New Architectures for Very Deep Learning

Doctoral Dissertation submitted to the
Faculty of Informatics of the Università della Svizzera Italiana
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Rupesh Kumar Srivastava

under the supervision of
Jürgen Schmidhuber

February 2018

Dissertation Committee

Dissertation accepted on 1 February 2018

Rupesh Kumar Srivastava
Lugano, 1 February 2018

Dedicated to Govind Lal Srivastava

We can only learn what we already almost know.

Patrick Winston

Abstract

Artificial Neural Networks are increasingly being used in complex real-world applications because many-layered (i.e., deep) architectures can now be trained on large quantities of data. However, training even deeper, and therefore more powerful networks, has hit a barrier due to fundamental limitations in the design of existing networks. This thesis develops new architectures that, for the first time, allow very deep networks to be optimized efficiently and reliably. Specifically, it addresses two key issues that hamper credit assignment in neural networks: *cross-pattern interference* and *vanishing gradients*.

Cross-pattern interference leads to oscillations of the network's weights that make training inefficient. The proposed Local Winner-Take-All networks reduce interference among computation units in the same layer through local competition. An in-depth analysis of locally competitive networks provides generalizable insights and reveals unifying properties that improve credit assignment.

As network depth increases, vanishing gradients make a network's outputs increasingly insensitive to the weights close to the inputs, causing the failure of gradient-based training. To overcome this limitation, the proposed Highway networks regulate information flow across layers through additional skip connections which are modulated by learned computation units. Their beneficial properties are extended to the sequential domain with Recurrent Highway Networks that gain from increased depth and learn complex sequential transitions without requiring more parameters.

Acknowledgements

I would like to thank my supervisor Jürgen Schmidhuber for the opportunity to conduct this research, and for the countless discussions that helped me understand how to think about artificial intelligence. A special thanks also goes to Faustino Gomez, who encouraged and supported me continuously throughout my PhD.

I am very grateful to my co-authors Jonathan Masci, Julian Zilly, Klaus Greff, Marijn Stollenga and Sohrob Kazerounian for our fruitful research collaborations. Together with the rest of IDSIA members, they gave me a fun and stimulating work environment for several years. In particular, I must thank Cinzia Daldini, who helped me in every official matter I was ever involved in during my PhD. I'm also grateful to Jianfeng Gao, Li Deng and Xiaodong He, who gave me the opportunity to spend an incredibly stimulating summer at Microsoft Research in Redmond.

Thanks to Bas Steunebrink, Faustino Gomez, Jan Koutník and Wonmin Byeon for reading and suggesting improvements to drafts of this thesis, and to my dissertation committee members – Antonio Carzaniga, Michael Bronstein, Ruslan Salakhutdinov and Sepp Hochreiter – for taking the time to review my work. Finally, I am forever grateful to my family for their love and support in pursuing this journey.

Contents

Figures

Tables

Acronyms

AI Artificial Intelligence

CNN Convolutional Neural Network

FNN Feedforward Neural Network

LCN Locally Competitive Networks

LSTM Long Short-Term Memory

LWTA Local Winner-Take-All

MLP Multilayer Perceptron

NN Neural Network

ReL Rectified Linear

RHN Recurrent Highway Network

RNN Recurrent Neural Network

WT Weight Tying

WTA Winner-Take-All

Chapter 1

Introduction

1.1 Motivation

The objective of building powerful computing machines is to program them to perform tasks that are too complex, time-consuming, expensive, dangerous or repetitive to be performed by humans. Many of these tasks require machines to exhibit intelligent behavior typically shown by animals, such as perception, reasoning or planning. For example, a computer programmed as a telemedicine agent is required to understand a patient's health condition through a series of questions and answers, and then suggest an appropriate course of action. To take another example, a robot may be required to assist in disaster relief efforts by planning the rescue of an affected population.

Broadly, there are two ways to program machines to exhibit desired behavior (i.e., produce certain outputs in response to certain inputs) within the limits of computer science [Gödel, 1931]. The first is to manually program the computer to produce the behavior based on expert knowledge, which has been the common approach in the design of many control systems. However, for many tasks sufficient expert knowledge is unavailable or incomplete. An example of this is in the field of computer vision where decades of research into hand-designed image processing algorithms has not resulted in high-performing general object recognition [Russakovsky et al., 2015]. Even if expert domain knowledge is available, writing programs for all of the large number of tasks humans are interested in is likely to take prohibitive amounts of time and effort. So the first approach is attractive for certain narrowly-defined tasks, but not a satisfactory long-term solution. The second approach is to define a specification that can be used to test the desired behavior, and then run a search procedure in the space of programs. The advantage of this *program search* approach is that it is very general, with the primary disadvantage being that the search may take too long to succeed in practice.

It was recognized early on that program search can be substantially sped up if, like intelligent animals, machines can use examples of correct behavior or experiences to guide the search for desired programs. The primary hurdle in obtaining these speedups is the *credit assignment problem* [Newell, 1955; Samuel, 1959; Minsky, 1961]. In order for a machine to improve its own behavior by learning from experience – what is called *machine learning* – it is necessary for it to properly attribute its successes and failures to its past decisions such that it can apply the appropriate changes.

Artificial neural networks (henceforth referred to as NNs or simply "networks") are a biologically inspired class of mathematical models used to implement programs in machine learning [Deng and Chen, 2014; Schmidhuber, 2015; Goodfellow et al., 2016]. They consist of a set of simple interconnected processing units, usually organized in groups called *layers*, which can in principle approximate arbitrary functions [Cybenko, 1989; Hornik et al., 1989] and run arbitrary programs [Siegelmann and Sontag, 1995]. Crucially, if the functions implemented by the processing units are differentiable, there exists an efficient procedure for credit assignment in arbitrary NNs: backpropagation [Linnainmaa, 1970, 1976; Werbos, 1981]. This algorithm computes the gradients of a network's outputs with respect its parameters efficiently, allowing gradient-based search techniques to be used for learning.

In recent years, this property combined with two key enabling factors has generated wide interest in NNs, in particular for perception problems [LeCun et al., 2015; Schmidhuber, 2015; Malik, 2017], although they have been studied since the 1960s. The first of these enablers is the abundance of labeled data, fueled by the ubiquity of the Internet and dedicated efforts to curate and label datasets for the purposes of machine learning. The second is availability of increasingly large amounts of computational power, which is required to learn from large datasets with high variability and complexity, or complicated simulations of physical systems.

Unfortunately, as more complex learning tasks are targeted, success of NNs is hampered by several difficulties in training them. In particular, the efficiency of gradient-based learning is highly dependent on the design of the neural network, i.e., its *architecture*. Credit assignment using backpropagation in increasingly large and complex networks becomes more and more difficult, and gradient based training algorithms face a multitude of obstacles such as plateaus, saddle points, and local minima. The number of layers in the network, the number of computation units in each layer, the non-linear functions implemented by the units, and their connectivity all play crucial roles in determining how fast the network can be trained in practice or if the training succeeds at all. These choices also affect generalization, i.e., performance on unseen input data, since an architecture with a more suitable inductive bias [Mitchell, 1980] will result in better generalization for certain problems.

A particularly critical issue in NNs is that as the number of layers connected in a chain (its *depth*) increases, credit assignment through backpropagation becomes increasingly difficult or even altogether infeasible [Hochreiter et al., 2001]. This is an important limitation since it is well known that increasing the number of layers is an efficient way to increase the potential complexity of functions that can be modeled using NNs [Bengio et al., 2013]. Many recent empirical breakthroughs in supervised machine learning have been achieved using large and deep NNs. Network depth played perhaps the most important role in these successes. For instance, within just a few years, the top-5 image classification accuracy on the 1000-class ImageNet dataset increased from ∼84% [Krizhevsky et al., 2012] to ∼95% [Szegedy et al., 2014; Simonyan and Zisserman, 2014] using deeper networks with rather small receptive fields per layer [Ciresan et al., 2011, 2012]. Similar trends in other domains such as speech recognition have also underscored the superiority of deeper networks in terms of accuracy and/or efficiency [Yu et al., 2013; Zeiler et al., 2013]. Therefore, network architectures that improve credit assignment and have suitable inductive biases for practical problems can improve and scale the application of machine learning methods across a variety of domains, from perception to modeling, control and decision-making.

Based on these motivations, the central motivation for this thesis is *to develop neural network architectures that permit efficient and reliable training of large and deep neural networks*. To this end, we design network architectures that improve credit assignment among computation units in the same layer through local competition, and across multiple layers through additional connections in the network which are modulated by learned computation units.

1.2 Contributions

In order to achieve our goals, we address two key issues that hamper credit assignment in neural networks: *cross-pattern interference* [Sutton, 1986] and *vanishing gradients* [Hochreiter et al., 2001].

First, we propose a new architecture based on local competition called Local Winner-Take-All (LWTA). In this architecture, units in a layer are arranged in groups, and competition among units in each group prevents interference between learning signals from multiple training patterns. LWTA networks match or exceed the performance of other network types in supervised learning experiments, and they also have a reduced tendency to forget previously learned tasks when abruptly trained on a new task.

Next, we explore a unifying interpretation of LWTA together with recently popular rectified linear [Jarrett et al., 2009] and maxout [Goodfellow et al., 2013b] networks as a collection of an exponential number of linear subnetworks. Several experiments are performed to understand the extent and utility of this interpretation: visualization of the active subnetworks, their training behavior, suitability of subnetwork identities for classification, and impact of training with dropout. The results confirm that implicit gating of input patterns based on their mutual similarity plays a key role in the success of the three network types.

A main limitation of the above developments is that local competition improves shallow NNs but not deep ones. In particular, NNs with more than about 20 layers still remain very hard to train, suggesting that competition between units improves credit assignment among units in each layer, but not across several layers. To address the vanishing gradient problem that causes this, we combine insights from LWTA and Long Short-Term Memory [Hochreiter and Schmidhuber, 1997b]. The resulting Highway network architecture overcomes the challenge of training very deep networks reliably and produces state-of-the-art results on the CIFAR-100 benchmark. When training a very deep network on two different tasks, we show that a Highway network automatically learns to utilize more layers for the "harder" task that intuitively requires more computational resources. Comparative experiments are used to demonstrate how variants of Highway networks can lead to similar or different results depending on the task.

Finally, we propose Recurrent Highway Networks (RHNs), which use Highway layers to extend and improve LSTM networks. They address another unsolved challenge related to depth: training recurrent neural networks with large depths in the recurrent transition function which maps previous states to new states during sequence processing. Using RHNs, we obtain increasingly improved results for progressively deeper networks with constant number of parameters, and set a new state of the art on the challenging `enwik8` and `text8` benchmarks for modeling text sequences from Wikipedia.

1.3 Overview

The rest of the thesis is organized as follows: Chapter 2 introduces basic concepts and establishes the relevant background for the rest of the thesis. Chapter 3 identifies the key difficulties that prevent the effective use of NNs for machine learning. It also provides a summary of past work that has attempted to address these difficulties. Chapter 4 introduces Local Winner-Take-All LWTA networks, a new architecture that utilizes local competition and guards against cross-pattern interference. Chapter 5 presents a unifying analysis of three different locally competitive architectures, including LWTA networks, and demonstrates the extent of their conceptual similarities. Chapter 6 develops the Highway network architecture that makes reliable training of very deep networks possible and evaluates its performance on several image classification datasets. Chapter 7 extends the Highway architecture to the sequential setting resulting in Recurrent Highway Networks. Profiting from their increased recurrence depth, Recurrent Highway Networks outperform the state-of-the-art on two challenging sequence modeling benchmarks. Finally Chapter 8 closes the thesis with a summary of the proposed architectures and a discussion of promising directions for future work.

Chapter 2

Background

This chapter reviews the basic concepts that underlie the use of NNs for machine learning. The methods developed in this thesis are studied in the context of supervised machine learning, so the treatment is limited to this setting. More detailed exposition can be found in various textbooks on the subject [Bishop, 2006; Murphy, 2012; Goodfellow et al., 2016].

2.1 The Credit Assignment Problem

The desire to speed up search for programs brought up the *credit assignment problem* early on in research on Artificial Intelligence (AI). This was one of the challenges that Newell [1955]; Samuel [1959]; Minsky [1961] and others identified in the context of developing programs to play games such as checkers and chess. These games were used as a proxy for understanding how a computer can successfully solve complex sequential decision making problems similar to humans. It was realized that it is particularly difficult to write expert programs for such problems, and so a learning system that could automatically design such programs would be necessary. Such a system would immediately face the credit assignment problem: if a machine makes several decisions to arrive at a result – say dozens of moves in a game of chess – how to know which decisions should have been different (and how) to obtain a better result? If this can not be known, any search for solution programs for complex problems involving a large number of intermediate decisions is unlikely to succeed in reasonable time. A method of measuring the impact of intermediate decisions (or outputs) of the program on the final outcome can direct the search towards better programs, leading to fast learning. For example, Newell [1955] suggested to break problems into subgoals, which would then provide additional signals for reinforcing useful decisions.

The credit assignment problem of sequential decision making is also relevant to the learning of static functions mapping inputs to outputs. In order to learn desired functions, a machine needs to identify which aspects of the data are important and to what extent. For example, a chess-playing machine might identify board configurations that are good or bad by selecting certain aspects of the configuration as *features* on which further computations are performed. It is this capability of learning methods that gives them an edge over hand-designed programs which may lack the features crucial for obtaining good results. If one considers a learning method that produces internal representations of the data during a multi-step computation of the output, then these representations act like intermediate decisions similar to chess moves. This challenge is related to the *problem of new terms* [Samuel, 1959; Dietterich et al., 1982], and is nowadays referred to as the problem of *representation learning* [Bengio et al., 2013]. Assigning appropriate credit to internal representations can substantially speed up learning and make it practical [Rumelhart et al., 1986].

Some researchers explicitly distinguish between two subproblems of the credit assignment problem [Sutton, 1984]: the *temporal* version, where appropriate credit for a final outcomes should be assigned to earlier decisions, and the *structural* version, where credit for a final outcome should be assigned to participating computing units. The techniques developed in this thesis primarily attempt to improve structural credit assignment. In certain cases, as done in Chapter 7, the same techniques can be used to aid in temporal credit assignment.

2.2 Supervised Learning

In this thesis, techniques for improving credit assignment are developed in the context of *supervised* learning problems, in particular the setting of single-label classification [Bishop, 2006]. We are given a *training set* $\mathscr{S}$ of independently and identically distributed (i.i.d.) pairs of input patterns and targets $(\mathbf{x}, y)$ for training. “Supervised” refers to the availability of targets for guiding the training. Each input pattern $\mathbf{x}$ is a real-valued vector associated with a single class label $y \in \{0, 1, \dots, \mathrm{K}-1\}$ where K is the total number of classes. The objective is to learn a *classifier*: a function $F(\mathbf{x})$ that maps the inputs to class labels such that the *classification error rate* (CER) or a related *objective function* is minimized for a *test set* of input-target pairs. The test set $\mathscr{S}'$ is sampled from the same distribution as the training set, and does not contain elements in common with the training set.

The CER is defined as

$$CER(F, \mathscr{S}') = \sum_{(\mathbf{x},y)\in\mathscr{S}'} \begin{cases} 0 & \text{if } (F(\mathbf{x}) = y), \\ 1 & \text{otherwise.} \end{cases} \tag{2.1}$$

Another problem setting, investigated in Chapter 7, which can be formulated as a supervised classification problem is that of modeling discrete sequences. These are sequential prediction problems such as predicting the next character or word in a natural language text or the next set of music symbols in a piano roll. Although the raw training data available in these cases are only samples of sequences, input-target pairs can be constructed by time-delaying input sequences to produce targets. For example, for the problem of word-level language modeling, training datasets are prepared by using words from raw sentences as input sequences, and the same word sequences shifted by one time step as target sequences. Once the data is prepared in this form, the methods for supervised classification are used to learn functions that predict the next symbol in the sequence at each time step. The main differences from the classification setting are that:

- The class of functions used for learning is different, capable of processing and producing sequential data.
- The learned function are interpreted and used as *generative* models of the data, rather than *discriminative* classifiers. Such models can be used for generating new samples from the data distribution, or for biasing/evaluating the outputs of another model.

2.2.1 Maximum Likelihood Estimation

A principled approach for supervised classification that results in a consistent training and evaluation methodology is to train *probabilistic* classifiers using *maximum likelihood estimation* (MLE) [Bishop, 2006; Murphy, 2012]. Assume that function $F(\mathbf{x})$ maps input patterns to probabilities $p(C_k|\mathbf{x}), k \in \{0, 1, \dots, K-1\}$ instead of classes. F is represented with a (sufficiently powerful) model such as an NN with a set of adjustable parameters arranged in a vector θ. The desired values of the parameters are fixed but unknown. The probability assigned by the classifier to the target class is $p(y|\mathbf{x}, \theta)$. The class prediction for computing the CER is taken to be the label with the highest assigned probability $\arg\max_k p(C_k|\mathbf{x})$.

The *likelihood function* $L(\theta)$ is defined as the probability assigned by the classifier to all samples in the training set for a given θ. Since the training samples are i.i.d.,

$$L(\theta) = p(\mathscr{S}|\theta) = \prod_{(\mathbf{x},y)\in\mathscr{S}} p(y|\mathbf{x},\theta). \tag{2.2}$$

Maximum likelihood training involves estimating the value of the parameters for which the likelihood is maximized:

$$\theta_{MLE} = \arg\max_{\theta} L(\theta). \tag{2.3}$$

Intuitively, maximizing the likelihood of correctly classifying the training set also minimizes the CER. It is common practice to minimize the negative log of the likelihood instead, giving

$$\theta_{MLE} = \arg\min_{\theta} \sum_{(\mathbf{x},y)\in\mathscr{S}} -\log p(y|\mathbf{x},\theta). \tag{2.4}$$

This does not change the results since log is a monotonically increasing function, but is done for multiple reasons: it increases the stability of computations, it is more compatible with optimization literature which often considers minimization instead of maximization problems, and it closely relates the quantity being minimized to those used in training approaches other than MLE.

2.2.2 Underfitting & Overfitting

MLE casts learning as a specific optimization (or search) problem. One can define a model F, the search space for θ, and then invoke a suitable optimization algorithm for minimizing the objective function. This may not always succeed: the obtained model may fail to produce low error rate on the training set. This can happen because the selected model is not powerful enough to capture all the information in the data for accurate prediction, or because the optimization procedure fails to exploit its full expressive power – a phenomenon termed *underfitting*. Underfitting can be addressed by using models that can learn more complex functions, but care must be taken since complex models can be more difficult to optimize.

