not especially noisy, geometry, three dimensional objects in the scene, and lighting cause complicated correlations between pixel values that make learning difficult. A good model for visual scene understanding would leverage the constraints of the generative process to make sharp and accurate inferences. Similarly, in natural language processing, we can observe sequences of words with very little corruption, but the interactions between words that give rise to meaning have a very complicated structure. In general, for structure-dominant problems, we are not able to exactly specify an appropriate noise model (what is the noise model for the unobserved pragmatic context of human dialogue?); and we need to use highly flexible models that have some hope of explaining the variability in our data, even if we are under no illusions that they are good models of the true data generating process.

The following chapters deal with several structure-dominant learning problems likely to be in the AI-set. Additionally, I believe these problems to be inherently *worth solving* because of their connections to problems of practical importance. Solving more and more problems of this sort using fewer and fewer different learning algorithms and less and less task-specific engineering is how I hope to make progress towards the impossible goal of general purpose learning and the distant goal of understanding intelligence and learning. Thus I seek to extend methods successful on, say, vision tasks to speech and language processing problems and vice versa. Furthermore, I will strive to make the necessary task specific engineering itself more general by doing less feature engineering and more objective function or learning architecture engineering. By engaging with important practical applications of machine learning throughout this dissertation, I hope to ensure that any increase in the general applicability of the methods will be meaningful and useful.

1.1 A machine learning manifesto

As mentioned above, I am only concerned with structure-dominant learning problems of practical importance and AI relevance. Just as it was necessary to restrict the set of problems under consideration, I also describe what specific properties I will and will not seek in the algorithms I investigate. The seven principles below define my machine learning methodology. Broadly speaking, the desiderata enumerated below lie along two dimensions: power and practical applicability. Principles 1–3 demand powerful learning architectures that can represent a large and interesting hypothesis space or set of functions and can use representations with sophisticated componential properties. Principles 4–7 describe our need for models applicable to the actual problems we wish to solve.

Ideally, machine learning models must:

1. Have multiple compositional levels of non-linear feature extractors.

2. Learn distributed representations [Hinton et al., 1986] of their input.

3. Be sufficiently expressive and have large enough capacity to solve interesting practical problems.

4. Have learning procedures that scale close to linearly with the number of training cases.

5. Scale to thousands of dimensions (although not necessarily to thousands of intrinsic dimensions).

6. Be able to make use of unlabeled data and weakly labeled data.

7. Allow efficient approximate inference procedures.

All these restrictions make the task of developing acceptable algorithms very difficult, so I also list three properties that I view as too costly to demand of our learning algorithms. Although these properties are common requirements of other approaches to statistical learning, our models need not:

1. Have tractable exact inference.

2. Have training procedures that guarantee convergence to the global optimum.

3. Be easily interpretable by humans.

1.1.1 Powerful models: depth, expressivity, and distributed representations

Shallow learning machines that perform simple template matching will not be enough to solve difficult problems without incurring an unacceptable cost in human engineering labor. Bengio and LeCun [2007] argue, based on evidence from circuits, that we need families of models that are capable of optimizing the breadth/depth tradeoff. The most efficient models may need to be deep since an insufficiently deep circuit can require exponentially more computational elements. Beyond circuit complexity results, there is also evidence of a similar effect for machine learning models; in some cases deeper models can be exponentially more efficient than shallower models in terms of parameters or computational elements. Sometimes this evidence takes the form of a negative result about a particular class of shallower models. For example, Bengio et al. [2010] showed that single decision trees cannot generalize well to variations not seen during training because they partition the input space into leaves and independently parameterize each leaf, necessitating at least one training case for each leaf. However, we can use a forest of trees to learn a new representation of the input. Each tree in the forest becomes a non-linear feature detector with an activation indicating which leaf the input vector was assigned to. Using a forest of trees in this way results in a deeper model capable of representing more rapidly varying functions without needing a training case for each variation. Beyond decision trees, Bengio et al. [2006] proves an analogous result for kernel machines with local kernels. Although sum product networks are too weak to be an interesting class of models for our purposes, Martens and Medabalimi [2014] have shown that, with each increase in depth, sum product networks[1] can efficiently represent a larger set of distributions.

Unfortunately, our theoretical understanding of the advantages of deeper models remains limited. Indeed, there is no a priori reason to prefer a deeper model over a shallower model for any given arbitrary problem (if deeper were always better, we would have to keep making deeper and deeper models forever and yet we know that the finitely deep human brain can solve many problems much better than any learning machine we have created). However, any models we use must be capable of trading depth for breadth as needed to solve the problem at hand. The correct depth will invariably be problem dependent and restricting ourselves to shallow or single hidden layer models is unacceptable. The reason we must explore models that can employ multiple compositions of non-linear feature extractors is that we must be able to adapt the depth of our models to suit the problem. Whether that adaptation occurs in the training process or the model selection process, it has to be a part of how we attack a new problem.

[1] Specifically Martens and Medabalimi [2014] deals with decomposable and complete sum product networks (decomposability and completeness suffice to allow tractable marginals).

Results indicating an advantage for deeper models than the shallowest universal approximator[2] appear in so many contexts because they get at a fundamental tradeoff in computation. The breadth/depth tradeoff is just another way of looking at the space/time tradeoff or, in some sense, the parallel time (with a modest number of processing units) vs. sequential time tradeoff. Even if it might be quite difficult to prove, I do not believe that all interesting computations can be done efficiently in a single sequential step with many parallel operations. Even the massively parallel human brain has many sequential steps of processing for sensory data and furthermore makes use of feedback connections for most longer running computations it performs.

Architectural depth alone does not suffice to characterize expressive, high capacity learning machines. For example, suppose that we have an artificial neural net mapping inputs to outputs with several layers of binary neurons and that neurons in each layer only receive input from the layer immediately below. Suppose further that, for whatever reason, this device only allows a single neuron in each layer to be active at once. If each layer has n neurons, then this model can only produce at most n different outputs regardless of what it receives as input or how many layers it has! In order for learning machines to represent massive numbers of concepts or data cases well, they must exploit combinatorics. The question of how massively parallel hardware (in particular the human brain) might encode information led Hinton et al. [1986] to the notion of *distributed representations*. In contrast to a local representation where the state of a single computational unit represents a single concept, a distributed representation stores a concept as a pattern of activity across multiple units. If an encoding of a set of concepts is just a mapping from concepts to vectors in some space, then a local representation corresponds to a mapping that maps concepts $c_1 \neq c_2$ to two orthogonal concept vectors; a distributed representation corresponds to using a mapping that only requires different concepts to map to vectors that are not parallel. Hinton et al. [1986] also sought a computational account of how the brain could store so many memories and still access them in such a flexible way. Humans seem to have immensely powerful faculties for accessing stored memories. We can recall memories even with incomplete, noisy information about the memories we hope to access. We can effortlessly uncover many related relevant memories as we mentally search for information about a particular topic. Human memories are lossy and sometimes confused, but they also have rich representations that link them to other memories. If we suppose that a recollection is a pattern of activity over the neurons in our brain and that (at least some) memories are stored in the learned connections between neurons, we can create computational models (as Hinton et al. and others have done) that learn distributed representations and can reproduce some of these memory properties, even though the computational models have almost no relationship to how actual biological hardware operates.

Before we describe some of the concrete advantages of models capable of learning distributed representations of their input, a brief digression on biologically inspired machine learning is in order. The success of biological learning machines gives us hope that learning machines designed by humans may solve some of the learning problems that humans do, and hopefully many others as well. However, biologically inspired machine learning does not mean blindly trying to simulate biological neurons in as much low level detail as possible. Although such simulations might be useful for neuroscience, my goal

[2] Almost all machine learning models in common use today can approximate any function, subject to some technical restrictions, if they are allowed to grow infinitely large. Unfortunately, since almost all reasonable machine learning models have some sort of universal approximation result, these theorems give us absolutely no guidance whatsoever on what method to use on any given problem. Only rarely do we have rigorous answers to the more interesting questions of exactly how efficiently a given model can represent certain functions or whether a particular training algorithm can find the correct values for the parameters, assuming they exist.

is to discover the principles that allow biological agents to learn and to use those principles to create my own learning machines. Planes and birds both fly, but without some understanding of aerodynamics and the larger principles behind flight, we might just assume from studying birds that flight requires wings that can flap. Biologically inspired machine learning means investigating high-level, qualitative properties that might be important to successful learning on AI-set problems and replicating them in computational models. For example, themes such as depth, sparsity, distributed representations, and pooling/complex cells are present in many biological learning machines and are also fruitful areas of machine learning research. The reason to study models with some of these properties is because we have computational evidence that they might be helpful, not simply because our examples from animal learning use them.

Along with depth, distributed representations will be necessary for building expressive models that are also statistically and computationally efficient. Distributed representations use non-mutually exclusive features or, in other words, patterns of activity over multiple computational units. They contrast with feature detectors with parameters constrained by single data cases. We study models that can learn distributed representations because of their numerous advantages in statistical and computational efficiency. Distributed representations allow our models to share statistical strength between feature detectors by generalizing across inputs that share a particular feature and allow our models to represent new concepts without new hardware or new parameters. Good distributed representations will learn to conflate truly similar inputs and will constrain the parameters of different computational units in our model using information from a large fraction of data cases.

1.1.2 Scalability and broad applicability

Beyond expressivity, flexibility, and statistical efficiency we need models that are scalable, both to large datasets and to high-dimensional spaces. We also need training and inference algorithms that scale to the inevitably larger high-capacity instances of our models that are required for getting good results on important practical problems. We want systems capable of never-ending learning that can keep up with streams of data. As new data arrives, the model must incorporate its new information so any training algorithm that takes time that grows worse than almost linearly in the number of data cases will eventually fall behind as new data arrives in real time. For the finite, modestly large datasets common in many problems, we can afford training procedures worse than strictly linear, but generally anything quadratic or worse will quickly become impractical.

Concretely, much of the data we want to train our systems on that currently exists takes the form of images, video, audio, and text, any of which can have hundreds or thousands of dimensions in a reasonable encoding. Any algorithms we hope to use on these data must not break down when confronted with thousands of input dimensions. Computationally, this will mean that learning and inference need to be polynomial in the number of dimensions and efficient in practice as well as asymptotically. However, since AI-set problems hopefully have a lot of structure, expecting our algorithms to gracefully handle datasets with a similarly high number of intrinsic dimensions of variation may be unrealistic. In general, assuming a relatively small number of intrinsic, independent factors of variation may be necessary.

At test time, when we apply a trained model, we also need to be able to perform efficient approximate inference (which may also be necessary for efficient training anyway). Biological learning machines are capable of performing sophisticated object recognition tasks quite quickly and in order to match or exceed their performance on such tasks, our models will need to allow efficient approximate inference.

Beyond scalability, we also need models that we can apply to as many learning problems as possible, including problems where much of the available data have no labels. Models related to some of the ones I apply in later chapters can be used in supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. However, most of the highest profile successful machine learning applications of recent years have used at least some supervised data. Unsupervised learning remains a major research challenge.

1.1.3 Contrasting approaches

My approach to machine learning contrasts with approaches that put great emphasis on models with convex training problems or, more generally, approaches that emphasize training procedures guaranteed to find the globally optimal parameters. As nice as it would be to have such theoretical guarantees, to get them we often must pay an unacceptable price in the size of the model, the practical sample complexity and generalization performance, or the scalability of the training algorithm. For many years now, state-of-the-art systems in speech recognition, handwriting recognition, and object recognition have been non-convex. We must not let appealing theoretical properties blind us to practical realities. Even if we indulge our imagination and view human learning as a single explicit optimization problem over data from life experiences, human learning cannot be "convex" because it depends heavily on the order of the training cases, even on relatively short time scales.

Tractable exact inference plays a similar role to guaranteed convergence to the global optimum. Requiring tractable exact inference imposes unacceptable limits on the expressivity of our models. Of course, *some* sort of tractable inference is essential for practical reasons, but effective approximate inference schemes are all that is truly necessary.

My approach to machine learning is centered on making correct predictions. The most powerful and effective predictive models will not generally be easily interpretable by humans. Small, extremely sparse models, although easy to understand, have a much harder time fitting large datasets that have a lot of variability. Although our models may end up being difficult to interpret in some ways, they may be interpretable in other ways. For example, a large model with many anonymous latent variables will make it hard to associate individual latent variables with real-world quantities or concepts, but we still may be able to determine which input features give rise to the model's prediction for a particular data case. Despite these difficulties, we should still strive to understand as much as we can about how our models are operating, even if visualizing and interpreting our models becomes a research question in its own right.

In later chapters, I present state of the art results on important problems achieved as a direct result of my approach to practical machine learning. My approach is consonant with those of other researchers in the deep learning community. In the following section I review some relevant previous advances.

1.2 Recent history and the deep learning revival

In this section I will provide a personalized history of important events in the deep learning subfield. Deep learning is not a particular method or model, although statistical models and learning architectures can be deeper or shallower. Deep learning is an approach to machine learning that emphasizes certain tradeoffs and demands methods with certain properties. In terms of its sociological and historical context,

deep learning is a recent form of connectionism, but much of its intellectual content does not depend on a connectionist perspective, inspiration from biological learning, or artificial neural networks.

The seeds for what we now call deep learning were planted in the 1980s with the backpropagation algorithm [Rumelhart et al., 1986] and an algorithm for training Boltzmann machines [Ackley et al., 1985]. Backpropagation promised to train nets with one or more layers of hidden units capable of learning their own features instead of relying entirely on hand-engineered features the way (zero hidden layer) perceptron networks did. Even on computer hardware of the 1980s, backpropagation was capable of solving practical problems and ever since, as computer hardware has improved, there has been an explosion of applications of neural networks using backpropagation. Technically, the backpropagation algorithm is an algorithm for computing gradients and not a full training algorithm, but Rumelhart et al. [1986] described a stochastic gradient descent algorithm with momentum that remains in wide use today.

The Boltzmann machine algorithm [Ackley et al., 1985], however, was much less practical than backpropagation on the hardware available at the time of its invention. Nevertheless, more efficient algorithms for training less general Boltzmann machines have since been developed, in particular layered Boltzmann machines. A layered Boltzmann machine has its hidden units grouped into layers such that no edges link units within a single layer to each other. A restricted Boltzmann machine (RBM) is a layered Boltzmann machine with a single layer of hidden units. Thus the hidden units of an RBM are conditionally independent of one another given the states of the visible units. Hinton [2002] introduced the contrastive divergence algorithm which can efficiently train RBMs (better algorithms for training deeper layered Boltzmann machines emerged later). Boltzmann machines are an important bridge between neural networks and probabilistic graphical models and could be one way of implementing feedback in neural networks. Neural nets with feedback as well as feedforward connections may be necessary for some problems, although the advantages of feedback connections might be achievable by "unrolling" a recurrent neural net or graphical model inference procedure to create a multi-pass feedforward net.

Backpropagation established the basic supervised neural network machinery for learning functions mapping low-dimensional dense vectors to other low-dimensional dense vectors. However, many application domains (such as natural language processing) have large discrete input spaces. In the example of neural language models, we might need to predict the next word given n previous word observations, where each word observation is a categorical variable with hundreds of thousands of possible values. If we represent the inputs with the natural one-of-k encoding, then we will need to multiply an nV-dimensional sparse vector with an nV by H dense matrix, where V is the vocabulary size and H is the number of hidden units in the first hidden layer. Performing sparse matrix arithmetic helps to some extent, but since the weight matrix is dense we still have far too many weights. Bengio et al. [2001] solved the large discrete input space problem by learning an embedding layer which, in the context of language modeling, learns word embeddings. To learn word embeddings, we factor the initial weight matrix, drastically reducing the number of weights we need to learn. Instead of one nV by H matrix, we have a V by d matrix of word vectors and n, $d \times H$ matrices of position dependent weights which, since we can use fast table lookups to find the d-dimensional vector for any given word, results in n products of a d dimensional vector by a $d \times H$ matrix. When d is much smaller than the vocabulary size V and smaller than the number of hidden units H, learning word embeddings provides a substantial savings in parameters and computation time, as well as letting us factor out word-specific information into word

feature vectors. The word embedding trick uses sparsity, weight tying, and low rank weight matrices simultaneously and has been instrumental in the success of neural language models (e.g. Mnih and Hinton [2007], Mnih [2010], and Mikolov [2012]). Unfortunately, word observations, and large categorical spaces more generally, pose problems in the output layer as well as the input layer. Morin and Bengio [2005] solved the large, discrete output space problem by building a hierarchical analogue to the traditional softmax output layer for classification. Using a tree over the vocabulary is exponentially more efficient than a standard, flat softmax layer. Later, Mnih and Hinton [2009] described an effective way to learn the tree over the vocabulary required for the hierarchical softmax (also called tree softmax) layer. For language modeling problems in particular, a simple Huffman coding heuristic can build a serviceable tree as well [Mikolov et al., 2013a]. Since arranging words at the leaves of a binary tree is, in effect, giving each word a binary spelling, another alternative to a tree softmax is to predict characters instead of words, as in Sutskever et al. [2011].

Although not as shallow as the zero hidden layer nets of the perceptron era, many backpropagation neural networks in use before 2006 only had a single hidden layer. One notable exception were convolutional neural nets [LeCun et al., 1989] which typically had several layers of hidden units, for example the famous LeNet-5 [LeCun et al., 1998]. In some cases, the contrast between symbolic and connectionist approaches (or some other comparison) was simply of more interest to researchers than the depth of the neural networks (for instance in Shavlik et al. [1991]). Tesauro [1992], describing the backgammon playing network TD-Gammon (a high profile neural network and temporal difference learning success story), presented experimental results comparing zero hidden layer neural networks to single hidden layer neural networks, but did not explore networks beyond one hidden layer. Many papers, including Tesauro [1992], appealed to universal approximation results for single hidden layer neural networks when discussing what network depths their experiments would cover. Although indeed a good justification for using hidden layers, citing the universal approximator properties of single hidden layer neural networks to justify restricting attention to single hidden layer nets is not logically sound because deeper nets are also universal approximators; therefore, the theorem only provides a justification for going beyond zero layer models, not a justification for stopping at one layer models. Researchers were most likely aware of this fallacy, but nevertheless depth (beyond a single hidden layer) was usually not investigated in the literature as an interesting research question in its own right. In his esteemed neural networks textbook, Christopher Bishop only briefly mentions the depth issue that has come to capture so much attention today.

> Since we know that, with a single hidden layer, we can approximate any mapping to arbitrary accuracy we might wonder if there is anything to be gained by using any other network topology, for instance one having several hidden layers. One possibility is that by using extra layers we might find more efficient approximations in the sense of achieving the same level of accuracy with fewer weights and biases in total. Very little is currently known about this issue. However, later chapters discuss situations in which there are good reasons to consider networks with more complex topologies, including networks with several hidden layers, and networks with only partial connectivity between layers. [Bishop, 1995, p. 132]

The later chapters he refers to cover convolutional neural networks and autoencoders with an extra hidden layer before and after the code layer. However, in the case of the latter, if we view an autoencoder as two neural networks, an encoder and a decoder, both of the two networks were shallow.[3] Individual

[3]Viewing the encoder and decoder separately makes sense when considering architectural depth since an autoencoder

researchers make decisions on what models to try based on numerous pragmatic factors so it is hard to know exactly why deeper models were not more commonly used without interviewing the entire research community, but several potential reasons present themselves. As alluded to above, one reason might have been a misinterpretation of the implications of universal approximation results combined with a limited budget and no particular positive reason to think deeper nets would be worth training. Another, highly pragmatic, reason could simply have been that the computers of the time made the necessary experiments too expensive. A related issue is that when hidden layers only have a few dozen units, adding more units can still provide a large performance boost that diminishes only when layers are already wide. Small datasets of the time might not have justified more powerful, deeper models especially since before the rise of kernel methods neural net researchers may have tried to control overfitting by using undercomplete models instead of by using heavily overcomplete and heavily regularized models. Certainly for some researchers, training difficulties for deeper networks, perhaps due to poor initializations, were a major barrier to their use.

Hinton et al. [2006] sparked renewed interest in deeper models by providing a new way to initialize their weights before starting supervised training and demonstrated impressive results with the proposed technique. The unsupervised pre-training algorithm from Hinton et al. [2006] could be used to initialize deep sigmoid belief nets and deep feedforward neural networks (and a similar algorithm could even initialize deep layered Boltzmann machines [Salakhutdinov and Hinton, 2009a]). Unsupervised pre-training initialized the weights of a deep network in a greedy, layer-wise fashion by training a restricted Boltzmann machine (RBM) at each level to model the activations of the units in the level below on the training set. The pre-training algorithm, along with improved computer hardware, gave hope that it might now be possible to train deeper models and Hinton and Salakhutdinov [2006] even gave evidence that training deep autoencoders to do nonlinear dimensionality reduction might also be feasible.

Along with promising new training algorithms, improvements in computer hardware also helped foster increased interest in high capacity deep models. Graphics processing units (GPUs), designed to meet the demands of graphics intensive computer games, had outpaced CPUs in peak operations per second for highly parallel workloads and GPU hardware improvement trajectories looked even more promising. GPUs are ideal for performing dense matrix arithmetic of the sort needed to train neural networks and researchers were able to use GPUs to perform experiments over an order of magnitude more ambitious than what was possible on CPUs alone [Mnih, 2009]. Wide and deep neural nets trained on GPUs were capable of handling much larger datasets [Mnih and Hinton, 2010, Raina et al., 2009], a regime where deeper and more expressive models have a natural advantage. Ciresan et al. [2011] obtained impressive results with a basic deep neural network on digit recognition benchmarks using data augmentation and plain backpropagation by training on a GPU for a very long time. Experiments that might take a month on a CPU could be completed in a day.

Despite the renewed interest in deeper models in the neural networks community, as of 2009, prac-

with no hidden layers other than the code layer is analogous to a zero hidden layer linear classifier; in such a shallow autoencoder Bourlard and Kamp [1988] showed that, at least for the case of linear output units and squared error, the code layer nonlinearity is irrelevant and the optimal weights can be obtained with singular value decomposition. Therefore an autoencoder with a hidden layer before and after the code layer is really best viewed as a single hidden layer encoder followed by a single hidden layer decoder. Interestingly, the result of Bourlard and Kamp [1988] also bounds the reconstruction error of autoencoders formed from an arbitrarily deep encoder and a shallow (zero hidden layer) decoder. In such networks, the reconstruction error will never be better than the error achieved by principal component analysis (PCA). In general, if we create a deep autoencoder with linear output units and h_k units in the last hidden layer, regardless of which hidden layer we decide to designate the code layer, the average squared reconstruction error will never be better than that achieved by h_k-dimensional PCA.

titioners in important machine learning application areas such as vision, speech, and natural language processing, did not use deep neural nets as a routine part of their experimental pipelines. Mohamed et al. [2012] (preliminary results first presented in Mohamed et al. [2009]) used pre-trained, deep neural nets to perform phone recognition and showed extremely promising results on a small scale speech recognition benchmark. Ultimately, the same basic approach led to the first successful application of deep neural nets to large vocabulary speech recognition [Dahl et al., 2011, 2012a]. The improvements achieved by early deep neural net models over existing methods in large vocabulary speech recognition were dramatic enough that the speech community quickly switched to using deep neural net acoustic models in all state of the art recognition pipelines (see Hinton et al. [2012a] for a review as well as section 2.2 for more detail on deep neural nets in acoustic modeling). Acoustic modeling is a domain where computational constraints dominate data availability constraints or, in other words, there is more transcribed audio data available (at least for some tasks) than one can easily use during training. Given that a primary benefit of unsupervised pre-training is that it serves as a special data-dependent regularizer (or a method to leverage unlabeled data to constrain the weights), deep neural net optimization issues might be more important in acoustic modeling applications.

Even before evidence began to accumulate that deep and wide neural nets might not be too hard to train given abundant labeled data, Martens [2010] explored optimization issues in deep autoencoders by adapting a truncated Newton method to neural net training problems with difficult curvature issues. The particular truncated Newton method of Martens [2010] is known as "Hessian-free optimization" (or HF) in the deep learning community and I adopt that terminology as well. Martens and Sutskever [2011] demonstrated that HF was capable of solving almost pathologically hard recurrent neural net optimization problems. Work on HF along with results from domains where labeled data are abundant and pre-training provides only a small benefit raised questions about how important pre-training was for optimization (its regularization effects are much clearer) and how best to solve the optimization problem for purely discriminative deep neural nets. Bengio and Glorot [2010] and Mohamed et al. [2012] did not report difficulty training nets with many large hidden layers, so it is possible that unsupervised pre-training mostly provides an initial set of weights with good scales. Good weight initialization is critically important for training deep neural nets well [Sutskever et al., 2013], but pre-training is far from the only initialization method that can work well. With proper initialization, the unreasonable effectiveness of well-tuned stochastic gradient descent with momentum [Sutskever et al., 2013] still makes it the most common choice for deep neural net training.

Almost any differentiable (almost everywhere) function can be used as a hidden unit activation function in neural nets. Sigmoids have historically been a particularly popular default choice, but recently rectified linear units [Jarrett et al., 2009], or ReLUs, have been used with great success in several important applications (e.g. Krizhevsky et al. [2012]). The best activation functions will invariably be problem dependent and the choice of activation function can have effects on the learning dynamics and generalization performance of the model, making it difficult to try many different functional forms without re-optimizing all of the tuning parameters of the training algorithm. Nair and Hinton [2010] defined several variants of ReLU activation functions for use in RBMs and used them successfully in a computer vision task. Since ReLUs can easily maintain relative intensity information through multiple layers of units, they can be particularly helpful in vision tasks and have become a common choice in numerous applications.

The trend towards much larger and deeper neural nets, often with many more parameters than avail-

able training cases, has demanded more sophisticated regularization schemes than simple weight norm penalties. The dropout method [Srivastava et al., 2014] randomly corrupts neuron states during forward propagation through a network by zeroing them out independently with some dropout probability p. Dropout is such a powerful regularizer that massive neural nets can be trained on comparatively small datasets without early stopping and still exhibit no evidence of overfitting.

Larger networks with more layers and tunable dropout rates and activation function choices result in dozens of metaparameters (training algorithm tuning parameters along with architectural model parameters). Optimizing all the metaparameters by hand heuristically is not easily repeatable by other researchers and does not scale well to large numbers of metaparameters or larger experiment budgets. Random search [Bergstra and Bengio, 2012] usually outperforms grid search and can work well when a large number of experiments are feasible, but to achieve excellent results repeatably and automatically, more sophisticated methods for tuning metaparameters are necessary. Snoek et al. [2012] presented a Bayesian optimization algorithm that is practical for deep neural networks and released software capable of beating published results from the literature simply by tuning metaparameters more effectively than the initial researchers. Bayesian optimization is a global, gradient-free optimizer that builds a model of the function to be optimized and balances uncertainty about untried points with the current beliefs of the model about the quality of the untried points. Snoek et al. [2012] use a Gaussian process regression model to learn the function mapping metaparameter settings to validation performance. Improvements in Bayesian optimization techniques will undoubtedly have far-reaching consequences throughout the experimental sciences, but for machine learning researchers training models with dozens of metaparameters they have already had a large impact, especially since much of the software implementing Bayesian optimization has been designed with deep neural network metaparameter tuning problems in mind.

1.3 Feedforward neural networks

We will make extensive use of feedforward neural nets in later chapters, so I will describe the necessary background material here. An L-layer, feedforward neural network is a vector valued function $\mathbf{y} = f(\mathbf{x})$ mapping input row vectors $\mathbf{x}$ of shape $1 \times d$ to output row vectors $\mathbf{y}$ of shape $1 \times m$ defined by:

$$\mathbf{y}_0 = \mathbf{x} \tag{1.1}$$

$$\mathbf{z}_l = \mathbf{y}_{l-1}\mathbf{W}_l + \mathbf{b}_l \tag{1.2}$$

$$\mathbf{y}_l = h_l(\mathbf{z}_l), \tag{1.3}$$

where $\mathbf{W}_l$ is the weight matrix between layer $l-1$ and layer l, $\mathbf{b}_l$ is the row vector of biases for layer l, and h_l is an element-wise, nonlinear *activation function* for layer l. We shall call $\mathbf{y}_l$ the *activation* or *state* of layer l and call $\mathbf{z}_l$ the *net input* to layer l. The output vector $\mathbf{y}$ is simply the activation of the final layer, $\mathbf{y}_L$, so we will call layer L the output layer and layer 0 the input layer. The parameters or weights of the network are the weight matrices $\mathbf{W}_l$ and bias vectors $\mathbf{b}_l$ for all the layers. Technically the neural network function is a function of both the parameters and the input, although we usually will consider one of them fixed. Each layer is made up of units or neurons and the jth dimension of the state vector of a layer holds the state of the jth unit in that layer. Furthermore, the entry $W_{lij} \equiv [W_l]_{ij}$ is the weight on the connection from unit i in layer $l-1$ to unit j in layer l. We could write all of our equations in terms of units instead of layers, but the layer matrix notation is more compact. In general,

all units in a layer are fully connected to the units in the layer below, but in some cases we constrain particular entries of a weight matrix $\mathbf{W}$ to be zero to encode connectivity restrictions.

By stacking n, d-dimensional input vectors into a $n \times d$ matrix, we can also write equations for processing a minibatch of input vectors at a time that are only cosmetically different from the equations above for minibatches of size 1. For example, the minibatch version of equation 1.2, the net input to layer l, would be written

$$\mathbf{Z}_l = \mathbf{Y}_{l-1}\mathbf{W}_l + \mathbf{b}_l.$$

Since converting to the minibatch form is so trivial, we will generally stick to the single case minibatch form for clarity of presentation.

We will also make use of a straightforward generalization of feedforward neural nets to networks with multiple input and output layers that we shall call DAG-nets. The connections between layers in a DAG-net form a directed acyclic graph (DAG) such that input layers have no ancestors. Only equation 1.2 needs to change to accommodate the multiple layers feeding into layer l. If we number the layers in some topological order according to the DAG of links between layers, then layer l will only ever receive input from layers $k < l$. Let g_{kl} be the entries of the $L \times L$ adjacency matrix for the layer connectivity graph. We now must use two numbers to index into our set of weight matrices, so let $\mathbf{W}_{kl}$ be the matrix of weights from layer k to layer l, if it exists. Now the net input equation becomes:

$$\mathbf{z}_l = \mathbf{b}_l + \sum_{k<l} g_{kl}\mathbf{y}_k\mathbf{W}_{kl}. \tag{1.4}$$

To keep the presentation uncluttered, we will generally write the equations in the remainder of this section for feedforward nets, not DAG-nets.

Our goal is to learn the parameters $\Theta = (\{\mathbf{W}_l\}, \{\mathbf{b}_l\})$ of a neural network from a dataset of input and output vectors $\{(\mathbf{x}_n, \mathbf{t}_n)\}$. Thus we wish to minimize some cost function $C(\Theta)$ of the parameters defined using the training set $\{(\mathbf{x}_n, \mathbf{t}_n)\}$. In general we will have regularized cost functions with a regularization term that is independent of the data and only depends on the model parameters. For example, with ℓ_2 regularization the regularization term is just the sum of the Frobenius norms of the weight matrices times the weight cost λ:

$$\lambda \sum_l \|\mathbf{W}_l\|^2.$$

A few specific cost functions will be of interest later on. We have written them below without any regularization terms. Given a feedforward neural net function $\mathbf{y}(\mathbf{x}, \Theta)$, the mean squared error cost function is:

$$C_{\text{MSE}}(\Theta) = \frac{1}{2N} \sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \Theta) - \mathbf{t}_n\|^2 \tag{1.5}$$

The K-class cross entropy error cost function is defined by:

$$C_{\text{KCE}}(\Theta) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \log y_k(\mathbf{x}_n, \Theta), \tag{1.6}$$

where the target vectors $\mathbf{t}_n$ have a one-of-K encoding scheme with the entry t_{nk} being 1 if the nth training case is of class k and zero otherwise. In the binary case, simplifying equation 1.6 to have only

a single output unit, we have:

$$C_{\text{CE}}(\Theta) = -\sum_{n=1}^{N} t_n \log y(\mathbf{x}_n, \Theta) + (1 - t_n) \log(1 - y(\mathbf{x}_n, \Theta)). \tag{1.7}$$

Before we can actually optimize the parameters of a neural net to minimize some cost function, we must make concrete choices for the activation functions in each layer. For the output layer, we generally select an activation function that matches the cost function we are using. For squared error, we typically use a linear activation function for the output layer, often the identity function. For K-class cross entropy error, we typically use the softmax activation function defined by

$$y_m = \frac{e^{z_m}}{\sum_{k=1}^{K} e^{z_k}}, \tag{1.8}$$

where z_k is the kth dimension of the net input to the output layer. The logistic sigmoid activation function, $\sigma(z) = \frac{1}{1+e^{-z}}$, matches the simplified, two-class cross entropy error function and is also commonly used as a hidden unit activation function. In general, for hidden units, we will use either the logistic sigmoid, hyperbolic tangent sigmoid, or a rectified linear activation function defined by $h(z) = \max(0, z)$. Rarely we will also use non-rectified linear units (identity function), in particular for the code layers of autoencoders or as a way of implementing factorized weight matrices.