On the other hand, successfully obtaining a low error rate on the training set may still result in poor performance on unseen data from the test set. This happens due to *overfitting* – the model's expressive power allows it to rely on unique properties of the randomly selected training samples instead of learning properties that can generalize to unseen test samples. Overfitting and underfitting are challenges faced in almost every learning task and combating these issues is a large and active field of study [Burnham and Anderson, 2003].

In supervised machine learning, overfitting is often controlled through model *regularization*. It refers to techniques for encouraging simpler model configurations over complex ones, according to some notion of simplicity, in accordance with Occam's razor [Pearl, 1978; Angluin and Smith, 1983; Blumer et al., 1987]. The simplest regularization method is to add an additional term $\lambda R(\theta)$ to the maximum likelihood objective function, where λ is a weighting factor and R is a regularization function of the model parameters. Most common regularization functions have a Bayesian interpretation as priors over the model parameters. For example, setting $\lambda R(\theta) = \theta^\top \theta$ (called L_2 *weight decay*) corresponds to assuming that the prior distribution of the model parameters is Gaussian [Bishop, 2006]. Some regularization techniques commonly used with NNs are discussed in subsection 2.4.2.

2.3 Neural Networks

NNs are used as models for a variety of machine learning problems, whether they be in supervised, unsupervised or reinforcement learning [Bishop, 2006; Sutton and Barto, 1998]. Some early models that can be characterized as types of NNs (e.g. Perceptrons [Rosenblatt, 1958], Neocognitron [Fukushima, 1980b]) were inspired by models of information processing in the human brain, while others such as GMDH networks [Ivakhnenko, 1971] were motivated by data analysis and control problems. Modern NNs are descendants of these models, but depart significantly in design from their biological inspirations. Schmidhuber [2015] provides a detailed survey and historical overview of NN developments. In the rest of the chapter, we cover fundamental NN concepts relevant to the rest of the thesis. More details on all these topics can be found in books by Bishop [1995, 2006]; Graves [2012]; Goodfellow et al. [2016].

A wide variety of NNs exist, but in this thesis we will only consider deterministic networks trained in a supervised setting. An NN can be described as a computational graph of simple processing units (also called nodes or neurons) connected by weighted directed edges. Each unit computes some function of the inputs that it receives from units connected to it and emits the obtained outputs (called its activation). The set of units in the graph are often divided into subsets called layers; the number of units in a layer is referred to as its *width*. Subsets of units that receive inputs to the network are called *input layers* and subsets that produce network outputs are called *output layers*. The term *hidden layers* is commonly used to refer to all other layers. The length of the longest path from the input to output layers is referred to as the *depth* of the network.

NNs with acyclic graphs are called *feedforward NNs* or FNNs, and those with cyclic graphs *recurrent* NNs or RNNs. Unlike FNNs, RNNs contain connections that have time lags – they transfer outputs emitted by source units in the past to destination units (which may be the same as source units) at future time steps during processing of sequential inputs. In mathematical notation, it is often clearer to write NN equations in terms of computations at the layer level rather than the unit level thanks to the succinctness of matrix notation, so this convention is adopted throughout the thesis, except when noted otherwise.

FNNs with a single hidden layer are known to be universal approximators [Cybenko, 1989; Hornik et al., 1989], i.e., given enough hidden units, they can approximate any continous function up to any desired accuracy. Similarly, RNNs are known to be Turing-complete [Siegelmann and Sontag, 1995] in theory.

A series of results, starting from the work of [Håstad, 1987; Håstad and Goldmann, 1991] for simple threshold circuits, and continuing to present day [Bengio et al., 2006; Bengio and Delalleau, 2011; Bianchini and Scarselli, 2014; Cohen et al., 2016] show that there are function classes which can be represented far more efficiently by deeper networks than shallower networks. In general, for most practical problems of interest, it is unknown if they fall in the above classes of functions. However, a pattern strong experimental results on challenging benchmarks in the domains of computer vision [Krizhevsky et al., 2012; Szegedy et al., 2014; Simonyan and Zisserman, 2014; He et al., 2016a] and speech recognition [Hinton et al., 2012a; Graves et al., 2013] indicates that this might be the case.

2.3.1 Types of Networks & Layers

Multilayer Perceptron

The simplest type of NNs consist of layers connected to form a linear graph. These networks are called *Multilayer Perceptrons* (MLP)s, and they typically consist of *fully-connected* layers.

An MLP represents a function $F : \mathbb{R}^m \rightarrow \mathbb{R}^n$ with L layers of computation units. The ℓ^{th} layer ($\ell \in \{1, 2, \dots, \mathrm{L}\}$) computes a non-linear transformation of its input $\mathbf{x}_\ell$ and produces its output $\hat{\mathbf{y}}_\ell$. Since an MLP is a linear graph, the input to each layer is always the output of the previous layer, i.e., $\mathbf{x}_\ell = \hat{\mathbf{y}}_{\ell-1}$. Thus, $\mathbf{x}_1 \in \mathbb{R}^m$ is the external input to the network and $\hat{\mathbf{y}}_L \in \mathbb{R}^n$ is the network's output. [1]

The transformation implemented by a fully-connected layer involves first

[1]To keep notation intuitive, we use $\hat{\mathbf{y}}$ to denote layer outputs and $\mathbf{y}$ to denote desired outputs in this chapter. However, the desired output is not used in the rest of this thesis, so we simply use $\mathbf{y}$ to denote layer outputs.

computing an affine function $\mathbf{z}_\ell$ of the input parameterized by weights $\mathbf{W}_\ell$ and bias $\mathbf{b}_\ell$, followed by application of a non-linear activation function f. To summarize:

$$\mathbf{x}_\ell = \hat{\mathbf{y}}_{\ell-1} \tag{2.5a}$$

$$\mathbf{z}_\ell = \mathbf{W}_\ell \mathbf{x}_\ell + \mathbf{b}_\ell \tag{2.5b}$$

$$\hat{\mathbf{y}}_\ell = f(\mathbf{z}_\ell). \tag{2.5c}$$

If layer ℓ has N_ℓ number of units, the dimensions of $\mathbf{W}_\ell$ are $N_\ell \times N_{\ell-1}$ and $\mathbf{b}_\ell$ is an N_ℓ-dimensional column vector. The set of all weights and biases is often collectively referred to as the parameters or weights of the network. Activation functions are typically unit-wise non-linearities such as the *logistic sigmoid* ($f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$) or the hyperbolic tangent ($f(x) = tanh(x)$), but in general they can take other forms or incorporate their own learnable parameters.

The procedure of computing the output of a network given an input pattern is termed a *forward pass*. It is the sequential activation of all the layers in the network according to Equation 2.5 in topological ordering for a given input. In the case of an MLP, $\mathbf{x}_1$ is set to the given input vector and then layers $1, 2, \dots, \mathrm{L}$ are activated in order to compute $\hat{\mathbf{y}}_\mathrm{L}$.

Fully Recurrent layer

There are many ways of adding time-lag connections to NNs, and as a result RNNs can have various topologies. See Lang et al. [1990]; Werbos [1990]; Elman [1990] for some early architectures. The simplest types of recurrent layer is called a *fully recurrent* layer or just, a *simple* RNN layer. Each unit in the layer has one-step time lag connections to all other units (called recurrent connections), and has incoming connections from all the inputs. As a result, the output of the layer at any time step t is a function of not just the inputs but also its output at the previous time step.

At any time step $t > 0$ of sequence processing, let $\mathbf{x}^{[t]} \in \mathbb{R}^m$ be the input to the RNN and $\hat{\mathbf{y}}^{[t]} \in \mathbb{R}^n$ be its output. $\hat{\mathbf{y}}^{[0]}$ is typically set to be the zero vector, but in some cases it is set to another constant vector or learned during training. Let us denote by $\mathbf{W} \in \mathbb{R}^{n \times m}$ the matrix of connection weights from the inputs to the units, by $\mathbf{b} \in \mathbb{R}^n$ a vector of bias weights and by $\mathbf{R} \in \mathbb{R}^{n \times n}$ the matrix of recurrent connection weights. The forward pass equations for a simple recurrent layer are:

$$\mathbf{z}^{[t]} = \mathbf{W}\mathbf{x}^{[t]} + \mathbf{R}\hat{\mathbf{y}}^{[t-1]} + \mathbf{b} \tag{2.6a}$$

$$\hat{\mathbf{y}}^{[t]} = f(\mathbf{z}^{[t]}). \tag{2.6b}$$

LSTM

The simple RNN layer described above is theoretically powerful, but ill-suited for gradient-based learning, as discussed in subsection 3.1.5. Due to this reason, the most successful recurrent architecture is *Long Short-Term Memory* (LSTM) [Hochreiter and Schmidhuber, 1997b], which was specifically designed to address the limitations of simple RNNs. The detailed architecture and its advantages are described in subsection 3.1.6.

Convolutional and Pooling layers

For data with spatial regularities such as images, the most popular choice of architecture used is a Convolutional Neural Network (CNN) [LeCun, 1989; LeCun et al., 1998]. Like an MLP, it is a linear graph of layers, with special layers that are designed to take advantage of the spatial relationships in the data. These layers are called *convolutional* layers, which apply a modification of the transformation used in fully-connected layers. In addition to convolutional layers, CNNs often utilize *pooling* or *sub-sampling* layers to reduce the spatial dimensionality of their inputs. Common types of pooling layers use the *max* or *mean* operations, applied over fixed size receptive fields of the inputs. Further details of these layers can be found in Goodfellow et al. [2016].

2.3.2 Output Activation Functions

The activation function used for the output layer of an NN depends on the interpretation of the outputs according to the problem. If the NN is being trained to predict unbounded real-valued targets, the identity activation $f(\mathbf{x}) = \mathbf{x}$ is commonly used. Similarly, if the targets are bounded in the range $[-1, 1]$, the *tanh* activation function is a natural choice.

As discussed in section 2.2, the outputs of an NN trained as a probabilistic classifier or discrete sequential predictor are interpreted as a categorical probability distribution over the labels. To enable this, the activation f of the output layer of the NN must be changed such that the outputs always form a valid probability distribution. The *softmax* activation function serves this purpose:

$$f(\mathbf{x}) = \text{softmax}(\mathbf{x}) = \frac{e^{\mathbf{x}}}{\Sigma\, e^{\mathbf{x}}} \tag{2.7}$$

2.4 Training Neural Networks

2.4.1 Objective Functions

The forward pass is used to compute the outputs or predictions of a network for a given input. To train the network on a given dataset using the method discussed in section 2.2, an objective (also called the loss or error) function should be minimized, which measures the discrepancy between the networks predictions and the targets. Based on the MLE approach (Equation 2.4), the error function to be minimized in the supervised classification setting is

$$\begin{aligned} E(\mathbf{y}, \hat{\mathbf{y}}_{\mathrm{L}}) &= \sum_{(\mathbf{x},y)\in\mathscr{S}} -\log p(\hat{\mathbf{y}} = \mathbf{y}|\mathbf{x}, \theta) && (2.8) \\ &= \sum_{(\mathbf{x},y)\in\mathscr{S}} -\mathbf{y}^{\top} \log \hat{\mathbf{y}}_{\mathrm{L}}, && (2.9) \end{aligned}$$

where $\mathbf{y}$ is a *one-hot* vector encoding of the scalar target y as a binary vector with all elements equal to zero except the y^{th} element which is set to one. For sequence prediction tasks, the error function is computed for each time step and then summed over the sequence length for each sequence.

2.4.2 Regularization

Apart from weight decay, three other commonly used NN regularization techniques are also used in this thesis. The particular choice of regularization for each experiment is primarily motivated by the need to ensure reasonably fair comparisons to recent work.

The first technique is to add random noise to the inputs, weights or activations of the network [Holmström and Koistinen, 1992; Bishop, 1995; Hinton et al., 2012b]. *Dropout* [Hinton et al., 2012b; Wan et al., 2013; Gal, 2015] is a modern and most effective variation of this technique which inserts multiplicative Bernoulli noise in the activations or weights of the network during training. For testing the network on a test set, the noise is removed and the weights are suitably scaled to account for the absence of noise.

The second technique is called *early stopping* [Holmström et al., 1989], and utilizes a held-out *validation set* (separate from the training and test sets) to detect overfitting during training. This is done by periodically monitoring a measure of performance (such as the CER) on both the training and validation sets during optimization, and stopping training when the training set performance continues to improve but the validation set performance starts to worsen.

Finally, *batch normalization* [Ioffe and Szegedy, 2015] is a recent technique which was introduced to improve the speed of gradient-based training of FNNs, but also acts as very effective regularization method in practice. Unlike the above two methods, this technique has not been widely adopted for preventing overfitting in RNNs.

2.4.3 Backpropagation

Backpropagation (often referred to as *backprop*) is the most common method of assigning credit in NNs. It is an algorithm for efficiently computing derivatives of the outputs of a network with respect to its parameters. While earlier work had already used the algorithm for efficient computation of derivatives in NN-like functions [Linnainmaa, 1970; Dreyfus, 1973], it was the work of Werbos [1974, 1981] that first developed NN-specific backprop[2]. Interestingly, mirroring the context surrounding the first discussions of the credit assignment problem, backprop was also brought to NNs with the goal of training sequential decision making systems [Werbos, 2006]. Rumelhart et al. [1986] popularized backpropagation in the context of learning pattern recognizers by demonstrated that useful internal representations emerged in NNs with hidden layers trained with the help of backprop. It was soon generalized for the case of RNNs [Robinson and Fallside, 1987; Werbos, 1988; Williams and Zipser, 1989]. Thus, backprop can be used to address both the structural and temporal credit assignment problems.

The procedure of applying backprop to compute derivatives of a network's parameters is called a *backward pass*, since it involves going through the layers in the reverse order of the forward pass. During the backward pass, the derivatives with respect to each of the layer's activations and parameters are computed with the help of activation values stored during the forward pass, and intermediate derivatives computed so far. This is the key feature of backprop – the complexity of the backward pass is the same as that of the forward pass.

The backward pass for the MLP described by Equation 2.5 proceeds as follows. First we define a useful quantity $\boldsymbol{\delta}_\ell$ for the layer ℓ, called the *deltas* for the layer:

$$\boldsymbol{\delta}_\ell := \frac{\partial \hat{\mathbf{y}}_\mathrm{L}}{\partial \mathbf{z}_\ell} \tag{2.10}$$

The objective is to compute $\{\frac{\partial \hat{\mathbf{y}}_\mathrm{L}}{\partial \mathbf{W}_\ell}, \frac{\partial \hat{\mathbf{y}}_\mathrm{L}}{\partial \mathbf{b}_\ell}\}, \ell \in \{1, 2, \ldots, \mathrm{L}\}$. This can be done by

[2]Schmidhuber [2015] and Griewank [2014] provide further details of backprop's developments.

starting at layer L and then using the following equations at each previous layer:

$$\boldsymbol{\delta}_L = f'(\mathbf{z}_\mathrm{L}), \tag{2.11a}$$

$$\boldsymbol{\delta}_\ell = \mathbf{W}_{\ell+1}^\top \boldsymbol{\delta}_{\ell+1} \cdot f'(\mathbf{z}_\ell), \tag{2.11b}$$

$$\frac{\partial \hat{\mathbf{y}}_\mathrm{L}}{\partial \mathbf{W}_\ell} = \boldsymbol{\delta}_\ell \mathbf{x}_\ell^\top, \tag{2.11c}$$

$$\frac{\partial \hat{\mathbf{y}}_\mathrm{L}}{\partial \mathbf{b}_\ell} = \boldsymbol{\delta}_\ell. \tag{2.11d}$$

These equations can be used to compute the gradient of any loss function L of the network's output with respect to its parameters using the chain rule:

$$\frac{\partial L}{\partial \mathbf{W}_\ell} = \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathrm{L}} \frac{\partial \hat{\mathbf{y}}_\mathrm{L}}{\partial \mathbf{W}_\ell} \tag{2.12a}$$

$$\frac{\partial L}{\partial \mathbf{b}_\ell} = \frac{\partial L}{\partial \hat{\mathbf{y}}_\mathrm{L}} \frac{\partial \hat{\mathbf{y}}_\mathrm{L}}{\partial \mathbf{b}_\ell} \tag{2.12b}$$

$$\tag{2.12c}$$

As long as the first term in the above products can be computed (simple if L is differentiable) or estimated, the second term can be computed using backprop.

When backprop is used for computing gradients for RNNs, it is termed Back-propagation Through Time (BPTT) [Werbos, 1990]. The main idea of the procedure is to *unroll* the RNN: converting an RNN processing a sequence of T time steps into an FNN with T layers, where the same set of parameters are shared by all the layers. This is intuitively based on the similarity between Equation 2.6 for a simple RNN and Equation 2.5 for the MLP. The RNN equations can be transformed (unrolled) into the MLP equations by replacing time indices with layer indices, and adding a time-varying input at each layer in additional to the output of the previous layer. After unrolling, backprop as usual can be applied for efficient computation of gradients with respect to the RNN parameters with the same time complexity as the forward pass.

2.4.4 Gradient Descent

The simplest algorithm for optimizing an objective function when the gradient with respect to its parameters can be computed is *gradient descent*. It is an iterative algorithm, which at the n^{th} iteration uses the gradient to apply an additive parameter update

$$\Delta\theta(n) = -\alpha \frac{\partial L}{\partial \theta(n)} \tag{2.13}$$

$$\Delta\theta(n+1) = \theta(n) + \Delta\theta(n) \tag{2.14}$$

The learning rate α is a *hyperparameter*. Similar to other hyperparameters such as the topology of the network, the regularization weight etc., its value is typically selected from a set of values based on validation set performance.

It is well known that always changing parameters in the exact direction of steepest descent can be harmful, since the learning rate should be adapted differently for different parameters [Sutton, 1986]. It is common to use modified versions of gradient descent due to this reason. The most common variant incorporates *momentum* [Polyak, 1964; Nesterov, 1983] which changes the parameter updates to

$$\Delta\theta(n) = \eta\Delta\theta(n-1) - \alpha \frac{\partial L}{\partial \theta(n)} \tag{2.15}$$

where η is a hyperparameter. Momentum makes it easier for the optimization algorithm to avoid getting stuck due to local minima and other tricky properties of the objective function [Sutton, 1986; Rumelhart et al., 1986; Plaut et al., 1987; Jacobs, 1988].

A final modification to gradient descent which makes it practical for training on large training sets is the use of small subsets of the training set (called *batches* or *minibatches*) to compute approximate gradients, instead of the full gradient computed for the entire training set. Weights are updated after forward and backward passes for each batch, instead of waiting for the entire training set to be processed which might require prohibitive amounts of time and memory. When each batch consists of only one training pair, the resulting algorithm is called *stochastic gradient descent*. Although many other variations of gradient descent exist (see subsection 3.1.2 for some alternatives) batch gradient descent with momentum is used for all the experiments in this thesis. It is simple, easy to implement, works well in practice, and the choice ensures that improvements obtained in training with the proposed architectures are not due to use of a special optimization method.

Having discussed the central ideas for setting up and training NNs for supervised learning, in the next chapter we discuss the issues arising in following these concepts and trying to apply NN-based learning methods to problems in practice. This discussion will set up the stage for novel NN architectures introduced Chapter 4 onwards that are designed to address these issues.

Chapter 3

Challenges & Related Work

Based on the tools discussed in the previous chapter, the recipe for attacking a given problem using NNs is as follows: First, design an NN that is expected to have sufficient representational power for the task. Then choose an appropriate objective function based on a learning strategy (such as MLE) and techniques to prevent overfitting. Finally, select and run an optimization algorithm to train the network.[1]

Unfortunately, for a large number of problems the above procedure does not result in a network that performs acceptably well on the test set. Assuming that the amount of data available is sufficient and the objective function is appropriate for the task, the reasons for failure can be broadly categorized into the following issues:

a) *Trainability gap*: The optimization algorithm fails to sufficiently minimize the objective function on the training set (in terms of desired performance or available time budget), even though the NN is in principle capable of learning the task very accurately (a type of underfitting).

b) *Generalization gap*: The network obtained as a result of training performs much worse on the test set (unseen data) than the training set (overfitting).

This chapter provides a discussion of these issues, and reviews important techniques from the literature that have been proposed to address them. Since the methods developed in this thesis are primarily motivated by the trainability gap, this issue is discussed in more detail. In particular, the problems arising from making NNs very deep are discussed, since as discussed earlier, depth is

[1]Due to the involvement of several hyperparameters in these steps, it is usually necessary to run several trials and choose the best performing hyperparameter values.

an important characteristic that makes them capable of modeling extremely complicated functions.

3.1 The Trainability Gap

In general training NNs is computationally intractable, i.e., there exist no efficient algorithms for training an arbitrary network on a arbitrary problem [Blum and Rivest, 1989; Judd, 1990; Hammer and Villmann, 2003]. There has been much research into understanding under which constraints the training of certain classes of NNs may be tractable [Anthony and Bartlett, 2009], but this quest is further complicated by the fact that a theoretical understanding and categorization of many practical problems of interest is missing. In practice, since there exists an efficient algorithm for computing gradients for arbitrary differentiable networks (backpropagation), most popular optimization algorithms for NNs are gradient-based, such as variants of steepest descent.

The ability to compute gradients efficiently is not a panacea. The objective function landscape for NNs is non-convex, with multiple local minima, saddle points and other characteristics that make optimization using gradient based methods tricky [Bishop, 1995]. These difficulties have motivated the development of a variety of techniques to improve and speed up the training of NNs which are discussed in this section. In the following chapters, new architectures are developed to specifically address two key issues that hamper credit assignment in NNs: *cross-pattern interference* (subsection 3.1.1) and *vanishing gradients* (subsection 3.1.5).

3.1.1 Cross-pattern Interference & Competitive Learning

Cross-pattern interference [Sutton, 1986] arises naturally in gradient-based training of NNs when several units in the network simultaneously try to learn useful computations for a varied set of input patterns. This can make learning inefficient, since the weights of the units oscillate due to conflicting information coming from different patterns presented to the network over time, and these oscillations increase the total time required for the weights to converge.

A consequence of cross-pattern interference is catastrophic forgetting [McCloskey and Cohen, 1989; Carpenter and Grossberg, 1988], which is a fundamental obstacle for the use of NN in continual learning systems [Ring, 1994]. In this setting, which is much more general than the supervised learning setting considered here, a machine's goal is to continually improve its behavior over a single lifetime through interactions with its environement. Since the distribution of input patterns that the machine observes can change significantly over time, useful computations learned by an NN at earlier times get completely disrupted by new information, meaning that the machine can not profit from past experiences.

Competitive Learning

Competitive interactions between units and neural circuits have long played an important role in biological models of brain processes. The effects of excitatory and inhibitory feedback found in many regions in the brain [Eccles et al., 1967; Anderson et al., 1969; Stefanis, 1969] were modeled with competitive interactions in several early models inspired by the brain's information processing mechanisms [von der Malsburg, 1973; Grossberg, 1976; Fukushima, 1980a; Kohonen, 1982]. These demonstrations of successfully combining competitive interactions with simple Hebbian-like [Hebb, 1949] learning rules inspired further work by Rumelhart and Zipser [1985] for feature learning, and by Schmidhuber [1989] who used them in RNNs. However, these and several following developments of competitive learning since the 1990s [Ahalt et al., 1990; Goodhill and Barrow, 1994; Terman and Wang, 1995; Wang, 1997] did not use gradient-based training.

The work of Jacobs et al. [1991a,b] was the first to leverage the properties of competitive interactions for the purpose of mitigating cross-pattern interference. They proposed to generalize competitive learning from the level of computational units to networks by employing a modular NN architecture. The architecture consisted of two types of networks, several *expert* networks and a *gating* network. The output of the gating network was used to combine the outputs of the experts to produce the final network output. This explicit modularity of the architecture enabled it to avoid cross-pattern interference by allocating different input patterns to different experts. Related architectures were also developed by Pollack [1987] and Hampshire and Waibel [1992].

3.1.2 Step Size Adaptation

A common difficulty arising in NN optimization is that it may not be the best to always follow the exact steepest descent direction. This is due to the presence of structures like *valleys* and *ravines* in the landscape [Sutton, 1986; Jacobs, 1988; Qian, 1999] which require the use of different step sizes along different dimensions. This can be achieved with momentum – a widely employed technique to accelerate gradient descent [Polyak, 1964; Nesterov, 1983] that was brought to NNs in the 1980s [Rumelhart et al., 1986; Plaut et al., 1987; Jacobs, 1988].

Another principled way to address the same issue is to use the inverse of the Hessian of the objective function to set the step sizes (i.e., Newton's method), and incorporating information about the curvature of the function in addition to the gradient. Since this operation is computationally too demanding for all but the smallest networks, many researchers have proposed methods to compute and use approximations of the Hessian [Sutton, 1986; Becker and LeCun, 1988; Roux et al., 2008; Martens, 2010, 2014; Grosse and Salakhudinov, 2015].

Instead of computing the Hessian of the objective function for use with second order methods, another approach to speed up NN training is to still employ first order gradient descent algorithms but use certain "tricks" to encourage favorable properties of the Hessian. These tricks are choices regarding the network design, initialization and data preprocessing. For convex optimization, the convergence rate of gradient descent methods near the optimum is heavily influenced by the Hessian, in particular by its *condition number* – the ratio of its largest and smallest eigenvalues – which should be close to unity for fast convergence. Some tricks for NN training that minimize the spread of eigenvalues are centering, normalizing and decorrelating the inputs [LeCun et al., 1998], using a scaled $tanh$ activation function instead of the logistic sigmoid [LeCun et al., 1991; Kalman and Kwasny, 1992], and careful initialization of the weights [LeCun et al., 1998; Glorot and Bengio, 2010; He et al., 2015].

A related set of techniques attempt to make the non-diagonal terms of the Hessian as close to zero as possible [Schraudolph, 1998a; Raiko et al., 2012; Desjardins et al., 2015; Ioffe and Szegedy, 2015; Salimans and Kingma, 2016]. Doing so makes first order methods perform better since only the diagonal terms (one term corresponding to each dimension) need to be approximated. This can be accomplished with the help of variants of gradient descent which automatically adapt the step sizes per dimension. Popular choices include AdaGrad [Duchi et al., 2011], AdaDelta [Zeiler, 2012], RMSProp [Tieleman and Hinton, 2012] and Adam [Kingma and Ba, 2014].

3.1.3 Initialization Strategies

The initial values of a network's weights have a direct impact on the success of optimization. One reason for this is that many activation functions are designed to *saturate* for large magnitude inputs, so that the unit outputs are always bounded. Saturation refers to the fact that the output of the activation function changes negligibly as the input changes i.e. its gradient becomes almost zero. For example, for the $tanh$ and hyperbolic sigmoid functions, large values of both positive and negative net input leads to saturation, which will impede gradient flow during the backward pass (subsection 2.4.3). In general, having too large or too small weights can severely attenuate signals (activations during forward pass, gradients during backward pass) as they propagate through a network. In conjunction with input data normalization [LeCun et al., 1991], normalized weight initialization techniques [LeCun et al., 1998; Glorot and Bengio, 2010; He et al., 2015] mentioned above are designed to make sure that the variance of the signals remains constant and close to one over the layers of the network. Of course, initialization can only guarantee favorable properties at the beginning of training, but practitioners have found that this is sufficient to enable successful training of small to medium-sized networks.

3.1.4 Activation Functions

In recent years, non-saturating activation functions have become popular, starting with the Rectified Linear (ReL) [Jarrett et al., 2009; Nair and Hinton, 2010; Glorot et al., 2011] function defined as $y = \max(0, x)$. Clearly this function does not saturate for positive inputs, and is therefore less likely to contribute to diminishing gradients during backpropagation. Moreover, while a complete theoretical understanding of this phenomenon is currently lacking, it has been found that if the network weights are initialized according to normalization techniques mentioned above, the saturation behavior of ReL for negative inputs does not impede training if the network depth is not large. Following the popularity of ReL in applications, its variants that do not saturate for negative inputs either have also been tried [Maas et al., 2013; He et al., 2015] with mixed success.

In addition to preventing saturation, the activation function also affects optimization through its effect on the properties of the Hessian. The ReL function causes a *bias shift* at each layer because its output is always non-negative. Exponential Linear Units [Clevert et al., 2015] use a modification of ReL that addresses this issue and pushes the mean activations at each layer closer to zero, which improves the conditioning of the Hessian. Recent work has further built upon this to develop *self-normalizing* networks [Klambauer et al., 2017] which preserve the mean and variance of signals during forward and backward propagation without requiring explicit normalization techniques.

3.1.5 The Challenges of Depth

The above issues contributing to the trainability gap are common to all NNs, but there is a specific challenge that fundamentally affects the trainability of powerful and efficient networks – gradient based training becomes increasingly infeasible as the depth of the credit assignment problem increases. It was first analyzed in the context of temporal credit assignment in RNNs [Hochreiter, 1991], and here we briefly review this analysis breifly.

Consider the simple RNN described in Equation 2.6. Ignoring the bias and inputs for clarity, the output of the RNN at time t is $\hat{\mathbf{y}}^{[t]} = f(\mathbf{R}\hat{\mathbf{y}}^{[t-1]})$. Then the temporal Jacobian $\mathbf{A}_t := \frac{\partial \hat{\mathbf{y}}^{[t]}}{\partial \hat{\mathbf{y}}^{[t-1]}}$ is

$$\mathbf{A}_t = \mathrm{diag}\big[f'(\mathbf{R}\hat{\mathbf{y}}^{[t-1]})\big]\mathbf{R}. \tag{3.1}$$

Let γ be a maximal bound on f' and $\sigma_{\max}$ be the largest singular value of $\mathbf{R}$. Then the norm of the temporal Jacobian satisfies

$$||\mathbf{A}_t|| \leq \Big\|\mathrm{diag}\big[f'(\mathbf{R}\hat{\mathbf{y}}^{[t-1]})\big]\Big\| \; ||\mathbf{R}|| \leq \gamma\sigma_{\max} \tag{3.2}$$

As discussed in subsection 2.4.3, to compute the *delayed* temporal Jacobian of the output $\hat{\mathbf{y}}^{[t_2]}$ with respect to $\hat{\mathbf{y}}^{[t_1]}$ where $t_2 > t_1$, we can use the chain rule and obtain

$$\frac{\partial \hat{\mathbf{y}}^{[t_2]}}{\partial \hat{\mathbf{y}}^{[t_1]}} := \prod_{t_1 < t \leq t_2} \mathbf{A}_t \tag{3.3}$$

Based on the above, the norm of the delayed temporal Jacobian is upper-bounded by $(\gamma\sigma_{\max})^{t_2 - t_1}$ implying that if the product $\gamma\sigma_{\max}$ is less than one, the delayed Jacobian's norm becomes exponentially smaller as the time delay increases. Similarly, it can be shown that if the spectral radius ρ of $\mathbf{A}_t$ is large enough, the norm of the delayed Jacobian can grow exponentially. Note that $\gamma = 1$ for $tanh$ and $\gamma = 0.25$ for the logistic sigmoid, so the necessary condition for the gradients to explode for these cases is $\rho > 1$ and $\rho > 4$.

The phenomena of the temporal gradients becoming exponentially small or large as the time delay increases are referred to as *vanishing* and *exploding* gradients respectively [Hochreiter, 1991; Bengio et al., 1994; Hochreiter et al., 2001; Pascanu et al., 2013b]. They are fundamental hindrances for successful training of RNNs as they imply that an RNN's outputs at future time steps quickly become either insensitive or too sensitive to the outputs at past time steps.

The same analysis applies to FNNs with slight modifications. As discussed in subsection 2.4.3, an RNN when "unrolled in time" for T time steps is mathematically equivalent to an FNN with T layers. Due to this equivalence, vanishing and exploding gradients also hamper learning in very deep FNNs, where the adjective "very deep" refers FNNs with a depth of approximately 20 or more. The two main differences are: a) the RNN has inputs at each time step while the FNN has inputs only at the first layer, and b) the RNN uses the same recurrent weights $\mathbf{R}$ at each time step to transform the outputs of the last step, while the FNN uses weights $\mathbf{W}_\ell$ for the ℓ^{th} layer. Experimentally it has been observed that traditional FNNs with depth up to 20 layers can be optimized successfully if certain tricks are employed (see e.g. Simonyan and Zisserman [2014], and experiments in subsection 6.3.2), but training traditional NNs beyond this depth has remained impractical.

There is a common misconception that vanishing gradients are only (or mostly) caused by the saturation of the activation functions. The above analysis clearly shows that this is incorrect – even for a fully linear network (i.e. $f(x) = x; f'(x) = 1 \; \forall x \in \mathbb{R}$) vanishing or exploding gradients will result based on the value of σ_{max} (Equations 3.2 & 3.3). However, in principle the interaction between the weight magnitudes and properties of the activation function can gaurd against vanishing gradients, as shown recently by Klambauer et al. [2017].

3.1.6 Techniques for Training Very Deep Networks

The techniques discussed in subsection 3.1.2 and subsection 3.1.3 are useful, and often essential, for training FNNs with depth up to 20 layers and RNNs that need to model temporal dependencies of similar lengths. In this section we discuss two lines of research specifically targeted at developing NN architectures that are more resilient to the pathologies resulting from large depth.

LSTM Vanishing and exploding gradients in NNs are fundamentally a consequence of their design, specifically the functional form of the transformation that maps inputs to outputs. To specifically combat vanishing gradients, Hochreiter and Schmidhuber [1997b] proposed a radical RNN design – Long Short-Term Memory (LSTM) that can assign credit across hundreds of time steps. The key design elements were the use of

a) a *memory cell* within each unit which has a recurrent self-connection with a constant weight equal to one.

b) additional units with weighted recurrent connections that control (or *gate*) the flow of information into and out of the cell.

Since the recurrent connections of the memory cell always have a weight of one, it can maintain constant error flow without attenuation during backpropagation through time. In fact, this mechanism is so successful at maintaining memory of past events that Gers et al. [2000] later added a *forget gate* unit whose output is multiplied with the cell's recurrent connection, enabled the memory to be reset by producing forget gate outputs close to zero, which is necessary for many sequential tasks. LSTMs with forget gates are now the most commonly used RNN architecture for problem involving learning from sequential data. For a detailed description of the various LSTM variants and their comparison, see the survey by Greff et al. [2017a].[2]

Skip connections Skip connections modify MLPs by adding direct weighted connections from *lower* layer (with lower layer indices ℓ) to *higher* layers. This is an intuitive way to improve gradient flow since such connections introduce shorter credit assignment paths in the network along which gradients can flow and support learning in the lower layers.

Skip connections for NNs have a long and winding history of development. It was clearly well known in the 1980s that the class of NNs that could be trained with backpropagation included those with skip connections [Rumelhart et al., 1986] but standard MLPs did not include them. A prominent exception is the work of Lang and Witbrock [1988] who used weighted connections from each layer to all higher layers in their experiments. They called the extra connections *short-cut* connections. This early work was also pioneering in that the motivation to use skip connections was to successfully train deeper networks for a difficult problem, although the vanishing gradient problem had not been formally identified yet by [Hochreiter, 1991].

The Cascade-Correlation architecture [Fahlman and Lebiere, 1990] was a way of incrementally adding computation units to an NN during supervised training. Since each added unit received connections from all existing units, this architecture naturally resulted in very deep networks with skip connections. However, weights of existing units were typically frozen, so the purpose of skip connections was not to improve credit assignment.

Lee and Holt [1992] connected inputs directly to the output of the network in their Direct Linear Feedthrough networks. Their motivation was to separate the learning of linear and non-linear contributions to speed-up learning and improve understanding of the non-linearity in the data. Kalman et al. [1993]; Kalman and Kwasny [1994] used similar connections and motivations in their TRAINREC system, referring to them as both "skip" and "shortcut" connections. Schraudolph [1998b,a] analyzed learning in such networks and found that the skip connections can in fact slow down learning. He then proposed a centering method for backpropagated gradients to ensure benefits from skip connections.

[2]This survery was co-authored by the author of this thesis.

Van Der Smagt and Hirzinger [1998] also advocated for skip connections to speed up training, but proposed to use the same weights for linear skip connections as the non-linear weights. The use of skip connections to speed up training by separately learning the linear and non-linear parts of the function was revisited by Raiko et al. [2012]. Instead of centering the gradient explicitly like Schraudolph [1998b], they proposed to transform the non-linear functions used by the units.