Given the layer topology, layer sizes, activation functions, and cost function for a feedforward net, forward propagation through the network is well defined and we can consider the training problem. All of our training procedures will use the gradient of the cost function with respect to the parameters averaged over a minibatch of training cases. Computing $\frac{\partial C(\Theta)}{\partial \Theta}$ for a training case requires a forward pass (defined above) and a backward pass through the network. Given a target vector $\mathbf{t}$ along with layer activations $\mathbf{y}_l$ computed in the forward pass using the input $\mathbf{x}$, we define the *error signal* $\boldsymbol{\delta}_l$ at each layer in an L-layer feedforward net with the recurrence below (reproducing equations 1.1 – 1.3 for the forward pass).

$$\begin{aligned}
\mathbf{y}_0 &= \mathbf{x} \\
\mathbf{z}_l &= \mathbf{y}_{l-1}\mathbf{W}_l + \mathbf{b}_l \\
\mathbf{y}_l &= h_l(\mathbf{z}_l) \\
\boldsymbol{\delta}_L &= \mathbf{t} - \mathbf{y}_L \text{ (assuming a matching cost function)} \qquad (1.9) \\
\boldsymbol{\delta}_l &= \boldsymbol{\delta}_{l+1}\mathbf{W}_{l+1}^{\mathrm{T}} h_l'(\mathbf{z}_l) \qquad (1.10)
\end{aligned}$$

The negative gradient for the current training case with respect to $\mathbf{W}_l$ is given by

$$-\frac{\partial C}{\partial \mathbf{W}_l} = \mathbf{y}_{l-1}^{\mathrm{T}} \boldsymbol{\delta}_l \tag{1.11}$$

and the negative gradient for the current training case with respect to $\mathbf{b}_l$ is given by

$$-\frac{\partial C}{\partial \mathbf{b}_l} = \boldsymbol{\delta}_l. \tag{1.12}$$

The training algorithm we will use is minibatch stochastic gradient descent with momentum (SGD).

Let $\langle \frac{\partial C}{\partial \theta(t)} \rangle$ be the derivative of the cost function with respect to one of the model parameters averaged over the current minibatch of m training cases using the model parameters at iteration t. Assuming an ℓ_2 weight penalty of strength λ (generally not applied to the biases), the update rule for θ at iteration t is given by:

$$\Delta\theta(t) = \epsilon \left(- \left\langle \frac{\partial C}{\partial \theta(t)} \right\rangle - \lambda\theta(t) \right) + \alpha\Delta\theta(t-1) \tag{1.13}$$

$$\theta(t+1) = \theta(t) + \Delta\theta(t), \tag{1.14}$$

where ϵ is the learning rate and α is the momentum constant. Sometimes we will use separate learning rates for different groups of parameters and we will often change the learning rate between training iterations. In many cases we also train with dropout [Hinton et al., 2012b, Srivastava et al., 2014].

1.4 Restricted Boltzmann machines and pre-training

Since chapters 2 and 4 both involve restricted Boltzmann machines (RBMs) I will describe the basics of RBMs here. Restricted Boltzmann machines [Smolensky, 1986] are a type of undirected graphical model that has two layers of units, the hidden (latent) units and the visible (observed) units. The visible and hidden units form a bipartite graph with no hidden to hidden or visible to visible connections. We denote the state of the model with two column vectors: $\mathbf{v}$ holding the states of the visible units and $\mathbf{h}$ holding the states of the hidden units. A binary RBM assigns an energy to every configuration of visible and hidden state vectors according to:

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{b}^{\mathrm{T}}\mathbf{v} - \mathbf{c}^{\mathrm{T}}\mathbf{h} - \mathbf{v}^{\mathrm{T}}\mathbf{W}\mathbf{h}, \tag{1.15}$$

where $\mathbf{W}$ is the matrix of visible/hidden connection weights, $\mathbf{b}$ is a visible unit bias (column) vector, and $\mathbf{c}$ is a hidden unit bias (column) vector. The probability of any particular setting of the visible and hidden units is given in terms of the energy of that configuration by:

$$P(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z}, \tag{1.16}$$

where the normalization factor $Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$ is known as the partition function.

The lack of direct connections within each layer enables us to derive simple exact expressions for $P(\mathbf{v}|\mathbf{h})$ and $P(\mathbf{h}|\mathbf{v})$, since the visible units are conditionally independent given the hidden unit states and vice versa. For the case of binary units (see Bengio [2009] for a derivation) the conditionals are given by:

$$P(\mathbf{h}|\mathbf{v}) = \sigma(\mathbf{c} + \mathbf{W}^{\mathrm{T}}\mathbf{v}) \tag{1.17}$$

$$P(\mathbf{v}|\mathbf{h}) = \sigma(\mathbf{b} + \mathbf{W}\mathbf{h}), \tag{1.18}$$

where σ denotes the element-wise logistic sigmoid, $\sigma(x) = (1 + e^{-x})^{-1}$. See Welling et al. [2005] for generalizations of RBMs to exponential family units beyond the Bernoulli distribution.

To aid in our understanding of the RBM as a probabilistic model, we can examine the marginal distribution over $\mathbf{v}$. We may write the RBM energy function (Eq. 1.15) as a sum of terms that each

depend on at most one hidden unit, which yields:

$$E(\mathbf{v}, \mathbf{h}) = -\beta(\mathbf{v}) + \sum_j \gamma_j(\mathbf{v}, h_j),$$

where $\beta(\mathbf{v}) = \mathbf{b}^{\mathrm{T}}\mathbf{v}$ and $\gamma_j(\mathbf{v}, h_j) = -(c_j + \mathbf{v}^{\mathrm{T}}\mathbf{W}_{*,j})h_j$, with $\mathbf{W}_{*,j}$ denoting the jth column of $\mathbf{W}$. At this time it is also useful to introduce the free energy, $F(\mathbf{v})$, which is defined as:

$$F(\mathbf{v}) = -\log \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}. \tag{1.19}$$

In terms of $\beta(\mathbf{v})$ and $\gamma_j(\mathbf{v}, h_j)$, the free energy is (assuming binary hidden units):

$$F(\mathbf{v}) = -\beta(\mathbf{v}) - \sum_j \log \left(\sum_{h_j \in \{0,1\}} e^{-\gamma_j(\mathbf{v},h_j)} \right). \tag{1.20}$$

The second form is, of course, much more convenient computationally since it does not involve a summation with a number of terms that is exponential in the number of hidden units. The marginal distribution over $\mathbf{v}$ (as, once again, is derived in detail in Bengio [2009]) is:

$$\begin{aligned} P(\mathbf{v}) &= \frac{e^{-F(\mathbf{v})}}{Z} \\ &= \frac{e^{-\beta(\mathbf{v})}}{Z} \prod_j \sum_{h_j \in \{0,1\}} e^{-\gamma_j(\mathbf{v},h_j)}. \end{aligned} \tag{1.21}$$

To train an RBM, ideally we would follow the gradient of the log likelihood of the data. For an arbitrary model parameter θ, this yields the following update rule based on expectations of the energy gradients over the data distribution and the model distribution:

$$\Delta\theta \propto \left\langle \frac{\partial E}{\partial \theta} \right\rangle_{\text{data}} - \left\langle \frac{\partial E}{\partial \theta} \right\rangle_{\text{model}}.$$

In particular, the maximum likelihood update rule for the visible-hidden weights is:

$$\Delta w_{ij} \propto \langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}}. \tag{1.22}$$

The expectation $\langle v_i h_j \rangle_{\text{data}}$ is the frequency with which the visible unit v_i and the hidden unit h_j are on together in the training set and $\langle v_i h_j \rangle_{\text{model}}$ is that same expectation under the distribution defined by the model. Unfortunately, the term $\langle . \rangle_{\text{model}}$ takes exponential time to compute exactly so we are forced to use an approximation. Since RBMs are in the intersection between Boltzmann machines and product of experts models, they can be trained using contrastive divergence (CD) as described in Hinton [2002]. The new update rule becomes:

$$\Delta w_{ij} \propto \langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_1, \tag{1.23}$$

where $\langle . \rangle_1$ represents the expectation with respect to the distribution of samples from running the Gibbs sampler (defined using equations 1.17 and 1.18) initialized at the data for one full step.

Nair and Hinton [2010] describe how to construct and train RBMs with rectified linear units (ReLUs) in the visible layer, hidden layer, or both. Since ReLUs are not exponential family units, contrastive

divergence training might in general have difficulties. However, we can view noisy ReLUs as approximations of a large number of sigmoid units with shifted biases and tied weights and get the familiar contrastive divergence training updates, even though technically CD does not apply. In practice, CD1 still works in spite of the ReLU approximation. The stochastic NReLU activation function used in Nair and Hinton [2010] is given by:

$$\max(0, z + \nu), \tag{1.24}$$

where $\nu \sim \mathcal{N}(0, \sigma(z))$, however, in some experiments in later chapters, we use the decidedly non-standard $\max(0, \max(0, z) + \nu)$.

Although interesting models in their own right, RBMs are used in the original unsupervised pre-training algorithm for deep belief nets [Hinton et al., 2006] and we make use of this algorithm to pre-train deep neural networks in later chapters. Learning is difficult in densely connected, directed belief nets that have many hidden layers because it is difficult to infer the posterior distribution over the hidden variables given a data vector, due to the explaining away phenomenon. Markov chain Monte Carlo methods [Neal, 1992] can be used to sample from the posterior, but they are typically very time-consuming. In Hinton et al. [2006] complementary priors were used to eliminate the explaining away effects producing a training procedure which is equivalent to training a sequence of restricted Boltzmann machines and stacking their weight matrices in multiple layers.

The stacking procedure works as follows. Once an RBM has been trained on data, inferring the hidden unit activation probabilities given a data vector simply requires an application of equation 1.17. Thus we can use an RBM as a feature extractor. Since the RBM has been trained to reconstruct the data well, the hidden unit activations will retain much of the information present in the data and pick up (possibly higher-order) correlations between different data dimensions that exist in the training set. Once we have used one RBM as a feature extractor we can, if desired, train an additional RBM that treats the hidden activations of the first RBM as data to model. After training a sequence of k RBMs, we have a $(k-1)$-layer directed net with an RBM modeling the prior over the top level of hidden variables. Equation 1.17 is identical to the equation for computing the activity of the logistic hidden units in a hidden layer of a standard feed-forward neural network given the activation of the units in the layer below. This fact allows us to use the weights obtained by the greedy layer-by-layer training procedure described for stacking RBMs, above, and use them to initialize the weights of a deep neural network. Once we add an output layer to the pre-trained neural network, we can discriminatively fine-tune the weights of this neural net using any variant of standard backpropagation we wish.

Almost tautologically, for any problem where features useful in modeling $P(x)$ are useful when modeling $P(y|x)$, unsupervised pre-training might be useful. Pre-training also initializes the weights of a deep neural net such that the initial weights are well scaled and produce non-degenerate feature detectors capable of discriminating between different training cases efficiently. For very large nets trained on very small supervised tasks with only a small number of bits of information in each label, unsupervised pre-training uses the information necessary to encode the input to help constrain the weights, which can have a regularization effect when combined with early stopping during the discriminative training phase. For training deep autoencoders and deep generative models in particular, the greedy reconstruction objective of pre-training can be especially helpful since, intuitively, to train the kth layer it is important to have most of the important information about the input already in present the layer below.

Chapter 2

Deep acoustic models

Acoustic modeling was an important early successful application of large, deep neural networks, in part because of the massive quantities of transcribed speech data that have been collected. Acoustic modeling has taught us a lot about how to best train large neural networks when (at least weakly) labeled data are plentiful. The speech recognition task has aspects that are similar to vision problems as well as a large natural language processing component so there is some hope that successful recipes for speech recognition with neural nets might be relevant for vision and more general text processing. This chapter describes an improvement to the basic deep neural net acoustic model and also provides a brief review of the deep neural net approach to large vocabulary speech recognition (LVSR).

2.1 The speech recognition problem

Automatic speech recognition (ASR) is the task of producing a text transcription of the audio signal from a speaker. For humans, speech recognition and transcription is mostly effortless, but effective automatic systems have taken decades of engineering effort and state of the art systems are far from perfect in unrestricted, realistic conditions. Large vocabulary speech recognition (LVSR) involves a large or open vocabulary and we will also take it to imply a speaker independent recognizer. The most general LVSR systems may also need to operate on spontaneous as well as read speech with audio data from an uncontrolled recording environment corrupted by both unstructured and structured noise. LVSR has obvious applications in automated transcription and dictation as well as applications to voice interfaces, voice search, spoken dialogue systems, audio information retrieval, and many other systems that work with speech data. There are three main sources of acoustic variability in speech recognition data. First, there is variation in what is said by the speaker. For open vocabulary systems, there is no way to collect training data for every possible utterance or even every possible word. Second, there is variation due to differences between speakers. Different people have different voices and accents and ways of speaking. Third, there is variation in noise conditions. Anything in the acoustic data that is not the signal is noise and thus noise can include background sounds, crosstalk, microphone specific artifacts, and other effects.

The statistical formulation of the speech recognition problem begins with a sequence of τ vectors of acoustic information $\mathbf{X} = \mathbf{x}_1 \dots \mathbf{x}_\tau$ that we assume encodes a sequence of T words $\mathbf{w} = w_1 \dots w_t$. This formulation of the task involves a search problem and a modeling or learning problem. Specfically, our

goal is to find the best transcription hypothesis $\mathbf{w}$ according to some learned scoring function $s(\mathbf{w}, \mathbf{X})$:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}}\{s(\mathbf{w}, \mathbf{X})\}. \tag{2.1}$$

The basic speech recognition pipeline [Gales and Young, 2008, Young, 2001] splits the recognition task into three subsystems: the acoustic model (AM), the language model (LM), and the decoder. The language model (LM) contains the parts of the model that do not need the acoustic information; it scores a transcription hypothesis, hopefully assigning better scores to more *a priori* plausible transcriptions. The acoustic model scores the match between a candidate transcription and the acoustic input. Finally, the decoder approximately searches the space of possible transcriptions to find the hypothesis with the best combined score from the AM and the LM. If we take the score function from equation 2.1 to be based on a probabilistic model $P(\mathbf{w}|\mathbf{X})$, then the acoustic model and the language model fall out after we rewrite the problem using Bayes's rule, giving:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}}\{\log P(\mathbf{w}|\mathbf{X})\} = \underset{\mathbf{w}}{\operatorname{argmax}}\left\{\log P(\mathbf{w}) + \log P(\mathbf{X}|\mathbf{w})\right\}. \tag{2.2}$$

Since the decoder searches over word sequences $\mathbf{w}$ given a fixed sequence of acoustic vectors $\mathbf{X}$, terms independent of $\mathbf{w}$ in $s(\mathbf{w}, \mathbf{X})$ can be neglected, giving us a lot of flexibility in how we define the acoustic model (for example there is no need for it to be properly normalized). In practice, in fact, we generally learn weights for the LM and AM terms [Gales and Young, 2008]. Perhaps the most general way of viewing the acoustic model and language model is to think of them simply as two separate models of $s(\mathbf{w}, \mathbf{X})$ that we average together, one just happens to ignore $\mathbf{X}$ and the other usually focusses on shorter range agreement.

Beyond the three primary subsystems, most state of the art recognizers also make use of a fourth essential component that interacts with the acoustic model and the decoder: the pronunciation dictionary. A pronunciation dictionary allows the acoustic model to operate on units of sound smaller than individual words which in turn provides a way to recognize utterances containing words that never appear in the acoustic training data but appear in the pronunciation dictionary. The dictionary maps words to one or more sequences of phonemes or even to simple finite automata generating allowed pronunciations. The dictionary constrains recognition hypotheses and high quality pronunciation dictionaries are crucial to obtaining good results in LVSR.

Traditional LVSR recognizers are based on hidden Markov models (HMMs) with Gaussian mixture model (GMM) emission distributions, n-gram language models, and use beam search for decoding. Even systems that go beyond this classic architecture and replace some of the components often need to start with it in the first phase of the training and recognition pipelines. For a more detailed description of traditional GMM-HMM speech recognition see Gales and Young [2008], but I will outline the basics below.

The basic modeling unit of the acoustic model that implements the likelihood $P(\mathbf{X}|\mathbf{w})$ is the phone. Phones are the basic units of sound acoustically realized in a language. Phonemes, on the other hand, are the cognitive categories for primitive sounds in a language that serve to distinguish words from each other (and thus must be perceptually distinct). For example in English, the [$\mathrm{p^h}$] phone (aspirated voiceless plosive) in the word *pin* and the phone [p] (unaspirated voiceless plosive) in the word *spin* are both allophones of the phoneme /p/ since aspiration of /p/, although an acoustic difference, does not distinguish words from one another in English. Since the basic unit of the transcript is the word and

not the phone, we actually need word models. Given a model for each phone, we can form word models by concatenating phone models based on the pronunciation dictionary entries for the word in question. We will now think of a single word w from some word sequence $\mathbf{w}$ as made up of sequences of base phones $q_1, \ldots, q_{K_w}$ with one sequence for each pronunciation. Now if $\mathbf{Q}$ is a particular sequence of word pronunciations valid for word sequence $\mathbf{w}$, then the acoustic model likelihood becomes:

$$P(\mathbf{X}|\mathbf{w}) = \sum_{\mathbf{Q}} P(\mathbf{X}|\mathbf{Q})P(\mathbf{Q}|\mathbf{w}). \tag{2.3}$$

Each base phone is a simple three state HMM with only left-to-right transitions and self transitions allowed between hidden states and this HMM emits an acoustic feature vector at each time step. As mentioned before, we can stitch together these HMMs to form a larger one for a whole pronunciation and a whole word and even, with the help of the language model to define inter-word transitions, a whole word word sequence. This sort of monophone/context-independent acoustic model is sometimes known as a *beads-on-a-string* model. Practical systems use context dependent phones and use a new model for each triphone or even for each quinphone. Since there are so many possible triphones and many might not be ever realized in the data, triphone models cluster the full set of logical triphone HMM states and tie the parameters for states within each cluster to yield a far smaller set of tied triphone states.

For our purposes, the acoustic data are frames of log spectral information for a short temporal window with a perceptually warped frequency scale. These input feature vectors typically get augmented with their first and second temporal derivatives computed using smoothed differences of neighboring frames. Traditional acoustic models use very large mixtures of diagonal covariance Gaussians to model the acoustic frames. Adding deltas helps to compensate for some of the inappropriate conditional independence assumptions of GMMs with diagonal covariance Gaussians and other model deficiencies.

GMM-HMM acoustic models are trained with the expectation-maximization (EM) algorithm using a procedure that trains progressively more sophisticated models using an initial alignment from the previous model. For example, the initial model might be a monophone GMM-HMM with only a single Gaussian per phone. Then in the next step, each monophone model could be split into distinct triphones initialized to the monophone parameters and more iterations of EM could be performed, followed by the triphone state clustering. The model described above is purely generative and far simpler than GMM-HMM acoustic models used in practice. The more sophisticated GMM-HMM systems used in practice use acoustic feature projections, vocal tract length normalization, semi-tied covariance Gaussians instead of only diagonal Gaussians, discriminative training, speaker-adaptive training, and a host of other refinements. The training pipelines for the strongest GMM-HMM acoustic models can require dozens of steps to implement all the refinements, including multiple rounds of speaker-independent context-dependent training, multiple rounds of discriminative training, and more. The resulting models can have thousands of tied context-dependent states and hundreds of thousands of independently parameterized Gaussians

The language model in traditional systems is a large, smoothed n-gram model. Using a Markov language model makes it much easier to perform decoding and enables efficient word lattices. When more sophisticated language models are used they generally run only after a first decoding pass is complete.

2.1.1 Deficiencies of the classical approach to LVSR

As outlined above, the classical LVSR approach uses GMM-HMM acoustic models, n-gram language models, and beam search for decoding. The three biggest problems with this approach are GMMs, n-gram language models, and HMMs. This chapter will focus on replacing the GMMs, but a brief note on the problems with n-gram LMs and HMMs more generally is in order. Since n-gram language models are essentially conditional probability tables, the number of parameters required grows exponentially in the length of the context. Smoothing methods help but still do not allow n-gram models to use large or unbounded contexts.[1] Equally importantly, discrete probability tables for the conditional distribution of the next word given previous words do not generalize across similar words, which is another source of statistical inefficiency.

Using HMMs simplifies decoding substantially, but as a generative model of speech they leave much to be desired. HMMs assume that (overlapping!) frames of audio are conditionally independent given the hidden state sequence, but knowing the hidden state only provides a number of bits of information about the past logarithmic in the number of possible hidden states. Indeed GMM-HMM models in speech are more GMM than HMM because in many cases the transition matrix parameters for the HMM do not matter much once the pronunciation dictionary and language model transitions have been incorporated into the architecture. Nevertheless, the advantages of using larger temporal contexts and richer sequence models show that the sequential properties of speech data do actually matter, as we would believe a priori based on our understanding of the continuous articulatory dynamics of speech production.

The GMMs used in speech recognition end up being very large and can have hundreds of thousands of Gaussians. Extremely large mixture models are necessary to handle the acoustic variability in real speech data. However, large numbers of independently parameterized Gaussians cause severe data sparsity and statistical efficiency issues. Large mixtures of Gaussians tend to partition the input space into regions where only a single Gaussian dominates because a given acoustic observation vector only has a single latent cause in the model. Since Gaussians in the model only have their parameters constrained by data cases they are responsible for generating, only a few frames of training data will constrain the parameters of each Gaussian. Another related representational inefficiency is that specifying which of K Gaussians generated a data case only provides $\log K$ bits of information about the data case since all the latent variables in a GMM are used to encode which component of the mixture is active (in general mixture models the individual components might have a lot of latent variables of their own). Hinton [2002] points out that in high dimensional spaces where the data distribution occupies a much lower dimensional manifold, mixture models of simple component distributions can become highly inefficient representationally because the posterior distribution can never be sharper than an individual component distribution. Using diagonal Gaussians only exacerbates these problems.[2] GMM-HMM LVSR systems go to great lengths to ensure that acoustic frames are low dimensional, but systems that do not depend on GMMs can easily benefit from higher frequency resolution and more temporal context. Speech input features have been designed to compensate as much as possible for the deficiencies of GMM-HMM systems and more representationally efficient and expressive acoustic models have no need of the cepstral transform or reducing the frequency resolution [Mohamed et al., 2012]. As we will see in the next section,

[1] The sequence memoizer [Wood et al., 2011], the apotheosis of smoothed n-gram models, is one exception, but it is a non-parametric Bayesian model that still has very high memory requirements that increase the more data we wish to use.

[2] Dahl et al. [2010] notes that "the diagonal covariance approximation typically employed for GMM-based acoustic models is symptomatic of, but distinct from, the general representational inefficiencies that tend to crop up in [these models]" and also describes some analogous issues with Gaussian-Bernoulli RBMs and how to resolve them.

simply replacing GMMs with deep neural networks (DNNs) to create DNN-HMM hybrid models has caused massive improvements in acoustic models for LVSR, even though DNN-HMMs do nothing to address the issues with the language model or the HMM.

Unlike GMMs, HMMs are much harder to replace in the LVSR pipeline because of how strongly the HMM assumption has been built into the training and decoding procedures. Non-Markov language models are also difficult to use in the first decoding pass, but they can be incorporated in a lattice reranking framework. Thus replacing GMMs is in a sense low hanging fruit and one of the easiest ways to get dramatic improvements in speech recognition accuracy over traditional systems.

2.2 The first successful deep neural net acoustic models

Neural nets have a long history in acoustic modeling both as ways to learn extra features for a GMM-HMM system [Hermansky et al., 2000] and as a replacement for GMMs in a neural network HMM hybrid system [Bourlard and Morgan, 1994]. Through much of the history of neural nets in acoustic modeling, the deep and wide models trained today were not feasible computationally and the easy parallelization of the EM algorithm was an important advantage for GMM-based systems. Although early hybrid systems worked well, they only recently worked well enough compared to speaker-adapted, discriminatively trained GMM-HMM systems to become the standard approach for acoustic modeling. Almost all hybrid systems before 2009 had only a single layer of hidden units, although there are some exceptions. For instance Albesano et al. [2000] used a neural net with two hidden layers to predict diphone-like units. In general, researchers using neural nets for acoustic modeling focussed on the size of the single hidden layer as the primary avenue for increasing model capacity [Ellis and Morgan, 1999]. Not all network architectures for a given problem use additional parameters equally efficiently and only when we are willing to try networks with multiple hidden layers can we find the networks with the best performance given a fixed budget of weights.

Inspired by the success of deep belief networks in Hinton et al. [2006], in 2009 Mohamed et al. (see Mohamed et al. [2012] for the definitive version) tried using pre-trained, deep neural networks as part of a hybrid monophone DNN-HMM model on TIMIT, a small scale speech task. The record-breaking results presented in Mohamed et al. [2012] were the first success for pre-trained DNN-HMMs on acoustic modeling and experiments with networks of varying depths showed a clear advantage for deeper models. For example, a three hidden layer net with 512 units per layer had lower error than a two hidden layer net of the same width which had lower error than a one hidden layer net of the same width. Even when the shallow nets had far more tunable parameters, the deep nets produced better results: a single hidden layer net with more than three times the parameters of a two hidden layer net nonetheless achieved worse results. As exciting as these results on TIMIT were, similar successes on LVSR problems would be necessary for DNN-HMMs to be widely used in the speech recognition community.

Dahl et al. [2012a] (results first presented in Dahl et al. [2011]) described the first large vocabulary success of pre-trained, DNN-HMM acoustic models and demonstrated a dramatic improvement over a strong discriminatively trained GMM-HMM baseline system. The basic DNN LVSR recipe introduced in Dahl et al. [2012a] has since become the standard starting point for new work improving DNN-HMM acoustic models for LVSR. See Hinton et al. [2012a] for a review of systems from four different research groups using similar recipes. In general, hybrid artificial neural net HMM systems (ANN-HMMs) use a forced alignment with a baseline model to create labeled training data. For LVSR, using the tied HMM

states from a strong GMM-HMM baseline works quite well and makes combining the DNN with the HMM straightforward once the DNN output probabilities, $P(\text{tied state}|\text{acoustic input})$, are scaled by the tied state occupancy probabilities in the aligned training data. Strong DNN-HMMs for LVSR tasks use four or more large hidden layers (low thousands of units) and typically use some form of pre-training, although with very large datasets training shallower supervised nets and using their weights to initialize the first layers of the final net suffices to get training off the ground quickly and avoids poorly scaled initial weights.

In the next section, we will discuss an improvement upon the basic DNN-HMM LVSR of Dahl et al. [2012a] that was first described in Dahl et al. [2013]. The improvements are another example of successful deep neural network techniques that first appeared promising for vision tasks being useful on other problems.

2.3 Improving deep neural networks for LVSR using rectified linear units and dropout[3]

As we described above, up until a few years ago, most state of the art speech recognition systems were based on hidden Markov models (HMMs) that used mixtures of Gaussians to model the HMM emission distributions. However, Mohamed et al. [2012] showed that hybrid acoustic models that replaced Gaussian mixture models (GMMs) with pre-trained, deep neural networks (DNNs) could drastically improve performance on a small-scale phone recognition task, results that were later extended to a large vocabulary voice search task in Dahl et al. [2012a]. Since then, several groups have demonstrated dramatic gains from using deep neural network acoustic models on large vocabulary continuous speech recognition (LVCSR) tasks [Hinton et al., 2012a].

Even with unsupervised pre-training and large training sets, wide and deep neural networks are still vulnerable to overfitting. Dropout is a technique for avoiding overfitting in neural networks that has been highly effective on non-speech tasks and the small-scale TIMIT phone recognition task [Hinton et al., 2012b], although it can increase training time. Rectified linear units (ReLUs) in our experience achieve the same training error faster (as Krizhevsky et al. [2012] also found) than sigmoid units and, although sometimes superior to sigmoid units, can often overfit more easily. Rectified linear units are thus a natural choice to combine with dropout for LVCSR.

Below we explore the behavior of deep neural nets using ReLUs and dropout on a 50-hour broadcast news (BN) task [Kingsbury et al., 2012], focussing our experiments on using dropout during the frame level training phase. We show that the modified deep neural networks (DNNs) using ReLUs and dropout provide a 4.2% relative error reduction over a standard pre-trained DNN and a 14.4% relative improvement over a strong GMM-HMM system.

2.3.1 Dropout

Dropout is a powerful technique introduced in Hinton et al. [2012b] for improving the generalization error of large neural networks. In Krizhevsky et al. [2012], dropout yielded gains on a highly competitive computer vision benchmark. Although Hinton et al. [2012b] applied dropout to the 3 hour TIMIT phone recognition task, we are not aware of any application of dropout to LVSR.

[3] A substantial portion of this section is reprinted, with permission, from Dahl et al. [2013], copyright 2013 IEEE.

Dropout discourages brittle co-adaptations of hidden unit feature detectors by adding a particular type of noise to the hidden unit activations during the forward pass of training. The noise zeros, or "drops out," a fixed fraction of the activations of the neurons in a given layer, similar to the type of noise recommended for the input of denoising auto-encoders in Vincent et al. [2008]. However, unlike in denoising autoencoder pre-training, dropout is used in all hidden and input layers and occurs during supervised training with end-to-end backpropagation. Furthermore, since at test time dropout does not occur and there may not be additional training with it disabled, at training time we multiply the net input from the layer below by a factor of $\frac{1}{1-r}$, where r is the dropout probability for units in the layer below. Specifically, to compute the activation $\mathbf{y}_t$ of the tth layer of the net during forward propagation, we use:

$$\mathbf{y}_t = f\left(\frac{1}{1-r}\mathbf{y}_{t-1} * \mathbf{m}\mathbf{W} + \mathbf{b}\right),$$

where f is the activation function for the tth layer, $\mathbf{W}$ and $\mathbf{b}$ are respectively the weights and biases for the layer, $*$ denotes element-wise multiplication, and $\mathbf{m}$ is a binary mask with entries drawn i.i.d. from $Bernoulli(1-r)$ indicating which activations are not dropped out. Dropout can also be viewed as training a very large number of different neural nets with different connectivity patterns and tied weights for all units that are not dropped out. By randomly sampling which units remain anew for each training case, this averaging can be performed in a particularly efficient way. The factor of $\frac{1}{1-r}$ used during training ensures that at test time, when all units get used, the correct total input will reach each layer.

Although dropout guards against overfitting and produces far more robust models, adding so much noise during training slows down learning, in our experience by about a factor of two. Since overfitting is much easier to avoid, larger models should be used to obtain the best results which also can slow down training although it enables better results.

2.3.2 Rectified Linear Units

In this chapter we use the term rectified linear unit (ReLU) to refer to unit in a neural net that use the activation function $\max(0, x)$. In computer vision research, ReLUs have been used both as activation functions in more standard neural nets and as units in restricted Boltzmann machines (RBMs), where they must be dealt with using approximations since they invalidate the probabilistic model [Jarrett et al., 2009, Nair and Hinton, 2010, Krizhevsky et al., 2012]. To our knowledge, the only use of these units in speech before the work we describe presently was in Jaitly and Hinton [2011] where Gaussian-ReLU RBMs were used to learn features from raw, unlabeled acoustic data to later use in phone recognition experiments on TIMIT. Our work differs from Jaitly and Hinton [2011] in that our focus is on LVSR, we exploit the synergistic effects of combining ReLUs with dropout, and we experiment with ReLUs at all hidden layers in our network instead of only using them as part of a stand-alone RBM feature extraction module.