The skip connections in the work mentioned above are mathematically the same as all other connections in the networks. The only difference lies in their inputs and outputs, as they change the connectivity of the network such that it is no longer linear graph i.e. an MLP. Novel architectures developed in this thesis (Chapter 6 onwards) also use skip connections, but they are of a different nature – more similar to those used by memory cells in the LSTM architecture.

Layer-wise training In contrast to the Cascade-Correlation Architecture, incremental layer-wise training *without* skip connections to improve credit assignment was developed by Schmidhuber [1992] for RNNs and by Hinton et al. [2006]; Hinton and Salakhutdinov [2006] for FNNs. Note that in contrast to the Cascade-Correlation architecture, these techniques were developed in the context pf *unsupervised* training of NNs to learn the regularities in a dataset for the purposes of compression or dimensionality reductions.

3.2 The Generalization Gap

The generalization gap exists for most model classes in machine learning, and NNs are no exception. Due to their high capacity for representing complicated functions, they are prone to overfitting [Vapnik, 2013; Bishop, 2006] to the training set essentially by memorizing the training data. Most well-defined measures of model complexity developed in statistical learning theory such as VC dimension [Vapnik, 2013] and Rademacher complexity [Bartlett and Mendelson, 2002] can not be directly applied to NNs, but there have been partially successful attempts to define such measures for specific network types [Bartlett, 1998; Neyshabur et al., 2015]. In practice, regularization techniques such as those mentioned in subsection 2.4.2 are almost always used during network training to aid in closing the generalization gap.

The techniques that aid in training NNs also have additional properties that affect generalization. This is because the geometry of the objective function around the local minimum reached by the optimization algorithm is related to the generalization ability of the network [Hochreiter and Schmidhuber, 1997a].

If an optimization algorithm naturally prefers *flat* minima, it is likely to produce solutions that generalize better. An example is the technique of Batch normalization [Ioffe and Szegedy, 2015], which was designed to normalize the outputs of each layer of a network to accelerate training, but also results in extremely effective regularization due to the stochasticity in the normalization procedure performed using batches sampled for training.

An important consideration in the context of generalization abilities of NNs is the role of architecture. For example, it is well known that CNNs are much more suitable for image analysis problems compared to simple fully-connected MLPs. Although MLPs are perfectly capable of matching the training set performance of CNNs, they do not generalize well to test data. The reason for this is that the *inductive bias* [Mitchell, 1980] of the CNN is more suitable for spatial signals such as images, i.e., the CNN is implicitly encouraged by design to implement the “correct” function for processing images. Of course, the implicitly “correct” functions are chosen by a scientist based on the properties of the domain, and these choices may not be theoretically optimal. Other architectural choices that together build up an NN such as connectivity, activation function, etc., are also similarly connected to generalization performance. They act as *priors* over the nature of functions/programs that are being modeled, and choosing the right prior implies that the underlying function can be learned using much less data. Thus, there is an interesting connection between the modeling capacity of deeper networks (as discussed in section 2.3 and their generalization that is relevant to this thesis. It has been noted by Bengio et al. [2013] that increasing depth is not only a way to increase the modeling capacity, but can also lead to improved generalization, since deeper NNs can model certain function classes significantly more efficiently than shallower ones.

It is notable that this interplay between architectural choices, speed of learning and generalization reflects the relationship between the two general approaches of programming computers discussed in Chapter 1. Expert knowledge may not be sufficient to write down the complete algorithmic solution to a problem, but it can still speed up (or slow down!) search in the space of solutions. Hence it is useful to identify and analyze a variety of feasible choices for the various building blocks of NNs, so that they can be employed when suitable for a given problem.

Chapter 4

Local Winner-Take-All Networks

Consider a layer of units in an MLP being trained to classify input patterns as being of one of the ten Arabic digits (0–9). The units are expected to learn to identify certain features in the inputs which are indicative of the digit's identity, such as edges oriented at various angles. Assume that the dataset is very simple, such that identifying horizontal and vertical edges would be sufficient to classify the data correctly. When the network is not completely trained, it is likely that many units respond to edges but do not have very high responses to these orientations. Instead, there are several units that respond to edges with intermediate orientations. Backpropagation will then assign some credit to all of these units, and they will all be slowly adjusted towards one of the desired directions at each learning step until the objective function is minimized.

The above problem of cross-pattern interference, as discussed in subsection 3.1.1, can potentially lead to inefficiencies during learning. The credit assignment is *diffused* among several units since any number of them may be learning the same feature. For an input pattern that only requires high response to horizontal edges, it would be helpful to assign the credit to a single unit that responds to horizontal edges. This would leave other units free to learn useful features required for other input patterns when they are presented next, such as vertical edges.

Competitive learning is a mechanism that was popular for supporting credit assignment in NNs before backprop became widely popular and offers a natural way to avoid cross-pattern interference, as discussed in subsection 3.1.1. In this chapter, we propose an architecture for gradient-based artificial neural networks that takes inspiration from early competitive learning approaches, and ultimately relies on Local Winner-Take-All (LWTA) behavior. In supervised learning experiments across a number of different networks and pattern recognition

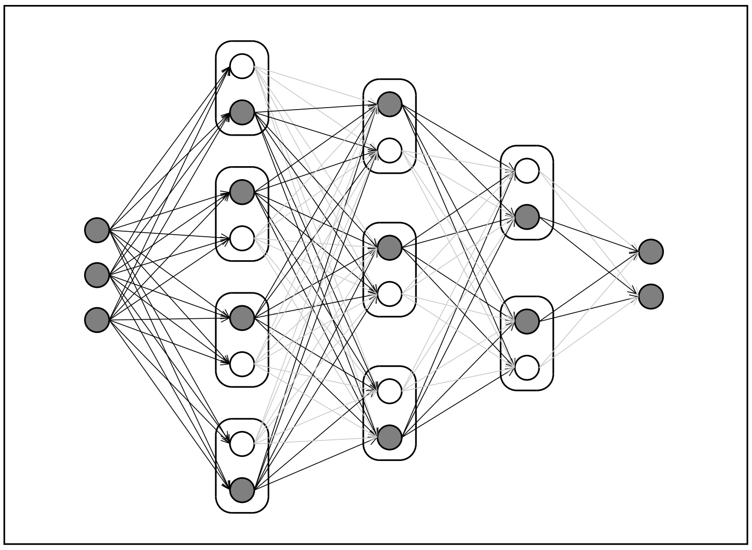


Figure 4.1. A fully-connection Local Winner-Take-All (LWTA) network with blocks of size two showing the winning unit in each block (shaded) for a given input pattern. Activations flow forward only through the winning units. Similarly, errors are backpropagated only through the winning units. Grey-ed out connections do not propagate activations. The active units constitute a subnetwork which changes depending on the input pattern.

tasks, LWTA speeds up learning and enables test performance that matches the state-of-the-art. Experiments show evidence that a type of modularity emerges in LWTA networks trained in a supervised setting, such that different modules (subnetworks) respond to different input patterns. LWTA also helps to prevent *catastrophic forgetting* [McCloskey and Cohen, 1989; Carpenter and Grossberg, 1988] in NNs when they are first trained on a particular task, then abruptly trained on a new task. This property is desirable in continual learning wherein learning regimes are not clearly delineated [Ring, 1994].

4.1 Networks with Local Winner-Take-All Blocks

This section describes a general network architecture with locally competing units. The network consists of a number of *blocks* organized into layers (Figure 4.1). Each block consists of n computational units, and produces an output vector $\mathbf{y}$, determined by the local interactions between the individual neuron activations in the block:

$$\mathbf{y} = g(\mathbf{h}), \tag{4.1}$$

where $g(\cdot)$ is a *competition* or *interaction function*, encoding the effect of local interactions in each block, and $\mathbf{h}$, is the vector of activations of the units in the block computed in the standard way by:

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{4.2}$$

where $\mathbf{x}$ is the input vector from units in the previous layer, $\mathbf{W}$ is the matrix of weights connecting the units in the previous layer to the units of the block, and f is the (generally non-linear) activation function. The output activations $\mathbf{y}$ are passed as inputs to the next layer. We define the interaction function g to be the hard Winner-Take-All (WTA) function such that the output of the j^{th} unit in the block is set to:

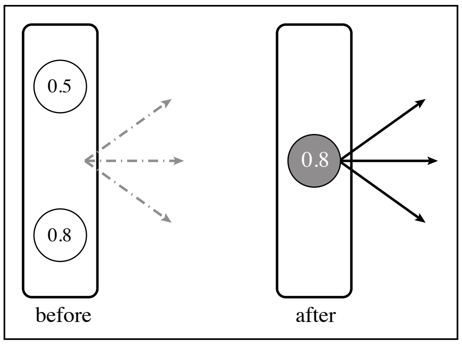
$$\mathbf{y}[j] = \begin{cases} \mathbf{h}[j] & \text{if } \mathbf{h}[j] \geq \mathbf{h}[k], k \in \{1, \ldots, \mathrm{n}\} \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

In the case of ties the winner is chosen by index precedence, so in an LWTA layer, there are as many units as there are blocks active at any one time for a given input pattern[1]. We denote a layer with blocks of size n as LWTA-n. To investigate the capabilities of the hard winner-take-all interaction function in isolation, the identity function is used as the activation function in Equation 4.2.

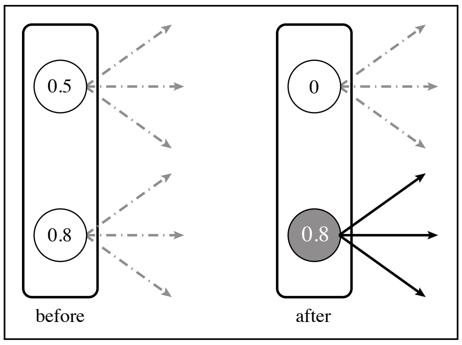
The difference between this Local Winner Take All (LWTA) network and a standard MLP is that no non-linear activation functions are used, and during the forward propagation of inputs, local competition between the units in each block "turns off" the activation of all units except the one with the highest activation. Therefore, LWTA can be interpreted as an activation function – but one that is not an unit-wise operation. For gradient computation, the error signal is only backpropagated through the winning units.

For each input pattern presented to an LWTA network, only a subgraph of the full network (a *subnetwork*) is active, e.g. the highlighted units and synapses in Figure 4.1. Training on a dataset consists of simultaneously training an exponential number of subnetworks that share parameters, as well as learning which subnetwork should be active for each pattern. In this way, input patterns consisting of very different sets of features can potentially be modeled more efficiently through specialization of units. This modular property is similar to that of networks with rectified linear (ReL units which were recently shown to be effective for several learning tasks [Krizhevsky et al., 2012; Maas et al., 2013]. Links between LWTA and ReL are discussed in subsection 4.2.3.

[1]There is always the possibility that the winning neuron in a block has an activation of exactly zero, so that the block has no output.



(a) max-pooling/maxout



(b) LWTA

Figure 4.2. Max-pooling/maxout vs. LWTA. (a) In max-pooling and maxout, each group of units in a layer has a single set of output weights that transmits the winning unit's activation (0.8 in this case) to the next layer, i.e. the layer outputs are subsampled. (b) In an LWTA block, the interaction function does not lead to subsampling. Output activations of the winning units flow into units in the subsequent layer via a different set of connections depending on the winning unit.

4.2 Comparison with Related Methods

Certain techniques already used with NNs have similarities with how LWTA is implemented and how it affects the activations in a network. This section compares and contrasts LWTA with these techniques to clarify its nature.

4.2.1 Max-pooling & Maxout

Neural networks with max-pooling layers [Zhou and Chellappa, 1988; Weng et al., 1992] have been found to be very effective for image classification tasks where they achieved state-of-the-art performance [Krizhevsky et al., 2012; Ciresan et al., 2012]. These layers are used in CNNs to subsample the representation obtained after convolving the input with learned weights, by dividing the representation into pools and selecting the maximum in each one. Max-pooling lowers the computational burden by reducing the number of connections in subsequent convolutional layers, and adds small translational/rotational invariance [Goodfellow et al., 2016].

Maxout [Goodfellow et al., 2013a] is a recently proposed architecture[2] for FNNs that pools together the activity of subsets of units using the max operation.

[2]The original study on Maxout was published a few months before LWTA.

It was designed to take advantage of training with dropout regularization [Hinton et al., 2012b] and produced the best results on several benchmark datasets. The difference between the two is that maxout outputs the maximum activity of independent units, while max-pooling outputs the maximum over a spatial representation produced by a single unit.

At first glance these operators seem very similar to LWTA, but there is a key difference: there is no pooling in an LWTA block and thus the representation dimension remains unchanged, instead the representation is "sparsified" (see Figure 4.2).

4.2.2 Dropout

Dropout [Hinton et al., 2012b] can be interpreted as a model-averaging technique that jointly trains several models sharing subsets of parameters and input dimensions, or as data augmentation when applied to the input layer [Hinton et al., 2012b; Goodfellow et al., 2013a]. This is achieved by probabilistically omitting ("dropping") units from a network for each example during training, so that those units do not participate in forward/backward propagation. Consider, hypothetically, training an LWTA network with blocks of size two, and selecting the winner in each block at random. This appears similar to training a neural network with a dropout probability of 0.5 since such dropout also turns off half of the units in the layer during the forward pass for each training example. Nonetheless, the two are fundamentally different.

Dropout is a probabilistic regularization technique that affects each unit independently while in LWTA the deterministic interaction between units in a block replaces the per-neuron non-linear activation. Like any regularization technique, dropout is not used when evaluating a trained network's performance on the test set. When performing the forward pass through a network during testing, the output weights from all units in a layer trained with dropout are scaled down to compensate for the fact that removal of dropout makes more units active [Hinton et al., 2012b]. In an LWTA network, there is no difference in operation between training and test times, and no output scaling is required. In fact, Dropout can be used in LWTA networks to insert noise in the local competition between units and improve generalization performance.

4.2.3 Rectified Linear Units

Rectified Linear (ReL) units are simply linear units that clamp negative activations to zero ($f(x) = x$ if $x > 0, f(x) = 0$ otherwise). Networks with this activation

Table 4.1. Comparison of rectified linear activation and LWTA.

		ReLU units		LWTA units	
x_1	x_2	y_1	y_2	y_1	y_2
$x_1 > x_2$					
Positive	Positive	x_1	x_2	x_1	0
Positive	Negative	x_1	0	x_1	0
Negative	Negative	0	0	x_1	0
$x_2 > x_1$					
Positive	Positive	x_1	x_2	0	x_2
Negative	Positive	0	x_2	0	x_2
Negative	Negative	0	0	0	x_2

function were shown to be useful for Restricted Boltzmann Machines [Nair and Hinton, 2010], outperformed sigmoidal activation functions in deep neural networks [Glorot et al., 2011], and have been used to obtain the best results on several benchmark problems across multiple domains [Krizhevsky et al., 2012; Dahl et al., 2013; Maas et al., 2013].

Compare an LWTA block with two units to two ReL units, where x_1 and x_2 are the weighted sum of the inputs to each unit. Table 4.1 shows the outputs y_1 and y_2 for all combinations of positive and negative x_1 and x_2. For both ReL and LWTA units, x_1 and x_2 are passed through as output in half of the possible cases. The difference is that in LWTA both units are never active or inactive at the same time, and the activations and errors flow always through exactly one neuron in the block. For ReL units, being inactive (saturation) is a potential drawback since units that do not get activated for any input patterns will not get trained, leading to wasted capacity.

While many of the above arguments for and against ReLU networks apply to LWTA networks, there is an important notable difference. During training of an LWTA network, units that are inactive for all patterns can become active (and get trained further) due to training of the other units in the same block. This suggests that LWTA networks are less likely to suffer from reduced network capacity due to degenerate units that become completely inactive for all data.

4.3 Experiments

In the following experiments, LWTA networks were tested on various supervised learning datasets, demonstrating their ability to learn useful internal representa-

Table 4.2. Test set errors on the permutation invariant MNIST dataset for methods without data augmentation or unsupervised pre-training

Activation	Test Error
Sigmoid [Simard et al., 2003]	1.60%
ReLU [Glorot et al., 2011]	1.43%
ReLU + dropout in hidden layers [Hinton et al., 2012b]	1.30%
LWTA-2	**1.28%**

tions without utilizing any other non-linearities.

In order to clearly assess the utility of local competition, no other strategies such as augmenting training data with transformations, noise or dropout were used. We also did not encourage sparse representations in the hidden layers by adding activation penalties to the objective function, a technique used by Glorot et al. [2011] for ReL units. Thus, our objective was to evaluate the utility of using LWTA in NNs rather than achieving the absolute best testing scores. This is also why the comparisons below only include published results of NN-based approaches.

All networks used in this chapter contained 2 units per block, and were trained using minibatch gradient descent, learning rate α_t and momentum m_t at epoch t given by

$$\begin{aligned} \alpha_t &= \begin{cases} \alpha_0 \lambda^t & \text{if } \alpha_t > \alpha_{min} \\ \alpha_{min} & \text{otherwise} \end{cases} \\ m_t &= \begin{cases} \frac{t}{T} m_i + (1 - \frac{t}{T}) m_f & \text{if } t < T \\ p_f & \text{if } t \geq T \end{cases} \end{aligned}$$

where λ is a learning rate *annealing factor*, α_{min} is the lower learning rate limit, and momentum is scaled from m_i to m_f over T epochs after which it remains constant at m_f. L2 weight decay was used for the CNN experiments, and max-norm regularization[3] for other experiments. This setup is based on that of Hinton et al. [2012b]. All experiments were conducted using an Nvidia GTX 580 GPU.[4]

4.3.1 Permutation Invariant MNIST

We first evaluate LWTA networks in the *permutation invariant* setting on the MNIST dataset (PI-MNIST; LeCun et al., 1998) of images of handwritten digits. In this setting, the task is to classify the digits without utilizing the 2D structure of the images, i.e. every input pattern is a vector of pixels.