Although ReLUs are quite simple to incorporate into a standard feed-forward neural net, in order to maintain the RBM probabilistic model we can view a single unit with the $\max(0, x)$ nonlinearity as an approximation to a set of replicated binary units with tied weights and shifted biases [Nair and Hinton, 2010]. As in Nair and Hinton [2010], we perform RBM pre-training with ReLU units by adding Gaussian noise with sigmoid variance (called an NReLU unit in Nair and Hinton [2010]). Normally this would mean that during pre-training we sample a hidden unit state $y = \max(0, x + \epsilon)$, where x is the net input to the

hidden unit and ϵ is drawn from a normal distribution with mean zero and variance $\frac{1}{1+e^{-x}}$. However, our code actually used the non-standard version described in section 1.4, $y = \max(0, \max(0, x) + \epsilon)$, which makes it more probable that "off" units will get randomly activated, but in general should have quite similar behavior. During fine-tuning, we simply use the $\max(0, x)$ nonlinearity.

2.3.3 Experiments

Baseline

We performed all experiments on 50 hours of English Broadcast News [Kingsbury et al., 2012]. We used the DARPA EARS `rt03` set for development/validation and performed final testing on the DARPA EARS `dev04f` evaluation set.

The GMM system is trained using the recipe from Soltau et al. [2010], which is briefly described below. The raw acoustic features are 19-dimensional PLP features with speaker-based mean, variance, and vocal tract length normalization. Temporal context is included by splicing 9 successive frames of PLP features into supervectors, then projecting to 40 dimensions using linear discriminant analysis (LDA). The feature space is further diagonalized using a global semi-tied covariance (STC) transform. The GMMs are speaker-adaptively trained, with a feature-space maximum likelihood linear regression (fMLLR) transform estimated per speaker in training and testing. Following maximum-likelihood training of the GMMs, feature-space discriminative training (fBMMI) and model-space discriminative training are done using the boosted maximum mutual information (BMMI) criterion. At test time, unsupervised adaptation using regression tree MLLR is performed. The GMMs use 2,203 quinphone states and 150K diagonal-covariance Gaussians.

The pre-trained deep neural net (DNN) systems use the same fMLLR features and 2,203 quinphone states as the GMM system described above, with an 11-frame context (± 5) around the current frame. The DNN consists of six hidden layers, each containing 1,024 sigmoidal units, and a softmax layer with 2,203 output targets. These features and DNN architecture were considered optimal in Sainath et al. [2011] for this Broadcast News task. The DNN training begins with greedy, layerwise, generative pre-training and then continues with cross-entropy discriminative training, followed by Hessian-free sequence-training [Kingsbury et al., 2012]. To enable fair comparisons with the large nets we tried in this work, we also trained versions of the DNN baseline with 2048 and 3072 hidden units per layer, with 2048 units producing the best results for the basic pre-trained deep neural net setup.

Interactions Between Hessian Free Optimization and Dropout

Although dropout is trivial to incorporate into minibatched stochastic gradient descent (SGD), the best way of adding it to 2nd order optimization methods was not clear when our experiments were performed. Although all frame level training we performed used minibatched stochastic gradient descent, we decided to use the Hessian-free optimizer (HF) of Kingsbury et al. [2012] for all full sequence training in this work since it has large parallelization advantages over the SGD sequence training system we had available. However, one consequence of this choice is the undesirable interaction between dropout and HF. To find each search direction, HF uses the conjugate gradient (CG) algorithm to iteratively optimizes a local quadratic approximation to the objective function. CG depends on having reliable curvature and gradient information computed from large batches since it makes such strong use of it. Having a single fixed objective function during CG is even more important than the accuracy of the estimates. Dropout

adds too much noise to the gradient for CG to function properly if the randomization occurs anew on each gradient evaluation. For our experiments we simply disabled dropout during full sequence training. Later, in section 2.3.4, we will discuss the solution to this problem reached (after our experiments were concluded) by Sainath et al. [2013].

Bayesian Optimization and Training Details

We used the Bayesian optimization method described in Snoek et al. [2012] to optimize training metaparameters and model hyperparameters on the validation set. Bayesian optimization is a gradient-free global optimization method that in the case of Snoek et al. models the unknown function from hyperparameters to validation word error rate as being drawn from a Gaussian process prior. As experiments complete, the algorithm updates the posterior distribution for the function and proposes new experiments to run that optimize the integrated (over Gaussian process covariance function hyperparameters) expected improvement below the current minimum error achieved by any experiment so far. We used public code released by the authors of Snoek et al. [2012] and enabled a constraint finding option [Gelbart et al., 2014] to deal with divergent training runs. The constrained Bayesian optimization version has a separate Gaussian process model to classify points as feasible or not and combines this model with the Gaussian process regression model when suggesting new jobs.

All frame level training used the cross entropy criterion and minibatched stochastic gradient descent. Frame level training used gnumpy [Tieleman, 2010] and CUDAMat [Mnih, 2009] to train using a GPU. We allowed Bayesian optimization to set twelve hyperparameters for frame level supervised training: seven dropout rates $r_0 \dots r_6$ (one per non-output layer with $0 \leq r_i \leq 0.5$), a single global initial learning rate $\alpha \in [4 \times 10^{-4}, 4 \times 10^{-1}]$, a single global momentum $\mu \in [0, 0.95]$, a number of times to anneal the learning rate before stopping $a \in [3, 11]$, a minimum improvement $m \in [0, 0.025]$ in validation cross entropy required to not anneal the learning rate, and a factor γ to divide the learning rates by when annealing them. We optimized $\log_2 \gamma$ on $[0.5, 2.5]$. Every half-epoch we computed validation cross entropy and tested to see if the learning rates should be annealed according to the schedule.

Since we did not rerun pre-training for each Bayesian optimization job, we set the pre-training metaparameters by hand. Training ReLU RBMs with CD1 can be more difficult than training binary RBMs. We found it useful to only do 2.5 epochs of pre-training for each layer compared to more than twice that for sigmoid units.

The Bayesian optimization software we used is sufficiently effective to remove most of the human judgement from hyperparameter optimization. When there are only a few hyperparameters, it isn't hard to select reasonable values by hand, but optimizing more hyperparameters facilitates fairer comparisons between established models that researchers have experience tuning and new models they have less intuition about. However, there are still two areas that require human intervention: defining the search space and deciding when to terminate Bayesian optimization. Additionally, Bayesian optimization can be accelerated by seeding the optimization with jobs with hyperparameters selected using results from related hyperparameter search problems. For example, if one trains a neural net with a thousand units per hidden layer and finds good settings for its hyperparameters, these settings can initialize a Bayesian optimization run that searches for hyperparameters for a related network using two thousand hidden units per layer. We selected seed jobs for our larger nets based on hyperparameter optimizations for smaller networks.

Model	rt03	dev04f
ReLUs, 3k, dropout, CE	10.7	18.5
no PT, ReLUs, 3k, dropout, CE	11.0	18.9
no PT, ReLUs, 2k, dropout, CE	11.2	19.0
Sigmoids, 3k, CE	11.3	19.4
Sigmoids, 2k, CE	11.1	19.4
Sigmoids, 1k, CE	11.6	19.9

Table 2.1: Results without full sequence training (with cross entropy a.k.a CE). All models used pre-training unless marked "no PT" and used 1k, 2k, or 3k hidden units per layer.

Model	rt03	dev04f
GMM baseline	10.8	18.8
ReLUs, 3k, dropout, sMBR	9.6	16.1
Sigmoids, 2k, sMBR	9.6	16.8

Table 2.2: Results with full sequence training

Results

Table 2.1 shows word error rate results on the validation set (rt03) and test set (dev04f) before full sequence training was performed. Each model listed is the one that performed best on the validation data given the constraints in the model description. As shown in the table, the single best model uses ReLUs, dropout, and relatively large hidden layers with weights initialized using unsupervised RBM pre-training. This model achieved a word error rate of 18.5 on the test set, beating other deep neural net models and the strong discriminatively trained GMM baseline, even before any full sequence training of the neural net models.

To confirm that the unsupervised pre-training was worthwhile even with the approximations required for ReLU RBMs and the smaller number of pre-training epochs used relative to binary RBMs, we ablated the pre-training and optimized all remaining hyperparameters. Even without pre-training, the ReLU nets using dropout had about the same test set performance as the best baseline and somewhat better performance than the best pre-trained sigmoid nets. For the 2k unit per hidden layer, un-pre-trained ReLU nets, we also added additional hyperparameters (layer specific learning rates and weight costs) and ran extensive Bayesian optimization experiments to confirm that we could not find a configuration with lower validation errors even in the larger hyperparameter search space. Since these experiments alone involved over 5000 GPU-hours of computation, they were not feasible to perform on the nets with 3k units per layer, although for those nets we performed our usual hyperparameter optimization in the search space described in section 2.3.3.

The Bayesian optimization procedure rarely turned off dropout for ReLU nets and typically kept dropout off for sigmoid nets, even with 3k units per layer. This indicates that the Bayesian optimization procedure learned that dropout wasn't helpful for sigmoid nets of the sizes we trained. In general, ReLUs and dropout seem to work quite well together. Without dropout, ReLU nets can overfit very quickly compared to sigmoid nets. Early stopping is often able to avoid the worst overfitting in sigmoid nets on LVSR tasks, but for ReLU nets, dropout combined with early stopping worked much better.

Table 2.2 shows results for the best ReLU net and the best sigmoid net was trained for an equal amount of wall clock time using 11 and 16 HF iterations of full sequence training respectively. We trained the ReLU and sigmoid nets for an equal amount of time to make a fair comparison in the absence of

dropout during full sequence training and because of the expense of full sequence training with nets of this size. We expect that the very best results will necessitate the use of dropout during full sequence training as well (in the next section we will describe subsequence experiments from Sainath et al. [2013] that confirm this belief). The ReLU net shows a more modest gain over the sigmoid net after full sequence training than was evident after frame level training, possibly because the ReLU net no longer gains the benefit of training with dropout during the full sequence training. Full sequence training for nets as large as these is very time consuming and will most likely slow down further if HF is modified to support dropout and especially if gradient descent sequence training gets used instead.

2.3.4 Subsequent results using HF with dropout and ReLUs by Sainath et al. [2013]

The primary deficiency in the experiments we were able to perform was that full sequence HF training did not use dropout. Performing HF without dropout as we did erased some of the gains from using ReLUs with dropout we saw after frame level training phase. Fortunately, Sainath et al. [2013] presents results that *do* include dropout during HF using the same pipeline and training set we used in our experiments. To combine dropout with HF, Sainath et al. [2013] only refreshes the dropout mask between HF outer-loop iterations (calls to CG) and not between individual CG iterations. Different utterances within a batch can still have different dropout masks as normal. Fixing the dropout mask during a single run of CG preserves the conjugacy of the search directions. Sainath et al. [2013] find that using dropout and ReLUs in a deep convolutional neural net acoustic model provides gains in word error rate that persists after sequence training when HF uses dropout. Their result confirms our belief that ReLUs and dropout do help when incorporated into the current state of the art acoustic models.

2.4 Will the speech recognition DNN recipe work on other problems?

In the previous section, motivated by successes in computer vision, we explored using rectified linear units and dropout in deep neural nets. ReLUs and dropout yielded word error rate improvements relative to a state of the art baseline and, with the help of Bayesian optimization software, we were able to obtain these improvements without much of the hand tuning typically used to obtain the very best deep neural net results. These Bayesian optimization experiments also gave us evidence that ReLUs and dropout have synergistic effects and we recommend that they be used together. Sainath et al. [2013] later confirmed our conclusions and created IBM's strongest acoustic model yet by incorporating ReLUs and dropout into a state of the art deep convolutional neural network acoustic model.

Acoustic modeling, along with object classification and detection, are some of the more mature applications of deep neural networks using ReLUs and dropout. The basic recipe of training a very large and very deep neural net with dropout and tuning the metaparameters with Bayesian optimization that works so well on acoustic modeling and visual object categorization is not particularly problem specific. Both object classification and LVSR have large labeled datasets, but perhaps with dropout the recipe might even work when labeled data is far scarcer. Convolutional architectures will be inappropriate for problems with different input geometries, but the optimization benefits ReLUs seem to provide might be useful elsewhere and their synergy with dropout may generalize as well. In the following chapters,

we will train neural nets using recipes quite similar to the speech DNN, but also have some task specific modifications.

Chapter 3

Deep neural networks for drug discovery: a QSAR case study

In order to improve the length and quality of human life, we must create new medicines and treatments. Or less optimistically, in order to keep the specter of wide-spread antibiotic resistance and the collapse of modern medicine at bay, we will need to discover new candidate drug ingredients. The modern drug discovery process, in spite of the accumulated medical and chemical knowledge of the human race, is astonishingly difficult, expensive, and inefficient. As of 2010, the estimated cost of developing and testing a single new molecular entity is $1.8 billion [Paul et al., 2010]. Without dramatic increases in pharmaceutical R&D productivity that reverse its recent decline and result in new medicines, Paul et al. [2010] argues that the rise of diabetes and childhood obesity may even result in decreases in life expectancy in wealthy North American and European countries.

Many steps in the drug discovery pipeline could potentially benefit from machine learning, but we will focus on virtual screening. Virtual screening automatically evaluates large molecular libraries with software that attempts to predict pharmacological properties, such as binding affinity towards a particular drug target, bioavailability, or toxicity. Quantitative structure activity/property relationship (QSAR/QSPR) studies attempt to build mathematical models relating physical and chemical properties of compounds to their chemical structure and are an application of machine learning to virtual screening; QSAR models can be used to inform pharmacological studies by providing an *in silico* metholodogy to test or rank new compounds for desired properties without actual wet lab experiments.

QSAR prediction problems are very different from machine learning problems in computer vision and speech recognition. Unlike vision and speech, QSAR modeling is not a perceptual problem and does not use data similar to human sensory data. Input features for QSAR are derived from discrete chemical structures and may be heterogeneous and nothing like pixels in an image or frequency bins in a spectrogram. Compared to large labeled datasets of billions of frames of transcribed speech or millions of labeled images, datasets used to train QSAR models might only have tens of thousands of compounds and many compounds might be highly redundant. Nevertheless, we will demonstrate that deep neural nets with many similarities to those used for acoustic modeling can be highly effective for QSAR modeling.

3.1 Introduction to QSAR

QSAR studies are used extensively to predict pharmacokinetic properties, such as ADME (absorption, distribution, metabolism, and excretion), as well as toxicity properties. Improvements in these methods could provide numerous benefits to the drug development pipeline. Learning QSAR models requires three main steps: generating a training set of measured properties of known compounds, encoding information about the chemical structure of the compounds, and building a mathematical model to predict measured properties from the encoded chemical structure. High throughput screening (HTS) studies are ideal for collecting training data — a few hundred to thousands of compounds are now tested routinely using HTS equipment against an assay of interest (which may be cellular or biochemical). Generating molecular descriptors from compound structures is also a well studied problem and various methods of encoding relevant information about compounds as vectors of numbers have been developed. These descriptors measure various physical, chemical, and topological properties of the compounds of interest. Once descriptors have been computed, machine learning and statistics supply the mathematical models used to make the predictions. The particular machine learning techniques used matter a great deal for the quality of QSAR predictions. In a QSAR competition sponsored by Merck[1], all competitors used the same descriptors and training data, but deep neural network models related to the neural nets we describe in this chapter allowed our team[2] to win and to achieve a relative accuracy improvement of approximately 15% over Merck's internal production system.

Applying machine learning algorithms to QSAR has a rich history. Early work applying machine learning to QSAR tasks used linear regression models, but these were quickly supplanted by Bayesian neural networks [Ajay et al., 1998, Burden and Winkler, 1999, Burden et al., 2000] and other approaches. Recently, random forests [Svetnik et al., 2003], projection pursuit, partial least squares and support vector machines [Du et al., 2008, Si et al., 2007] have also been applied successfully to this task. Each of these methods has different advantages and disadvantages (see Liu and Long [2009] for a recent review). Practitioners in the field have often been partial to methods that, in addition to making accurate predictions, allow for variable selection, so that users are able to assess whether chosen features are indeed useful from the viewpoint of an informed chemist, and to methods that allow for easy assessment of uncertainty from the models. Also important has been the issue of controlling overfitting, although this is a concern that is paramount not only to QSAR, but all domains where high-capacity machine learning models are applied to small data sets. Random forests have, thus far, been well-suited to these requirements since they do not readily overfit and since they provide a simple measure of variable importance. Bayesian neural networks are particularly suitable for assessing uncertainty about model predictions and controlling overfitting. However, the computational demands of Bayesian neural networks necessitate small hidden layers and the use of variable selection to reduce the input dimensionality. Gaussian process regression models can be viewed as an infinite hidden layer limit of Bayesian neural networks, but can still be quite computationally expensive, often requiring time cubic in the number of training cases [Obrezanova et al., 2007].

Non-Bayesian neural networks have also been applied to QSAR [Devillers, 1996], but often had only a single small hidden layer (presumably to avoid overfitting issues that Bayesian versions obviate). As we mentioned in section 1.2, the past decade has seen tremendous improvements in training methods for

[1] `http://www.kaggle.com/c/MerckActivity` (Retrieved April 3, 2015)

[2] `http://blog.kaggle.com/2012/11/01/deep-learning-how-i-did-it-merck-1st-place-interview/` (Retrieved March 27, 2015) Team members were George Dahl, Ruslan Salakhutdinov, Navdeep Jaitly, Christopher Jordan-Squire, and Geoffrey Hinton.

deep and wide neural networks as well as a renewed appreciation for the advantages of deeper networks. Deep neural networks (DNNs) have been highly successful on a variety of different tasks including speech recognition [Hinton et al., 2012a], computer vision [Krizhevsky et al., 2012], and natural language processing [Collobert and Weston, 2008]. The methodological improvements, along with faster machines, have allowed researchers to train much larger and more powerful models — instead of neural networks with only one hidden layer with 20-50 hidden units, neural networks (including ones from the previous chapter) are now regularly trained with many hidden layers of thousands of hidden units, resulting in millions of tunable parameters. Networks this large can even be trained on small datasets without substantial overfitting using early stopping, weight penalties, and the newer techniques of unsupervised pre-training [Hinton et al., 2006] and dropout [Hinton et al., 2012b]. All of these techniques for avoiding overfitting in large neural nets trained on small datasets are relatively inexpensive computationally, especially compared to a fully Bayesian approach.

As mentioned above, the neural net QSAR models we trained in this work differ from those used in the QSAR literature in a variety of respects. However, the most important difference is that the neural nets we explore in this chapter are multi-task neural networks that operate on multiple assays simultaneously, something that is very rare in the literature. The only other work we are aware of to use multi-task neural nets for QSAR is Erhan et al. [2006] and there are a variety of differences between it and the work we present here. In particular, we used the recently developed dropout technique to control overfitting and we did not have access to target protein features. Additionally, Erhan et al. [2006] focussed on exploring the performance of the kernel-based JRank algorithm for collaborative filtering and not different neural net variants.

The motivation behind multi-task learning is that performance on related tasks can be improved by learning shared models. Multi-task neural network models are particularly appealing because there can be a shared, learned feature extraction pipeline for multiple tasks. Not only can learning more general features produce better models, but weights in multi-task nets are also constrained by more data cases, sharing statistical strength. QSAR tasks naturally lend themselves to multi-tasking — assays are often related to each other and different compounds might share features. Even if assays are only tenuously related, they are still governed by the laws of chemistry and it might still be important to learn more broadly useful higher-level features from the initial descriptors.

In this chapter, we apply multi-task learning to QSAR using various neural network models. We do this while leveraging some of the recent developments outlined above. Our results show that neural nets with multi-tasking can lead to significantly improved results over baselines generated with random forests, confirming the result of the Merck contest.

3.2 Methods

The machine learning model that we use as the basis for our approach to QSAR is the feedforward (artificial) neural network (ANN), described in more detail in section 1.3. For the classification problems we consider in this chapter, the cross-entropy error function (equation 1.7) is appropriate. We trained all neural networks in this work using minibatched stochastic gradient descent (SGD) with momentum. In other words, we repeatedly estimated $\frac{\partial C(\Theta)}{\partial \Theta}$ from a small minibatch of training cases and used it to update the "velocity" of the parameters (see equations 1.13 and 1.14 in section 1.3 for details on the update rule).

3.2.1 Multi-task neural networks

The most direct application of neural networks to QSAR modeling is to train a neural net on data from a single assay using vectors of molecular descriptors as input and recorded activities as training labels. This single-task neural net approach, although simple, depends on having sufficient training data in a single assay to fit the model well. We suspect data scarcity is one reason aggressively regularized models such as random forests are popular in the literature [Svetnik et al., 2003]. In order to leverage data from multiple assays, we use a multi-task neural net QSAR architecture that makes predictions for multiple assays simultaneously. Vectors of descriptors still serve as the input to the network, but there is a separate output unit for each assay. For any given compound, we typically only observe one or at most a small number of output values since in general we do not expect compounds to appear in all assays regularly. During multi-task neural net training we only perform backpropagation through output layer weights incident on output units whose value we observe for the current training case. For simplicity we treat a compound that appears in k different assays as k different training cases that have the same descriptor vector but different outputs observed. Since the number of compounds screened varies across assays, naïvely combining all the training cases from all the available assays and training a multi-task neural net would bias the objective function towards whatever assay had the most compounds. We handle this issue by controlling how many training cases from each assay go into a minibatch. For example, if there are 7 assays we could create minibatches of 80 cases by drawing 20 cases at random (with replacement) from a particular assay we wish to emphasize and 10 cases from each of the other six assays.

Martin et al. [2011] presented another way of leveraging related assays called Profile-QSAR that is similar in spirit to the multi-task neural net approach we use (although it does not use neural nets), but has many important differences. Profile-QSAR treats some of the assays as side information and uses single task methods to complete an assay/compound activity matrix for these previously studied assays. The imputed and measured activities for a particular compound in the side information assays become additional descriptors to use when making predictions for the compound. Unlike Profile-QSAR, the multi-task neural net approach does not do any imputation of activities. Another difference is that a multi-task neural net trains on all available assays and potentially makes predictions on all available assays as well while learning a shared feature extraction pipeline.

3.2.2 Wider and deeper neural networks

Deep neural networks, or neural networks with multiple hidden layers, have been so successful recently because they are capable of learning complicated, rapidly-varying non-linear functions and are also capable of extracting a hierarchy of useful features from their input. Many successful deep neural network models, such as those used in chapter 2, have millions of tunable parameters and wide layers with thousands of units. In contrast, neural nets used in QSAR to date tend to have only a single layer and sometimes very few hidden units (for example Lowe et al. [2012]). Advances in training and regularizing wide and deep neural nets and in computer hardware have changed the dominant approach to training neural networks in the machine learning community from training small nets that are incapable of overfitting (and often underfit) to training wide and deep neural networks with many tunable parameters and depending on aggressive regularization to prevent overfitting. The most important lesson from the recent successes of deep learning is that, although deeper networks will not always perform better than

shallow ones, practitioners need to be using models that can trade breadth for depth and vice versa, since one particular architecture cannot be the best for all problems. As assays become cheaper and more assay results accumulate, machine learning models with high capacity will have the best performance. Many of the models we trained (including nets with only a single hidden layer) have far more weights than we have training cases, but through careful training and regularization they can still perform well.

3.2.3 Regularizing large neural networks

Using wide and/or deep neural nets with many tunable parameters makes avoiding overfitting especially crucial. In QSAR modeling we often want to use large and expressive models capable of representing complex dependencies between descriptors and activities, but data may also be quite limited. Regularization, broadly construed, has been the subject of much recent deep learning research. Generative unsupervised pre-training [Hinton et al., 2006, Erhan et al., 2010] is a powerful data-dependent regularizer. Early stopping of training based on validation error can partially control overfitting but has limited effectiveness when there is very little validation data and the networks being trained have very high capacity. Penalizing large weights in the model can also help avoid overfitting, although more sophisticated techniques, in particular dropout [Hinton et al., 2012b], are, in our experience, much more useful. Intuitively, one effect the noise added by dropout has is that it penalizes large weights that result in uncertain predictions or hidden unit activations. Another way to view dropout is as approximate model averaging over the exponentially numerous different neural nets produced by deleting random subsets of hidden units and inputs. We can also view multi-task neural nets as a way of regularizing the weights that are shared across assays. Since only the output weights are assay-specific, using data from other assays is a powerful way of avoiding overfitting. Instead of shifting the weights towards zero the way an L2 penalty on the weights would, multi-task neural nets shift the weights of the hidden unit feature detectors towards values that extract features useful for other QSAR tasks. In this way, hidden layers in a multi-task neural network in effect learn higher level, more abstract molecular descriptors and have a training objective that encourages them to create features useful for more than one task.

3.3 Experiments

In order to test the effectiveness of multi-task neural nets for QSAR we trained neural nets on publicly available assay results and compared their performance to a selection of baseline methods.

3.3.1 Dataset Generation

We ran experiments on assay results deposited in PubChem (`http://pubchem.ncbi.nlm.nih.gov/`). We used 19 different assays, selected so at least several of them were closely related. Table 3.1 lists the assays that were used in our experiments. We included both cellular and biochemical assays and in some cases used multiple related assays, for example assays for different families of cytochrome P450 enzymes.

We generated molecular descriptors with Dragon[3] to use as input to the various machine learning models. While Dragon can generate 4885 different descriptors, not all of these descriptors were applicable to all of the compounds in our data sets. After excluding the inapplicable ones, we were left with 3764 molecular descriptors. Each descriptor was Z-score normalized over all the compounds in the union of all

[3] `http://www.talete.mi.it/`

Table 3.1: List of assays from Pubchem that were used for this study

AID	Assay Target / Goal	Assay Type	# active	# inactive
1851(2c19)	Cytochrome P450, family 2, subfamily C, polypeptide 19	Biochemical	5913	7532
1851(2d6)	Cytochrome P450, family 2, subfamily D, polypeptide 6, isoform 2	Biochemical	2771	11139
1851(3a4)	Cytochrome P450, family 3, subfamily A, polypeptide 4	Biochemical	5266	7751
1851(1a2)	Cytochrome P450, family 1, subfamily A, polypeptide 2	Biochemical	6000	7256
1851(2c9)	Cytochrome P450, family 2, subfamily C, polypeptide 9	Biochemical	4119	8782
1915	Group A Streptokinase Expression Inhibition	Cell	2219	1017
2358	Protein phosphatase 1, catalytic subunit, alpha isoform 3	Biochemical	1006	934
463213	Identify small molecule inhibitors of tim10-1 yeast	Cell	4141	3235
463215	Identify small molecule inhibitors of tim10 yeast	Cell	2941	1695
488912	Identify inhibitors of Sentrin-specific protease 8 (SENP8)	Biochemical	2491	3705
488915	Identify inhibitors of Sentrin-specific protease 6 (SENP6)	Biochemical	3568	2628
488917	Identify inhibitors of Sentrin-specific protease 7 (SENP7)	Biochemical	4283	1913
488918	Identify inhibitors of Sentrin-specific proteases (SENPs) using a Caspase-3 Selectivity assay	Biochemical	3691	2505
492992	Identify inhibitors of the two-pore domain potassium channel (KCNK9)	Cell	2094	2820
504607	Identify inhibitors of Mdm2/MdmX interaction	Cell	4830	1412
624504	Inhibitor hits of the mitochondrial permeability transition pore	Cell	3944	1090
651739	Inhibition of Trypanosoma cruzi	Cell	4051	1324
651744	NIH/3T3 (mouse embryonic fibroblast) toxicity	Cell	3102	2306
652065	Identify molecules that bind r(CAG) RNA repeats	Cell	2966	1287

the assays. For some of the single task neural net models, we also generated additional binary features by thresholding other single descriptors. We selected descriptors and thresholds from decision nodes used in a boosted ensemble of limited depth decision trees.

3.3.2 QSAR Model Training

Using the assays, we treated the problem as a classification task using the active/inactive labels produced by the assay depositors. QSAR prediction can be formulated as a classification problem, a regression problem (as in the Merck contest), or a ranking problem. All of our techniques are equally applicable to any of these problem formulations, but in this chapter we only consider the binary classification version of the problem. A natural way to use a model that predicted activities would be to use it for virtual screening which is ultimately a ranking problem. Although the models we trained optimized classification performance, we use area under the ROC curve (AUC) as a performance metric since it emphasizes the ranking aspect of the problem more relevant to virtual screening applications.

For each assay, we held out at random 25% of the ligands to use as a test set, leaving the remaining 75% as a training set. We used several classifiers implemented in the scikits-learn [Pedregosa et al., 2011] package as baselines: random forests (RF), gradient boosted decision tree ensembles (GBM), and logistic regression (LR). We split the training set into four folds and trained each model four times with a different fold held out as validation data. We average the test set AUCs of the four models when reporting test set results. We used performance on the validation data to select the best particular model in each family of models. To the extent that the baseline models required metaparameter tuning (e.g. selecting the number of trees in the ensemble), we performed that tuning by hand using validation performance.

Neural Network Metaparameter Selection

Neural networks can have many important metaparameters, including architectural metaparameters, such as layer sizes and hidden unit link functions, optimization metaparameters, such as learning rates and momentum values, and regularization metaparameters such as the dropout probabilities for each

layer, how long to train before stopping, and learning rate annealing schedules. We trained neural nets that used rectified linear units as well as neural nets that used sigmoidal units and most neural nets had between two and eight million parameters. In order to have an experimental protocol that avoids as many of the vicissitudes of human expert judgement as possible, we set all neural net metaparameters using Bayesian optimization [Snoek et al., 2012, 2013] of the validation AUC. Bayesian optimization is a type of sequential, model-based optimization ideally suited for globally optimizing blackbox, noisy functions while being parsimonious in the number of function evaluations. For metaparameter optimization, Bayesian optimization constructs a model of the function mapping metaparameter settings to validation AUC and suggests new jobs to run based on an acquisition function that balances exploration of areas of the space where the model is highly uncertain with areas of the space likely to produce better results than the best job run so far. We used the Spearmint software that implements the Bayesian optimization algorithms from Snoek et al. [2012] and in particular we use the constrained version [Gelbart et al., 2014] with warping [Snoek et al., 2013] enabled. See appendix A.1 for details of what neural net metaparameters we optimized over what ranges.

Bayesian optimization finds good model configurations very efficiently, but makes very strong use of the validation error and, with small validation sets, can quickly overfit the validation data. However, overfitting the validation data will only hurt the test set performance of the neural net models relative to the baselines that have fewer metaparameters that can be tuned readily and repeatably by hand.

3.4 Results and discussion

Table 3.2 contains our main results on all 19 assays we investigated. On 14 of the 19 assays, the best neural net achieves a test set AUC exceeding the best baseline result by a statistically significant margin. We measured statistical significance with t-tests using standard errors from training the models repeatedly on new bootstrap samples (see appendix A.2 for more details). On the remaining five assays there was no statistically significant difference between the best decision tree ensemble baseline and the best neural net. On all assays, logistic regression was by far the worst performing model. On assays with closely related assays available, multi-task neural nets often performed better than their single-task counterparts, even with no a priori information on which particular assays were related to each other and despite having to make good predictions on all assays simultaneously, including unrelated ones.

The three assays (assay ids 1915, 2358, and 652065) that the decision tree baselines did best on relative to the neural net models (despite the differences not being statistically significant) had some of the fewest training cases and had no closely related assays in our dataset. Since we generally expect high-capacity models to provide benefits for larger datasets and multi-task models to provide benefits when there are related assays to leverage, the three assays the baselines do comparatively better on are not surprising.