The last 10,000 examples in the training set were used for tuning the hyperparameters (learning rate, momentum, regularization and network size) using a small grid search. The network with the best hyperparameter setting was trained until convergence on the full training set. Minibatches of size 20 were used during training. The best model obtained, which gave a test set error of **1.28%**, consisted of three LWTA-2 layers of 500 blocks followed by a 10-way softmax layer. Table 4.2 compares our results with other comparable networks. The performance of LWTA is comparable to that of a network whose units all use the ReL activation function with dropout in the hidden layers, which in turn performs better than sigmoidal networks. This is an encouraging result since it indicates that local competition can be a viable alternative to conventional activation functions used in NNs. We note again that performance on this task can be easily improved, for example by using dropout in input layers as well [Hinton et al., 2012b; Goodfellow et al., 2013a].

4.3.2 CNNs on MNIST

In this experiment on the MNIST dataset, the compatibility of LWTA with CNNs which are designed to exploit the spatial structure of images was tested. The architecture used consisted of convolution units with a receptive field of 7×7 pixels in the first layer and 6×6 pixels in second layer, with 16 and 32 maps respectively. These layers were followed by two fully-connected layers each with 64 LWTA blocks and finally a 10-way softmax output layer. Every convolutional layer was followed by a 2×2 max-pooling operation.

We tested two variants: one in which LWTA-2 was used was used only in the two fully-connected layers (the convolutional layers used the ReL activation function), and one in which all hidden layers used LWTA-2. L2 weight decay with coefficient 0.05 was found to be beneficial to improve generalization. The results are summarized in Table 4.3 along with other NN-based approaches that do not

[3]This technique sets a maximum limit on the norm of the incoming weights for all units. The weights are rescaled whenever this limit is exceeded [Srebro et al., 2005].

[4] To speed up experiments, the Gnumpy [Tieleman, 2010] and CUDAMat [Mnih, 2009] libraries were used.

Table 4.3. Test set errors on MNIST dataset for convolutional NNs with no data augmentation. Results marked with an asterisk use layer-wise unsupervised feature learning to pre-train the network and global fine tuning.

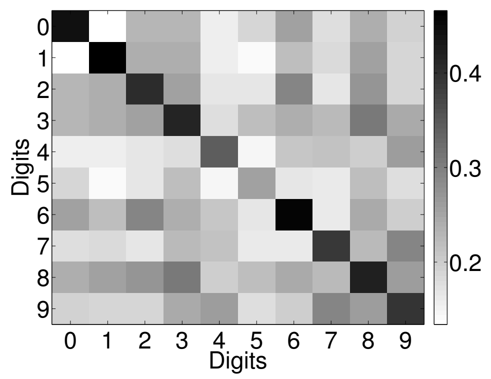
Architecture	Test Error
2-layer CNN + 2 layer MLP [Ranzato et al., 2007] *	0.60%
2-layer ReLU CNN + 2-layer LWTA	**0.57%**
3-layer ReLU CNN [Zeiler et al., 2013]	0.55%
2-layer CNN + 2 layer MLP [Jarrett et al., 2009] *	0.53%
2-layer LWTA CNN + 2-layer LWTA MLP	**0.53%**
3-layer ReLU CNN + stochastic pooling [Zeiler and Fergus, 2013]	0.47%
3-layer maxout + dropout [Goodfellow et al., 2013a]	0.45%

use data augmentation. Networks with LWTA are again highly competitive with other high-performing architectures. They are outperformed with the help of special regularization techniques (stochastic pooling in Zeiler and Fergus [2013] and dropout in Goodfellow et al. [2013a]). These results indicate that LWTA is also a promising activation for CNNs which are widely used in a variety of domains.

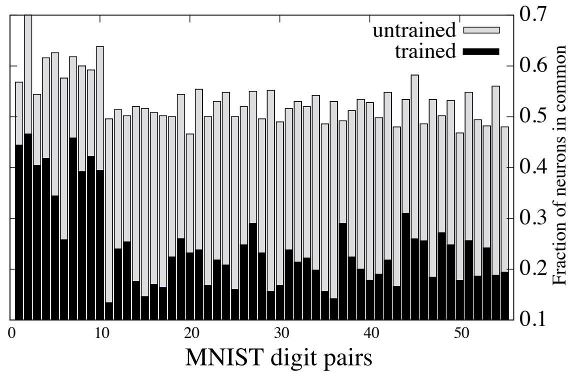
4.3.3 Amazon Sentiment Analysis

Next, LWTA networks were tested on the Amazon sentiment analysis dataset [Blitzer et al., 2007], since the ReL activation was shown to perform well in this domain [Glorot et al., 2011]. We used the balanced subset of the dataset consisting of 1000 positive and 1000 negative reviews of four categories of products: *Books, DVDs, Electronics* and *Kitchen appliances*. The task is to classify the text reviews as positive or negative. The text of each review was converted into a binary feature vector encoding the presence or absence of unigrams and bigrams. Following Glorot et al. [2011], the 5000 most frequent vocabulary entries were retained as features for classification. The data was then divided into 10 equally balanced folds, and networks were tested with cross-validation, yielding the mean test error over all folds.

Glorot et al. [2011] reported an overall accuracy of 78.95% for the ReL activation function on this dataset in the context of unsupervised learning with auto-encoding NNs to obtain sparse feature representations which were then used for classification. An LWTA-2 network with three 500-blocks layers was trained in a fully supervised setting to directly classify each review as positive



(a)



(b)

Figure 4.3. Analysis of subnetworks in trained LWTA networks. (a) Each entry in the matrix denotes the fraction of units that a pair of MNIST digits has in common, on average, in the subnetworks that are most active for each of the two digit classes. (b) The fraction of units in common in the subnetworks of each of the 55 possible digit pairs, before and after training.

or negative using a 2-way softmax output layer. This resulted in mean test set accuracies of *Books*: 80%, *DVDs*: 81.05%, *Electronics*: 84.45% and *Kitchen*: 85.8%, for an overall accuracy of **82.82%**. This result demonstrates the utility of local competition on a problem in a domain from previous experiments, where the input data consists of preprocessed binary vectors instead of continuous pixel values for images.

Together, these results provide strong evidence that LWTA networks can be used to match or outperform networks with the ReL activation function, which at the time of writing is widely used in applications. This finding warrants further investigation into their properties, which we continue in the following sections. Note that soon after the publication of the above results, Wang and JaJa [2014] used LWTA to obtain further state of the art results on various benchmark problems in object recognition.

4.4 Analysis of Subnetworks

A network with a single LWTA-m layer of N blocks consists of m^N subnetworks which can be selected and trained for individual input patterns while training over a dataset. After training, we expect that the subnetworks consisting of active units for patterns from the same class to have more units in common compared to subnetworks being activated for different classes. In the case of relatively simple

Table 4.4. Local competition guards against catastrophic forgetting: LWTA networks outperform logistic sigmoid and ReL activation functions in remembering dataset P1 after training on dataset P2.

Testing error on P1	**LWTA**	**Sigmoid**	**ReLU**
After training on P1	$1.55 \pm 0.20\%$	$1.38 \pm 0.06\%$	$1.30 \pm 0.13\%$
After training on P2	$6.12 \pm 3.39\%$	$57.84 \pm 1.13\%$	$16.63 \pm 6.07\%$

datasets like MNIST, it is possible to examine this by recording the units that were active in the layer for each pattern in a subset of 10,000 input patterns. For each class, the subnetwork consisting of units active for at least 90% of the examples was designated the representative *mean* subnetwork, which was then compared to mean subnetworks for all other classes by counting the number of units in common.

Figure 4.3a shows the fraction of units in common between the mean subnetworks of each pair of digits. Digits that are morphologically similar such as "3" and "8" have subnetworks with more units in common than the subnetworks for digits "1" and "2" or "1" and "5" which are intuitively less similar. To verify that this subnetwork specialization is a result of training, the fraction of common units between all pairs of digits for the same 10,000 patterns both before and after training were compared (Figure 4.3b). The figure shows that the subnetworks were much more similar prior to training, and the network has learned to partition its parameters to reflect the structure of the data.

4.5 Implicit Long Term Memory

This section examines the effect of the LWTA architecture on catastrophic forgetting [McCloskey and Cohen, 1989; Carpenter and Grossberg, 1988] to answer the question: does the ability of LWTA networks to act as a collection of several subnetworks allow it to retain information about dataset A, even after being trained on a different dataset B? To test for this *implicit long term memory*, the MNIST training and test sets were each divided into two parts, P1 containing only digits $\{0,1,2,3,4\}$, and P2 consisting of the remaining digits $\{5,6,7,8,9\}$. Three different network architectures were compared: (1) three LWTA-2 hidden layers each with 500 blocks each, (2) three hidden layers each with 1000 units and the logistic sigmoid activation, and (3) three hidden layers of 1000 units each with the ReL activation. All networks had a 5-way softmax output layer representing the probability of an example belonging to each of the five classes. All networks were initialized with the same parameters, and trained with a fixed learning rate and momentum.

Each network was first trained to reach a negative log-likelihood of 0.03 on the P1 training set. This value was chosen heuristically to indicate successful training within reasonable time for all three network types. The weights for the output layer (corresponding to the softmax classifier) were then stored, and the network was trained further, starting with new randomly sampled output layer weights, to reach the same negative log-likelihood on P2. Finally, the output layer weights saved from P1 were restored, and the network was evaluated on the P1 test set. The experiment was repeated for 10 different initializations to account for stochasticity in initialization and training.

Table 4.4 shows that the LWTA network *remembers* what was learned from P1 much better than sigmoid and ReL networks, though it is notable that the sigmoid network performs much worse than both LWTA and ReL. Over multiple trials, the test error values depended on the learning rate and momentum used, but LWTA networks tended to remember better than the ReLU networks by about a factor of two in most cases, and sigmoid networks always performed much worse. Although standard network architectures are known to suffer from catastrophic forgetting, these results show that ReL networks are in fact more robust than sigmoidal networks, and moreover, they are outperformed by LWTA.

Note that [Goodfellow et al., 2014] have conducted further experiments in different settings extending the study here, and reported that on certain task pairs, the use of dropout regularization during training has a strong impact on implicit long-term memory of FNNs.

4.6 Discussion

In this chapter, we demonstrated that it is possible to fore-go traditional unit-wise activation functions in NNs and replace them with a form of local competition among units – local winner-take-all (LWTA) interactions. Although such interactions have a long and rich history of use in NNs, it was not known that networks employing them can be trained using backpropagation and common gradient based optimization algorithms. During training, such networks automatically self-modularize into a collection of multiple parameter-sharing subnetworks responding to different input representations. They match or improve upon results of comparable methods on digit recognition and sentiment analysis, and also avoid catastrophic forgetting, retaining useful representations of one set of input patterns even after being trained to classify another. This has implications for continual learning agents that should not forget representations of parts of their environment when being exposed to other parts. In our experiments, we also found that LWTA networks train faster (i.e. achieve lower loss for the same number of weight updates) compared to networks with sigmoidal non-linearities – a behavior also shown by ReL networks.

A potential limitation of using LWTA interactions is that they introduce jump discontinuities in the unit outputs, which can lead to instability during gradient-based training. However, the particular experiments conducted here did not suffer from this issue. Note that the total contribution of an LWTA block to the next layer can in principle be continuous if all units learn the same outgoing weights. In this case, the block behaves like a maxout [Goodfellow et al., 2013a] block. Another limitation is that certain activation functions are known to benefit from appropriate initialization of weights [LeCun et al., 1998; Glorot and Bengio, 2010; He et al., 2015], but unfortunately such an initialization has not been identified for LWTA networks. The initialization proposed by Glorot and Bengio [2010] for the *tanh* activation function was used here, but this may not be the best choice in general.

It is notable that Maass [1999, 2000] showed that FNNs with WTA dynamics as the only non-linearity are computationally as powerful as networks with threshold or sigmoidal units, and networks employing only soft WTA competition are universal function approximators. Moreover, these results hold even when the network weights are strictly positive—a finding which has ramifications for our understanding of biological neural circuits, as well as the development of neural networks for pattern recognition.

All of the above properties are a direct result of the simple competition mechanism by way of which credit is assigned to linear computation units in a network during backpropagation. It has been noted before that the ReL activation function implements similar behavior in NNs [Nair and Hinton, 2010; Glorot et al., 2011], but this phenomenon has not been investigated in detail so far. This raises interesting open questions: to what extent is the assignment of different input patterns to different linear subnetworks responsible for the performance of the networks? Does this assignment happen early in training and remains constant later — such that initially assigned subnetworks continue to get trained on the assigned patterns — or does the assignment continue to change throughout the training duration? Further, does the evolution of assignments during training differ between LWTA, ReL, and the related maxout networks? Since all of these three lead to improved results and faster training compared to networks with sigmoidal activation functions, answering these questions can provide generalizable insights for improving credit assignment in NNs. This the goal for the analysis presented in the next chapter.

Chapter 5

Understanding Locally Competitive Networks

In the previous chapter, we demonstrated that local competition between units implemented by way of LWTA is sufficient to train powerful FNNs. These results are not too surprising once we note that other recently successful activation functions such as ReL [Jarrett et al., 2009] and maxout [Goodfellow et al., 2013a] are also *locally competitive*. In this chapter we examine this commonality in detail with several experiments designed to understand how local competition regulates credit assignment in these networks, and how understanding this phenomenon may be useful in practice.

5.1 Locally Competitive Networks

We first briefly review the three activation functions described above as locally competitive: ReL, LWTA and maxout. In a typical FNN, the total weighted input or *pre-synaptic activation* z is first computed as $z = \mathbf{w}^\top \mathbf{x} + b$, where $\mathbf{x}$ is the vector of inputs to the unit, $\mathbf{w}$ is a trainable weight vector, and b is a trainable bias.[1] When the ReL activation function is used, the output or post-synaptic activation of each unit is simply $\max(z, 0)$, which can be interpreted as competition between its total input and a fixed value of 0. For LWTA, the units in a layer are divided into blocks of a fixed size and the output of each unit is Iz, where I is an indicator which is one if the unit has the maximum z in its group and zero otherwise. In maxout, the units in a block compete via the max operation, so that the output

[1]This equation describes the unit-level computation instead of the layer-level equations in section 2.3

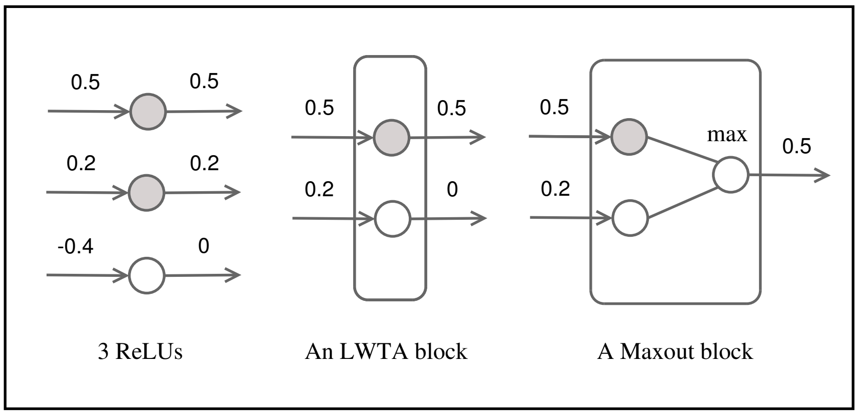


Figure 5.1. Comparison of Rectified Linear Units (ReLUs), Local Winner-Take-All (LWTA), and maxout activation functions. The pre- and post-synaptic activations of the units are shown on the left and right side of the units respectively. The shaded units are 'active' – non-zero activations and errors flow through them. The main difference between maxout and LWTA is that the post-synaptic activation can flow through outgoing connections with different weights depending on the winning unit in LWTA. For maxout, the outgoing weight is the same for all units in a block due to downsampling.

is that of the unit with the highest z in the block.[2] A maxout block can also be interpreted as an LWTA block with shared outgoing weights among the units. A comparison of the three activation functions is shown in Figure 5.1.

5.2 Hypothesis

Our approach to analyzing Locally Competitive Networks (LCNs) starts at the observation that in these networks, only a subset of units (i.e. a "subnetwork" or "submodel") has non-zero activations or gradients for any given input pattern. As an illustrative example, consider the activation of a neural network with rectified linear units (ReLUs) in a single hidden layer. An examination of the subnetworks activated for 100 randomly selected input patterns shows that a large number of different subnetworks are activated (Figure 5.2). The identity of the units composing the active subnetwork is decided through rather simple local competition, as discussed above. Put another way, this competition enables **hard** credit assignment, as opposite to **diffused** credit assignment present in a

[2]In our terminology, the terms *unit* and *block* correspond to the terms *filter* and *units* in Goodfellow et al. [2013a].

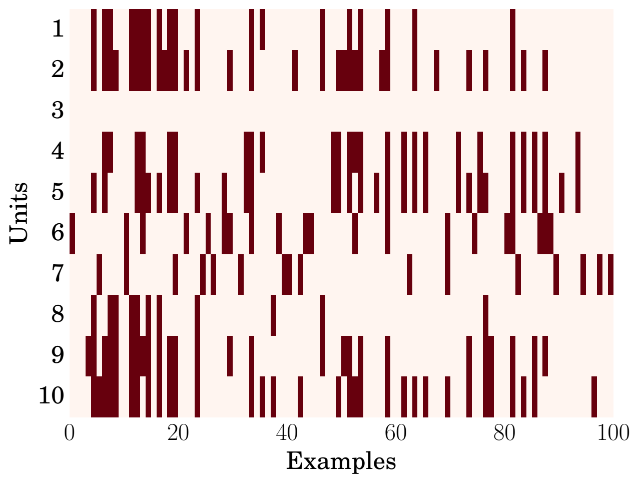


Figure 5.2. Subnetworks for 100 examples for 10 ReLUs. The examples activate many different possible subsets of the units, shown in dark. In this case, unit number 3 is inactive for all examples.

network without such competition, that mitigates the problem of cross-pattern interference [Sutton, 1986]. In fact, a *linear* subnetwork of the network performs the effective computation for any single input pattern, but this subnetwork is different for different input patterns. This observation has been made for ReL networks by Nair and Hinton [2010]; Glorot et al. [2011]. Instead of treating a neural network as a complex, global function approximator, the expressive power of the network can be interpreted to be coming from its ability to activate different subnetworks of linear units for different patterns.

The above observation leads to our central hypothesis: locally competitive networks act as a models that can switch between linear submodels such that *similar submodels respond to similar patterns*, and more importantly, in the classification setting the *mutual similarity of active subnetworks is highly indicative of class membership*. We call this the *model of models* hypothesis and in the remainder of this chapter, demonstrate its validity through visualizations and numerical comparisons. While it is not surprising that the linear subnetworks that respond to patterns of the same class have several units in common, our results show that the relationship between the subnetworks and class membership is strong enough that a simple binary encoding of the active units in a trained network can be used to represent queries for classification or retrieval tasks.