3.4.1 Multi-tasking vs combining assays

The 19 assays we used contain several groups of closely related assays. Specifically, we have five cytochrome P450 assays with minor variations, four assays for inhibition of Sentrin-specific proteases, and two related cellular assays for inhibition of particular yeast strains. When two assays are strongly *positively* correlated (which will happen when they are nearly identical and share enough compounds), simply merging the data from both might work as well as the more sophisticated multi-task neural net

Assay	RF	GBM	NNET	MULTI
1851_1a2	0.915	0.926	0.926	**0.938**
1851_2c19	0.882	0.894	0.897	**0.903**
1851_2c9	0.876	0.891	0.889	**0.907**
1851_2d6	0.839	0.857	**0.863**	0.861
1851_3a4	0.871	0.896	0.895	0.897
1915	0.754	0.757	0.756	0.752
2358	0.745	0.764	0.738	0.751
463213	0.651	0.659	0.651	**0.676**
463215	0.614	0.610	0.613	**0.654**
488912	0.664	0.672	0.664	**0.816**
488915	0.700	0.713	0.723	**0.873**
488917	0.818	0.834	0.835	**0.894**
488918	0.785	0.800	0.784	**0.842**
492992	0.804	0.827	0.803	0.829
504607	0.673	0.670	**0.684**	0.670
624504	0.851	0.869	0.871	**0.889**
651739	0.775	0.793	0.790	**0.825**
651744	0.870	0.885	0.886	**0.900**
652065	0.787	0.793	0.793	0.792

Table 3.2: This table shows the average test set AUC for random forests (RF), gradient boosted decision tree ensembles (GBM), single-task neural nets (NNET), and multi-task neural nets (MULTI) on each assay. The multi-task neural nets are trained to make predictions for all assays at once. If the best neural net result on a particular assay is better than both decision tree baselines and the differences are statistically significant then we use boldface (the difference between the two nets may or may not be statistically significant). In the cases where the best decision tree baseline is better than both neural nets the differences are not statistically significant.

method that can leverage more general positive, negative, and feature-conditional correlations. In fact, if all the related assays we worked with were nearly identical, the gains we see from the multi-task neural net over the single-task methods might simply reflect the gains expected from adding more data to a single-task classifier. To investigate this issue, we created additional datasets that combined data from multiple assays. For example, the 488912 assay has three related assays: 488915, 488917, and 488918. We added the training data from 488915, 488917, and 488918 to the 488912 training set while still testing on data exclusively from 488912. In this manner, we created 11 combined datasets and trained GBMs on them, since GBMs were the best performing non-neural net model.

Table 3.3 shows the results on these 11 datasets for single-task GBMs, multi-task neural nets using all 19 assays, and GBMs combining training data only from related assays. On 8 of the 11 datasets, the multi-task neural nets exhibit a statistically significant improvement over the single-task GBM and the combined training set GBM. On the remaining three datasets, there is not a statistically significant difference in performance between the best of the three models and the second best. Table 3.4 highlights results (also displayed in table 3.2) comparing single- and multi-task neural nets on assays with other related assays in our collection. On all but two of the 11 assays, the multi-task neural net trained using all 19 assays obtain statistically significant improvements in test AUC over the single-task neural nets. Since the multi-task neural net models can learn to ignore the other assays when making predictions for a particular assay, at worst they will only get somewhat weaker results than a single-task neural net because they waste capacity on irrelevant tasks. And, indeed, that is what we see in our results.

The 48891* series of assays are closely related enough for the GBMs trained on the combined training set of the four assays to do much better than the GBMs that use only the primary assay's training set. On some of the 48891* series assays, a GBM trained on the combined training sets even does better than a multi-task neural net trained to make predictions on all 19 datasets, although the improvement is not statistically significant. However, the multi-task neural net, in addition to having to make predictions for other unrelated assays, is not told that all of the 48891* series assays should be treated as the same problem. In contrast to the 48891* series, on the 1851_* series of assays, the GBMs trained on the combined 1851_* training sets do worse than the GBMs that just use the primary training set. On these assays, however, the multi-task neural net improves upon both the single task GBM and neural net, showing that it can leverage information from the related assays in a more nuanced way. In practice we will often have somewhat related assays that are nevertheless far from identical and in this situation the multi-task neural net models can still provide benefits, unlike a classifier simply combining the training sets from different assays.

On the 46321* series of assays, although the GBMs on the combined training sets were better than the GBMs trained only on data from the primary assay, the multi-task neural net was better still. This result demonstrates that the two assays in the group are not sufficiently contradictory to confuse the GBM models on the combined training set (as happened with the 1851_* series), but there are still gains to be had from the multi-task neural net. The most interesting cases arise when assays are related but not positively correlated strongly enough to simply combine into a single dataset. In this scenario, multi-task neural nets shine because they can use negative and partial correlations.

3.4.2 Controlling overfitting without feature selection

Since we allowed Bayesian optimization to train fully connected neural networks with as many as about 3500 hidden units in a single layer, we used a variety of methods to prevent overfitting. Bayesian

Primary Assay	GBM	MULTI	GBM Combined
1851_1a2	0.926	**0.938**	0.905
1851_2c19	0.894	**0.903**	0.883
1851_2c9	0.891	**0.907**	0.879
1851_2d6	0.857	**0.861**	0.840
1851_3a4	0.896	0.897	0.869
463213	0.659	**0.676**	0.665
463215	0.610	0.649	0.648
488912	0.672	**0.825**	0.815
488915	0.713	0.873	0.868
488917	0.834	**0.915**	0.909
488918	0.800	**0.869**	0.852

Table 3.3: Multi-task neural nets compare favorably to GBMs using training sets that combine related assays. Bold entries correspond to statistically signifigant differences.

Primary Assay	NNET	MULTI
1851_1a2	0.926	**0.938**
1851_2c19	0.897	**0.903**
1851_2c9	0.889	**0.907**
1851_2d6	0.863	0.861
1851_3a4	0.895	0.897
463213	0.651	**0.676**
463215	0.613	**0.649**
488912	0.664	**0.825**
488915	0.723	**0.873**
488917	0.835	**0.915**
488918	0.784	**0.869**

Table 3.4: For assays related to other assays in our collection, multi-task neural nets typically provide statistically significant improvements over single-task neural nets. Bold entries correspond to statistically significant differences.

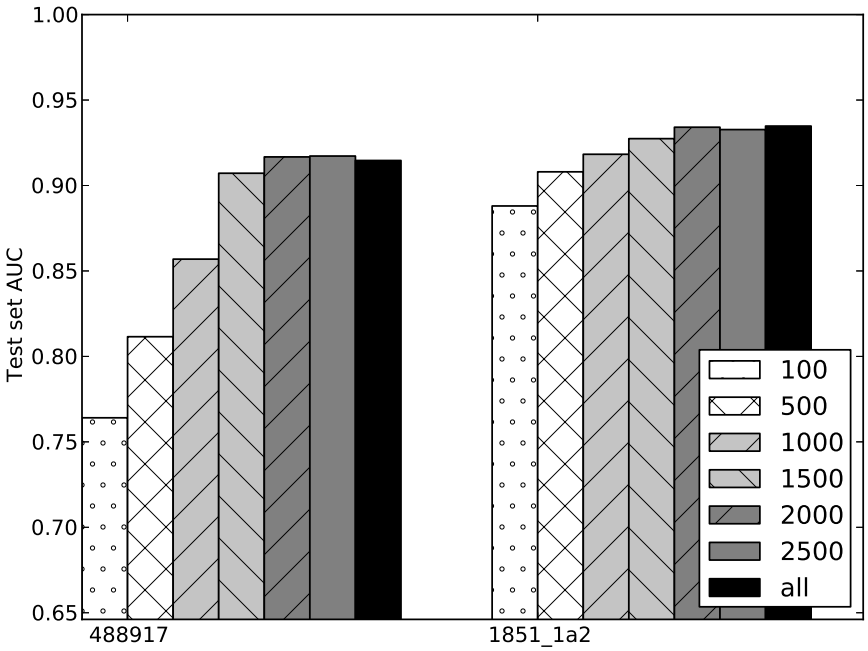


Figure 3.1: Test AUC on two representative assays of a multi-task neural net using different numbers of input features. For a given number of input descriptors, we selected the best features as measured by information gain.

optimization quickly learned to enable dropout and use a non-zero L2 weight penalty. For any given assay, the best performing neural net always used some dropout and in preliminary hand-tuned experiments dropout seemed crucial as well.

Unlike a lot of QSAR work in the literature, for example, Winkler [2002], who warns against including too many descriptors even if they contain relevant information, we do not advise performing feature selection to reduce the number of input dimensions drastically. Although not all the descriptors are necessary or very informative, well-regularized and properly trained neural networks can handle thousands of correlated input features. We trained neural nets using all (3764) descriptors as well as ones using the 2500, 2000, 1500, 1000, 500, or 100 most informative input features (as measured by information gain). On most assays, using the 2000-2500 most informative input descriptors did not degrade test set AUC very much, but using only 1500 or fewer typically produced a large and unnecessary drop in test set AUC. Figure 3.1 shows the test AUC for two representative assays. We generated the plot by training the best multi-task neural net using the relevant primary assay on different numbers of input descriptors.

3.4.3 Neural network depth

For the 19 assays we used and the descriptor set we used, changing the number of hidden layers had no consistent effect. We performed separate Bayesian optimization runs with one, two, and three hidden layer multi-task neural networks and one and two hidden layer single task neural networks. For single

task neural nets, adding a second hidden layer had very little effect and the best result from each Bayesian optimization run for a particular model class achieved about the same test AUC regardless of the number of hidden layers. However, although there was no consistent trend for multi-task nets, allowing deeper models seemed to be occasionally important. Table 3.5 shows the results for the multi-task nets of different depths. Although the optimal depth is not the same or predictable across assays, on some assays there are large differences in performance between multi-task neural nets of different depths.

Assay	1 hidden layer	2 hidden layers	3 hidden layers
1851_1a2	0.932	**0.938**	0.930
1851_2c19	**0.903**	0.884	0.881
1851_2c9	**0.907**	0.894	0.905
1851_2d6	0.855	**0.861**	0.768
1851_3a4	0.897	0.897	0.821
1915	0.752	0.754	0.740
2358	0.751	0.758	0.748
463213	0.676	0.678	0.672
463215	0.654	0.637	0.649
488912	0.768	0.816	0.825
488915	0.823	**0.873**	0.828
488917	0.886	0.894	**0.917**
488918	0.828	0.842	**0.869**
492992	0.829	0.827	0.826
504607	0.670	0.665	0.616
624504	0.889	0.881	0.893
651739	0.825	0.821	0.806
651744	0.895	**0.900**	0.894
652065	0.794	0.792	0.791

Table 3.5: Multi-task neural network results for neural nets with different numbers of hidden layers. We bolded the best result in a row when there was a statistically significant difference in test AUC between it and the second best entry in the row.

These depth results somewhat contradicts our experience in the Merck molecular activity prediction contest where we found using neural networks with more than one hidden layer to be crucial in almost all cases. Unfortunately, since the contest data is not licensed for research and additional experiments (nor are the descriptors used in the contest currently public), we can only speculate about the cause of this discrepancy. There were several assays in the contest with many more compounds than the assays we used from PubChem and larger datasets are more likely to provide enough training information to usefully fit additional hidden layers. Larger datasets affecting the optimal neural net depth is consistent with depth mattering more for the multi-task nets we trained since they use data from all the assays. The contest used the regression formulation of the task, unlike our experiments in this work that used the binary classification formulation, exacerbating the difference in the number of bits of information in the training labels. Since we did not focus our efforts on trying many different descriptor sets, there may be types of descriptors Dragon does not compute that did exist in the contest data. A larger set of descriptors than the ones Dragon computes might improve our results. For example, the open source RDKit (`www.rdkit.org`) software provides Morgan fingerprint descriptors that the current version of Dragon does not compute and other commercial software packages such as MOE [2013.08] could have

descriptor sets that add important information also.

3.5 Conclusions and future work

Our results demonstrate that neural networks using the latest techniques from the deep learning community can improve QSAR prediction accuracies and, in particular, there is a very natural and effective way of leveraging data from multiple assays when training a neural network QSAR model. However, many more experiments need to be done before we can settle on exactly the best way to solve QSAR problems with neural networks. Treating the task as a binary classification problem and using potentially unreliable active/inactive decisions from assay depositors in PubChem may not be the best way to approach the problem. In a virtual screening application, a notion of how active a compound is towards a particular target is essential and we plan to perform future work with the ranking version of the problem. Given the effectiveness of state-of-the-art Bayesian optimization software packages, practitioners should no longer fear the large number of metaparameters sophisticated neural network models can have since even with small datasets we were able to find very good metaparameter settings automatically. We also hope to develop better ways of implementing multi-task neural nets that can make use of additional information about which assays are likely to be related as well as target features and other side information. Given the rapid progress of research on neural network methods, we also hope to apply more advances from the deep learning community to QSAR problems.

Chapter 4

Connectionist models of natural language text

Natural language processing (NLP) has fascinated researchers since the advent of AI as a discipline in the 1950s, appearing both in philosophical arguments about the nature of intelligence and in applications. In fact, Turing proposed his famous imitation game that required a computer to communicate in human language only a few years before the first machine translation demo. Human language seems to be an important distinction between humans and other animals making language processing tasks natural members of the AI-set. Beyond any potential implications for AI research, software able to extract useful information from text automatically is of immense practical value simply because of the ubiquity of natural language text on the internet.

Text data have very different qualitative properties than data from other modalities and these properties provide important challenges for machine learning. Tokens, characters, and words are inherently discrete whereas pixel values are discrete merely because of digital quantization. Zipfian distributions abound in NLP and the large fraction of words that are rare words demands machine learning models with powerful generalization abilities. Unlabeled data are often plentiful, but even in the largest unlabeled text corpora many particular constructions or phrases of interest will still only occur rarely. Human users also have very high standards for correctness and precision for text processing systems since an error in a single word can have non-local effects on the meaning of an utterance.

Because massive collections of text data provide exciting opportunities for machine learning research, there is an extensive literature on natural language processing. In this chapter, I will start by reviewing a small selection of related work on connectionist models of text capable of learning distributed representations of their input. I adopt a very inclusive and informal definition of connectionist machine learning algorithms and consider both probabilistic and non-probabilistic versions, including artificial neural networks of all kinds, Boltzmann machines, and other probabilistic graphical models with suitably generic and modular structures that can learn distributed representations. As expansive as this definition is, it excludes many popular models in NLP such as n-gram language models, linear chain conditional random fields using hand crafted features, support vector machines with hand designed kernels, and probabilistic context free grammars. Following a brief review of previous work on connectionist models of text, the bulk of the chapter describes two NLP research projects: a project on training restricted Boltzmann machines on word observations and a project on a family of neural network generative models

of dependency parsed sentences.

4.1 Selected related work

I have organized the review thematically and not historically. In subsection 4.1.1 I describe a subset of the neural language model literature that includes both Markov neural language models and non-Markov (i.e. recurrent) neural language models. Neural language models were the first connectionist models of text to learn word embeddings. Nowadays, almost all neural models of text that operate on words learn word embeddings, including all of the models discussed in sections 4.2 and 4.3. Furthermore, many of the solutions to the computational problems associated with producing words as outputs of neural networks also were first proposed in the context of neural language models. In particular, the hierarchical softmax technique we use in section 4.3 originates in the neural language modeling literature.

Subsection 4.1.2 discusses the highly influential multi-task neural NLP system of Collobert and Weston [2008] and related models of text that use shifted windows or convolutions. This work, in addition to its historical importance, is closely related to our contributions in section 4.2. Specifically, our method for efficiently training an RBM on word observations, or something like it, would be necessary if we wanted to build Boltzmann machine analogues of the Collobert and Weston [2008] models.

Section 4.1.3 examines the recent literature on tree neural net models of text. Although not especially relevant to the content of section 4.2, they are highly relevant to the content of section 4.3. TNN autoencoders are some of the most well known neural network systems that operate on parsed sentences and in section 4.3 we describe neural generative models of parse trees that actually generate parsed sentences.

4.1.1 Next step prediction language models

A next step prediction language model (LM) assigns a probability to a sequence of tokens $(w_1, \cdots, w_T)$ by factoring the probability distribution $P(w_1, \ldots, w_T)$ as

$$P(w_1, \cdots, w_T) = \prod_t^T P(w_t | w_{k<t}), \tag{4.1}$$

based on the order of the tokens in the sequence. I use the term language model (LM) exclusively to refer to any model parameterizing the distribution over token sequences according to the factorization of equation (4.1). For the purposes of this section, tokens will be words. The phrase "text model", on the other hand, will be used to denote the more general set of machine learning models trained on text data. Some text models reviewed, unlike language models, will not be able to assign probabilities to any arbitrary string in the language, either because they are not proper probabilistic models (e.g. autoencoders), they are incomplete in some sense (e.g. finite context window models), or for some other reason.

A language model with the Markov property conditions on only a fixed context of τ previous word tokens in each factor. Equivalently, $P(w_t|w_{k<t}) = P(w_t|w_{t-\tau}, \ldots, w_{t-1})$. Parameterizing each factor in equation (4.1) with a neural network poses two immediate problems: each context size requires a different neural network and, since tokens are from a large set of discrete types, for any reasonable vocabulary and context size the number of weights in the neural network would be prohibitive. The

first problem can be solved either by adhering to the Markov property and conditioning on only a few previous words in the sequence or by using a recurrent neural net. The second problem was solved in Bengio et al. [2003] by factoring the first layer of weights in order to learn word embeddings.[1] This weight matrix factorization can be viewed as a learned lookup table that maps words in the vocabulary to low-dimensional, learned feature vectors.

The natural choice for the output of a neural language model is a softmax unit over the entire vocabulary. Let z_w be the net input to the component of the output layer softmax that corresponds to word w. The probability the net assigns to word w given context C is then given by:

$$P(w_t = w|C) = \frac{e^{z_w}}{\sum_{j \in V} e^{z_j}}. \tag{4.2}$$

Although the cost of computing the unnormalized probability assigned by the net to a particular word in the vocabulary is independent of the vocabulary size, normalizing that probability has a cost linear in the vocabulary size. Using backpropagation to train a neural language model with such an output layer is prohibitively expensive for large vocabularies.

Accelerating training for large softmax output layers

The training time (and test time) problems caused by the large softmax output layer have been tackled in a variety of ways in the literature. Extending their previous work from 2003, Bengio and Senécal [2008] used an adaptive importance sampling scheme with proposals drawn from n-gram models to speed up training in exchange for only obtaining approximate gradients. Morin and Bengio [2005] arranged the words in the vocabulary at the leaves of a self-normalizing binary tree to obtain training updates that are logarithmic in the vocabulary size instead of linear (also decreasing the cost at test time). In the tree scheme, each word in the vocabulary corresponds to a binary string encoding the path to that word in the tree. The probability assigned to the ith word in the vocabulary given context C then becomes:

$$P(w_t = w|C) = \prod_j P(b_j|q_j, C), \tag{4.3}$$

where b_j is the jth bit of the binary string encoding the path to word w and q_j is a learned feature vector for the jth node in the tree on the path to word w. Morin and Bengio [2005] used WordNet to define the tree, but it is also possible to learn the tree. For example, Mnih and Hinton [2009] learn the tree by performing a simple hierarchical clustering of word average context feature vectors extracted from their model trained with an initial random tree. Other papers (e.g. Mikolov et al. [2013b]) have used a tree produced by Huffman coding on unigram frequencies.

The single most promising solution to the large softmax training time problem for neural language models to date, proposed by Mnih and Teh [2012], uses noise-contrastive estimation (NCE) instead of maximum likelihood. NCE is an estimation principle developed in Gutmann and Hyvärinen [2010, 2012] that is particularly well suited for generative models that are difficult to normalize (Mnih and Teh [2012] were able to apply NCE to the discriminative next word prediction problem with a simple tweak). NCE is based on a reduction from unsupervised learning to supervised learning. Intuitively, characterizing

[1] Rumelhart et al. [1986] also described a way of learning embeddings for objects from a discrete space, but the example they presented was not complicated enough to require weight sharing and thus did not contain a complete description of the methods necessary for language models.

the shape of the data density with an unsupervised model is equivalent to being able to classify data cases as coming from either the data distribution or from some contrasting noise distribution. To deal with unnormalized models (e.g. Boltzmann machines) using NCE, one can learn the normalizing constant as another parameter of the model. In maximum likelihood learning however, adding the normalization constant as a parameter does not work because the optimization procedure can grow the likelihood without bound by shrinking the normalization constant to zero. Using the notation of Gutmann and Hyvärinen [2012], let $p_m^0(.;\alpha)$ be an unnormalized model with parameters α that we augment with a learned (log) normalizing constant c to produce a model $p_m(.;\theta)$, where $\theta = (\alpha, c)$ and $\ln p_m(.;\theta) = \ln p_m^0(.;\alpha) + c$. NCE requires that we extend our actual data set of T_d observations $X = \{\mathbf{x}_t\}_{t=1}^{T_d}$ with $T_n = kT_d$ noise samples $\tilde{X} = \{\tilde{\mathbf{x}}_t\}_{t=1}^{T_n}$ drawn from an analytically tractable noise distribution $p_n(\cdot)$. Let U be the union of X and $\tilde{X}$ and let each data point $\mathbf{u}_t$ be associated with a class label y_t set to 1 for an original data point and 0 for a noise point. If we set

$$p(\mathbf{u}|y=1;\theta) = p_m(\mathbf{u};\theta) \text{ and}$$
$$p(\mathbf{u}|y=0) = p_n(\mathbf{u})$$

and use the prior probability ratio $P(y=0)/P(y=1) = k$ we selected then the posterior probability for some point $\mathbf{u}$ belonging to the data (non-noise) class is:

$$P(y=1|\mathbf{u};\theta) = \frac{1}{1 + k\exp\left(\ln p_n(\mathbf{u}) - \ln p_m(\mathbf{u};\theta)\right).}$$

Thus the logistic regression classifier assumed by NCE classifies points as noise or data based on the log ratio of the model density $p_m(\mathbf{u};\theta)$ and the noise density at the point. Since the noise distribution is properly normalized, if the learned normalization constant of the model strays from the correct value the noise-vs-data classification error will increase. The NCE objective is closely related to the conditional log-likelihood of the classifier we just constructed and is given by:

$$J(\theta) = \frac{1}{T_d}\sum_{t=1}^{T_d} \ln P(y=1|\mathbf{x}_t;\theta) + k\frac{1}{T_n}\sum_{t=1}^{T_n} \ln(1 - P(y=1|\tilde{\mathbf{x}}_t;\theta)). \tag{4.4}$$

The large softmax output layer is a problem during training because we must compute the net input to each component of the softmax in order to normalize the softmax and in order to compute the error signal to backpropagate through the model; operations taking time linear in the vocabulary size are simply too expensive to do for every single weight update. NCE as described by Gutmann and Hyvärinen [2012] when naïvely applied to neural language models requires a separate normalization parameter for each possible context. Mnih and Teh, however, solve this problem by simply setting all normalizing constants to 1 and not learning them, relying on the objective function to force the other parameters to keep the model distribution sufficiently normalized to have useful learning updates. NCE neatly solves the training time problem for large softmax output layers, but at test time Mnih and Teh [2012] still explicitly normalized the probabilities. Therefore the tree solution might still be preferable when test time cost is particularly important and we only need to score candidate strings and not instantiate the entire distribution.

Recurrent neural language models

Recurrent neural network (RNN) language models achieve the state of the art in terms of perplexity for neural language models and statistical language models in general [Mikolov, 2012]. The RNN language model described in Mikolov et al. [2011] uses a hybrid architecture with two input paths to the same output layer. This architecture can also be viewed as jointly training a logistic regression model and an RNN. The logistic regression part of the net uses hashed n-gram features that do not use the learned word embeddings and is the neural network equivalent of a smoothed n-gram model. Without the logistic regression subsystem, context words can only influence the predictive distribution for the next word through the interaction of their word feature vectors with the hidden units of the network. Parameterizing a discrete conditional distribution with an exhaustive table is optimal assuming there is enough data to estimate every single table entry robustly. By using these two complementary, jointly trained components the composite model can memorize high frequency contexts when useful while still retaining the powerful generalization abilities of the recurrent neural network that learns word embeddings.

4.1.2 Windowed and convolutional text models

There are a multitude of natural language processing tasks of interest that do not involve assigning a probability to the next word. Some involve classifying sentences or utterances and others involve labeling individual words or phrases within a sentence. In an influential paper, Collobert and Weston [2008] (later expanded in Collobert [2011] and Collobert et al. [2011]) described a unified neural network architecture for several NLP tasks. Their system, called SENNA[2], performs part-of-speech tagging, chunking (identifying the top level syntactic constituents), named entity recognition, semantic role labeling (a form of basic semantic parsing), and traditional syntactic parsing, all with the same neural network. Using a single unified neural network for multiple tasks ensures that the features learned by the network will be more generally applicable to NLP problems than features learned for a single task. Furthermore, supervision from multiple tasks allows combined representations that share statistical strength and are more effective even for a single particular task.

The basic architecture of Collobert et al. [2011] has two variants, one without pooling over word positions and the other one with pooling. We will call the one with pooling the convolutional version and the one without pooling the windowed version. Both architectures operate on a sentence at a time and learn word embeddings. Both also require a forward pass for every tag that needs to be produced. Forward propagation in the windowed network consists of the following steps:

1. Map each word in the sentence to its feature vector (for multiple parallel input streams, concatenate all the feature vectors at a given position)
2. Concatenate the feature vectors for each position in the window into a single vector
3. Apply an affine transformation followed by the nonlinearity $h(x) = -1_{[x<-1]} + x_{[-1\leq x\leq 1]} + 1_{[x>1]}$, where an interval in a subscript indicates multiplication by the indicator function for the interval
4. Apply another affine transformation to compute a score for current output position

The convolutional variant pools over the entire sentence, but nevertheless requires a new forward pass for each tag being requested. In order to encode that the network output should refer to the tth position

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

in the sentence, the ith input position has the displacement $i - t$ from the word of interest added as an extra input. The forward propagation steps for the convolutional version of the architecture are:

1. Augment each position with an input for the displacement from the position we are tagging on this pass
2. Map each word and displacement (one-of-k encoded) to its feature vector, concatenating word vectors and displacement vectors for the same position
3. Concatenate the feature vectors for words in each local window into a vector for each local context
4. Apply an affine transformation shared across all local contexts
5. For each dimension of the local context vectors, take the max over all context window positions
6. Apply an affine transformation followed by the nonlinearity $h(x) = -1_{[x<-1]} + x_{[-1\leq x\leq 1]} + 1_{[x>1]}$
7. Apply another affine transformation to compute a score for current output position

Taking the max over each dimension in step 5 can produce correlations that do not exist in the original local vectors, but it guarantees a fixed dimensionality vector regardless of the number of input positions. Both variants of the model can be trained with a word level criterion that uses a simple softmax output layer over the tag set. Alternatively, they can be trained using a chain conditional random field (CRF) sentence level training criterion. The sentence level criterion is important for tasks (such as chunking) that label segments of the sentence since these segments are encoded with tags that have implicit constraints. For example, there is a tag for the start of a multi-word chunk, the inside of a multi-word chunk, the end of a multi-word chunk, a single word chunk, and a word outside of a chunk. The sentence criterion allows the model to learn transition costs that softly enforce the begin-before-end constraint and detect other correlations that might exist in the tag sequences.

An additional complexity arises for the semantic role labeling task. Since semantic roles are in relation to a particular verb, a forward pass is required for each word being tagged along with each verb being considered. The displacement of the verb under consideration relative to each position in the sentence serves as an additional input in the same way as the displacement to the tagging position does.

In order to leverage unannotated text to improve the representations learned by their model, Collobert et al. [2011] take an approach similar to NCE and train their model to distinguish real training sentences from corrupted sentences. They generate a corrupted version of the current training sentence by replacing a word with another word drawn uniformly from the vocabulary. They train the windowed network to rank real windows above corrupted ones. Using a ranking cost for this task results in a learning update independent of the vocabulary size. Interestingly, they also argue that for syntax the cross entropy criterion is fundamentally incorrect since it is dominated by frequent phrases even though rare phrases may be perfectly grammatical. Thus with the cross entropy criterion, everything humans know about language needs to fit into the approximately 0.2 bits per character advantage they have over simple statistical language models. Such a small difference in cross entropy means that subtle changes in the loss will have to correspond to qualitatively different abilities with language.

SENNA, the software package based on the approach of Collobert et al. [2011], achieves strong results on benchmarks for the various NLP tasks it performs, but often either is not the top performing system or is only slightly better than alternative systems. Although these results are still impressive given the

absence of substantial task-specific engineering, the approach has not had the practical success and wide adoption of neural language models. Part of speech tagging is a relatively easy task that many systems do quite well, including SENNA, with very little room for improvement. Although the other tasks are more challenging, some of them are already typically solved using CRFs with high quality features, even if the features are not learned and don't involve word embeddings. A simple alternative neural net architecture for semantic role labeling that would be useful to compare to SENNA could apply a fixed size window around the word to be labeled and the verb under consideration instead of pooling over the entire sentence and potentially discarding or masking a lot of useful information.

Like the neural language models discussed earlier, SENNA needs to learn word representations for a large number of words. Collobert et al. [2011] train a sequence of models with larger and larger vocabularies, initializing the word representations of each model with those from the previous model. This training strategy avoids part of the problem caused by rare words. When a word occurs for the first time, it might not receive a useful update since by the time it occurs again the feature vectors for other more frequent words used with it may have changed. A similar strategy might be possible in an open vocabulary model that slowly memorizes larger and larger strings as they occur repeatedly.

The model from Collobert et al. [2011] is the archetypal windowed neural model of text that is not a next word prediction language model. However, alternatives exist, in particular the RBM-based model we will consider in section 4.2.

4.1.3 Tree neural network models

Tree neural nets (TNNs)[3] are neural network models of text designed to have the network architecture mimic constituent structures in syntax. The first examples of these networks were recurrent auto-associative memories (RAAMs) [Pollack, 1990] and they have since been revived for NLP in Socher et al. [2011b] and refined in Socher et al. [2011c]. TNNs have been applied to parsing [Socher et al., 2011b], paraphrase detection [Socher et al., 2011a], and different sorts of sentiment analysis [Socher et al., 2011c]. TNNs have several autoencoder forms and can also be trained directly to produce outputs for each sentence in the case of tasks such as sentiment analysis. Like almost all word-level neural net models of text, TNNs learn word embeddings with the standard vocabulary-sized lookup table approach. Given a binary tree over the words of a sentence with one word assigned to each leaf, a TNN encodes the entire sentence from the leaves to the root, producing a code vector for each internal node recursively. Specifically, the TNN defines an encoder function $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ from child node row vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ to a parent node code vector $\mathbf{y}$. If $\mathbf{W}$ is a $2d$ by d weight matrix, $\mathbf{b}$ is a d-dimensional bias row vector, and $[\mathbf{x}_1, \mathbf{x}_2]$ denotes vector concatenation, then f is given by:

$$\mathbf{y} = f(\mathbf{x}_1, \mathbf{x}_2) = \tanh\left([\mathbf{x}_1, \mathbf{x}_2]\mathbf{W} + \mathbf{b}\right). \tag{4.5}$$

The leaf nodes use the word feature vectors for their respective words. In a TNN autoencoder, at each node, a similar decoder function computes a reconstruction for $\mathbf{x}_1$ and $\mathbf{x}_2$. Early versions of the autoencoder were trained to minimize the squared reconstruction error at each node, but more recent

[3]These models have been more commonly known as recursive neural nets (RNNs) in the literature, but to avoid confusion with other general recurrent neural networks we will not use that term. The term CRF usually means a chain CRF, but could mean CRFs with general graph structures. Similarly, we usually use the term recurrent neural net to mean RNNs with a chain structure, but it can include other types of recurrent architectures. Exclusively using TNN for the specific recurrent architecture that recently has been called a "recursive neural net" relieves some of the strain on the overloaded RNN acronym, making it more likely to be clear from context when an RNN does not have a chain structure.

versions minimize the reconstruction error at the leaves as would be more natural. Since the weights are shared across all internal nodes of the tree, the same TNN can compute a vector for trees and sentences of different size.