5.3 Subnetwork Analysis

We propose a simple encoding of subnetworks as bit strings called *submasks* for analysis. For input pattern i, the submask $s_i \in \{0, 1\}^u$ represents the corresponding subnetwork by having a '1' in position j, $j = 1..u$, if the corresponding unit *would have non-zero gradients during backpropagation* for the pattern i, and '0' otherwise, where u is the number of units in the full network. There are two important points to note:

- Instead of extracting submasks from all the layers in a network, they will only be extracted from a single layer to keep analysis computationally tractable. Typically, the analysis will focus on the last hidden layer, i.e. the layer whose outputs are fed directly into the final output layer in networks trained for classification.

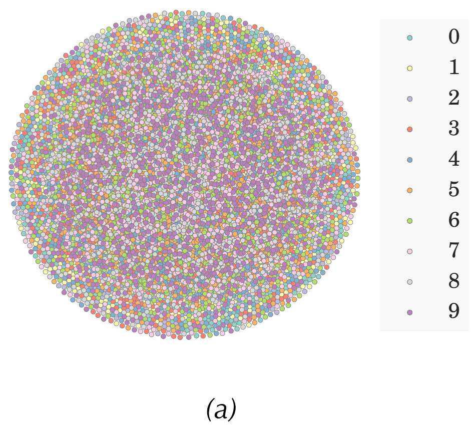
- In ReL and LWTA layers, the units which would have non-zero gradients during backpropagation are simply those for which the output activations are non-zero. For maxout layers, submasks are constructed by binarizing the unit activations such that only the units producing the maximum pre-synaptic activation are represented by a '1' and the rest by '0'.

The submasks uniquely and compactly encode each subnetwork in a format that is amenable to analysis, and as subsection 5.4.2 shows, facilitate efficient data retrieval.

In the next subsection, the t-SNE dimensionality reduction algorithm [Van der Maaten and Hinton, 2008] is used to visualize the relationship between subnetworks that emerge during training. Later in subsection 5.3.2, the evolution of subnetworks during training is examined. subsection 5.3.3 shows that the submasks obtained from a trained network can directly be used for classification using a simple nearest neighbors approach. Finally, subsection 5.3.4 analyses how using dropout during training affects the utility of the submasks. All experiments in this section were performed on the MNIST dataset [LeCun et al., 1998]. This familiar dataset was chosen because it is relatively small and easy for NNs, and therefore provides a tractable setting in which to verify the repeatability of our results. More complex datasets are used in section 5.4 to demonstrate the utility of submasks for classification and retrieval.

5.3.1 Visualization through Dimensionality Reduction

In order to visualize the relationship between submasks for a large number of input patterns, MLPs with three hidden layers and various activation functions



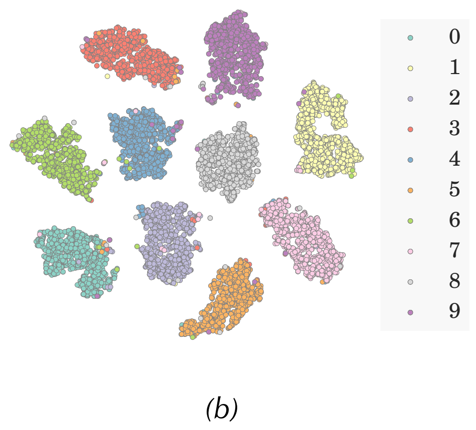
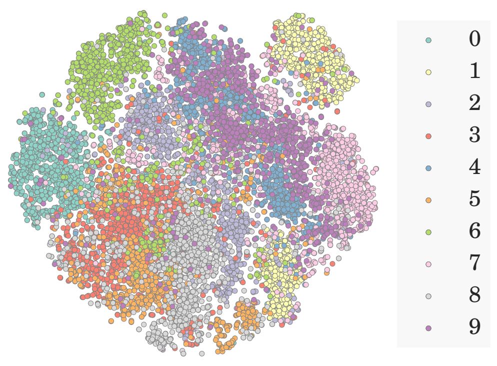


Figure 5.3. 2-D Visualization of submasks from the last of the 3 hidden layers in ReL network for the MNIST test set. (a) shows the submasks before training which lack any discernible relationship. (b) shows submasks from a trained network, showing clearly demarcated clusters relevant to the supervised classification task. 'Mistakes' made by the network can also be observed, such as mistaking '4's for '9's.

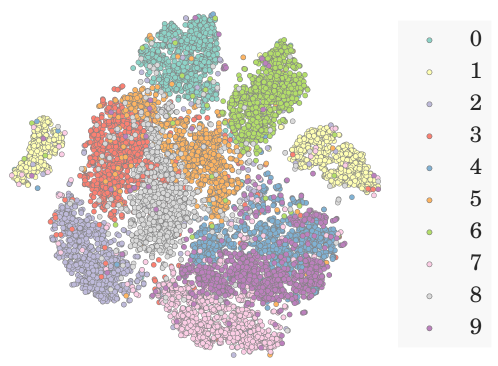
were trained on the MNIST training set, stopping when the error on a separate validation set did not improve for 50 epochs. The submasks for the entire test set of 10K examples were then extracted and embedded into two dimensions using t-SNE for visualization. Each submask is a bit string of length corresponding to the size of the network's last hidden layer (1000 in this case).

Figure 5.3 shows the visualization of the submasks from the last hidden layer of the ReL network before (a) and after (b) training. Before training, i.e. for a randomly initialized network, there is no discernible structure to be observed. After training, ten distinct clusters of submasks are present, one for each MNIST class. It is notable that irrespective of the actual activation values which are lost in the binary representation, the submasks for the test examples are clearly highly indicative of class memberships. In other words, the subnetworks for input patterns of the same class are much more similar to each other compared to those of patterns from different classes.

Figure 5.4 shows that some class-indicative structure can already be observed *before* training if submasks from the first hidden layer (instead of the third) are visualized. This indicates that at the start of network training, the random projection in the first hidden layer already leads to slightly different subsets of units (on average) responding to patterns from different classes. subsection 5.3.2 quantifies how often unit responses change during training.



(a) Untrained 1st LWTA layer.



(b) Untrained 1st ReL layer.

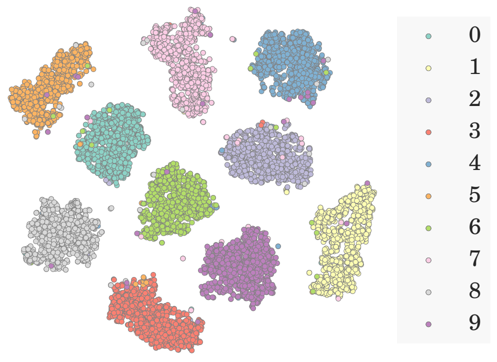
Figure 5.4. 2-D visualization of submasks obtained **before training** from the 1st (closest to the input) hidden layer of 3-hidden-layer LWTA and ReL networks on MNIST test set.

Visualizations of submasks from trained LWTA and maxout networks show class-specific clusters similar to those from the ReL network, as shown in Figure 5.5. Note that the visualizations also show many instances where the network makes incorrect predictions on the test set. The submasks for some examples lie in the cluster of submasks for the wrong class, indicating that the 'wrong' subnetwork was selected for these examples.

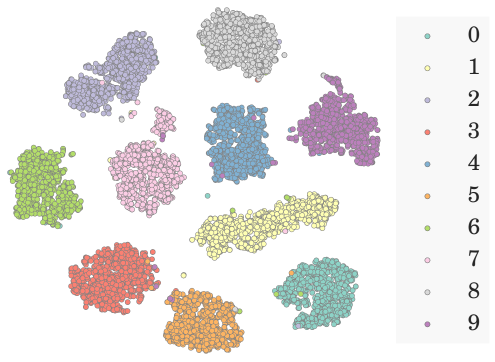
5.3.2 Behavior during Training

How do submasks for different examples become organized over the course of training a network? To answer this question, the submasks of each sample in the training set were recorded at each epoch. Figure 5.6 characterizes the change in the subnets over time by counting the number of input patterns for which a unit flips from being on to being off, or vice-versa, from one epoch to the next. The plot in the figure shows the fraction of patterns for which an inter-epoch flip occurred, averaged across all units in the network. Higher values indicate that the assignment of subnets to patterns is not stable. The batch size for this experiment was 100, which means that each pass over the training set consists of 500 weight updates. For the run shown, the average fraction of flips starts at 0.2, but falls quickly below 0.05 and keeps falling as training proceeds, indicating that, the assignment of subnetworks to individual examples stabilizes quickly.

From a model of models perspective, Figure 5.6 suggests that during training,



(a) Trained LWTA layer.



(b) Trained Maxout layer.

Figure 5.5. 2-D visualization of submasks from the last hidden layer of 3-hidden-layer LWTA and maxout networks on MNIST test set. Organization of submasks into distinct class-specific clusters similar to that in ReL networks is observed.

the subnetworks rapidly organize in an early transient phase such that subnetworks responding to similar examples have more parameters in common than those responding to dissimilar examples. This allows for better specialization of subnetworks due to the reduced interference from dissimilar examples and shared parameters for similar examples. In the later fine-tuning phase when the network already has a low loss, training slows down and much less re-assignment of subnetworks takes place.

5.3.3 Evaluating Submasks

Since the visualization of submasks for the test set shows task-relevant structure, it is natural to ask: how well can a submask represent the input pattern that produces it? If the submasks for similar examples are similar, perhaps they can be used as data descriptors for tasks such as similarity-based retrieval. Class-representative submasks that can be generated without explicit training for retrieval are an attractive possibility since sparse informative binary codes enable efficient storage and retrieval for large and complex datasets. Consequently, learning to generate them is an active research area [Gong et al., 2013; Masci et al., 2014b,a; Grauman and Fergus, 2013]. To quantify the extent of similarity between submasks that result for similar input patterns, we trained LCNs with fully-connected layers for image classification and then used a simple k nearest neighbors (kNN) algorithm to classify images based only on the generated submasks. Training was performed

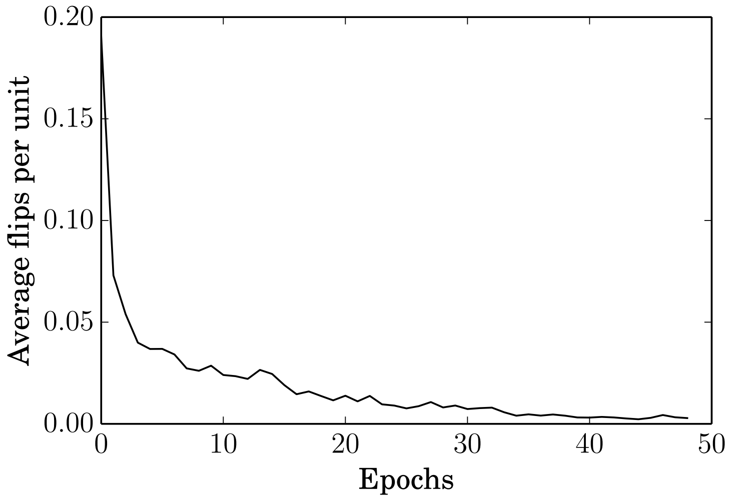


Figure 5.6. The plot shows the mean of the fraction of examples (total 10K) for which units in the layer flip (turn from being active to inactive or vice-versa) after every pass through the training set. The units flip for up to 20% of the examples on average in the first few epochs, but quickly settle down to less than 5%.

MNIST training set, and the value of k with the lowest validation error was selected to classify the test set. If this approach results in a classification error that is close to that obtained using the network's softmax layer's output, we can conclude that submasks are indeed highly informative.

The results are shown in Table 5.1. In each case, the accuracy of the kNN predictions is close to that obtained using the network's outputs. Note that in case of the maxout network, the softmax layer processes the outputs of the last maxout layer, but the submasks were obtained from unit activations before the pooling operation in the layer. So in this case it is more appropriate to compare the kNN classification error for submasks to that obtained if kNN is used to classify the unit activations before pooling as well, which resulted in 121 errors.

Submasks can also be obtained from convolutional layers in trained CNNs. The maxout CNN from Goodfellow et al. [2013a] yields 52 errors on the MNIST test set. Since the penultimate layer in this model is convolutional, the submasks were constructed using the pre-synaptic unit activations from this layer for all convolutional maps. Visualization of these submasks showed similar structure to that obtained from fully-connected layers in Figure 5.5b, and kNN classification on the submasks resulted in 65 errors. It is evident that for all well-trained networks, the kNN performance is close to the performance of the network's final output layer.

Table 5.1. Classification results on the permutation invariant MNIST test set using the networks last layer outputs (softmax) vs. *k*NN on the submasks. All submasks were extracted from the last hidden layer. Classification based on submasks incurs only a minor degradation in performance.

Network	No. test errors	
	Softmax	*k*NN
ReL (trained without dropout)	161	158
LWTA (trained with dropout)	142	154
Maxout (trained with dropout)	116	131

5.3.4 Effect of Dropout

Regularization through dropout [Hinton et al., 2012b] has proven to be an efficient technique for improving the generalization of large networks, and has been often used in combination with locally competitive activation functions [Krizhevsky et al., 2012; Goodfellow et al., 2013a; Zeiler and Fergus, 2013]. In order to examine how dropout may affect the organization of useful submasks, a 3-hidden-layer network with 800 ReLUs in each hidden layer was trained without dropout (denoted by ND) on PI-MNIST, starting from five different initializations until the validation set error did not improve. The networks were then trained again from the same initialization with dropout (WD) until the validation error matched (WD-M) and then fell below (WD-B) the lowest validation error from the ND case. In both cases, minibatch gradient descent with momentum was used. Finally, the submasks from both ND and WD networks were compared in terms of *k*NN classification performance.

The networks which were trained to result in lower classification error with dropout (WD-B) also yielded better submasks in terms of *k*NN classification performance. On the other hand the submasks from ND and WD-M networks give similar results. These results support the interpretation of dropout as a regularization technique that prevents “co-adaptation of feature detectors” (units) [Hinton et al., 2012b], leading to better representation of data by the subnetworks. In the model of models view, dropout improves generalization by injecting noise in the assignment of submodels, making them more robust.

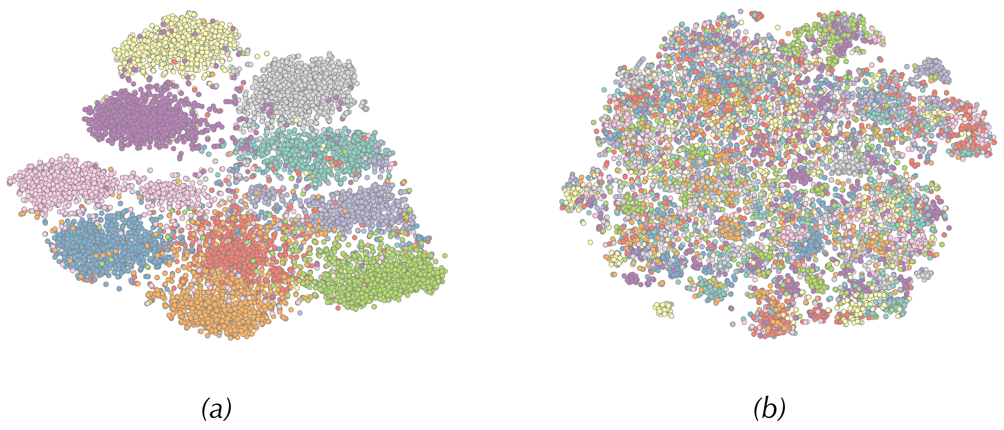


Figure 5.7. 2-D visualizations of the submasks obtained from the penultimate layer of the trained maxout networks reported in Goodfellow et al. [2013a]. (a) The CIFAR-10 test set. The 10-class structure is visible, although the clusters are not as well separated as in the case of MNIST. This corresponds with the higher error rates obtained using both *k*NN and the full network. (b) The CIFAR-100 test set. It is difficult to visualize any dataset with 100 classes, but several clusters are still visible. The separation between clusters is much worse, which is reflected in the high classification error.

5.4 Experiments on Larger Datasets

The following experiments apply the methods described in the previous section to more challenging benchmark problems: CIFAR-10, CIFAR-100, and ImageNet. For the CIFAR experiments, the models described in Goodfellow et al. [2013a] are used since they use locally competitive activations (maxout), are trained with dropout, and good hyperparameter settings for them are available [Goodfellow et al., 2013b]. We report the classification error on the test set obtained using the softmax output layer, as well as kNN classification on both the penultimate layer unit activations and submasks for comparison. The best value of k was obtained based on the validation set, though $k = 5$ with distance weighting was found to usually work well.

5.4.1 CIFAR-10 & CIFAR-100

CIFAR-10 and CIFAR-100 [Krizhevsky, 2009] are datasets of 32×32 color images divided into 10 classes. The results obtained on the test sets for these datasets are summarized in Table 5.2. kNN on submasks resulted in a loss in accuracy of 1.25% on the CIFAR-10 dataset, and 2.26% on the CIFAR-100 dataset on average, compared to using unit activations. Figure 5.7a shows the 2-D visualization of the test set submasks for CIFAR-10. Some classes can be seen to have highly representative submasks, while confusion between classes can be seen in the lower half of the plot. The clusters of subnetworks are not as well separated as they were with MNIST [3] , reflecting the relatively worse classification performance of the full network. Submask visualization for CIFAR-100 (Figure 5.7b) reflects the high error rate for this dataset. Although any visualization with 100 classes can be hard to interpret, many small clusters of submasks can still be observed.

5.4.2 ImageNet

In this section, the utility of the submasks obtained from large CNNs trained on the ImageNet Large Scale Visual Recognition Challenge 2012 (ImageNet 2012) [Deng et al., 2012] dataset is evaluated, pointing to potential practical uses of submasks.

We first evaluate classification using kNN as before, but report top-5 instead of top-1 accuracy, as is common practice for this dataset. Two networks are

[3]In general, this depends on the value of the *perplexity* hyperparameter in t-SNE. However, well separated clusters similar to MNIST were not obtained for various values of perplexity.

Table 5.2. Classification errors on CIFAR datasets comparing maxout network performance, *k*NN on activation values, *k*NN on pre-activations (before maximum pooling) and *k*NN on binary submasks. Results are averaged over 5 runs.

Dataset	Softmax	*k*NN (activations)	*k*NN (pre-activations)	*k*NN (submasks)
CIFAR-10	9.94 ± 0.31%	9.63 ± 0.21%	10.11 ± 0.16%	11.36 ± 0.22%
CIFAR-100	34.49 ± 0.22%	37.54 ± 0.14%	41.37 ± 0.26%	43.63 ± 0.18%

Table 5.3. Top-5 Classification accuracy on validation set when performance of two different networks on ImageNet is compared to performance of submasks obtained from each of them. Note that as network accuracy improves by about 6%, submask accuracy improves by about 10%.

Network	Softmax	*k*NN on submasks
DeCAF [Donahue et al., 2013]	19.2%	29.2%
Convnet	13.5%	20.4%

compared: DeCAF [Donahue et al., 2013] and Toronto Convnet[4]. Table 5.3 compares the results obtained from both networks originally as well as those obtained using *k*NN on the submasks from the penultimate layer. Submasks retain a large amount of information on this difficult large-scale task, while greatly improving storage efficiency since only one bit per dimension is required for storage. It is also important to note that submasks obtained from a better trained network result in better performance. An improvement of 5.7% in the network error translates to 8.8% improvement in the performance of *k*NN on submasks. This progression, together with similar trends observed for small scale datasets in the previous sections, indicates that using more accurate networks, highly accurate and efficient submasks can be obtained.

There are many approaches dedicated to constructing binary codes for fast similarity-based search and retrieval (Grauman and Fergus [2013] provide a review). Krizhevsky et al. [2012] have suggested that the activations from a trained convolutional network can be compressed to binary codes using auto-encoders. As a cheap alternative to these approaches, the results here show that submasks from a trained network can be utilized directly for efficient data

[4]https://github.com/torontodeeplearning/convnet/

Table 5.4. Comparison of mean average precisions at various thresholds using binary codes obtained using different techniques on the ImageNet 2012 dataset. Submasks are obtained directly from networks trained for classification without any further training. Up to mAP@100 the submasks show a constant performance degradation of about 3 points.

Technique	mAP@5	mAP@10	mAP@100
Submasks	58.3	56.7	46.7
Diffhash	61.0	59.3	49.5

retrieval based on similarity, without the use of any pair-wise supervision during training of the original network.

Submask based retrieval results on ImageNet 2012 were compared to those obtained using binary codes produced by DiffHash, a supervised similarity-preserving hashing method proposed by Strecha et al. [2012] trained on the non-binarized last hidden layer features from the network. The network used was Toronto Convnet, with a top-5 classification error of 13.5% on the validation set. DiffHash learns a linear projection and is trained by providing similar and dissimilar pairs of points, for which a ground-truth similarity measure is known, i.e., sharing the same class or not. Precision-recall curves obtained for the two approaches are shown in Figure 5.8, and Table 5.4 reports results for mean average precision; $mAP = \sum_{r=1}^{R} P(r) \cdot rel(r)$, where $rel(r)$ indicates the relevance of a result at a given rank r, $P(r)$ is the precision at r, and R is the number of retrieved results.

The performance obtained using submasks is comparable to but strictly lower than that of DiffHash codes. Nevertheless, the results of this comparison are very encouraging considering the experimental setting, and there are clear indications that submasks from a network with lower classification error may further close the gap. A few sample retrieval results for examples from the ImageNet 2012 dataset are shown in Figure 5.9. An interesting avenue for future research is exploration of additional regularization techniques during supervised training of the network to further improve the utility of submasks for search and retrieval.

5.5 What about Sigmoidal Networks?

In principle it is possible to use sigmoidal activation functions to obtain binary codes by thresholding the activation values after supervised or unsupervised [Salakhutdinov and Hinton, 2009] training. But there are two limitations

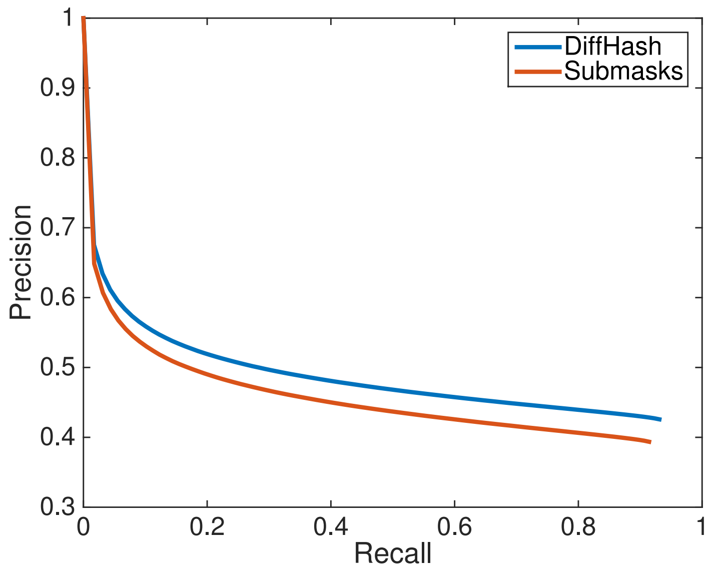


Figure 5.8. Comparisons of precision-recall curves on ImageNet 2012 when using binary codes obtained using different techniques. The performance of submasks is competitive and decays only for high recall values where supervised hashing obtains a better ranking of the results due to the pair-wise supervision.

of this approach:

- The main limitation is that large sigmoidal networks are slow to train. Due to this, obtaining useful activations from them is impractical for large datasets which are common in retrieval settings. LCNs have been crucial for the successful application of neural networks to such datasets.
- The thresholding is somewhat arbitrary and the best threshold needs to be selected by trial-and-error. For LCNs, the binarization is natural and inherent to the process of credit assignment in these networks.

5.6 Discussion

To recap, we presented the following evidence in support of the model of models hypothesis put forth at the beginning of this chapter:

- In LCNs, local competition within each layer assigns credit to similar subsets of units (having many units in common) for similar input patterns (as determined by the objective function used for training).

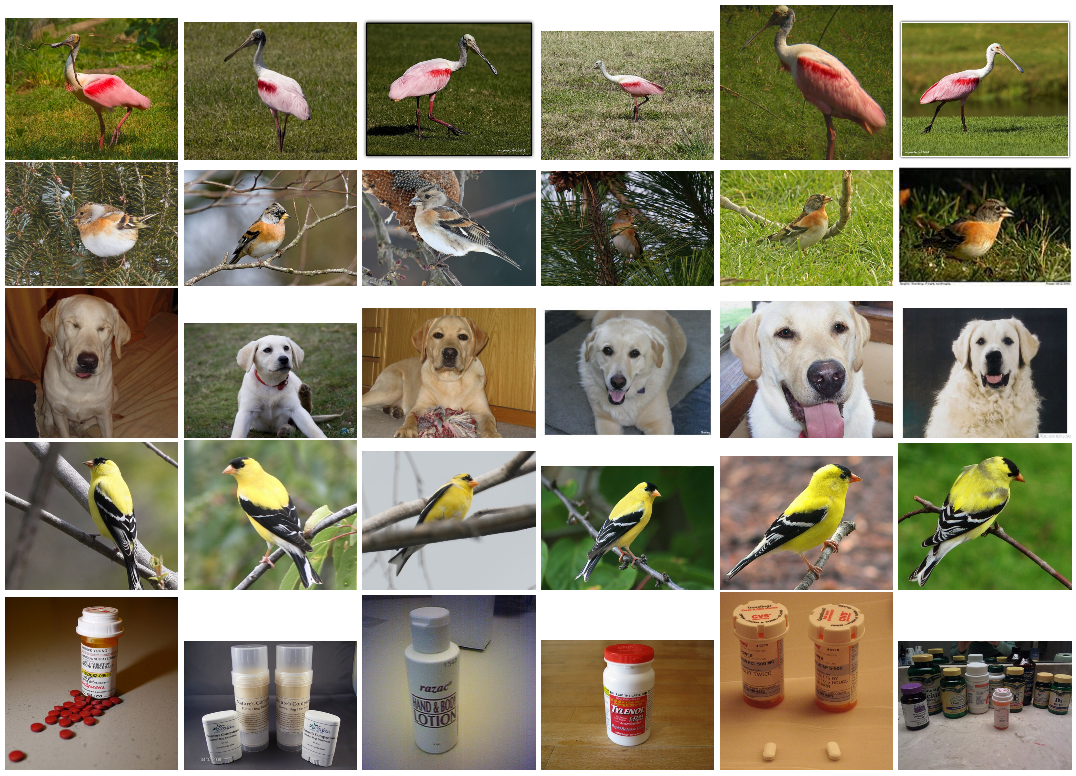


Figure 5.9. Retrieval based on finding images with similar submasks on the ImageNet 2012 dataset. The first image in each row is the query image, the other five are the responses retrieved using nearest neighbor search.

- During training of networks, a large degree of reassignment of subnetworks happens early in the training as the networks loss improves rapidly. The reassignment slows down later in training in the fine tuning phase.
- The submask encoding representing the credit assignment is highly informative and can be used for making predicting the class membership of test data instead of the unit outputs. For highly accurate networks, the predictive accuracy approaches that of the outputs.

LCNs enable easier and faster training on complex pattern recognition tasks compared to networks with sigmoidal or similar activation functions. The findings in this chapter indicate that low interference due to hard credit assignment among subnetworks is the key property that speeds up learning by reducing cross-pattern interference in NNs.

The core principles that underlie the beneficial properties of LCNs are essentially the same as those that motivated competitive learning approaches developed in the past several decades (subsection 3.1.1). In particular, a key difference to the Mixture of Experts architecture developed by Jacobs et al. [1991a,b] is that LCNs are not composed of separate expert networks that specialize in processing different sets of input patterns. Instead, every possible subset of units of in a network acts as a very “local” expert, and increasingly different input patterns activate increasingly different subsets of units.

Chapter 6

Highway Networks

This chapter addresses the challenge of designing NN architectures with the purpose of improving the training of very deep networks by deriving inspirations from multiple sources. The first inspiration comes from the benefits of improved regulation of information flow in locally competitive NNs studied in chapters 4 and 5. The second is the work of Jacobs et al. [1991a,b] whose modular networks explicitly regulate information flow between network modules through a separately learned gating network. Finally, ideas from the LSTM architecture for RNNs that ease temporal credit assignment are borrowed and extended to the case of feedforward networks. Together, these ideas lead to *Highway networks*, a class of feedforward networks that can be reliably trained even when they have hundreds of layers.

6.1 Architecture

For simplicity, in this chapter the non-linear function implemented by the ℓ^{th} layer in a plain MLP (Equation 2.5) is written compactly as:

$$\mathbf{y}_\ell = H(\mathbf{x}_\ell), \tag{6.1}$$

where applying the function H typically consists of an affine transformation (with weights and biases) of the input $\mathbf{x}_l$ followed by a non-linearity, but in general it can take other forms.

Local competition supports implicit, local gating of information to constituent subnetworks in contrast to the explicitly-learned, global gating network in the work of Jacobs et al. [1991a,b], as mentioned at the end of the previous chapter. As a first step towards an architecture that extends the benefits of information gating to very deep networks, the best of both approaches can be combined such that the gating of information is explicit, but still implemented locally at each unit. A layer in such an explicit-but-local network can then be defined as

$$\mathbf{y}_\ell = H(\mathbf{x}_\ell) \cdot G(\mathbf{x}_\ell). \tag{6.2}$$

where G is a gating transformation similar to H, implemented by an additional set of units. We refer to this as a *Mixture of Units* (MoU) architecture. Compared to a plain layer of units, each unit in the MoU layer is paired with an additional gating unit (with its own weights and bias) which adjusts its output dynamically, i.e. differently for each input pattern, through a multiplicative connection.[1]

Preliminary experiments on the MNIST dataset using *NN*s with the above architecture were conducted using the logistic sigmoid activation function in G and *tanh* in H. The results and speed of training observed were similar to ReL and LWTA networks, even though the network used only sigmoidal non-linearities. Since plain networks with sigmoidal non-linearities are generally slow to train, we can surmise that the introduced local gating mechanism improves credit assignment and enables performance comparable to locally competitive networks. However, the MoU architecture does not fundamentally address the difficulties encountered when training very deep networks since, like locally competitive networks in previous chapters, they only improve credit assignment among units in the same layer.

To address this limitation, we take inspiration from the architecture of LSTM, and further extend the MoU architecture by adding additional units, implemented with a non-linear transformation C, that learn to regulate the flow of information across layers in addition to within layers:

$$\mathbf{y}_\ell = H(\mathbf{x}_\ell) \cdot T(\mathbf{x}_\ell) + \mathbf{x}_\ell \cdot C(\mathbf{x}_\ell) \tag{6.3}$$

where the gating transformation, G. has been renamed since there are now two types of gating units. T implements the *transform gates*, which learn how much the regular transformation, H, of the input contributes to the total output $\mathbf{y}$ of the layer. C implements the *carry gates*, which learn how much of the input is carried over to the output. The gates are analogues of the *input* and *forget* gates in the LSTM architecture, while a skip connection from input to output with a constant weight of one is analogous to the recurrent self-connection used by the LSTM cell. We refer to networks with this architecture as **Highway networks**, since they allow unimpeded flow of information across layers.

[1]To support this interpretation, G should be defined such that it produces values in $[0,1]$ although in practice this may not be a hard constraint.

The transformation implemented by a Highway layer is much more flexible than Equation 6.1. In particular, observe that for particular outputs of T and C,

$$\mathbf{y}_\ell = \begin{cases} \mathbf{x}_\ell, & \text{if } T(\mathbf{x}_\ell) = \mathbf{0}, C(\mathbf{x}_\ell) = \mathbf{1}, \\ H(\mathbf{x}_\ell), & \text{if } T(\mathbf{x}_\ell) = \mathbf{1}, C(\mathbf{x}_\ell) = \mathbf{0}. \end{cases} \tag{6.4}$$

Similarly, for the derivative of the layer outputs with respect to its inputs,

$$\frac{d\mathbf{y}_\ell}{d\mathbf{x}_\ell} = \begin{cases} \mathbf{I}, & \text{if } T(\mathbf{x}_\ell) = \mathbf{0}, C(\mathbf{x}_\ell) = \mathbf{1}, \\ H'(\mathbf{x})_\ell, & \text{if } T(\mathbf{x}_\ell) = \mathbf{1}, C(\mathbf{x}_\ell) = \mathbf{0}. \end{cases} \tag{6.5}$$

Thus, depending on the output of the transform and carry gates, a Highway layer can vary its behavior between that of H, a layer that simply passes its inputs through, and their unit-wise combination. This design enables credit assignment across large depths since during the backward pass, gradients can flow backwards without diminishing through paths on which carry gate outputs are non-zero. We define the following terminology for subsequent sections: just as a plain layer consists of multiple computing units each computing its output $y = H(\mathbf{x}_l)$, a Highway layer consists of multiple *blocks*, with each block computing a *block state* $H(\mathbf{x}_l)$, *transform gate output* $T(\mathbf{x}_l)$ and *carry gate output* $C(\mathbf{x}_l)$. Finally, it produces the *block output* $y = H(\mathbf{x}) * T(\mathbf{x}) + x * C(\mathbf{x})$, which is connected to the next layer. Note that x is the scalar output of the corresponding block in the previous layer, while $\mathbf{x}$ is the vector of all block outputs from the previous layer.

6.2 Variants of Highway Networks

Just like it is possible to construct several variants of the LSTM architecture by adding/removing units or connections [Greff et al., 2017a; Jozefowicz et al., 2015], corresponding variants of Highway networks can also be constructed. For most experiments in the remainder of this thesis, the *coupled* variant is used by setting $C(\mathbf{x}_\ell) = 1 - T(\mathbf{x}_\ell)$, so that the block output is a convex combination of $H(\mathbf{x}_l)$ and $\mathbf{x}_l$:

$$\mathbf{y}_\ell = H(\mathbf{x}_\ell) \cdot T(\mathbf{x}_\ell) + \mathbf{x}_\ell \cdot (1 - T(\mathbf{x}_\ell)). \tag{6.6}$$

Several further variants can be constructed by restricting the gate outputs to constant values instead of learning weights that control them:

Full. The original Highway architecture in which both transform and carry gates are learned, i.e., Equation 6.3.

Table 6.1. Test set classification accuracy for pilot experiments on the MNIST dataset. Results reported for Maxout [Goodfellow et al., 2013b] and Deeply Supervised Nets (DSN) [Lee et al., 2015] are included for comparison.

Network	**Highway Networks**		**Maxout**	**DSN**
	10-layer (width 16)	10-layer (width 32)		
No. of parameters	39 K	151 K	420 K	350 K
Test Accuracy (in %)	99.4±0.03	99.54±0.02	99.55	99.61

MoU. No skip connections, obtained if carry gates are always set to zero, i.e., Equation 6.2.

Multiplicative-skip Transform gates are always set to zero, i.e., $\mathbf{y}_\ell = \mathbf{x}_\ell \cdot C(\mathbf{x}_\ell)$.

Additive skip or *Residual*. Proposed by He et al. [2016a], in which both transform and carry gate are set to one, i.e., $\mathbf{y}_\ell = H(\mathbf{x}_\ell) + \mathbf{x}_\ell$.

C-Only. Retains learned carry gates but transform gates are set to one i.e. $\mathbf{y}_\ell = H(\mathbf{x}_\ell) + \mathbf{x}_\ell \cdot C(\mathbf{x}_\ell)$.

T-Only. Retains learned transform gates but carry gates are set to one i.e. $\mathbf{y}_\ell = H(\mathbf{x}_\ell) \cdot T(\mathbf{x}_\ell) + \mathbf{x}_\ell$.

Section 6.5 presents experimental comparisons between variants obtained in this manner.[2]

For all variants, the dimensionality of $\mathbf{x}_\ell, \mathbf{y}_\ell, H(\mathbf{x}_\ell)$, $T(\mathbf{x}_\ell)$, and $T(\mathbf{C}_\ell)$ must be the same for the unit-wise multiplicative gating in Equation 6.3 to be valid. To change the size of the intermediate representation, one can replace $\mathbf{x}_\ell$ with $\hat{\mathbf{x}}_\ell$ obtained by suitably sub-sampling or zero-padding $\mathbf{x}_\ell$. The strategy used in this thesis is to simply use a plain layer instead of a Highway layer whenever the size of the representation in the network needs to be changed.