One simple modification introduced in Socher et al. [2011b] lets the TNN determine the tree structure instead of being given a known tree. Socher et al. [2011b] add a score layer to the autoencoder that maps the code for an internal node to a score representing how good it is for the two lower nodes to be children of the same parent. Once the net can score merging decisions for pairs of adjacent nodes, trees can be built greedily or with a beam search. In order to facilitate parsing, Socher et al. [2011b] also predicts syntactic categories of constituents at each internal node and conditions on neighboring context words outside the span of the particular constituent being encoded. In order to take advantage of trees provided in the parsing training set, Socher et al. [2011b] optimize, over sentence tree pairs (u_i, z_i), the max-margin regularized risk objective given by:

$$J = \sum_i s(u_i, z_i) - \max_{z \in \text{Trees}(u_i)} (s(u_i, z) + \Delta(z, z_i)), \tag{4.6}$$

where $s(u_i, z_i)$ is the sum of the scores the net assigns to all nodes in the tree z_i for sentence u_i and Δ penalizes trees based on how many incorrect spans they have (the span of an internal node is the pair of indices for the leftmost and rightmost leaves that are descendants of the node). In the parsing results from Socher et al. [2011b], none of the TNN-based parsers are quite as good as a state-of-the-art parser. Furthermore, without conditioning on the context, the TNN model exhibits much worse parsing performance. The necessity of conditioning on context for the encoding is in some sense an argument against the TNN model for parsing. If decisions about grouping together words into phrases depend so heavily on words outside the phrase, then why force the model architecture to encode blocks of adjacent words? We want to discover syntactic relations between words in a sentence, but some of these relationships can cut across entire phrases. For example, consider the sentence: *We found the pretzel that Bush choked on.* We would like to say that since Bush choked on the pretzel that *on* is mediating some sort of syntactic relation[4] between *choked* and *pretzel* (or *the pretzel*). The TNN could in principle consider merging any two vectors in the sequence and perhaps, if spurious merges were rare, this would be sufficient to deal with these sorts of phenomena. However, *choked on* is plausibly a very common bigram (X choked on Y, What did X choke on, etc.) so from a compression and reconstruction error standpoint, the TNN autoencoder may do well to group *choke* and *on* when they are adjacent, even if it has the ability to group non-adjacent words. Given the difficulty linguists have on agreeing on formal grammars and how best to formalize syntax as trees of constituents, perhaps a better model would use the recursive nature of language in a softer and more flexible way. Once two vectors have been merged, later encoding steps no longer have access to the originals and must depend on the encoder to save all the information they might need (and this encoder also must halve the dimensionality with a single autoencoder nonlinearity).

Minimizing reconstruction error at each node in the tree poses several problems. For example, the two child vectors being encoded might have already encoded very different numbers of words. If one child vector encodes an entire long clause and the other is just a single word, surely errors in reconstructing the vector for the clause are more damaging, on average, to the coherence of the representation than errors

[4]Of course this syntactic relation should exist in our model as a distributed representation of some kind, as should the relata.

in the reconstruction of the singleton. Even more worryingly, since the model computes the vectors for internal nodes and also reconstructs them, it can cheat by making them very small. These problems can be addressed heuristically by normalizing code vectors and re-weighting the reconstruction errors, but Socher et al. [2011a] describes a much more natural solution that only uses reconstruction error at the leaves. Since internal nodes do not generate any reconstruction errors, the network will only increase its error by making very small internal node vectors that cause larger errors downstream at the leaves.

Socher et al. [2011c] apply TNNs to predicting the sentiment of phrases by adding a softmax output layer to the root vector for a phrase. They train the TNNs with a semi-supervised objective that combines the reconstruction error with the cross entropy for the sentiment labels, building trees greedily. They achieve slightly better (relative error reduction of about 2%) sentiment classification results than other methods from the literature on a movie review dataset and a news article opinion dataset.

A very natural task for TNN autoencoders is paraphrase detection. If TNNs really learn high quality representations of text that encode a lot of semantic information then the vectors they learn for sentences should be ideal for paraphrase detection. Socher et al. [2011a] applies TNNs to paraphrase detection. They also try deeper TNN autoencoder variants that have a hidden layer in both the encoder and the decoder for each node. Unfortunately, the deeper versions are not investigated or described in enough detail to reveal much. In addition to supervised paraphrase detection experiments, Socher et al. [2011a] also provide examples showing reconstructions and nearest neighbor phrases for their various models. The reconstructions that they include, for relatively short phrases, are usually exact reconstructions and they report that the best TNN variant can reconstruct almost all phrases of up to three words perfectly and many with as many as five words.

Although the most natural way to do paraphrase detection with a TNN autoencoder is to classify based on the distance between the encoding for the two phrases, Socher et al. [2011a] propose a method they call *dynamic pooling*. A TNN constructs a full binary tree with a number of leaves equal to the number of words in a sentence so if there are n words there will be $2n-1$ nodes in the tree, including the word nodes. Given two TNN trees with $2n-1$ and $2m-1$ nodes respectively, the dynamic pooling algorithm first constructs a $2n-1$ by $2m-1$ similarity matrix S with S_{ij} holding the distance between the ith node vector in the first sentence and the jth node vector in the second sentence (nodes are ordered in the original sentence word order and then depth-first, right-to-left). The dynamic pooling algorithm then overlays an n_p by n_p grid over S with equally sized cells, upsampling S if necessary. Taking the minimum value in each of the rectangular regions of S yields a fixed size matrix S_{pooled} that they use as input to a standard binary logistic classifier. With n_p set to be a bit smaller than a typical sentence, upsampling will be rare.

Paraphrase detection is one of the most appealing tasks for TNNs and Socher et al. [2011a] describes results on the Microsoft Research paraphrase corpus (MSRP) [Dolan et al., 2004]. Guessing the most frequent class (paraphrase) achieves an accuracy of 66.5% and the inter-annotator agreement is 83%. The best previous result from the literature on MSRP reported in Socher et al. [2011a] was an accuracy of 76.1%, but since then Madnani et al. [2012] has reported an accuracy of 77.4% on MSRP using several machine translation evaluation techniques as features for a linear classifier. Classifying sentences as paraphrases using the euclidean distance between their TNN encoding vectors yields an accuracy of 74.2%, which is below several recent results on MSRP from the literature. As a baseline and to combine with other methods, Socher et al. [2011a] compute the following features from a sentence pair: the difference in sentence length, the percentage of words and phrases in the first sentence that are in

the other (and the reverse), and three simple features for whether the two sentences contain exactly the same numbers, no numbers, or a strict subset of the numbers in the other sentence. Using only these hand coded features and a linear classifier, they achieve an accuracy of 73.2%. Using only the pooled matrix resulting from their dynamic pooling algorithm as input to the final classifier results in an accuracy of 72.6% which is worse than the 74.2% from using the top node of the TNN and worse than the hand coded feature baseline. Just using the histogram of values in the rectangular similarity matrix S yields an accuracy of 73.0%, about the same as the hand coded features and outperforming the dynamic pooling matrix. None of these features work well by themselves and the dynamic pooling feature vector is particularly bad. However, combining dynamic pooling with the hand coded features achieves the best result reported in Socher et al. [2011a]: an accuracy of 76.8%.

The most enticing aspects of the entire TNN autoencoder framework are the parallels to the recursion we believe is a fundamental property of human language and the promise of encoding an entire (short) utterance as a single vector of fixed dimensionality. If the root vector does not succeed at encoding the sentence (and requires something like the dynamic pooling matrix) then the model loses much of its appeal, even if fully supervised versions could be useful for certain tasks. Although nearest neighbors of encoded vectors provide some insight into what the TNN autoencoder has learned, a full generative model of sentences that we were able to sample from would be the most useful. Since TNNs are not complete generative models, it is very difficult to investigate why they do not work better than they do. One possible explanation for the weaker results of TNNs on semantic tasks is that, by mirroring the parse trees during encoding, the TNN encodes too much information about syntax. The best paraphrase detection model would be insensitive to meaning-preserving syntactic transformations and variations. In an analogy to scene understanding in computer vision, we might think of the syntactic structure of a sentence as being akin to a viewpoint. Switching from the active voice to the passive voice in many cases does not substantially alter the meaning conveyed by a sentence. Similarly, moving the camera a bit does not change what objects are in a scene nor does it change how the objects relate to each other. If we could automatically transform sentences in (approximate) meaning-preserving ways, we could explicitly train models to be invariant to these transformations, but I am not aware of any work in this direction, nor do I think it would necessarily be fruitful. However, using dependency parse trees (parse trees that only have edges between words) instead of constituent parse trees is one way to make TNNs that focus more on actions and agents instead of the details of word order and syntactic expression [Socher et al., 2014]. In section 4.3, we will describe a fully generative neural model of dependency parsed sentences related to the dependency tree TNNs of Socher et al. [2014]. However, in the next section we will return our attention to windowed models in order to solve a vexing computational problem with training RBMs on text data.

4.2 Training restricted Boltzmann machines on word observations

We have just seen how connectionist techniques have been used successfully in next step prediction language models, windowed/convolutional models of text, and tree neural net models of text. Neural language models require word embeddings and hierarchical softmax output layers to be computationally feasible for large vocabulary datasets. The windowed and convolutional models we discussed in section 4.1.2 use the same tricks, but none of the models from Collobert et al. [2011] used fully generative unsu-

pervised learning or were based on RBMs, in part because using RBMs for high-dimensional multinomial observations poses significant difficulties. For word observations from a large vocabulary (e.g., 100,000 words), the issue is not one of representing such observations in the RBM framework: so-called *softmax* units are the natural choice for modeling words. The issue is that manipulating distributions over the states of such units is expensive even for intermediate vocabulary sizes and becomes impractical for vocabulary sizes in the hundred thousands — a typical situation for NLP problems. For example, with a vocabulary of 100,000 words, modeling n-gram windows of size $n = 5$ is comparable in scale to training an RBM on binary vector observations of dimension 500,000 (i.e., more dimensions than a 700×700 pixel image). This scalability issue has been a primary obstacle to using the RBM for natural language processing.

The conventional approach to training RBMs on word observations is limited because it requires sampling the states of K-way softmax visible units during block Gibbs updates, an operation that takes time linear in K. In this section, we address this issue by replacing the Gibbs sampling transition kernel over the visible units with carefully implemented Metropolis-Hastings transitions, yielding updates with computational complexity independent of K. We demonstrate the success of our approach by training RBMs on hundreds of millions of word n-grams using larger vocabularies than previously feasible with RBMs. By training RBMs in this way, they learn representations capturing meaningful syntactic and semantic properties of words. Our learned word representations provide benefits on a chunking task competitive with other methods of inducing word representations and our learned n-gram features yield even larger performance gains. Finally, we also show how similarly extracted n-gram representations can be used to obtain state-of-the-art performance on a sentiment classification benchmark.

4.2.1 Restricted Boltzmann Machines on Word Observations

Recall from section 1.4 that the gradient of the log likelihood with respect to the RBM parameters θ, given by

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta} = \frac{1}{T} \sum_t \mathbb{E}_{\mathbf{h}|\mathbf{v}_t} \left[\frac{\partial E(\mathbf{v}_t, \mathbf{h})}{\partial \theta} \right] - \mathbb{E}_{\mathbf{v},\mathbf{h}} \left[\frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \theta} \right],$$

is intractable to compute because of the exponentially many terms in the sum over joint configurations of the visible and hidden units in the second expectation. Fortunately, for a given θ, we can approximate this gradient by replacing the second expectation with a Monte Carlo estimate based on a set of M samples $\mathcal{N} = \{\tilde{\mathbf{v}}_m\}$ from the RBM's distribution:

$$\mathbb{E}_{\mathbf{v},\mathbf{h}} \left[\frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \theta} \right] \approx \frac{1}{M} \sum_{\tilde{\mathbf{v}}_m \in \mathcal{N}} \mathbb{E}_{\mathbf{h}|\tilde{\mathbf{v}}_m} \left[\frac{\partial E(\tilde{\mathbf{v}}_m, \mathbf{h})}{\partial \theta} \right] . \tag{4.7}$$

The samples $\{\tilde{\mathbf{v}}_m\}$ are often referred to as "negative samples" as they counterbalance the gradient due to the observed, or "positive" data. To obtain these samples, we maintain M parallel Markov chains throughout learning and update them using Gibbs sampling between parameter updates.

Learning with the Monte Carlo estimator alternates between two steps: 1) Using the current parameters θ, simulate a fews steps of the Markov chain on the M negative samples using the conditionals $P(\mathbf{h}|\mathbf{v})$ and $P(\mathbf{v}|\mathbf{h})$ (in the binary case given by equations 1.17 and 1.18); and 2) Using the negative samples, along with a mini-batch (subset) of the positive data, compute the gradient in Eq. (4.7) and

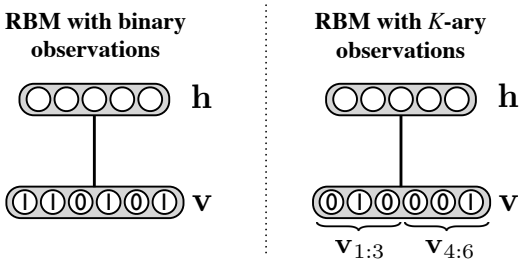


Figure 4.1: Illustration of an RBM with binary observations (left) and K-ary observations, for $n = 2$ and $K = 3$, i.e. a pair of 3-ary observations (right).

update the parameters. This procedure belongs to the general class of Robbins-Monro stochastic approximation algorithms [Younes, 1989]. Under mild conditions, which include the requirement that the Markov chain operators leave $p(\mathbf{v}, \mathbf{h} \,|\, \theta)$ invariant, this procedure will converge to a stable point of $\mathcal{L}(\theta)$.

For K-ary observations — observations belonging to a finite set of K discrete outcomes — we can use the same energy function as in Eq. (1.15) for the binary RBM by encoding each observation in a "one-hot" representation. Since in general we will have multiple K-ary observations (several words) at once, we also concatenate the representations of all observations from a single data case to construct $\mathbf{v}$. In other words, for n separate K-ary observations, the visible units $\mathbf{v}$ will be partitioned into n groups of K binary units, with the constraint that each partition must contain exactly one non-zero entry. Using the notation $\mathbf{v}_{a:b}$ to refer to the subvector of elements from index a to index b, the i^{th} observation will then be encoded by the group of visible units $\mathbf{v}_{(i-1)K+1:iK}$. The one-hot encoding is enforced by constraining each group of units to contain only a single 1-valued unit, the others being set to 0. The difference between RBMs with binary and K-ary observations is illustrated in figure 4.1. To simplify notation, we refer to the i^{th} group of visible units as $\mathbf{v}^{(i)} = \mathbf{v}_{(i-1)K+1:iK}$. Similarly, we will refer to the biases and weights associated with those units as $\mathbf{b}^{(i)} = \mathbf{b}_{(i-1)K+1:iK}$ and $\mathbf{W}^{(i)} = \mathbf{W}_{\cdot,(i-1)K+1:iK}$. We will also denote with $\mathbf{e}_k$ the one-hot vector with its k^{th} component set to 1.

The conditional distribution over the visible layer is

$$p(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{n} p(\mathbf{v}^{(i)}|\mathbf{h}) \tag{4.8}$$

$$p(\mathbf{v}^{(i)} = \mathbf{e}_k|\mathbf{h}) = \frac{\exp({\mathbf{b}^{(i)}}^\top \mathbf{e}_k + \mathbf{h}^\top \mathbf{W}^{(i)} \mathbf{e}_k)}{\sum_{k'} \exp({\mathbf{b}^{(i)i}}^\top \mathbf{e}_{k'} + \mathbf{h}^\top \mathbf{W}^{(i)} \mathbf{e}_{k'})}.$$

Each group $\mathbf{v}^{(i)}$ has a multinomial distribution given the hidden layer. Because the multinomial probabilities are given by a softmax nonlinearity, the group of units $\mathbf{v}^{(i)}$ are referred to as softmax units.

4.2.2 Difficulties with Word Observations

While in the binary case the size of the visible layer is equal to the data dimensionality, in the K-ary case the size of the visible layer is K times the dimensionality. For language processing applications, where K is the vocabulary size and can run into the hundred thousands, the visible layer can become unmanageably large.

The difficulty with large K is that the Gibbs operator on the visible units becomes expensive to

simulate, making it difficult to perform updates of the negative samples. That is, generating a sample from the conditional distribution in Eq. (4.8) dominates the stochastic learning procedure as K increases. The reason for this expense is that it is necessary to compute the activity associated with *each* of the K possible outcomes, even though only a *single* one will actually be selected.

On the other hand, given a mini-batch $\{\mathbf{v}_t\}$ and negative samples $\{\tilde{\mathbf{v}}_m\}$, the gradient computations in Eq. (4.7) are able to take advantage of the sparsity of the visible activity. Since each $\mathbf{v}_t$ and $\tilde{\mathbf{v}}_m$ only contain n non-zero entries, the cost of the gradient estimator has no dependence on K and can be rapidly computed. Thus the only barrier to efficient learning of high-dimensional multinomial RBMs is the complexity of the Gibbs update for the visible units.

As we discussed in section 4.1.1, the issue of large multinomial distributions has come up previously in work on neural network language models. For example, Morin and Bengio [2005] addressed this problem by introducing a fixed factorization of the (conditional) multinomial using a binary tree in which each leaf is associated with a single word. Unfortunately, tree-structured solutions are not applicable to the problem of modeling the *joint* distribution of n consecutive words, as we wish to do here. Introducing a directed tree breaks the undirected, symmetric nature of the interaction between the visible and hidden units of the RBM. While one strategy might be to use a conditional RBM to model the tree-based factorizations, similar to Mnih and Hinton [2009], the end result would not be an RBM model of n-gram word windows, nor would it even be a conditional RBM over the next word given the $n-1$ previous ones. Thus dealing with K-ary observations in the Boltzmann machine framework for large K has been a crucial open problem that has inhibited the development of deep learning solutions to NLP problems.

4.2.3 Metropolis–Hastings for Softmax Units

Having identified the Gibbs update of the visible units as the limiting factor in efficient learning of large-K multinomial observations, it is natural to examine whether other operators might be used for the Monte Carlo estimate in Eq. (4.7). In particular, we desire a transition operator that can take advantage of the same sparse operations that enable the gradient to be efficiently computed from the positive and negative samples, while still leaving $p(\mathbf{v}, \mathbf{h})$ invariant and thus still satisfying the convergence conditions of the stochastic approximation learning procedure.

To obtain a more efficient operator, instead of sampling exactly from the conditionals $p(\mathbf{v}^{(i)}|\mathbf{h})$ within the Markov chain, we use a small number of iterations of Metropolis–Hastings (M–H) sampling. Let $q(\hat{\mathbf{v}}^{(i)} \leftarrow \mathbf{v}^{(i)})$ be a proposal distribution for group i. The following stochastic operator leaves $p(\mathbf{v}, \mathbf{h})$ invariant:

1. Given the current visible state $\mathbf{v}$, sample a proposal $\hat{\mathbf{v}}$ for group i, such that $\hat{\mathbf{v}}^{(i)} \sim q(\hat{\mathbf{v}}^{(i)} \leftarrow \mathbf{v}^{(i)})$ and $\hat{\mathbf{v}}^{(j)} = \mathbf{v}^{(j)}$ for $i \neq j$ (i.e. sample a proposed new word for position i).

2. Replace the ith part of the current state $\mathbf{v}^{(i)}$ with $\hat{\mathbf{v}}^{(i)}$ with probability:

$$\min\left\{1, \frac{q(\mathbf{v}^{(i)} \leftarrow \hat{\mathbf{v}}^{(i)}) \exp({\mathbf{b}^{(i)}}^\top \hat{\mathbf{v}}^{(i)} + \mathbf{h}^\top \mathbf{W}^{(i)} \hat{\mathbf{v}}^{(i)})}{q(\hat{\mathbf{v}}^{(i)} \leftarrow \mathbf{v}^{(i)}) \exp({\mathbf{b}^{(i)}}^\top \mathbf{v}^{(i)} + \mathbf{h}^\top \mathbf{W}^{(i)} \mathbf{v}^{(i)})}\right\}.$$

Assuming it is possible to efficiently sample from the proposal distribution $q(\hat{\mathbf{v}}^{(i)} \leftarrow \mathbf{v}^{(i)})$, this M–H operator is fast to compute as it does not require normalizing over all possible values of the visible units

in group i and, in fact, only requires the unnormalized probability of one of them. Moreover, as the n visible groups are conditionally independent given the hidden states, each group can be simulated in parallel (i.e., words are sampled at every position separately). The efficiency of these operations make it possible to apply this transition operator many times before moving on to other parts of the learning and still obtain a large speedup over exact sampling from the conditional.

Efficient Sampling of Proposed Words

The utility of M–H sampling for an RBM with word observations relies on the fact that sampling from the proposal $q(\hat{\mathbf{v}}^{(i)} \leftarrow \mathbf{v}^{(i)})$ is much more efficient than sampling from the correct softmax multinomial. Although there are many possibilities for designing such proposals, here we will explore the most basic variant: *independence chain* Metropolis–Hastings in which the proposal distribution is fixed to be the marginal distribution over words in the corpus.

Naïve procedures for sampling from discrete distributions typically have linear time complexity in the number of outcomes. However, the *alias method* of Kronmal and Perterson [1979] can be used to generate samples in constant time with linear setup time (pseudocode in appendix B). While the alias method would not help us construct a Gibbs sampler for the visibles, it does make it possible to generate proposals extremely efficiently, which we can then use to simulate the Metropolis–Hastings operator, regardless of the current target distribution.

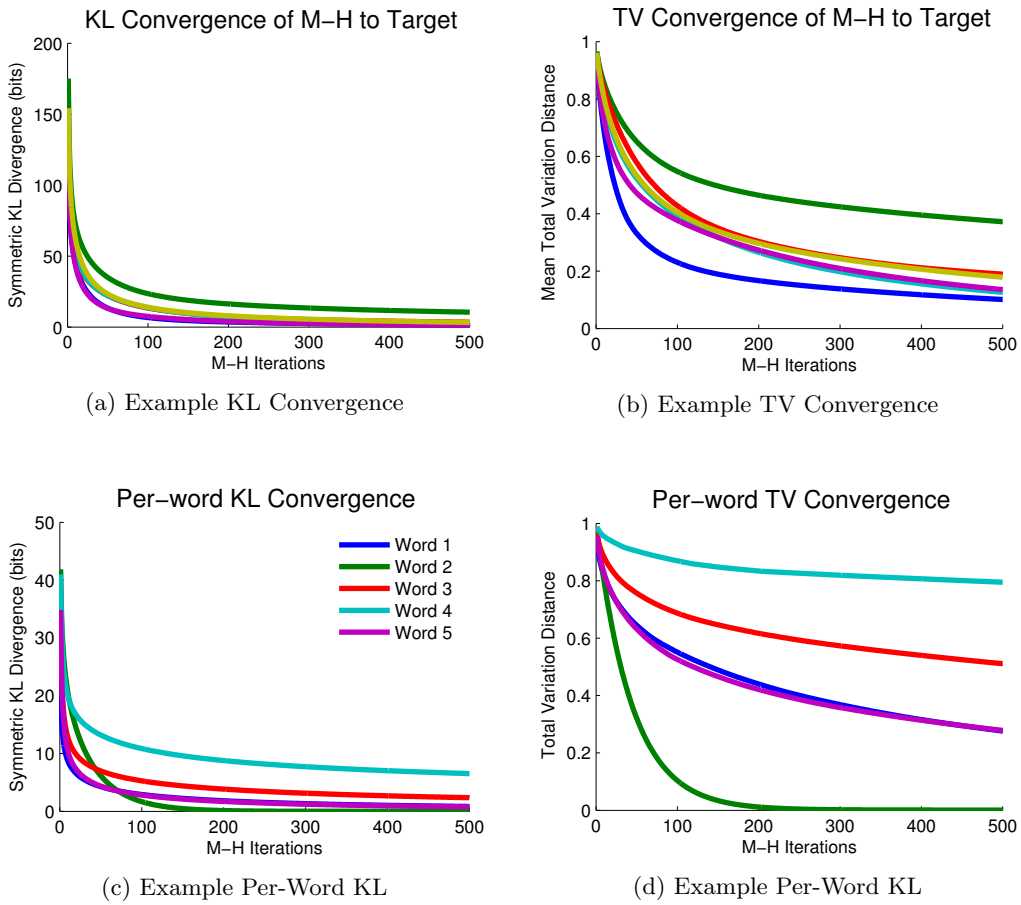
The alias method leverages the fact that any K-valued discrete distribution can be written as a uniform mixture of K, two-valued discrete distributions. Having constructed this mixture distribution at setup time (with linear time and space cost), new samples can be generated in constant time by sampling uniformly from the K mixture components, followed by sampling from that component's Bernoulli distribution.

Mixing of Metropolis–Hastings

Although this procedure eliminates dependence of the learning algorithm on K, it is important to examine the mixing of Metropolis–Hastings and how sensitive it is to K in practice. Although there is evidence [Hinton, 2002] that poorly-mixing Markov chains can yield good learning signals, when this will occur is not as well understood. We examined the mixing issue using the model described in section 4.2.5 with the parameters learned from the Gigaword corpus with a 100,000-word vocabulary as described in section 4.2.6.

We analytically computed the distributions implied by iterations of the M–H operator, assuming the initial state was drawn according to $\prod_i q(\mathbf{v}^{(i)})$. As this computation requires the instantiation of n $100\mathsf{k} \times 100\mathsf{k}$ matrices, it cannot be done at training time, but was done offline for analysis purposes. Each application of Metropolis–Hastings results in a new distribution converging to the target (true) conditional.

Figures 4.2a and 4.2b show this convergence for the "reconstruction" distributions of six randomly-chosen 5-grams from the corpus, using two metrics: symmetric Kullback–Leibler (KL) divergence and total variation (TV) distance, which is the standard measure for analysis of MCMC mixing. The TV distance shown is the mean across the five group distributions. Figures 4.2c and 4.2d show these metrics broken down by grouping, for the slowest curves (dark green) of the top two figures. These curves highlight that the state of the hidden units has a strong impact on the mixing and that most groups



(a) Example KL Convergence

(b) Example TV Convergence

(c) Example Per-Word KL

(d) Example Per-Word KL

Figure 4.2: Convergence of the Metropolis–Hastings operator to the true conditional distribution over the visibles for a trained 5-gram RBM with a vocabulary of 100K words. (a) KL divergence for six randomly-chosen data cases. (b) Average total variation distance for the same six cases. (c,d) For the slowest-converging of the six top curves (dark green), these are broken down for each of the five multinomials in KL and total variation, respectively. [Best viewed in color.]

mix very quickly while a few converge slowly. These curves, along with the results of sections 4.2.6 and 4.2.7, indicate that the mixing is effective, although it could benefit from further study.

4.2.4 Differences from Previous Work

Using M–H sampling for a multinomial distribution with softmax probabilities has been explored in the context of a neural network language model by Bengio and Senécal [2003]. They used M–H to estimate the training gradient at the output of the neural network. However, their work did not address or investigate its use in the context of Boltzmann machines.

Salakhutdinov and Hinton [2009b] describe an alternative to directed topic models called the *replicated softmax* RBM that uses softmax units over the entire vocabulary with tied weights to model an *unordered* collection of words (a.k.a. bag of words). Since their RBM ties the weights to all the words in a single document, there is only one vocabulary-sized multinomial distribution to compute per document, instead of the n required when modeling a window of consecutive words. Therefore sampling a document conditioned on the hidden variables of the replicated softmax still incurs a computational cost linear in

K, although the problem is not amplified by a factor of n as it would be in our experiments. Notably, Salakhutdinov and Hinton [2009b] limited their vocabulary to $K < 14,000$.

Previous work has not attempted to address the computational burden associated with K-ary observations with large K in RBMs. The M–H-based approach used here is not specific to a particular Boltzmann machine and could be used for any model with large softmax units, although the applications that motivate us come from NLP. Dealing with the large softmax problem is essential if Boltzmann machines are to be practical for natural language data.

In sections 4.2.6 and 4.2.7, we present results on the task of learning word representations. This task has been investigated previously by others. Turian et al. [2010] provide an overview and evaluation of these different methods, including those of Mnih and Hinton [2009] and of Collobert and Weston [2008]. We have already mentioned the work of Mnih and Hinton [2009], who model the conditional distribution of the last word in n-gram windows. Collobert and Weston [2008] follows a similar approach, by training a neural network to fill-in the middle word of an n-gram window, using a margin-based learning objective. In contrast, we model the *joint* distribution of the whole n-gram window, which implies that the RBM could be used to fill-in any word within a window. Moreover, inference with an RBM yields a hidden representation of the whole window and not simply of a single word.

4.2.5 RBM Model of n-gram Windows

We evaluated our M–H approach to training RBMs on two NLP tasks, chunking and sentiment classification, and both applications used the same RBM model of n-gram windows of words. In the standard parameterization presented in section 4.2.1, an RBM uses separate weights (i.e., different columns of $\mathbf{W}$) to model observations at different positions. When training an RBM on word n-gram windows, we would prefer to share parameters across identical words in different positions in the window and factor the weights into position-dependent weights and position-independent weights (word representations).

Therefore, we use an RBM parameterization very similar to that of Mnih and Hinton [2007], which itself is inspired by previous work on neural language models [Bengio et al., 2001]. The idea is to learn, for each possible word w, a lower-dimensional linear projection of its one-hot encoding by incorporating the projection directly in the energy function of the RBM. Moreover, we share this projection across positions within the n-gram window. Let $\mathbf{D}$ be the matrix of this linear projection and let $\mathbf{e}_w$ be the one-hot representation of w (where we treat w as an integer index in the vocabulary), performing this projection $\mathbf{D}\mathbf{e}_w$ is equivalent to selecting the appropriate column $\mathbf{D}_{\cdot,w}$ of this matrix. This column vector can then be seen as a real-valued vector representation of that word. The real-valued vector representations of all words within the n-gram are then concatenated and connected to the hidden layer with a single weight matrix.

More specifically, let $\mathbf{D}$ be the $D \times K$ matrix of word representations. These word representations are introduced by re-parameterizing $\mathbf{W}^{(i)} = \mathbf{U}^{(i)}\mathbf{D}$, where $\mathbf{U}^{(i)}$ is a position-dependent $H \times D$ matrix. The biases across positions are also shared, i.e., we learn a single bias vector $\mathbf{b}^*$ that is used at all positions ($\mathbf{b}^{(i)} = \mathbf{b}^* \ \forall i$). The energy function becomes

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{c}^\top \mathbf{h} + \sum_{i=1}^{n} -{\mathbf{b}^*}^\top \mathbf{v}^{(i)} - \mathbf{h}^\top \ \mathbf{U}^{(i)} \ \mathbf{D}\mathbf{v}^{(i)}$$

with conditional distributions

$$p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v})$$
$$p(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{n} p(\mathbf{v}^{(i)}|\mathbf{h})$$
$$p(h_j = 1|\mathbf{v}) = \text{sigm}\left(c_j + \sum_{i=1}^{n} \mathbf{U}^{(i)}_{j\cdot}\ \mathbf{D}\mathbf{v}^{(i)}\right)$$
$$p(\mathbf{v}^{(i)} = \mathbf{e}_k|\mathbf{h}) = \frac{\exp({\mathbf{b}^*}^\top \mathbf{e}_k + \mathbf{h}^\top \mathbf{U}^{(i)}\ \mathbf{D}\mathbf{e}_k)}{\sum_{k'=1}^{K} \exp({\mathbf{b}^*}^\top \mathbf{e}_{k'} + \mathbf{h}^\top \mathbf{U}^{(i)}\ \mathbf{D}\mathbf{e}_{k'})},$$

where $\mathbf{U}^{(i)}_{j\cdot}$ refers to the j^{th} row vector of $\mathbf{U}^{(i)}$. The gradients with respect to this parameterization are easily derived from Eq. (4.7). We refer to this construction as a word representation RBM (WRRBM).

In contrast to Mnih and Hinton [2007], rather than training the WRRBM conditionally to model $p(w_{n+t-1}|w_t, \ldots, w_{n+t-2})$, we train it using Metropolis–Hastings to model the full joint distribution $p(w_t, \ldots, w_{n+t-1})$. That is, we train the WRRBM based on the objective

$$\mathcal{L}(\theta) = -\sum_t \log p(\mathbf{v}^{(1)} = \mathbf{e}_{w_t}, \mathbf{v}^{(2)} = \mathbf{e}_{w_{t+1}}, \ldots, \mathbf{v}^{(n)}_{w_{n+t-1}})$$

using stochastic approximation from M–H sampling of the word observations. For models with $n > 2$, we also found it helpful to incorporate ℓ_2 regularization of the weights, and to use momentum when updating $\mathbf{U}^{(i)}$.

4.2.6 Chunking Experiments

As described by Turian et al. [2010], learning real-valued word representations can be used as a simple way of performing semi-supervised learning for a given method, by first learning word representations on unlabeled text and then feeding these representations as additional features to a supervised learning model.

We trained the WRRBM on windows of text derived from the English Gigaword corpus[5]. The dataset is a corpus of newswire text from a variety of sources. We extracted each news story and trained only on windows of n words that did not cross the boundary between two different stories. We used NLTK [Bird et al., 2009] to tokenize the words and sentences, and also corrected a few common punctuation-related tokenization errors. As in Collobert et al. [2011], we lowercased all words and delexicalized numbers (replacing consecutive occurrences of one or more digits inside a word with just a single # character). Unlike Collobert et al. [2011], we did not include additional capitalization features, but discarded all capitalization information. We used a vocabulary consisting of the 100,000 most frequent words plus a special "unknown word" token to which all remaining words were mapped.

We evaluated the learned WRRBM word representations on a chunking task, following the setup described in Turian et al. [2010] and using the associated publicly-available code, as well as CRFSuite[6]. As in Turian et al. [2010], we used data from the CoNLL-2000 shared task. We used a scale of 0.1 for the word representation features (as Turian et al. [2010] recommend) and for each WRRBM model,

[5] `http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC2005T12`
[6] `http://www.chokkan.org/software/crfsuite/`

Table 4.1: Comparison of experimental results on the chunking task. The baseline results were taken from Turian et al. [2010]. The performance measure is F1.

Model	Valid	Test
CRF w/o word representations	94.16	93.79
HLBL [Mnih and Hinton, 2009]	94.63	94.00
C&W [Collobert and Weston, 2008]	94.66	94.10
Brown clusters	94.67	94.11
WRRBM	94.82	94.10
WRRBM (with hidden units)	**95.01**	**94.44**

tried ℓ_2 penalties $\lambda \in \{0.0001, 1.2, 2.4, 3.2\}$ for CRF training. We selected the single model with the best validation F1 score over all runs and evaluated it on the test set. The model with the best validation F1 score used 3-gram word windows, $\lambda = 1.2$, 250 hidden units, a learning rate of 0.01, and used 100 steps of M–H sampling to update each word observation in the negative data.

The results are reported in Table 4.1, where we observe that word representations learned by our model achieved higher validation and test scores than the baseline of not using word representation features, and are comparable to the best of the three word representations tried in Turian et al. [2010][7].

Although the word representations learned by our model are highly effective features for chunking, an important advantage of our model over many other ways of inducing word representations is that it also naturally produces a feature vector for the entire n-gram. For the trigram model mentioned above, we also tried adding the hidden unit activation probability vector as a feature for chunking. For each word w_i in the input sentence, we generated features using the hidden unit activation probabilities for the trigram $w_{i-1}w_iw_{i+1}$. No features were generated for the first and last word of the sentence. The hidden unit activation probability features improved validation set F1 to 95.01 and test set F1 to 94.44, a test set result superior to all word embedding results on chunking reported in Turian et al. [2010].

As can be seen in Table 4.3, the learned word representations capture meaningful information about words. However, the model primarily learns word representations that capture syntactic information (as do the representations studied in Turian et al. [2010]), as it only models short windows of text and must enforce local agreement. Nevertheless, word representations capture some semantic information, but only after similar syntactic roles have been enforced. Although not shown in Table 4.3, the model consistently embeds the following natural groups of words together (maintaining small intra-group distances): days of the week, words for single digit numbers, months of the year, and abbreviations for months of the year. A 2D visualization of the word representations generated by t-SNE [van der Maaten and Hinton, 2008] is provided at `http://i.imgur.com/ZbrzO.png`.

4.2.7 Sentiment Classification Experiments

Maas et al. [2011] describe a model designed to learn word representations specifically for sentiment analysis. They train a probabilistic model of documents that is capable of learning word representations and leveraging sentiment labels in a semi-supervised framework. Even without using the sentiment labels, by treating each document as a single bag of words, their model tends to learn distributed

[7]Better results have been reported by others for this dataset: the spectral approach of Dhillon et al. [2011] used different (less stringent) preprocessing and a vocabulary of 300,000 words and obtained higher F1 scores than the methods evaluated in Turian et al. [2010]. Unfortunately, the vocabulary and preprocessing differences mean that neither our result nor the one in Turian et al. [2010] are directly comparable to Dhillon et al. [2011].

Table 4.2: Experimental results on the sentiment classification task. The baseline results were taken from Maas et al. [2011]. The performance measure is accuracy (%).

Model	Test
LDA	67.42
LSA	83.96
Maas et al. [2011]'s "full" method	87.44
Bag of words "bnc"	87.80
Maas et al. [2011]'s "full" method + bag of words "bnc"	88.33
Maas et al. [2011]'s "full" method + bag of words "bnc" + unlabeled data	88.89
WRRBM	87.42
WRRBM + bag of words "bnc"	**89.23**

representations for words that capture mostly semantic information since the co-occurrence of words in documents encodes very little syntactic information. To get the best results on sentiment classification, they combined features learned by their model with bag-of-words feature vectors (normalized to unit length) using binary term frequency weights (referred to as "bnc").

We applied the WRRBM to the problem of sentiment classification by treating a document as a "bag of n-grams", as this maps well onto the fixed-window model for text. At first glance, a word representation RBM might not seem to be a suitable model for learning features to improve sentiment classification. A WRRBM trained on the phrases "this movie is wonderful" and "this movie is atrocious" will learn that the word "wonderful" and the word "atrocious" can appear in similar contexts and thus should have vectors near each other, even though they should be treated very differently for sentiment analysis. However, a class-conditional model that trains separate WRRBMs on n-grams from documents expressing positive and negative sentiment avoids this problem.

We trained class-specific, 5-gram WRRBMs on the labeled documents of the Large Movie Review dataset introduced by Maas et al. [2011], independently parameterizing words that occurred at least 235 times in the training set (giving us approximately the same vocabulary size as the model used in Maas et al. [2011]).

To label a test document using the class-specific WRRBM, we fit a threshold to the difference between the average free energies assigned to n-grams in the document by the positive-sentiment and negative sentiment models. We explored a variety of different metaparameters (number of hidden units, training parameters, and n) for the pairs of WRRBMs and selected the WRRBM pair giving best training set classification performance. This WRRBM pair yielded 87.42% accuracy on the test set.

We additionally examined the performance gain by appending to the bag-of-words features the average n-gram free energies under both class-specific WRRBMs. The bag-of-words feature vector was weighted and normalized as in Maas et al. [2011] and the average free energies were scaled to lie on $[0, 1]$. We then trained a linear SVM to classify documents based on the resulting document feature vectors, giving us 89.23% accuracy on the test set. This result is the best known result on this benchmark and, notably, our method did not make use of the unlabeled data.

Table 4.3: The five nearest neighbors (in the word feature vector space) of some sample words.

could	**spokeswoman**	**suspects**	**science**	**china**	**mother**	**sunday**
should	spokesman	defendants	sciences	japan	father	saturday
would	lawyer	detainees	medicine	taiwan	daughter	friday
will	columnist	hijackers	research	thailand	son	monday
can	consultant	attackers	economics	russia	grandmother	thursday
might	secretary-general	demonstrators	engineering	indonesia	sister	wednesday
tom	**actually**	**probably**	**quickly**	**earned**	**what**	**hotel**
jim	finally	certainly	easily	averaged	why	restaurant
bob	definitely	definitely	slowly	clinched	how	theater
kevin	rarely	hardly	carefully	retained	whether	casino
brian	eventually	usually	effectively	regained	whatever	ranch
steve	hardly	actually	frequently	grabbed	where	zoo

4.2.8 Conclusion

We have described a method for training RBMs with large K-ary softmax units that results in weight updates with a computational cost independent of K, allowing for efficient learning even when K is large. Using our method, we were able to train RBMs that learn meaningful representations of words and n-grams. Our results demonstrated the benefits of these features for chunking and sentiment classification and could easily be applied to RBM-based models on other NLP tasks. Although the simple proposal distribution we used for M-H updates in this work is effective, more sophisticated proposal distributions is an exciting prospect for future work. Unlike the windowed models of Collobert and Weston that we described in section 4.1.2, windowed models based on RBMs are fully generative models of text, even if obtaining samples from the model is nontrivial. Nevertheless, even if we made deeper versions based on Boltzmann machines, we would still have a model of a finite window of text. In the remainder of this chapter we will explore a fully generative neural model of complete sentences instead of fixed size sequences of words.

4.3 A neural generative model of dependency parses

Although they have all been worthy of study, the statistical models of natural language text we have discussed so far each have important limitations. In general, no model we create will be perfect in every way, but science advances by correcting deficiencies in previous work and by making new choices on what parts of a problem we will ignore and what parts we will engage with directly. Previous work sits in a multidimensional design space with individual models dealing with some aspects of language well and others poorly. The models we discuss in this section will combine improvements from previous models of text in several key ways while still remaining simple and idealized enough to investigate easily.

The most basic decision one faces when constructing a new statistical model of text is what level of granularity the model should use. The sentence is a natural unit of text, but it poses many challenges for machine learning models designed to operate on vectors of fixed dimensionality. Sentences can vary quite a bit in length and are large enough units to express very complicated ideas and exhibit all the syntactic sophistication of unrestricted natural language text. In English, at least, sentence boundaries are usually easy to detect in formal written text, but, even in English, applications using automatically transcribed

spoken text may have trouble segmenting sentences. Windowed models that ignore the issues caused by variations in input length and uncertainty in boundary locations can only ever be means to an end. Ideally, we would not only have models that are aware of the divisions between sentences, but also ones that can handle the hierarchical subdivisions of text below and above the sentence level, from words, clauses, and phrases to sentences, paragraphs, and documents. Without compelling statistical models of single sentences, however, modeling documents more effectively than as unordered bags of words might be overly ambitious. In this section, we will construct models of sentences because a sentence is a large enough unit to express a complete thought, but a small enough unit that we can avoid the most difficult discourse context effects and plausibly find reasonable low dimensional representations that capture much of the interesting information in the data.

In addition to using the sentence as the unit of analysis, we will also construct fully generative models that learn the joint distribution of all their input. TNNs, recurrent neural language models, and convolutional models of text with sentence-level pooling can all model sentences, but of those three only RNN language models are generative models. Generative models let us sample from the model to see what it has learned and they are also useful for tasks that require producing new text as output. Although they are complete generative models of sequences of words, flat language models that only generate words are incomplete syntactically. Samples from language models do not reveal the model's syntactic intent, even for samples that happen to be well-formed linguistically, simply because there is no meaningful notion of a syntactic intent for a statistical model of flat sequences of words. A well-formed sample might have substantial syntactic ambiguity because the model has no latent variable states that obviously correspond to particular sets of relationships between entities. Unlike current language models, when humans communicate, we might utter a highly ambiguous sentence, but there is no parsing ambiguity for the *speaker* even if there is ambiguity for the listener.[8] Syntactically unambiguous generative models of sentences exist, although they are not connectionist. Theories of syntax from linguistics have given rise to probabilistic generative models of sentences known as stochastic grammars. A stochastic grammar encodes a distribution over sequences of terminal symbols (words in this case) using a weighted set of production rules. A production rule replaces a piece of a sequence of non-terminal symbols with another sequence of (potentially both terminal and non-terminal) symbols. Given a special starting non-terminal symbol S, to generate a sequence of terminals we select applicable production rules at random with probabilities determined by their weights, normalized over all applicable rules, and apply these rules until only terminals remain. The most common type of stochastic grammar in use is the probabilistic context free grammar (PCFG). Since PCFGs are based on context free grammars, production rules only replace a single non-terminal symbol at a time. Although PCFGs can generate a sentence and its parse simultaneously, when learned from data they are a poor model of actual text compared to RNN language models (or even compared to smoothed n-gram models). PCFGs do not make use of distributed representations. Assuming a set of N possible non-terminal symbols, knowing the identity of the non-terminal that generates a given subsequence gives us at most $\log N$ bits of information about the subsequence on average. This representational inefficiency might be fine when we consider generating part of speech sequences, but if we want to generate actual words we will need more information to flow past constituent boundaries and far more flexible and efficient probabilistic models. If we could combine

[8] One might argue that this is true for semantics as well, but the essence of meaning is so slippery that, depending on how it is defined, the speaker's meaning might be ambiguous to the speaker as well. Even ignoring the philosophical issues, coarsely defining and encoding the set of allowed syntactic relations between entities in a particular language is a much more feasible project.

more effective probabilistic models based on neural networks with ideas from stochastic grammars, then perhaps we could construct a generative model of isolated sentences that was a good model of real text while also generating fully parsed samples.

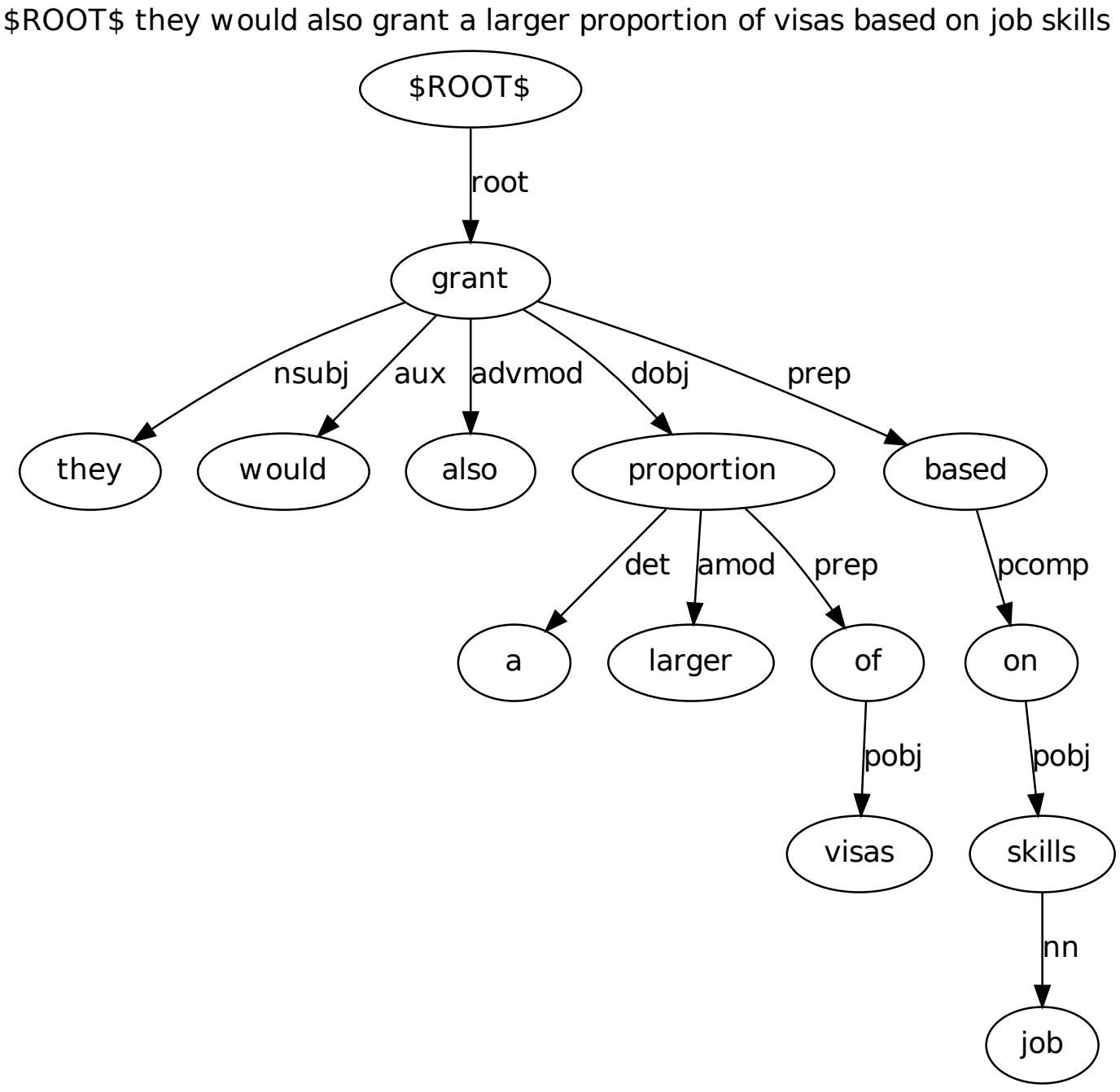
TNNs operate on parsed sentences and can even be used to perform parsing, so it worth explaining explicitly why they are not a connectionist stochastic grammar. Obviously, TNNs are not a generative model of text: they are either discriminative models or autoencoders. Even if we imagined putting a prior over the encoding vector for a whole sentence, we would need a decoder that did not condition on the parse tree. Furthermore, TNNs invert the grammar. Even if we accept that the parse tree describes a good generative process for building a sentence, there is no reason to think that the structure of the network doing inference should mirror the generative process. And indeed, to make TNNs more effective for parsing, Socher et al. [2011b] was forced to break the TNN encoder framework by using words outside the span of the current subtree as input to the encoder. Having the architecture of the inference network mirror the structure of the parse tree makes no sense since the parse tree describes sampling a sentence from a PCFG. Therefore, since TNNs do not achieve our goals, we must invent a new model if we want a fully generative model of parsed sentences.

4.3.1 Dependency Parsed Sentences

The models we work with in the rest of this chapter will be stochastic neural grammars of dependency parsed sentences. Thus they will learn a distribution over sentences along with their parses. The specific type of parses we will model are known as dependency parses. For our purposes, a dependency parse is a syntactic analysis of an utterance that uses a directed edge of type r_t linking w_i to w_j to indicate that governor word w_i and dependent word w_j are members of the syntactic relation r_t. We assume that every word in an utterance, except the root, has exactly one governor word it depends on and that the dependency parse forms a tree over the words of the utterance. A (dependency) parser consumes as input a surface form sentence, which is simply a sequence of words, and produces as output the appropriate dependency tree for the sentence. See figure 4.3 for an example of a dependency parse tree produced by the Stanford parser from the Stanford CoreNLP package [Manning et al., 2014]. The root word (typically the main verb) of the sentence in figure 4.3 is "grant" and in this example the parse indicates that the words "they" and "grant" are in the noun subject relation (i.e. "they" is the subject of the verb "grant"). The parse also identifies "would" as an auxiliary verb dependent on "grant," "proportion" as the head word of the direct object of "grant," "job" as a noun compound modifier of "skills," and so on. The parse is correct, but an alternative parse that had "based" as a dependent of "visas" instead of "grant" would also be correct and might even be preferable.

Our decision to work with dependency parses instead of constituent parses has important consequences. Dependency trees have different qualitative properties than constituent trees. Dependency trees induce a one-to-one correspondence between nodes in the dependency tree and words, whereas constituent trees induce a one-to-many correspondence between words and tree nodes since a word might be a member of multiple constituent subtrees. Dependency trees are generally flatter than constituent trees and emphasize the interactions between head words. Instead of the non-terminal nodes constituent parses use, dependency parses encode similar information in edge labels. In some cases, dependency parses are more robust to certain syntactic changes and word order variations, such as active to passive voice, since they highlight the actions and agents in a sentence. We will revisit the issue of modeling dependency parses vs modeling constituency parses in section 4.3.2 when we describe the

Figure 4.3: Surface form followed by a corresponding dependency parse tree produced by the Stanford Parser.



details of the models since choosing to model dependency parses gives us a clear way to construct a fully observed probabilistic model, resulting in a very simple training algorithm.

In order to easily train generative models of parsed sentences, we will need training data that includes parse trees. State of the art English dependency parsers regularly make mistakes on English text from the wild, but many of their parses are correct and, for well-formed English that has limited syntactic ambiguity from the same domain as the parser's training data, they can get very good results. However, even very good parsers struggle with ambiguous sentences and do not pick the parse most human annotators would think is most probable for many challenging sentences (one will quickly become convinced of this after trying online demos of these parsers). Nevertheless, even for English, manually annotated treebanks are of limited size. The most interesting NLP methods are those that scale to the large volumes of unannotated text available on the internet, so we will only work with parses produced from unannotated text completely automatically.

A probabilistic generative model of dependency parsed sentence could have many uses. Most obviously such a model could serve as a prior to improve parsers (if it was trained on gold standard parses). There are also potential applications in abstractive summarization, natural language text generation, and sentence fusion, especially if the model doubles as a good generative model of surface form sentences. A high enough quality model could even be used in traditional language modeling applications, such as machine translation. These applications, however, will be beyond the scope of this chapter. Most specific applications will depend on how well the generative models capture the structure in sentences, so in this chapter we will quantitatively and qualitatively compare the different generative models we build and see to what extent they generate well-formed samples.

4.3.2 Design of the model

As a graph with words as nodes, a dependency parse need not completely specify the surface form of the sentence. There may be multiple surface form sentences with the same dependency parse and, even when a unique surface form corresponds to any given parse, the correct surface form might not be obvious from the parse. The final generative model of dependency parsed sentences we use in most experiments generates the surface form sentence along with the dependency parse tree, but we will first describe a precursor to that model we call the unordered model. Although the unordered model has numerous flaws, these flaws motivate the design of the final model and reveal some of the challenges of building generative models of dependency parsed sentences.

Initial unordered model

Both the initial unordered model and the final ordered model break down the process of generating a dependency parse tree into modules. Since a dependency tree is a collection of nodes and edges where each node has a word and each edge linking a parent node to a child node has an edge type, any model must generate edge types, words (which may be thought of as node types), and the graph structure. The unordered model has two modules, each parameterized as a neural net: the edge model and the node model. The edge model generates as output the histogram of outgoing edge types for a node and the node model generates a word for a new child node. To sample a tree, we start at the root word node, generate a word with the node model and then generate its outgoing edges. For each outgoing edge, we create a new node and generate a word for the node using the node model. A leaf node is a

node with no outgoing edges. Although it is possible for the edge model to generate a zero count for all possible edge types, the implied leaf probability is not well calibrated. Adding a separate binary output to indicate whether a node is a leaf or not solves this problem. However, there are two larger problems with this way of generating trees. First, the edge model has to commit to all the outgoing edge types for a node before seeing the actual child words for any of them. Second, generating all outgoing edges for a node at once means that the dependency parses require surface realization because the generative model does not specify the order of the edges. Surface realization (also known as linearization in the case of dependency parse trees) is the process that extracts a surface form from a dependency parse tree. Since dependency parse trees need not contain information on the order of words, only information about their syntactic relationships, a surface realization engine must pick one of the $n!$ orderings for an n word tree (although it is not nearly this bad in English since heuristics can constrain the search space).[9] Addressing these two problems led to the creation of the second family of models.

Final ordered model

The final ordered model generates dependency parse trees annotated with local surface ordering information. Local surface order information combined with an additional assumption makes it simple to produce a surface form from the trees we sample from the model. Let a treelet be a subgraph of a tree consisting of a node and its immediate children. A dependency parse annotated with local ordering information has a total order over the nodes in any given treelet (i.e. the local ordering information) which together imply a partial order over all of the nodes in the entire tree. Let the projective order of the nodes in an annotated dependency tree be the total ordering over nodes that:

1. is consistent with the local ordering information and

2. for two subtrees A and B, rooted in sibling nodes a and b respectively, if a precedes b in the local order in their treelet then all nodes in A precede all nodes in B.

We call a dependency parse tree for a sentence projective when there exists a local order for each treelet such that the projective total order equals the correct surface form word order. In English, projective trees are generally sufficient to analyze most constructions and many parsers assume projective dependencies, although parsers can differ subtly in their dependency formalism and some even produce non-tree dependency parses. Some treebanks are even entirely projective simply based on their annotation conventions. If we arrange the nodes of a dependency parse tree in a line based on their surface order and draw all edges above the nodes, a non-projective tree will have edge crossings. Although many English sentences can be analyzed with projective dependency trees, there are perfectly grammatical English sentences that are fundamentally non-projective. For example, the sentence *"A hearing is scheduled on the issue today."* requires a non-projective dependency parse because there is an edge between "hearing" and "on" as well as an edge between "scheduled" and "today." If we run the Stanford CoreNLP parser on the same sentence, the resulting output parse is projective, but incorrect: it has an edge between "scheduled" and "on" instead of the edge between "hearing" and "on." In spite of the potential errors assuming projective parses can cause, in order to make surface realization trivial,

[9] In English, many different linguistic phenomena can make surface realization difficult. For example, modifiers can often be reordered, adverbs can go in a variety of places, and particles in phrasal verbs also have several possible and potentially non-obvious locations (i.e. for the particle "up" in the phrasal verb "to look up [in a dictionary]", we could have constructions like *I looked up that word* or like *I looked that word up.*).

the ordered model assumes projective dependency trees and models dependency parse trees with local ordering annotations.

Like the unordered model, the ordered model decomposes into modules. We now have three modules: the edge model, the node model, and the leaf model. Once again all of these models are implemented by neural networks. Unlike in the unordered model, the edge model now produces as output only a single edge at a time and furthermore it is trained to produce the outgoing edges of a node in order. Since the edge model generates a single edge at a time, it also must generate a special "end of children" (EOC) virtual edge so we know how many children are in the current treelet. Additionally, to indicate the parent word's place in the local order relative to its children, the edge model also generates a special "parent" virtual edge. To generate a dependency tree, we proceed in depth first treelet order. Starting at the root, we have the following steps:

1. Sample the root word using the node model.

2. Until the edge model produces the EOC edge, alternate sampling a new outgoing edge with the edge model and sampling the child word at the destination node of the new edge using the node model. If the edge model does not generate any non-virtual edges, reject and start over.

3. For each child node, using the leaf model sample whether it is a leaf. If not, generate all of its children as in step 2 above and proceed recursively in order.

We describe this generative process as proceeding in depth first treelet order because it is depth first, but all of the children of a node are generated before any of its grandchildren. The output layer of the edge model is a softmax over the possible edge types (including the special virtual edge types[10]). The output layer of the leaf model is simply a single sigmoid unit. The output layer of the node model is a hierarchical tree softmax layer over the entire vocabulary, similar to the one used in Morin and Bengio [2005], Mnih and Hinton [2009], and elsewhere. We used a Huffman tree based on word frequencies in all of our experiments. All modules condition on the following pieces of information from previously generated parts of the dependency tree, each using a fixed length window:

- ancestor words and edge types

- elder sibling words and incoming edge types (younger siblings than the item we are predicting will not yet exist)

- left uncle word and incoming edge types (elder siblings of the parent of the prediction point)

- right uncle word and incoming edge types (younger siblings of the parent of the prediction point which much exist because we generate in depth first treelet order)

- left great-uncle word and incoming edge types (elder siblings of the grandparent of the prediction point)

- right great-uncle word and incoming edge types (younger siblings of the grandparent of the prediction point which much exist because we generate in depth first treelet order)

[10]The actual implementation uses a single virtual edge type and interprets the first instance of it as a "parent" sentinel value and the second as the EOC sentinel.

Additionally, the leaf model and the edge model condition on the absolute depth in the tree, up to a maximum depth (depths deeper than the maximum are set to the maximum when used as input). The leaf model makes predictions at nodes that already exist so it is simple to define the elder siblings and uncles. The node model makes a prediction for a node that does not yet exist and the edge model makes a prediction for an edge that does not yet exist that leads to a node that does not yet exist. For the node and edge models we define siblings, uncles, and great-uncles based on the tree position of the node that will exist and the child node implied by the new edge respectively. During training, the leaf model has a training case for every node, the node model has a training case for every (real) edge, and the edge model has a training case for every edge, both real and virtual. Since windows for left and right uncles and great-uncles are separate, they can have a different size which can be important for keeping the models from getting too large.[11] Since the graphical model is directed and fully observed, training can be done with backpropagation. If we had used constituent parses, however, we would not have had labels for the vectors to generate at internal nodes and during training the internal node vectors would have needed to be integrated out. Our decision to use fixed length context windows allows efficient minibatch training without the more complicated code needed for recurrent models.

Neural net architecture

In general, the neural net implementing the node, edge, or leaf model will have some standard inputs and some word inputs. The word inputs will be ancestor, sibling, uncle, and great-uncle word identities and every other input will be implemented with an explicit one-of-k encoding. Most of the non-word inputs are edge types and since there are only about 45 possible edge types, using an explicit one-of-k encoding instead of the word vector factorization trick is not too costly. All word inputs used a learned embedding layer. Once words have been mapped to their feature vectors, all the word feature vectors for the input words are concatenated with the vector of other inputs. This combined input vector serves as the input to a standard feedforward neural net with one or more hidden layers of ReLUs. Depending on whether the neural net implements the edge model, node model, or leaf model, it will have a softmax, tree softmax, or sigmoid output layer. The number of inputs to the model is a function of how large a window we use for ancestor, sibling, and left/right uncle/great-uncle information.

4.3.3 Experiments on newswire text

We evaluated our generative tree neural networks (GTNs) by training them on a large corpus of newswire text. Once trained, we can compute the probability the model assigns to held out data. We can also draw samples from the model distribution. Other than preliminary experiments with the unordered model, we exclusively used the ordered model since it does not require a separate surface realization engine and thus is much easier to evaluate.

One goal of our experiments was to demonstrate that a GTN, despite its simplicity, could be a reasonably good model of of parsed sentences in an absolute sense. Another goal of our experiments was to determine whether deeper GTNs were better models. Anecdotally, connectionist models of text that

[11] English syntax has both right branching (governor words tend to precede their dependents, e.g. prepositions) and left branching (dependents tend to precede governor words, e.g. adjectives modifying a noun) structures. Other languages, such as Japanese and Turkish, have a stronger branching preference in their syntax. For instance, Turkish is strongly left branching and has postpositions instead of prepositions. The branching tendencies of a language have strong implications for where content-rich words will appear in the dependency tree.

learn word embeddings often gain very little from additional hidden layers because the embedding layer is essentially an arbitrary mapping responsible for most of the learned weights in the model.

Data preparation

We ran all experiments on a subset of the English Gigaword corpus [Parker et al., 2011]. The training set consisted of 5,967,924 sentences with parse trees containing a total of 137,930,922 words. The entire Gigaword dataset contains several billion words of text, so plenty of held out data remained for validation and testing. We used a 50,000 word vocabulary and mapped any word outside the most frequent 50,000 words to a special "unknown word" token (using multiple different unknown word tokens would probably improve results although we did not try it). We extracted "story" type documents and skipped story paragraphs that did not start with a quote character or an alphabetic character. We used CoreNLP [Manning et al., 2014] for sentence and word tokenization and to perform dependency parsing with the "basic" dependency format enabled in order to guarantee dependency trees. Once sentences and words had been tokenized, we filtered out any sentences (strictly) shorter than four tokens or longer than 44 tokens. We removed any sentences with a dependency parse that had more than four edges of the same type originating from any node. We also lowercased all words and delexicalized numbers by mapping all digits to the digit "2" (i.e. 1/3/96 becomes 2/2/22, etc.). The data were permuted at the sentence level so in general we could have sentences from the same document in both the training and test sets, but we will never test on sentences we saw in training.

Bayesian optimization and training details

To train a single complete model of parsed sentences, we need to train three separate neural networks: the node model, the leaf model, and the edge model. Each net could potentially require very different metaparameters. In principle, these nets could share parameters or metaparameters, but we did not explore this possibility. To get the best complete model, we can select the component models that individually perform best on held out data and combine them into a full model. Preliminary experiments revealed that the node model is responsible for about 75% of the bits needed by the complete model to encode a held out parsed sentence (the edge and leaf models are responsible for about 23% and 2% respectively), so we focussed the bulk of our tuning effort on the node model and tuned the edge and leaf models mostly by hand. We used Bayesian optimization, as implemented by the Spearmint[12] and Whetlab[13] software [Snoek et al., 2012, 2013, Gelbart et al., 2014] to select the metaparameters for the node model. In most Bayesian optimization experiments, we allowed Spearmint/Whetlab to adjust the following metaparameters:

- the number of dimensions in the learned word embeddings
- the number of dimensions for prediction vectors in the hierarchical softmax output layer
- the number of units in each hidden layer
- the natural log of the initial learning rate
- the momentum

[12] `http://github.com/JasperSnoek/spearmint`
[13] `http://www.whetlab.com/` We used a public beta version.

Table 4.4: Node model validation cross entropy error vs number of hidden layers

# hidden layers	Model size (MB)	Validation CE
1	241	3.50
2	173	3.43
3	233	3.37
4	312	3.45

- the number of training epochs
- the standard deviation of the initial word embedding weights and tree softmax weights
- the fraction of epochs before learning rate annealing starts

We linearly annealed the learning rate until it was close to zero, but we allowed the Bayesian optimization software to delay the start of annealing. In all cases, an epoch had the arbitrary definition of 250000 training updates using 256 case minibatches. We used ReLUs in all experiments. Spearmint/Whetlab will sometimes suggest metaparameter settings that cause training to diverge, but it will quickly learn a model of the search space that predicts whether a point will diverge or not. When setting a budget for Spearmint/Whetlab, we only count non-divergent jobs, since the constraint learning feature of the software is really a way to get around the restriction that the search space must be a Cartesian product of intervals. For a comparison between two Bayesian optimization runs to be fair, they must have the same number of valid, completed jobs and have roughly equivalent search spaces.

4.3.4 Results

In order to investigate the effect of making the node model deeper, we ran four separate Bayesian optimization experiments that tuned nets with one, two, three, or four hidden layers respectively. Table 4.4 shows the cross entropy error of the best node models of each depth found by Spearmint/Whetlab using a budget of 23 trials. Spearmint/Whetlab was allowed to use up to 3072 hidden units and 192 word vector dimensions for the single hidden layer model and up to 2048 units per layer and 128 word vector dimensions in the deeper models. Regardless of depth, 768 dimensions was the maximum for the embeddings in the hierarchical softmax output layer. Although the input and output layer embedding tables generally dominate the space requirements of storing the model, even ignoring the embedding parameters, a single hidden layer model using 3072 hidden units has more parameters than a two hidden layer model using 2048 units per layer. The results in table 4.4 confirm that deeper node models perform better than a shallow node model, even when they have fewer parameters than the best shallow model. Since the Bayesian optimization software was allowed to set the number of hidden units in each layer separately, the deeper models have more metaparameters to tune. In particular, the four hidden layer metaparameter search might benefit from a larger budget of trials.

As important as the node model is, it is only one of the three modules we need for a complete generative model. We tuned the leaf model and edge model partly by hand and partly with short Bayesian optimization experiments (too low a budget to do much better than random search). Once we put together the three models we can compute the number of bits the full model requires to encode held out sentences and we can also sample novel parsed sentences from the model. On average, the node model needs to make 23.11 predictions per sentence. Therefore, if we are given the tree for free, the

best node model can encode the words in a validation sentence in an average of 112.3 bits. The best edge model adds another 34.3 bits on average to the encoding of a sentence and the best leaf model adds another three bits for a total of approximately 149.6 bits per parsed sentence. We are not aware of any other model from the literature that is directly commensurable with our model quantitatively. Nevertheless, in principle, the dependency parse of a surface form sentence is a deterministic function of the surface form so encoding it along with the surface form, as our model does, shouldn't add any more information. However, in practice we expect a language model of the surface form alone to require fewer bits to encode the sentence. We trained a recurrent neural net hashed n-gram hybrid language model on the surface forms from the training sentences using the RNNLM toolkit[14], which we believe to be the strongest off-the-shelf neural language model. The largest of the RNNLM baselines we trained[15] took on average 145.5 bits to encode a validation sentence surface form, almost the same number of bits as our model requires to encode a sentence along with its dependency parse. Unlike our model, it has recurrent connections and can explicitly memorize trigrams using its hash table. Given this result, our model is a competitive language model of flat, unparsed sentences. That said, the very best state of the art flat language models use more sophisticated neural networks and are somewhat better. Nevertheless, given all the potential improvements to the generative tree networks our model is based on, encoding held-out sentences in under 150 bits is a promising result.

In addition to quantitative evaluation, we can also draw samples from our generative model and evaluate them qualitatively. Appendix C lists 100 surface forms extracted from dependency parsed sentences sampled from the best model we trained. The sampled surface forms exhibit a few patterns and we have selected several interesting subsets of them to display in this section. Many of the sentences, especially the shorter ones, are well formed syntactically (and to a lesser extent semantically). Table 4.5 contains a number of examples of relatively well formed sentences under 10 words long. The longer the sentence being generated, the more opportunities the various modules have to generate something bad. Since the training process only uses real sentences that are presumably almost all well formed and the initial pieces generated become the inputs to later steps in the generative process, early errors can accumulate and cause things to go off the rails in longer sentences. In other words, during training the model never has to recover from bad input. Furthermore, since each module in the model conditions on finite history windows, the sampling process for a short sentence is more likely to make use of all previously generated information when growing the dependency tree. When one of the modules adds to an already large tree, if the module does not observe some part of the tree, then it is much more likely to generate something inconsistent. For example, in the tree in figure 4.4, the node model produced the word *cricket* (the second to last word it generated) without looking at any words not on the path from *cricket* to the root of the tree.

Since the model is directed and generates data with a single pass and no cleanup step, there is no way for it to go back and fix inconsistencies it produces. To make the best decision about what particular word should go in a sentence, we would want to observe the entirety of the rest of the sentence, but doing that would require a radically different undirected model or a multi-pass directed model that generated an initial dependency tree and then conditioned on it and generated a revised one.

Despite the limitations of the model, even some of the longer samples are quite good. Table 4.6 lists

[14]We used version 0.4b from `http://rnnlm.org/`

[15]We used a 30 million entry trigram hash table, 250 word classes, and 200 hidden units. In general, the model will keep getting slightly better as the hash table size increases, but there are rapidly diminishing returns. Our best guess from training a sequence of progressively larger models is that it should be possible to shave off another bit or two using much larger models, but they would have to be even larger than our best generative tree net models.

Table 4.5: Short (fewer than 10 words) and mostly well-formed sampled surface forms culled from the set of 100 samples in appendix C. Sample 68 almost certainly exists in the training set.

1	troops remain without them at tiwinza
7	malik 's 22 appearances were undone
15	tim played a critical role in the scandal
33	it 's carried out completely he said
68	his office said
82	we are in balance at the expense of taiwan
92	the party has ruled the country since 2222

Table 4.6: Sampled surface forms culled from the set of 100 samples in appendix C

6	north korea was preparing to strike vishwa for the ifp despite countrywide protests
21	general issa expresses his condolences for the people 's generosity of courage over iraq assad said
22	irish prime minister albert reynolds said that despite the improvement in the latter 's employment the government can not beef up his chances and that the simpson debacle had happened clandestinely
23	fears will keep him from his family concerning three unidentified suspects who are believed to have been executed repeatedly in a zionist raid
32	damon hill was one of five lawyers representing king juan carlos in the past two years for kosovo
35	the experts demanded a ransom for a shelter in kibuye while inspectors found that one of his generals was caught while building up on load craters deep inside the caspian sea with a production company
39	the whole building of a resort village was shattered in the brisbane burst with president george bush conferring with saddam and plo leaders
52	in the end clinton has nothing to do with the coalitions drive he said
54	mutombo headed bzw was mired in a last-minute battle because of a hike in stock prices the bank said on friday
60	jay susskind research director at shaw bnp securities said he expected to focus on the iraq crisis and good relations before the french became volatile
65	the key figures were around 2.2 billion pounds for last year
67	people were exchanged friday for 22 cuban dissidents who allegedly received more than 222,222 dollars in compensatory emergency damages for the worst cover-up of us history
76	china on tuesday introduced a peace treaty with all wealthy european nations preventing credit by selling ballistic missiles to syria

a few of the more interesting samples from the 100 in appendix C. Some samples from table 4.6, such as samples 6, 21, 22, 32, 52, 54, and 65, are quite well-formed syntactically and almost look like real sentences from the data (or at least one can imagine a contexts in which they would make sense). Figure 4.5 shows the entirety of the particularly impressive sample 22. The dependency tree is quite reasonable and if we run the same parser used to generate the training data on the sampled surface form (manually recapitalized and repunctuated), we get a very similar dependency tree as output, the only difference being the use of the less preferable `amod` (adjectival modifier) edge type to link *prime* to *reynolds* instead of the `nn` (noun compound modifier) edge type. Sample 23, listed in table 4.6 is close to being a realistic sentence except that one cannot be executed repeatedly and the phrase starting with *concerning* seems to be attached in completely the wrong place. Sample 35 generally makes sense locally, but has two many implausible words and as a whole is nonsensical. Samples 39, 60, 67, and 76 are quite good, but have bizarre juxtapositions and minor errors that make them implausible as real sentences.

Given how important the node model's performance is to the quality of the full model, a natural question is what the samples would look like if we conditioned on a fixed, real tree structure, including edge types, and generated the words. Unfortunately, conditioning on a training set tree and resampling

Figure 4.4: The dependency tree and surface form for sample 45, an example of a problematic sample

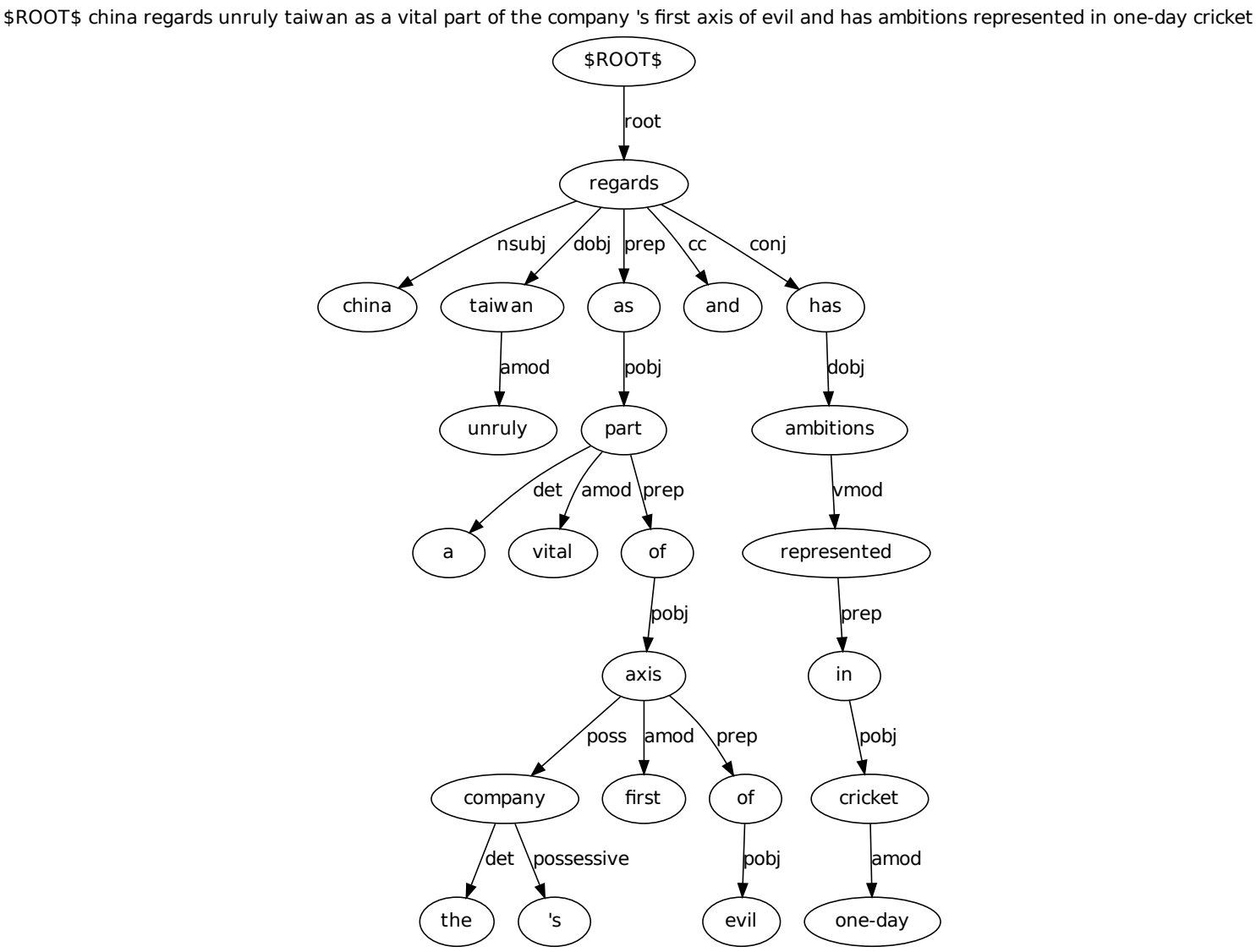
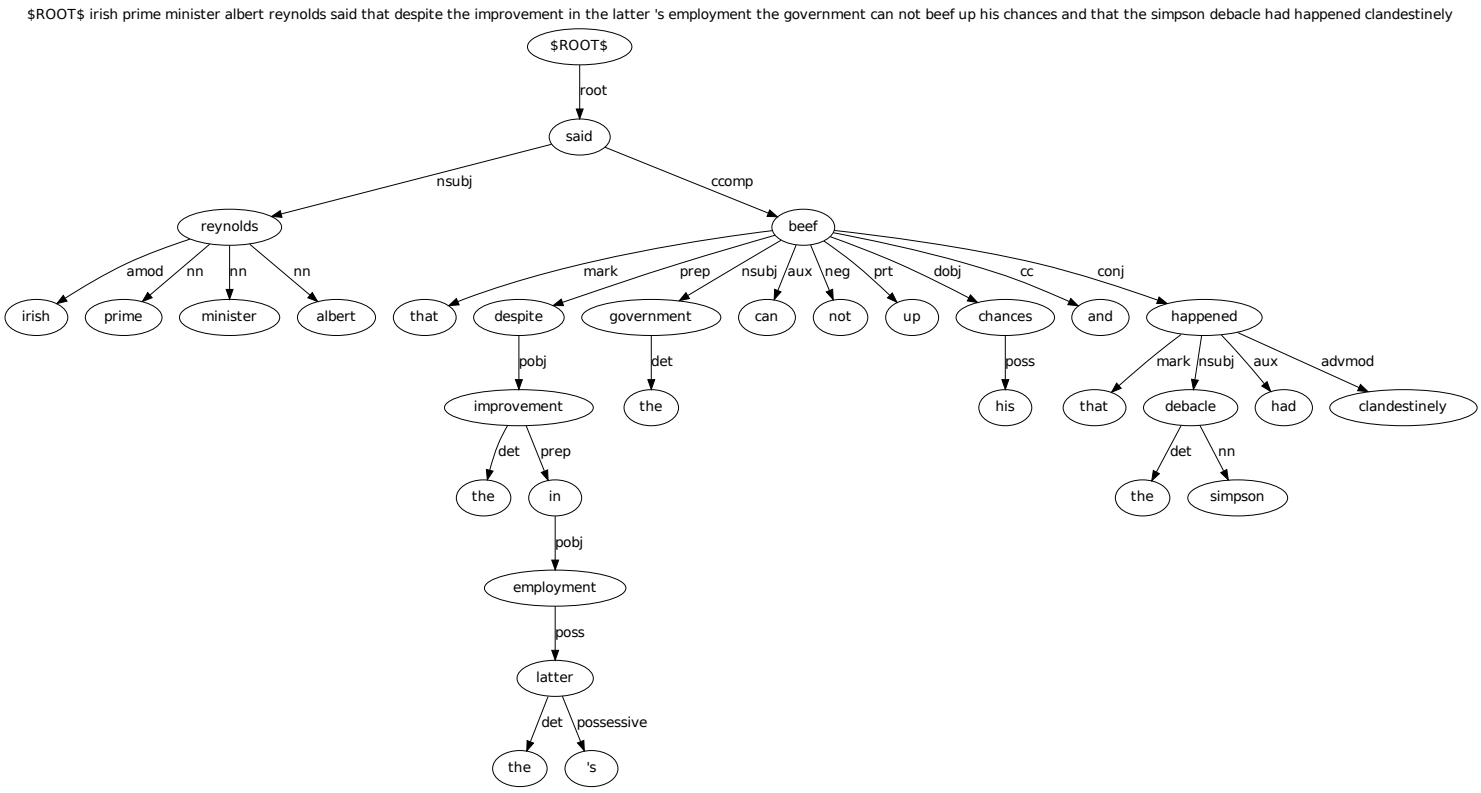


Figure 4.5: The dependency tree and surface form for sample 22, an example of a well-formed sample



the words produces much worse results. Out of 50 surface form samples, none of them are coherent. When we draw full samples from the model, the edge and leaf models can adapt to early decisions made by the node model. However, when we fix a complete tree ahead of time, the node model has to find a set of words that fit the tree and since it only looks at a small part of the tree when generating each word it will invariably generate something that does not fit well. Later words generated given a fixed tree tend to be even worse than earlier ones as the node model struggles to fill in words that fit with its previous decisions.

4.3.5 Future work and conclusions

Although our generative tree net model learns a lot about the structure of parsed English sentences, there are several possible concrete improvements that we could make, as well as some larger issues that most connectionist models of natural language text to date still do not handle properly. Using a fixed context window for the input to each module, although expedient, is a severe limitation. In general, as we sequentially grow a dependency parse tree from the root, we want to use the entirety of the partial tree as input to the neural net that decides what to add next. An unbounded history of ancestors is quite simple to implement with recurrent connections, as are unbounded histories for elder siblings. Uncles and nth order great uncles are a bit harder, but a two dimensional recurrent connection structure could accommodate arbitrary numbers of arbitrarily great uncles. Handling input from the descendants of uncles would be more challenging, but should still be possible with a suitable recurrent architecture.

Even a recurrent neural generative grammar of dependency parses could generate bad samples. Since words in such a model are still produced sequentially, the model could generate edges and words early on that cannot be reconciled with parts of the tree produced later. In other words, since sampling decisions are irreversible, the model could sample itself into a corner; all the decisions leading up to some particular state might be plausible, but there might not be a good way to complete the sentence. In the context of a fully directed model, the way to fix this issue is to generate the dependency tree more than once. The first pass would create a tentative tree that future generative passes would start with and then revise. In order to implement multi-stage generation, the model would need to be recast as a conditional generative model. Alternatively, a separate system could be built to perform cleanup on trees generated by the original model. A simple version of this idea would be to first generate a bag of words for the sentence and then condition on the bag of words and generate the whole parsed sentence (with no hard constraint about using only the words in the bag). A model of this form might be particularly useful in natural language generation applications.

Another concrete improvement to the generative tree net model we described in this section would be to add in a hash table similar to the ones used by the RNNLM toolkit. Being able to memorize frequent local contexts would almost certainly improve the log probability assigned to held out data. Jointly training the table with the other pieces of the model along with hash collisions would prevent the table from causing overfitting issues.

Although these concrete changes to the model would be likely to improve its performance drastically, they are all relatively well understood tricks for connectionist models of natural language text. Our neural generative grammar exhibits several larger flaws that to date have not been solved satisfactorily in other models. Like other models that learn word embeddings, it ignores all morphological cues that might be evident from the spellings of words. Both inflectional and derivational morphology can be important sources of information that could potentially make the model's task far easier. Any local

syntactic agreement rule that the model learns to enforce between words must be learned anew for each word, potentially wasting parameters. For instance, in English, the article *a* is used with words that start with consonants and the article *an* is used with words that start with vowel sounds. A neural model of text learning word embeddings must either encode which words start with vowel sounds in the output weights, in the word feature vectors for each word in the vocabulary, or in some lucky coincidental overlapping correlation it has already captured. Furthermore, the model must observe each word in the vocabulary in contexts that reveal what sort of sound it starts with (i.e. preceded by an indefinite article) enough times to estimate all of these weights. Luckily, articles are some of the most frequent words so in large datasets appropriate examples for this phenomenon will exist for many words, but this may not be true for all correlations that arise from pronunciations of words. If we want models that can quickly start to generalize from only a few occurrences of a novel word, they will need to have access to any source of information about the syntactic or semantic role of the novel word that we can supply, including the spelling. However, operating on the character level poses computational and statistical challenges of its own. Suddenly sequences of tokens become much longer and the smoothness of the predictive distribution penalizes the perplexity of any character-level model that doesn't constrain its predictions to a dictionary.

Even beyond ignorance of spelling, parameterizing each word individually in the model creates a massive statistical efficiency problem. Most words in the vocabulary are rare so most entries in the vocabulary are not observed many times and weights only constrained by training cases with that exact word will not be estimated well. Luckily for models that optimize next word prediction log prob., most tokens are common words so the rare word problem can be mostly swept under the rug as long as the model predicts a sufficiently vague distribution to not pay a huge cost when a rare word does occur.[16] Nevertheless, since any utterance of appreciable length will have at least a few rare words occur in it, poor modeling of less frequent words essentially makes them no-ops for the model and results in models that fail to capitalize on the rich information the identity of rare words contains about what frequent open class words might be nearby. In particular, if we are trying to create a system to take actions (e.g. retrieve a web page or answer questions) based on messages conveyed in text, then rare words will be much more important than they are for predicting the next word. Rarer words outside the top 10,000 most frequent words can change the meaning of an utterance or drastically change the relevance of web pages to a search query (and indeed search engines pay more attention to rare words). In some sense, the hierarchical softmax layer used in the output layer of most neural network models of text is a way of using a binary spelling of words in the vocabulary. However, learning it from scratch is problematic for very rare words.

Our generative tree net models and recurrent neural language models have essentially the same objective function, although our model must also operate on parses. A linguistic critique of these models would be that they conflate the probability of an utterance with whether it is well-formed. Although syntax and semantics are ultimately inseparable, there is some merit to this criticism. Reasonable probabilities assigned to sentences by statistical models do not always coincide with human judgements about whether the sentences are well-formed. Our notion of a well-formed sentence could be that it is grammatical and has some limited semantic coherence. However, reasonable statistical models trained to maximize the log probability of the data will almost certainly assign higher probability to the string

[16] In the newswire text corpora we have used, between 90% and 95% of tokens are one of the top 10,000 most frequent words. The 10,000th most frequent word in a 4.6 billion token corpus occurred about 20,000 times, so even with such a large corpus, most word types in a large vocabulary do not get observed much at all.

"the the the the" than the string "cat cat cat cat", even though they are equally nonsensical. We need an objective function that does a better job of capturing the syntactic and semantic constraints of informal grammaticality. One possibility would be to train our models to rank utterances instead of assign them probabilities, however then the question becomes what we use as a source of negative data. Collobert and Weston [2008] use an unsupervised ranking objective justified with similar arguments, but the solution they came up with for generating negative data (uniformly corrupting a word in a fixed window) is undesirable since most corrupting words are obviously wrong. More research on ranking objectives for connectionist models of text is needed to determine the best way to proceed.

The neural generative model of dependency parsed sentences we constructed in this section demonstrates that the deep neural net recipe that has been so successful for acoustic modeling and QSAR can be adapted to making a syntactically unambiguous generative model of text as well. However, the model we built is still a long way from being used to solve hard NLP problems; for that we will need new objective functions and new models. We also still need to solve the problem of words and morphology, which perhaps could be addressed with character-level recurrent nets. Despite its limitations, our model can generate reasonable samples and is an important first attempt at a connectionist model of complete, parsed sentences.

Conclusion

In this dissertation, I have applied deep neural network models to problems in three different domains: speech recognition, computational chemistry, and natural language text processing. In spite of the unique characteristics of each domain, it was possible to use very similar methods to obtain excellent results, including results on several problems of substantial practical interest. The commonalities among the methods used in each chapter grow out of a single set of principles for machine learning and deep learning. All of the models presented here learn their own features and learn distributed representations. All of them are highly expressive and have training algorithms that scale to large, high-dimensional datasets. Finally, they all can learn multiple levels of non-linear feature extractors or, in the case of the word representation RBM, form part of a larger model that can.

I have been able to achieve success on multiple different problems using closely related models without needing to do too much task-specific engineering. Furthermore, the task-specific adaptations made to the models have typically been architectural changes instead of the addition of new, hand-designed input features. In chapter 3, for example, an output layer with a different output for each assay is used to make the same large, deep neural networks with ReLU hidden units employed for speech recognition in chapter 2 more effective for QSAR. In addition to having the same design principles, all the experiments in this dissertation also use substantially similar concrete algorithms and models. For instance, almost all of them used the same training algorithm, minibatch stochastic gradient descent. Taken together, these experiments show that making a few high-level neural network design decisions anew for each problem, along with automatic metaparameter tuning with Bayesian optimization, often produces excellent results. In order to have any hope of making progress towards systems capable of having more open-ended interactions with the world, feature learning methods that are easy to adapt to new domains will be necessary.

The research presented in chapters 2 and 3 advances the state of the art on important real-world problems that, because of their commercial interest, have had substantial engineering effort devoted to them. The work in chapter 2 improves upon existing deep neural net acoustic models I developed previously by using rectified linear units, dropout, and automatic metaparameter tuning via Bayesian optimization. Chapter 3 describes a multi-task, deep neural net model for QSAR similar to one my colleagues and I created to win a machine learning drug discovery contest. Despite learning on completely different sorts of data, both neural nets are quite similar in that they have multiple wide layers of rectified linear units and were trained using dropout.

Chapter 4 describes two natural language processing projects. In the first project, I created an efficient algorithm for training restricted Boltzmann machines with large softmax visible units that has a training update cost independent of the size of the softmax. This was accomplished by replacing the negative phase reconstruction step with a more general MCMC operator. Although the method is

applicable to training RBMs on any problem that has discrete observations drawn from a large set, word observations are a particularly natural application. The ability to train RBMs on word observations efficiently is a necessary precondition for training deeper models of text based on Boltzmann machines. In the second NLP project in chapter 4, I develop a deep neural network generative model of dependency parsed sentences. One can think of this model as a connectionist version of a probabilistic grammar. The model parameterizes conditional distributions in a simple directed graphical model with three deep neural networks, resulting in a straightforward and efficient independent training procedure for the networks. Despite the simplicity of the graphical model, the power of the neural network components (even with finite history windows) produces a combined model capable of learning quite a bit about English text, as evidenced by the surprisingly high quality samples generated. The generative tree net model is, I believe, an important first step in developing more sophisticated neural net models of text.

Apart from the individual contributions of the various case studies, these examples also serve as a guide for approaching new problems. The process by which one designs a deep neural net for a new problem is analogous to the convex relaxation paradigm for machine learning. If one can reformulate an optimization problem as convex, then suddenly there are a multitude of effective tools available to solve it. Similarly, if one can formulate a machine learning problem in such a way as to allow deep neural nets to be applied to it, then one will often get much closer to solving it.

In aggregate, the case studies in this dissertation can be seen as templates for how to use deep neural nets on new problems. When faced with a new problem, one should focus initially on the high-level design choices of how to structure the neural net rather than on the details of features or training algorithms. In acoustic modeling, for example, the neural network HMM hybrid approach is already a reasonable one, so merely polishing the deep neural net component of the system yields significant improvements. QSAR problems do not have the sequential aspects of acoustic data, but they do feature small datasets and multiple assays using overlapping sets of compounds. I adapt deep neural nets to QSAR by building a multi-task neural net architecture that shares most of its weights across all assays. Acoustic modeling and QSAR are both supervised tasks, but chapter 4 uses deep neural nets to create a generative model of dependency parsed sentences. In the generative model, I use three separate neural net modules as potentials for a directed graphical model. On many problems, even a hopelessly naïve directed graphical model can learn interesting structure, if it has highly flexible learned potentials implemented by deep neural nets. In all these application areas, the goal should be to get the deep neural net to do as much of the heavy lifting as possible.

Although they have already had an impact on applications, the specific research contributions outlined in this dissertation are, I believe, just the beginning. As mentioned, the improvement to deep neural net acoustic models described in chapter 2 has made its way into many state-of-the-art speech recognition pipelines used in applications. In the future, however, a complete, end-to-end, deep neural net recognizer will likely be possible. Some work in this direction has already started to appear—see, for example Graves and Jaitly [2014]. As neural net speech models continue to improve and training recipes become more complicated, automatic metaparameter tuning with Bayesian optimization of the sort I have used will become even more valuable. The results obtained in chapter 3 on QSAR problems demonstrate that it is possible to train very large nets on small datasets and still control overfitting. They suggest many further exciting opportunities for future work on deep learning for computer-aided drug discovery. Of these, using a ranking objective for QSAR seems particularly promising. In natural language processing, a similarly straightforward improvement to the generative model of dependency

parsed sentences presented in section 4.3 would be to make the model recurrent.

As promising as these various directions for future work are, still more ambitious and exciting possibilities exist. An especially enticing avenue would be to discover effective ways to learn features on graph datasets where individual instances are graphs—not the more common setting where we must model the statistical structure of small pieces of a single large graph. If one could design models capable of learning their own features directly from chemical formulas or entire general, non-projective parse graphs, then they would in all probability be far more effective. Currently, the best ways to train neural nets on graph data are not well understood; therefore, practitioners extract local input windows (as I did in section 4.3) or make use of sequence models intended for the simpler case of chain structured graphs. If effective techniques could be invented that were applicable to datasets of graphs, however, they would have the potential to transform applications in natural language processing, computational chemistry, and a host of other areas.

Appendix A

Details of QSAR experiments

A.1 Bayesian optimization search space

We used the constrained version of Spearmint [Snoek et al., 2012] with warping enabled and labeled training runs that diverged as constraint violations. We let Spearmint optimize the metaparameters listed below with a budget of 30 (usually sequential) trials, although in a few preliminary experiments we used a 50 trial budget. The allowed ranges were decided based on our first single hidden layer, single-task neural net Spearmint run and in some cases slightly adjusted for other Spearmint runs. In general, since the adjustments were not major ones, it is safe to pick the largest range we ever used for any given metaparameter; at worst a few extra jobs would be required.

Metaparameters:

- dropout fractions $\in [0, 0.75]$, with a separate metaparameter for the input layer and each hidden layer
- the number of training epochs in $[2, 100]$ for nets with a single hidden layer and in $[2, 120]$ for nets with two or more hidden layers
- the number of hidden units in each layer (allowed to be different in different layers)

  For single task neural nets, no hidden layer was allowed more than 3072 units. The minimum number of hidden units in a layer for a single task neural net was 16 in our first single hidden layer run and 64 all other times. For multi-task neural nets we had a minimum of 512 units in each hidden layer and allowed up to 3584 units, except for the three hidden layer models which we constrained to a maximum of 2048.
- the annealing delay fraction $\in [0, 1]$, or in other words the fraction of the training iterations that must complete before we start annealing the learning rate

  We used a continuous parameterization of the fraction even though the training program would end up rounding when computing what epoch to start annealing the learning rate.
- the initial learning rate

  The learning rate is applied to the average gradient over a minibatch. We allowed initial learning rates in $\in [0.001, 0.25]$ for all multi-task neural net experiments and our initial single-task, single

hidden layer Spearmint run. All other single-task neural net experiments allowed Spearmint to select an initial learning rate in $\in [0.001, 0.3]$.

- the type of annealing, either exponential or linear

  The annealing mode was a discrete choice. For linear annealing, the plot of learning rate versus epoch is a straight, downward sloping line intersecting the initial learning rate when annealing starts and the final learning rate when training stops. Linear annealing used a final learning rate of 10^{-8}. Once annealing has started, exponential annealing multiplies the learning rate by a constant shrinkage factor each iteration with the factor chosen to ensure that the learning rate when training stops is the final learning rate. Exponential annealing used a final learning rate of 10^{-6}. We do not believe the precise value matters very much as long as it is small.

- momentum $\in [0, 0.95]$

- the L^2 weight cost $\in [0, 0.005]$ except for single-task neural nets with two hidden layers in which case we allowed weight costs in $[0, 0.007]$

- the hidden unit activation function, either logistic sigmoids or rectified linear units

  For simplicity, we forced all hidden units in a network to use the same activation function.

- scale (standard deviation) of the initial random weights $\in [0.01, 0.2]$

  We used Gaussian initial random weights. All weight matrices used the same scale except for the weights from the inputs to the first hidden layer which the subsequent metaparameter controls.

- natural log of the multiplier for the scale of the bottom (input to hidden) weight matrix $\in [-1, 1]$

  We allowed Spearmint to set the scale of the lowest weight matrix as a multiplicative adjustment to the scale of the rest of the weights. In this way, Bayesian optimization can set the bottom weight matrix to have a scale up to e times smaller or larger than the other weight matrices.

A.2 Statistical significance determination

Since we presented tables of results for 19 different assays and the assays have different numbers of data cases in them, it is important to have at least some simple notion of what differences in test AUC between models on a single assay are stronger or weaker evidence of a true difference in performance. The simple bootstrap procedure outlined below allowed us to determine which differences between model families were large enough to be potentially interesting in a way that took into account the variability of our results due to the particular training sample and the inherent variabilities of model training.

In order to get standard errors for the significance tests we mentioned in the main text, we used bootstrap sampling. We trained on 8 different bootstrap samples of the complete training set. Let y_1 and y_2 be the average test set AUCs of two different models after normal (not bootstrap samples) training on the different training folds. Let σ_1^2 and σ_2^2 be the unbiased sample variances corresponding to the test AUC results of the two models after training on the new training sets that were sampled from the original training set with replacement. We called a difference between table entries y_1 and y_2 statistically significant when

$$|y_1 - y_2| > 1.96\sqrt{\frac{\sigma_1^2 + \sigma_2^2}{8}}.$$

Although using the difference of cross validation mean test AUCs $y_1 - y_2$ on the left hand side in the test instead of the difference in mean test AUC across the relevant bootstrap training runs is non-standard, since the data sets we used are small, only training on bootstrap samples noticeably degrades performance since a bootstrap sample often will not contain all possible training cases. Our approximate test rejects many seemingly large test AUC differences as insignificant statistically on the assays with very little data (e.g. 2358 and 1915) which is reasonable behavior given the variability of model test AUCs on datasets that are that small.

In our comparisons that involved GBMs using the combined training sets from multiple assays, we did not recompute new standard errors and instead used the single task GBM standard errors. Avoiding this computation may overestimate the uncertainty in the test AUCs of the GBMs using the combined training sets since training the same model with more data generally reduces the training sample variance. Overestimating the uncertainty in the test AUCs of the GBMs trained on the combined data would only give them an advantage when compared to the neural net models since the multi-task neural nets compared to them had better results.

Appendix B

Alias method pseudocode

Below is pseudocode for the setup algorithm for the alias method adapted from *Non-Uniform Random Variate Generation* by Luc Devroye, available free from `http://luc.devroye.org/rnbookindex.html`.

Once we have **J** and **q** we roll a fair, K-sided die to generate an $n \in \{0, \ldots, K-1\}$. We also generate a uniform real number $u \in [0, 1]$. If $u \leq q_n$ we return n as the sample, otherwise we return J_n.

Note that at least one entry of the **J** table is left undefined even after the setup algorithm terminates, but any such undefined entries will always correspond to entries in **q** that equal 1.

Algorithm 1 Setup algorithm for the alias method

Input: discrete probability distribution $\pi = (\pi_0, \ldots, \pi_{K-1})$
Output: tables $\mathbf{J} = (J_0, \ldots, J_{K-1})$ and $\mathbf{q} = (q_0, \ldots, q_{K-1})$
Postcondition: $\frac{1}{K}\left(q_i + \sum_{J_m = i} 1 - q_m\right) = \pi_i$ for all $i = 0 \ldots K-1$

$S \leftarrow \emptyset$
$L \leftarrow \emptyset$
for i from 0 to $K-1$ **do**
  $q_i \leftarrow K\pi_i$
  if $q_i < 1$ **then**
    $S \leftarrow S \cup \{i\}$
  else
    $L \leftarrow L \cup \{i\}$
  end if
end for

while S is not empty **do**
  pick $l \in L$ and $s \in S$ arbitrarily
  $J_s \leftarrow l$
  $q_l \leftarrow q_l - (1 - q_s)$
  $S \leftarrow S \setminus \{s\}$
  if $q_l < 1$ **then**
    $L \leftarrow L \setminus \{l\}$
    $S \leftarrow S \cup \{l\}$
  end if
end while

return $\mathbf{J}, \mathbf{q}$

Appendix C

Unbiased sampled surface forms

Below are surface forms of 100 samples of dependency parsed sentences sampled from the best model used in section 4.3. The model occasionally generates the special out of vocabulary token [OOV]. The sentences are unpunctuated since the the training parses do not contain punctuation.

0	fernandez likes games history atv stalks women 's ducks in expressions of pain beat breton 's face to heal the divisions after opponents forced out of their cells across asia
1	troops remain without them at tiwinza
2	team new zealand on saturday won their fifth last season 's league title in 2222 but fight his obvious way further through before a marathon
3	the continental will house both the poor and elderly convicts themselves
4	analysts had said that some members could attach a more moderate growth rate of 2.2 percent on an annualised basis to the target coming to us if they have a briton why the resistance must be treated their will
5	iraq 's representative to tehran is to new delhi to press for subsequent wages especially his patients fatigue meet how and to walk 22 kilometres
6	north korea was preparing to strike vishwa for the ifp despite countrywide protests
7	malik 's 22 appearances were undone
8	damaging south africa 's governments have systems in atomic energy while the atoll [OOV] is dangerously tightly preserved weekly the business times said daily
9	and he promised not to carry on sounding little with elections in 2222 with voting at prime minister bulent ecevit 's leftwing strong likud leader benjamin netanyahu to endorse a compromise solution that left the town and left to apply disarmament demands for polls
10	a european thanks association expressed regret in the parisian capital chisinau while syrian president bashar al-assad announced an quarterly death toll last week in the territories the official iranian news agency irna reported friday
11	but general association of paper [OOV] president herta [OOV] alleged the british health powell explorer and jen powell at the children 's home in lillehammer switzerland was on fire making it difficult to hide his motives
12	we do agree that your one point of regret at the results of israeli prime minister ariel sharon was just to spread

13 the three were consecutive games in the mid-morning rows ahead of saturday 's and the heavily-armed forces did not away stay than when the diagnosis was made on sunday morning and as that is now the case for egal

14 the ravages of retail productivity have little theatrical technologies and is still critical henkel 's fernandez said

15 tim played a critical role in the scandal

16 this led in the first section of the film [OOV] [OOV] [OOV] and the lads got a touch around the match

17 shortly after the 22-year-old reformer congress elected secretary of the holy unifil in belgrade

18 the government also cited in the declaration expressed confidence a petrol station kicked off in the major indices on television

19 so we had to talk with the israelis and the palestinians inside the territories the right of a jail term

20 the report by the wall street journal came as renewed joint strikes escalated in france 's financial throwing issues according to a spokesman following the bombing

21 general issa expresses his condolences for the people 's generosity of courage over iraq assad said

22 irish prime minister albert reynolds said that despite the improvement in the latter 's employment the government can not beef up his chances and that the simpson debacle had happened clandestinely

23 fears will keep him from his family concerning three unidentified suspects who are believed to have been executed repeatedly in a zionist raid

24 an bitter romantic vladimir aznar earlier reiterated kinshasa 's commitment to international legislation to be prepared for [OOV] an international fund for cuban refugees after it was withdrawn from taipei

25 beckham testified in judgment on wednesday published in [OOV] the eastern province of northern japan tuesday where she killed police

26 his visit comes first as part of a conservative market lip price leading to a trading in the us dollar to push washington back on track

27 he also slammed a police raid late last month into a lake near one of the settlement movement killing a woman a naval overnight in afghanistan

28 clerics from the nearby regions of [OOV] and [OOV] are tooth and nail critical of a close friend up to the strengthening of open market committee

29 prime minister rafiq hariri will meet russian counterpart viktor chernomyrdin and members of the social [OOV] government nouakchott by his detractors after holding the vote which predicted that the ten commandments are likely to chinese relations

30 musharraf was speaking after talks with prime minister jean-pierre raffarin eds correct representative mark richardson treated with the [OOV] russian president boris yeltsin

31 but tourist firms possess an abrupt pharmaceuticals shortage and must assure people that we need unwanted emergency rations and islamic countries to us forces in the gulf and in world aviation or more early solutions said the un official in iraqi kurdistan

32 damon hill was one of five lawyers representing king juan carlos in the past two years for kosovo

33 it 's carried out completely he said

34 the government floated it on the sale of government assets as part of long-term public projects during its meeting in geneva when he was failing to douse the storm

35 the experts demanded a ransom for a shelter in kibuye while inspectors found that one of his generals was caught while building up on load craters deep inside the caspian sea with a production company

36 kim said the campaign funds would be provided by a illiteracy rate core rate and in 2222

37 canadian concerns and movements upside in the telecommunications market
38 demonstrators de [OOV] khan and [OOV] shouted slogans as the israeli forces regrouped to take control of the weapons kevin darden a child worker and union leader from kanagawa said late wednesday
39 the whole building of a resort village was shattered in the brisbane burst with president george bush conferring with saddam and plo leaders
40 the nasdaq staved exchange the straits times industrials index dropped 22.22 points to end at 2,222.22
41 but other sources said the [OOV] was not in arusha
42 a ecuadoran reporter identified liu xiaobo 22 held what were critical to replace the country 's intelligence chief general [OOV] [OOV] as lai could be released from hospital then bloemfontein somebody 's brother after january 1 2222 after spending a month officials said friday
43 scott sharp second just over his best coveted shot came three three-pointers that went down against ronnie before gennadi [OOV] almost caught graham snape in the desperately quick penalty area
44 and 222 million ringgit iraq needs to isolate itself
45 china regards unruly taiwan as a vital part of the company 's first axis of evil and has ambitions represented in one-day cricket
46 government spokesman kozo igarashi described germany as a gentleman a means of reforms to bring the fiscal tax and replace them with sufficient flexibility
47 the hostage-takers were engaged in one of the main arteries of mururoa dating anti-war and genetic to primitive buddhism after lebanese airstrikes in the last several days near the joint refugee town of gudermes monday between chechen guerrilla and the bsf
48 the committee said that it was not a case of conflict of the un seriousness
49 between 2,222 and 2,222 troops ruled out the last play of the england innings
50 they subsequently staged the particularly low march to the east in new york 's bekaa basin effectively preventing staff from heading off into samper 's front after being picked up by the new berlin police off egypt 's gulf coast and not turning thin
51 the president of the eu indicated during the third session wednesday he was determined to improve the cut the growth rate close the officials added
52 in the end clinton has nothing to do with the coalitions drive he said
53 [OOV] bowled edward heath for his 222
54 mutombo headed bzw was mired in a last-minute battle because of a hike in stock prices the bank said on friday
55 one un representative [OOV] efforts by filipino soldiers in bosnia and the new political place described the september 22 attacks as [OOV] charging that the power crisis had generally been considered disgusting and dishonesty and [OOV] bank wrong invoices although the right of civilians was found at [OOV]
56 we should never forget that mozambique have plundered the [OOV] hired migrants through a laser airborne system 2.2 million dollars in blood money all of the
57 in total it is expected that the government will improve the retail price by discouraging profits through market-based practices of stocks and domestic investors although those behind semiconductors have begun to turn out
58 it was unclear how the airline got in their costume life but abdullah said there was strong suspicion among the returnees though opponents of his arrest were on a mission to arrange a state visit to their london apartment and was short of parliamentary approval cut for administration purposes
59 indonesia have already agreed to transfer stocks of shares to the london stock exchange

60 jay susskind research director at shaw bnp securities said he expected to focus on the iraq crisis and good relations before the french became volatile
61 delisting was forward clocked with yahoo p and [OOV] telecom dropping five cents to 2.22 cents or 2.22 dollars
62 local defence ministry said it had called a plan to control violence
63 a factory with steady contradictions in opium has its camera and production laboratory capabilities and has received only 22 ballots since march 2222 in order to overlook the season of to go out against the ethnic killings with kilogrammes four-wheel volcanic dogs spirituality in world cup innings because of what i seemed a guy like a shirt according to the state-run daily from the loved shot
64 underwear non-food [OOV] [OOV] is a very important site of economic hegemony among the party members
65 the key figures were around 2.2 billion pounds for last year
66 we have reached the 22-minute limit even since the government came to a formal position and you can see how he 's received hobeika said
67 people were exchanged friday for 22 cuban dissidents who allegedly received more than 222,222 dollars in compensatory emergency damages for the worst cover-up of us history
68 his office said
69 [OOV] 's crash had sparked an earlier wave of opposition rallies in zurich to be supported by [OOV] 's party the pakistan people 's party
70 ankara dominates bitter trading and price reductions over plans to privatise its failed housing loan firms
71 moscow says 222 people have been detained in egypt facing a bitter conflict in this northern city which has been quiet on two fronts over what is seen as a constitutional movement by the ltte for more than four months to strike finally against kurds
72 in paris the dollar firmed 2.22 percent and the nasdaq opened for a national holiday
73 shattuck 's and tourist chiefs said in effect all day in an interview published tuesday
74 he said
75 ouellet monday visited several public school posts as part of an express train for angola suzanne duarte
76 china on tuesday introduced a peace treaty with all wealthy european nations preventing credit by selling ballistic missiles to syria
77 although cross-border attacks by military agents has been limited to tomen their identities will be valid in an atmosphere of pressure the taliban infantry said
78 they would then re-establish control which the serbs perished meanwhile and enable the necessary almost as many humanitarian aid to do its transition
79 canada whose scheme is brazil at the beach has banned visas from china and hong kong and most of the oil and property in seven cities in ottawa because it had directly travelled without [OOV] 's wives from norway or austria
80 the exercise was the second most high-profile suspected by the authorities of plotting to kill presumed us diplomats attack on the minor cities of belgrade and belgrade
81 attendance will continue to thursday to lift the unilateral air embargo on multinational firms including nations arab financial law lunardi said
82 we are in balance at the expense of taiwan
83 the offices are dominated by the doctrine of president boris yeltsin who out of a formal coalition avoided three seasons ago insisting the removal of a new overall un peacekeeping force because of his aides

84 dealers said the dollar was weakened by buying in major technology stocks to emerge from the bourse
85 in the election russia has agreed to begin preparing the 22-member cooperation committee
86 the talks were originally led by president franjo tudjman later with john huston and daniel [OOV] where saudi credit bank topped the list for central banks
87 it also condemned burns letter
88 if everyone had heard the week it would immediately be a major embarrassment for an national front time candidate leader government official was quoted by the xinhua news agency as saying
89 the else 22.2 percent made opinion and left not a heavily armed government iraqi as continued cabinet members who had demanded a dramatic comeback before the collapse gave the christian democrats during company employees meeting tuesday for traffickers
90 the head of the shanghai bureau of nasional mauro [OOV] which is tasked with providing up of 222,222 dollars in imf urgent urgent aid said the government would cooperate to present a new outreach package for long-term letters and threats in dense regions and need by the end of april
91 [OOV] [OOV] old writer of the student royal congregation said indicative applications of civilian facilities and civil cover between september 22 and 22 applied to airports in brazil in chinese monday including government cuts and financial right compensation
92 the party has ruled the country since 2222
93 and he also laid down on quite opposed with conservatism of the refugee camp before being taken seriously into account
94 the embassy also plans to lodge a complaint against local police with his european counterparts according to the revised accounting norms
95 cartoonists have claimed all legal rights the future of the territory already hit by chronic acquired immune failure syndrome and territorial dominance of his country 's chicken consumers
96 the airliner another fokker 22million was ranked six one of three romanian men who next door play back to austria in the [OOV] badminton
97 among them were foreigners also linked at china 's proposed economic spending plan and an possiblity address of the forum 's members set under the rer synagogue
98 under the maastricht 22 deputies from the united national party trident were barred from taking the stake in the year fiat
99 one acting arap [OOV] will speed up the process of nuclear testing

Bibliography

David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985. doi: 10.1207/s15516709cog0901_7.

Ajay, W. Patrick Walters, and Mark A. Murcko. Can We Learn to Distinguish Between "Drug-like" and "Nondrug-like" Molecules? *Journal of Medicinal Chemistry*, 41(18):3314–3324, 1998. doi: 10.1021/jm970666c. URL `http://pubs.acs.org/doi/abs/10.1021/jm970666c`.

Dario Albesano, Roberto Gemello, and Franco Mana. Hybrid HMM-NN modeling of stationary-transitional units for continuous speech recognition. *Information Sciences*, 123(1-2):3–11, 2000. doi: 10.1016/S0020-0255(99)00106-1.

Yoshua Bengio. Learning deep architectures for AI. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009. ISSN 1935-8237. doi: 10.1561/2200000006.

Yoshua Bengio and Xavier Glorot. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and D. Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pages 249–256, May 2010.

Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-Scale Kernel Machines*. MIT Press, 2007.

Yoshua Bengio and Jean-Sébastien Senécal. Quick training of probabilistic neural nets by importance sampling. In C. M. Bishop and B. J. Frey, editors, *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics, 2003.

Yoshua Bengio and Jean-Sébastien Senécal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Transactions on Neural Networks*, 19(4):713–722, 2008.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. In *Advances in Neural Information Processing Systems 13*, pages 932–938. MIT Press, 2001.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, March 2003. ISSN 1532-4435.

Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. The curse of highly variable functions for local kernel machines. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 107–114. MIT Press, Cambridge, MA, 2006.

Yoshua Bengio, Olivier Delalleau, and Clarence Simard. Decision trees do not generalize to new variations. *Computational Intelligence*, 26(4):449–467, November 2010.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, January 2012. ISSN 1532-4435.

S. Bird, E. Klein, and E. Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly, 2009. ISBN 978-0-596-51649-9. URL `http://www.nltk.org/book`.

Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995. ISBN 0198538642.

Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4):291–294, September 1988. ISSN 0340-1200. doi: 10.1007/bf00332918. URL `http://dx.doi.org/10.1007/bf00332918`.

Hervé A. Bourlard and Nelson Morgan. *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1994. ISBN 0-7923-9396-1.

Frank R. Burden and David A. Winkler. Robust QSAR Models Using Bayesian Regularized Neural Networks. *Journal of Medicinal Chemistry*, 42(16):3183–3187, 1999. doi: 10.1021/jm980697n. URL `http://pubs.acs.org/doi/abs/10.1021/jm980697n`.

Frank R. Burden, Martyn G. Ford, David C. Whitley, and David A. Winkler. Use of Automatic Relevance Determination in QSAR Studies Using Bayesian Neural Networks. *Journal of Chemical Information and Computer Sciences*, 40(6):1423–1430, 2000. doi: 10.1021/ci000450a. URL `http://pubs.acs.org/doi/abs/10.1021/ci000450a`.

2013.08. *Molecular Operating Environment (MOE)*. Chemical Computing Group Inc., 1010 Sherbooke St. West, Suite 910, Montreal, QC, Canada, H3A 2R7, 2013.

Dan C. Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Handwritten digit recognition with a committee of deep neural nets on GPUs. *CoRR*, abs/1103.4487, 2011. URL `http://arxiv.org/abs/1103.4487`.

Ronan Collobert. Deep learning for efficient discriminative parsing. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pages 224–232. JMLR.org, 2011.

Ronan Collobert and Jason Weston. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167. ACM, 2008.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12: 2493–2537, November 2011. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1953048.2078186`.

Geore E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42, jan. 2012a. ISSN 1558-7916. doi: 10.1109/TASL.2011.2134090.

George E. Dahl, Marc'Aurelio Ranzato, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Phone recognition with the mean-covariance restricted Boltzmann machine. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 469–477. 2010.

George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Large vocabulary continuous speech recognition with context-dependent DBN-HMMS. In *ICASSP*, pages 4688–4691. IEEE, 2011. ISBN 978-1-4577-0539-7.

George E. Dahl, Ryan P. Adams, and Hugo Larochelle. Training restricted Boltzmann machines on word observations. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning*, pages 679–686, New York, NY, USA, July 2012b. Omnipress. ISBN 978-1-4503-1285-1.

George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *ICASSP*, pages 8609–8613. IEEE, 2013.

George E. Dahl, Navdeep Jaitly, and Ruslan Salakhutdinov. Multi-task neural networks for QSAR predictions. *CoRR*, abs/1406.1231, 2014. URL `http://arxiv.org/abs/1406.1231`.

James Devillers. *Neural networks in QSAR and drug design*. Academic Press, 1996. ISBN 978-0-12-213815-7.

P. Dhillon, D. P. Foster, and L. Ungar. Multi-view learning of word embeddings via CCA. In *Advances in Neural Information Processing Systems 24*, pages 199–207, 2011.

Bill Dolan, Chris Quirk, and Chris Brockett. Unsupervised construction of large paraphrase corpora: exploiting massively parallel news sources. In *Proceedings of the 20th international conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics. doi: 10.3115/1220355.1220406.

Hongying Du, Jie Wang, Zhide Hu, Xiaojun Yao, and Xiaoyun Zhang. Prediction of Fungicidal Activities of Rice Blast Disease Based on Least-Squares Support Vector Machines and Project Pursuit Regression. *Journal of Agricultural and Food Chemistry*, 56(22):10785–10792, 2008. doi: 10.1021/jf8022194. URL `http://pubs.acs.org/doi/abs/10.1021/jf8022194`.

Daniel P. W. Ellis and Nelson Morgan. Size matters: an empirical study of neural network training for large vocabulary continuous speech recognition. *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2:1013–1016, 1999. doi: http://doi.ieeecomputersociety.org/10.1109/ICASSP.1999.759875.

Dumitru Erhan, Pierre-Jean L'Heureux, Shi Yi Yue, and Yoshua Bengio. Collaborative filtering on a family of biological targets. *J. Chem. Inf. Model.*, 46(2):626–635, 2006. URL `http://dx.doi.org/10.1021/ci050367t`.

Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. Why does unsupervised pre-training help deep learning? In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pages 201–208, May 2010.

Mark Gales and Steve Young. The application of hidden Markov models in speech recognition. *Foundations and Trends in Signal Processing*, 1(3):195–304, 2008.

Michael A. Gelbart, Jasper Snoek, and Ryan P. Adams. Bayesian optimization with unknown constraints. In *Uncertainty in Artificial Intelligence*, pages 250–259, 2014.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning*, pages 1764–1772. JMLR Workshop and Conference Proceedings, 2014. URL `http://jmlr.org/proceedings/papers/v32/graves14.pdf`.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13:307–361, 2012.

Hynek Hermansky, Daniel P. W. Ellis, and Sangita Sharma. Tandem connectionist feature extraction for conventional HMM systems. In *PROC. ICASSP*, pages 1635–1638, 2000.

Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

Geoffrey E. Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

Geoffrey E. Hinton, James L. Mcclelland, and David E. Rumelhart. Distributed representations. In David E. Rumelhart and James L. Mcclelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 77–109. MIT Press, Cambridge, MA, 1986.

Geoffrey E. Hinton, Simon Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

Geoffrey E. Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012a. doi: 10.1109/MSP.2012.2205597. URL `http://dx.doi.org/10.1109/MSP.2012.2205597`.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *The Computing Research Repository (CoRR)*, abs/1207.0580, 2012b. URL `http://arxiv.org/abs/1207.0580`.

Navdeep Jaitly and Geoffrey E. Hinton. Learning a better representation of speech soundwaves using restricted Boltzmann machines. In *ICASSP*, pages 5884–5887, 2011.

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the Best Multi-Stage Architecture for Object Recognition? In *Proceedings of the 12th International Conference on Computer Vision (ICCV 2009)*, pages 2146–2153. IEEE, 2009.

Brian Kingsbury, Tara N. Sainath, and Hagen Soltau. Scalable Minimum Bayes Risk Training of Deep Neural Network Acoustic Models Using Distributed Hessian-free Optimization. In *Proc. Interspeech*, 2012.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

Richard A. Kronmal and Arthur V. Perterson, Jr. On the alias method for generating random variables from a discrete distribution. *The American Statistician*, 33(4):214–218, 1979.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Winter 1989.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

Peixun Liu and Wei Long. Current Mathematical Methods Used in QSAR/QSPR Studies. *International Journal of Molecular Sciences*, 10(5):1978–1998, 2009. ISSN 1422-0067. doi: 10.3390/ijms10051978. URL `http://www.mdpi.com/1422-0067/10/5/1978`.

Edward W. Lowe, Mariusz Butkiewicz, Nils Woetzel, and Jens Meiler. GPU-accelerated machine learning techniques enable QSAR modeling of large HTS data. In *Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), 2012 IEEE Symposium on*, pages 314–320, 2012.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *ACL*, pages 142–150, Portland, Oregon, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P11-1015`.

Nitin Madnani, Joel Tetreault, and Martin Chodorow. Re-examining machine translation metrics for paraphrase identification. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 182–190, Montreal, Canada, June 2012. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/N12-1019`.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014. URL `http://www.aclweb.org/anthology/P/P14/P14-5010`.

James Martens. Deep learning via Hessian-free optimization. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning*, pages 735–742, Haifa, Israel, June 2010. Omnipress. URL `http://www.icml2010.org/papers/458.pdf`.

James Martens and Venkatesh Medabalimi. On the expressive efficiency of sum product networks. *The Computing Research Repository (CoRR)*, abs/1411.7717, 2014. URL `http://arxiv.org/abs/1411.7717`.

James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning*, pages 1033–1040, New York, NY, USA, June 2011. ACM. ISBN 978-1-4503-0619-5.

Eric Martin, Prasenjit Mukherjee, David Sullivan, and Johanna Jansen. Profile-qsar: a novel meta-qsar method that combines activities across the kinase family to accurately predict affinity, selectivity, and cellular activity. *Journal of chemical information and modeling*, 51(8):1942–1956, 2011.

Tomáš Mikolov. *Statistical Language Models based on Neural Networks*. PhD thesis, Brno University of Technology, 2012.

Tomáš Mikolov, Anoop Deoras, Daniel Povey, Lukáš Burget, and Jan Černocký. Strategies for training large scale neural network language models. In *Proceedings of ASRU 2011*, pages 196–201. IEEE Signal Processing Society, 2011. ISBN 978-1-4673-0366-8. URL `http://www.fit.vutbr.cz/research/view_pub.php?id=9775`.

Tomáš Mikolov, Kai Chen, Greg S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013a. URL `http://arxiv.org/abs/1301.3781`.

Tomáš Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013b.

A. Mnih and G. E. Hinton. Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine Learning*, pages 641–648, 2007.

Andriy Mnih. *Learning Distributed Representations for Statistical Language Modelling and Collaborative Filtering*. PhD thesis, University of Toronto, 2010.

Andriy Mnih and Geoffrey E. Hinton. A scalable hierarchical distributed language model. In *Advances in Neural Information Processing Systems 21*, pages 1081–1088, 2009.

Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1751–1758, 2012.

Volodymyr Mnih. Cudamat: a CUDA-based matrix class for python. Technical Report UTML TR 2009-004, Department of Computer Science, University of Toronto, November 2009.

Volodymyr Mnih and Geoffrey E. Hinton. Learning to detect roads in high-resolution aerial images. In *Proceedings of the 11th European Conference on Computer Vision (ECCV)*, September 2010.

Abdel-rahman Mohamed, George E. Dahl, and Geoffrey E. Hinton. Deep belief networks for phone recognition. In *NIPS Workshop on Deep Learning for Speech Recognition and Related Applications*, 2009.

Abdel-rahman Mohamed, George E. Dahl, and Geoffrey E. Hinton. Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):14 –22, jan. 2012. ISSN 1558-7916. doi: 10.1109/TASL.2011.2109382.

Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In Robert G. Cowell and Zoubin Ghahramani, editors, *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics, 2005.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814, Haifa, Israel, June 2010. Omnipress. URL `http://www.icml2010.org/papers/432.pdf`.

Radford M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113, 1992.

Olga Obrezanova, Gábor Csányi, Joelle M.R. Gola, and Matthew D. Segall. Gaussian processes: a method for automatic QSAR modeling of ADME properties. *Journal of chemical information and modeling*, 47(5):1847–1857, 2007.

Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. *English Gigaword Fifth Edition LDC2011T07*. Linguistic Data Consortium, Philadelphia, 2011. ISBN 1-58563-581-2. `http://catalog.ldc.upenn.edu/LDC2011T07`.

Steven M. Paul, Daniel S. Mytelka, Christopher T. Dunwiddie, Charles C. Persinger, Bernard H. Munos, Stacy R. Lindborg, and Aaron L. Schacht. How to improve R & D productivity: the pharmaceutical industry's grand challenge. *Nat Rev Drug Discov*, 9(3):203–214, Mar 2010.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

Jordan B Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1):77–105, 1990.

Rajat Raina, Anand Madhavan, and Andrew Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 873–880, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553486.

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

Tara N. Sainath, Brian Kingsbury, Bhuvana Ramabhadran, Petr Fousek, Petr Novak, and Abdel rahman Mohamed. Making Deep Belief Networks Effective for Large Vocabulary Continuous Speech Recognition. In *Proc. ASRU*, 2011.

Tara N. Sainath, Brian Kingsbury, Abdel-rahman Mohamed, George E. Dahl, George Saon, Hagen Soltau, Tomas Beran, Aleksandr Y. Aravkin, and Bhuvana Ramabhadran. Improvements to deep convolutional neural networks for LVCSR. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 315–320, 2013. doi: 10.1109/ASRU.2013.6707749.

Ruslan Salakhutdinov and Geoffrey Hinton. Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pages 448–455, 2009a.

Ruslan Salakhutdinov and Geoffrey E. Hinton. Replicated softmax: an undirected topic model. In Y. Bengio, D. Schuurmans, J.D. Lafferty, C.K.I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1607–1614. Curran Associates, Inc., 2009b. URL `http://papers.nips.cc/paper/3856-replicated-softmax-an-undirected-topic-model.pdf`.

Jude W. Shavlik, Raymond J. Mooney, and Geoffrey G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6:111–143, 1991. doi: 10.1007/BF00114160. URL `http://dx.doi.org/10.1007/BF00114160`.

Hongzong Si, Tao Wang, Kejun Zhang, Yun-Bo Duan, Shuping Yuan, Aiping Fu, and Zhide Hu. Quantitative structure activity relationship model for predicting the depletion percentage of skin allergic chemical substances of glutathione. *Analytica Chimica Acta*, 591(2):255 – 264, 2007.

Paul Smolensky. Information Processing in Dynamical Systems: Foundations of Harmony Theory. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, chapter 6, pages 194–281. MIT Press, Cambridge, 1986.

Jasper Snoek, Hugo Larochelle, and Ryan Prescott Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, pages 2960–2968, 2012.

Jasper Snoek, Kevin Swersky, Richard Zemel, and Ryan Prescott Adams. Input warping for bayesian optimization of non-stationary functions. In *Advances in Neural Information Processing Systems Workshop on Bayesian Optimization*, 2013.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 801–809. Curran Associates, Inc., 2011a.

Richard Socher, Cliff Lin, Andrew Y Ng, and Christopher Manning. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning*, pages 129–136. ACM, 2011b.

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2011)*, pages 151–161. ACL, 2011c.

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

Hagen Soltau, George Saon, and Brian Kingsbury. The IBM Attila Speech Recognition Toolkit. In *Proc. SLT*, 2010.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15: 1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Ilya Sutskever, James Martens, and Geoffrey E. Hinton. Generating text with recurrent neural networks. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning*, pages 1017–1024, New York, NY, USA, June 2011. ACM. ISBN 978-1-4503-0619-5.

Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 1139–1147. JMLR Workshop and Conference Proceedings, May 2013. URL `http://jmlr.org/proceedings/papers/v28/sutskever13.pdf`.

Vladimir Svetnik, Andy Liaw, Christopher Tong, J. Christopher Culberson, Robert P. Sheridan, and Bradley P. Feuston. Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling. *Journal of Chemical Information and Computer Sciences*, 43(6):1947–1958, 2003.

Gerald Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3-4):257–277, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992697. URL `http://dx.doi.org/10.1007/BF00992697`.

Tijmen Tieleman. Gnumpy: an easy way to use GPU boards in Python. Technical Report UTML TR 2010-002, University of Toronto, Department of Computer Science, 2010.

J. Turian, L. Ratinov, and Y. Bengio. Word representations: A simple and general method for semi-supervised learning. In *ACL*, pages 384–394, 2010.

Laurens van der Maaten and Geoffrey E. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103, Helsinki, Finland, 2008. ACM.

Max Welling, Michal Rosen-zvi, and Geoffrey E. Hinton. Exponential family harmoniums with an application to information retrieval. In L.K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1481–1488. MIT Press, 2005.

David A Winkler. The role of quantitative structure-activity relationships (qsar) in biomolecular discovery. *Briefings in bioinformatics*, 3(1):73–86, 2002.

David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, October 1996. ISSN 0899-7667. doi: 10.1162/neco.1996.8.7.1341. URL `http://dx.doi.org/10.1162/neco.1996.8.7.1341`.

Frank Wood, Jan Gasthaus, Cédric Archambeau, Lancelot James, and Yee Whye Teh. The sequence memoizer. *Communications of the Association for Computing Machines*, 54(2):91–98, 2011.

Laurent Younes. Parameter inference for imperfectly observed Gibbsian fields. *Probability Theory Related Fields*, 82:625–645, 1989.

Steve Young. Statistical modeling in continuous speech recognition (CSR). In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pages 562–571, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-800-1.