[2]Pilot optimization experiments (subsection 6.3.2) on training very deep networks were also successful with a more complex design based on an LSTM block "unrolled in time", but this design is not explored further here since simpler designs already work well.

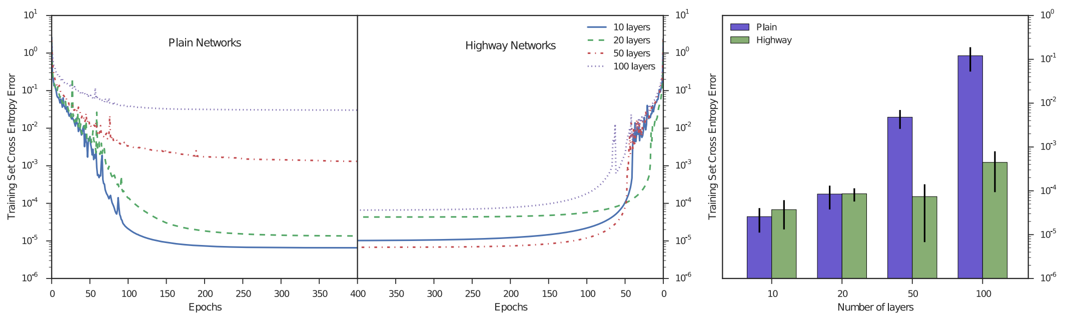


Figure 6.1. Comparison of optimization of plain networks and Highway networks of various depths. All networks are trained for MNIST digit classification. *Left:* The training curves for the best hyperparameter settings obtained for each network depth. *Right:* Mean performance of top 10 (out of 100) hyperparameter settings. Plain networks become much harder to optimize with increasing depth, while Highway networks with up to 100 layers can still be optimized well.

6.3 Experiments

6.3.1 Setup

We implemented transform gates as $T(\mathbf{x}_\ell) := \sigma(\mathbf{W}_T\mathbf{x}_\ell + \mathbf{b}_T)$, where $\mathbf{W}_T$ is the weight matrix and $\mathbf{b}_T$ the bias vector for the transform gates. Note that $\sigma(x) \in (0,1), \forall x \in \mathbb{R}$, so the conditions in Equation 6.4 would never be met exactly. This suggests a simple initialization scheme that is independent of the form of H: $\mathbf{b}_T$ can be initialized with a negative value (e.g. -1, -3 etc.) such that the network is initially biased towards *carry* behavior. This scheme is based on the proposal by Gers et al. [2000] to initially bias the gates in an LSTM network in order to help bridge long-term temporal dependencies early in learning. In our experiments, it was found that a negative bias initialization for the transform gates was sufficient for training to proceed in very deep networks with more than 1000 layers for various zero-mean initial distributions of the weights and different activation functions used by H. In general the initial bias is best treated as a hyperparameter, but as a first guess, values of -1, -2 and -3 for Highway networks of depth approximately 10, 20 and 30 respectively worked well.

Convolutional Highway networks were constructed by simply employing the coupled Highway transformation (Equation 6.6) in CNN layers, using convolution operations with the same receptive fields and spatial strides for all transformation functions in a layer. All networks were trained using SGD with momentum. An exponentially decaying learning rate was used in subsection 6.3.2. For the rest of the experiments, a simpler commonly used strategy was employed where the learning rate starts at a value and decays according to a fixed schedule by a factor. These hyperparameters and schedule were selected once based on validation set performance on the CIFAR-10 dataset, and kept fixed for all experiments in subsection 6.3.5. All convolutional Highway networks used the rectified linear activation function [Glorot and Bengio, 2010] to compute the block state H. To provide a better estimate of the variability of classification results due to random initialization, results are reported based on 5 runs wherever available. Experiments were conducted using Caffe [Jia et al., 2014], Brainstorm (`https://github.com/IDSIA/brainstorm`), and Torch [Collobert et al., 2011] frameworks. Source code, hyperparameter search results and related scripts are publicly available at the URL: `http://people.idsia.ch/~rupesh/very_deep_learning/`.

6.3.2 Optimization Stress Test

To support the hypothesis that training of Highway networks does not fail when depth increases dramatically, a series of rigorous optimization experiments were conducted, comparing them to plain networks with normalized initialization [Glorot and Bengio, 2010; He et al., 2015]. Plain and Highway networks of varying depths were trained on the MNIST digit classification dataset. All networks were *thin*: each layer had 50 blocks for highway networks and 71 units for plain networks, yielding roughly identical numbers of parameters (≈5000) per layer. In all networks, the first layer was a fully-connected plain layer followed by 9, 19, 49, or 99 fully-connected plain or Highway layers. The network output was produced by a *softmax* layer. A random search of 100 runs was performed for both plain and Highway networks to find good settings for the following hyperparameters: initial learning rate, momentum, learning rate exponential decay factor & activation function (either *rectified linear* or *tanh*). For Highway networks, an additional hyperparameter was the initial value for the transform gate bias (between -1 and -10). Weights for the H and T transformations were initialized using the same normalized initialization as plain networks.

The training curves for the best performing networks for each depth are shown in Figure 6.1. As expected, 10 and 20-layer plain networks exhibit very good performance (mean loss $< 1 \times 10^{-4}$), which significantly degrades as depth increases, even though network capacity increases. Highway networks do not suffer from an increase in depth, and 50/100 layer highway networks perform similar to 10/20 layer networks. The 100-layer highway network performed more than 2 orders of magnitude better than a similarly-sized plain network. It was also observed that Highway networks consistently converged significantly faster

Table 6.2. Experimental comparison between Fitnets and Highway networks for CIFAR-10 object recognition. Architectures tested were based on *Fitnets* trained by Romero et al. [2014] using two-stage hint based training. Highway networks were trained in a single stage without hints, matching or exceeding the performance of Fitnets.

Network	**Depth**	**Parameters**	**Accuracy (%)**
Fitnets (reported by Romero et al. [2014])			
Teacher	5	≈9M	90.18
Fitnet A	11	≈250K	89.01
Fitnet B	19	≈2.5M	91.61
Highway networks			
Highway A (Fitnet A)	11	≈236K	89.18
Highway B (Fitnet B)	19	≈2.3M	**92.28±0.16**
Highway C	32	≈1.25M	91.20

than plain ones.

6.3.3 MNIST Digit Classification

As a sanity check for the generalization capability of Highway networks, 10-layer convolutional Highway networks on MNIST were trained, using two architectures, each with 9 convolutional layers followed by a softmax output layer. The number of filter maps (width) was set to 16 and 32 for all the layers. The trained networks resulted in test set performance competitive with state of the art methods but had much fewer parameters, as show in Table 6.1.

6.3.4 Comparison to Fitnets on CIFAR-10

Due to improved credit assignment through local competition, maxout networks [Goodfellow et al., 2013b] can cope much better with increased depth than those with traditional activation functions. However, Romero et. al. [Romero et al., 2014] recently reported that training on CIFAR-10 through plain backpropagation was only possible for maxout networks with a depth up to 5 layers when the number of parameters was limited to ≈250K and the number of multiplications to ≈30M. Similar limitations were observed for higher computational budgets. Training of deeper networks was only possible through the use of a two-stage training procedure and addition of soft targets produced from a pre-trained shallow teacher network (hint-based training).

Highway networks with number of parameters and operations comparable to those of Fitnets were found to be easy to train in a single stage using minibatch gradient descent with momentum. As shown in Table 6.2, networks Highway A and Highway B, which were based on the architectures of Fitnet A and Fitnet B, respectively, obtained similar or higher accuracy on the test set. It was further found that the networks could be made thinner and deeper: for example a 32-layer highway network consisting of alternating receptive fields of size 3×3 and 1×1 with $\approx$1.25M parameters performed better than the earlier teacher network [Goodfellow et al., 2013b].

6.3.5 Results on CIFAR-10 and CIFAR-100

It is possible to obtain high performance on the CIFAR-10 and CIFAR-100 datasets by utilizing very large networks and extensive data augmentation. This approach was popularized by Ciresan et al. [2012] and recently extended by Graham [2014]. Since our aim is only to demonstrate that deeper networks can be trained without sacrificing ease of training or generalization ability, here we performed experiments in the more common setting of global contrast normalization, small translations and mirroring of images. Following Lin et al. [2014], the fully connected layer used in the Highway B network from the previous section was replaced with a convolutional layer having a receptive field of size one followed by a global average pooling layer. The hyperparameters from the last section were re-used for both CIFAR-10 and CIFAR-100. The test set results obtained for these datasets are tabulated in Table 6.3. The results are close to the best results obtained in a comparable setting for CIFAR-10, and significantly better for CIFAR-100. Together, these results demonstrate that not only does the Highway architecture allow direct training of deeper networks with popular gradient descent algorithms, it also results in networks that generalize well on unseen test data.

In the next section, we conduct introspective experiments to gain a qualitative understand of the computations performed by coupled Highway networks (the variant used in experiments so far) trained for supervised learning.

Table 6.3. Test set accuracy of convolutional Highway networks on the CIFAR-10 and CIFAR-100 object recognition datasets with typical data augmentation. For comparison, the accuracy reported by recent studies in similar experimental settings is listed as well.

Network	**CIFAR-10 Accuracy (%)**	**CIFAR-100 Accuracy (%)**
Maxout [Goodfellow et al., 2013b]	90.62	61.42
dasNet [Stollenga et al., 2014]	90.78	66.22
NiN [Lin et al., 2014]	91.19	64.32
DSN [Lee et al., 2015]	92.03	65.43
All-CNN [Springenberg et al., 2014]	**92.75**	66.29
Highway Network	92.31±0.12	**67.61±0.15**

6.4 Analysis

Figure 6.2 illustrates the inner workings of the best[3] 50-hidden-layer fully-connected Highway networks trained on MNIST (top row) and CIFAR-100 (bottom row). In each row, the x-axis corresponds to Highway blocks and y-axis corresponds to Highway layers, with the top most layer being the closest to the network's input. The first three columns show the bias, the mean activity over all training samples, and the activity for a single random sample for each transform gate respectively. Block outputs for the same single sample are displayed in the last column.

The transform gate biases of the two networks were initialized to -2 and -4 respectively, and Figure 6.2 shows that most biases decreased further during training. For the CIFAR-100 network the biases increase gradually with depth. They are lowest near the input (top) and highest near the output (bottom), forming a color gradient in the figure. Interestingly, this pattern is the inverse of the average activity of the transform gates which decreases from input to output, as seen in the second column. This indicates that the strong negative biases of transform gates near the network's input are not used to shut down the gates, but to make them more selective. This behavior is also suggested by the fact that the transform gate activity for a single example (column 3) is very sparse. The effect is more pronounced for the CIFAR-100 network, but can also be observed to a lesser extent in the MNIST network.

The last column of Figure 6.2 displays the block outputs and visualizes the

[3]obtained via random search over hyperparameters to minimize the best training set error achieved using each configuration

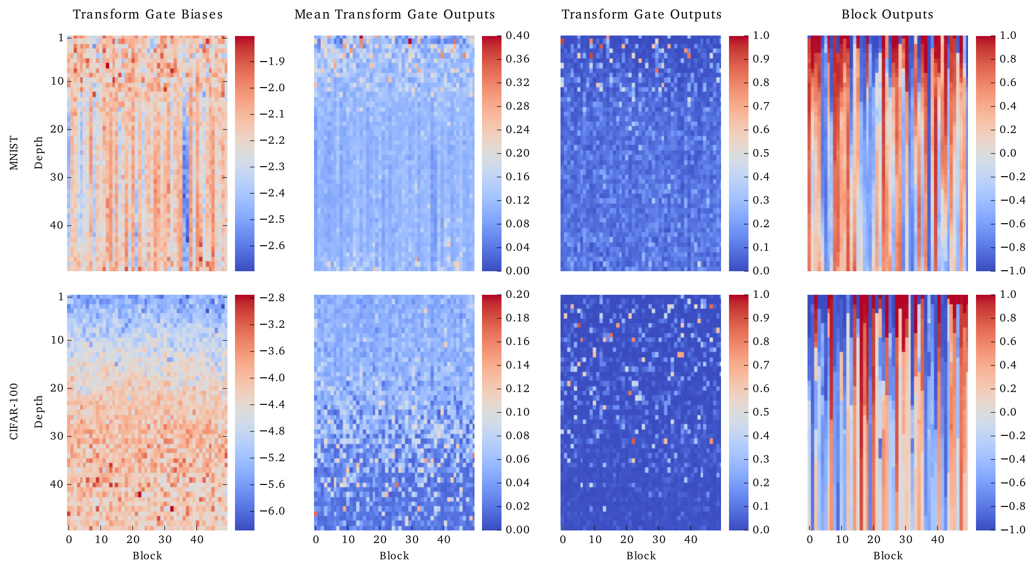


Figure 6.2. Visualization of best 50 hidden-layer highway networks trained on MNIST (top row) and CIFAR-100 (bottom row). The first hidden layer is a plain layer which changes the dimensionality of the representation to 50. Each of the 49 highway layers (y-axis) consists of 50 blocks (x-axis). The first column shows the transform gate biases, which were initialized to -2 and -4 respectively. In the second column the mean output of the transform gate over all training examples is depicted. The third and fourth columns show the output of the transform gates and the block outputs (both networks using *tanh*) for a single random training sample. Best viewed in color.

concept of "information highways". Most of the outputs stay constant over many layers forming a pattern of stripes. Most of the change in outputs happens in the early layers ($\approx$ 15 for MNIST and $\approx$ 35 for CIFAR-100).

6.4.1 Routing of Information

An important feature of the Highway architecture over hard-wired shortcut connections is that the network can learn to dynamically adjust the routing of the information based on the current input. This begs the question: does this behavior manifest itself in trained networks or do they just learn a static routing that applies to all inputs similarly. A partial answer can be found by looking at the mean transform gate activity (second column) and the single example transform gate outputs (third column) in Figure 6.2. Especially for the CIFAR-100 case, most transform gates are active on average, while they show very selective activity for

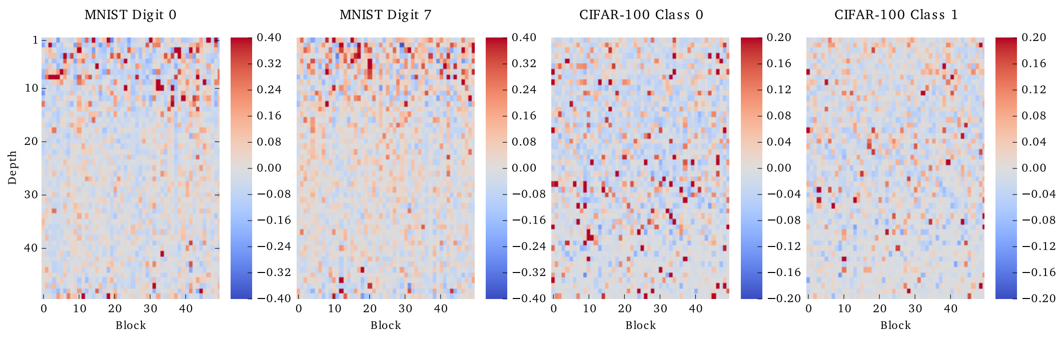


Figure 6.3. Visualization showing the extent to which the mean transform gate activity for certain classes differs from the mean activity over all training samples. Generated using the same 50-layer highway networks on MNIST on CIFAR-100 as Figure 6.2. Best viewed in color.

the single example. This implies that for each sample only a few blocks perform transformation but different blocks are utilized by different samples.

This data-dependent routing mechanism is further investigated in Figure 6.3. Each of the columns show how the average over all samples of one specific class differs from the total average shown in the second column of Figure 6.2. For MNIST digits 0 and 7 substantial differences can be seen within the first 15 layers, while for CIFAR class numbers 0 and 1 the differences are sparser and spread out over all layers. In both cases it is clear that the mean activity pattern differs between classes. The gating system acts not just as a mechanism to ease training, but also as an important part of the computation in a trained network.

6.4.2 Layer Importance

Since all the transform gates were initially biased towards being closed, in the beginning of training every layer mostly copies the activations of the previous layer. Does training indeed change this behavior, or is the final network still essentially equivalent to a network with a much fewer layers? To shed light on this issue, we investigated the extent to which *lesioning* a single layer affects the total performance of trained networks from subsection 6.3.2. Lesioning means manually setting all the transform gates of a layer to 0, forcing it to simply copy its inputs. For each layer, the network was evaluated on the full training set with the gates of that layer closed. The resulting performance as a function of the lesioned layer is shown in Figure 6.4.

For MNIST (left) it can be seen that the error rises significantly if any one of

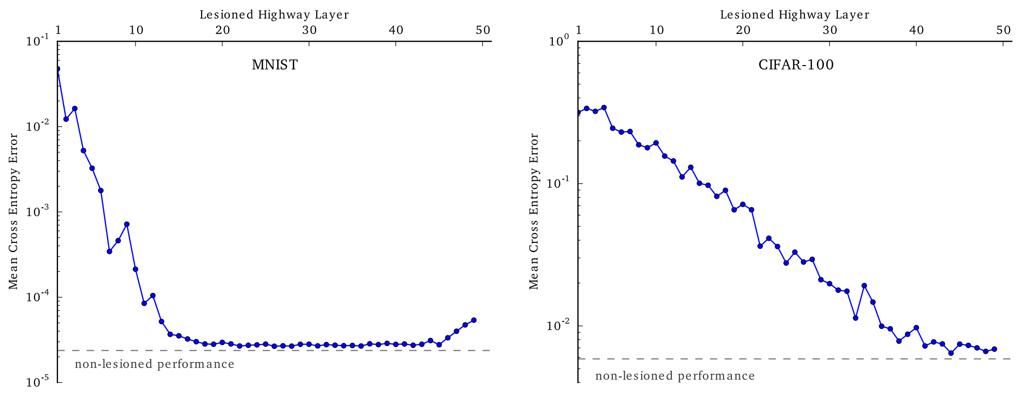


Figure 6.4. Lesioned training set loss (y-axis) of the best 50-layer highway networks on MNIST (left) and CIFAR-100 (right), as a function of the lesioned layer (x-axis). Evaluated on the full training set while forcefully closing all the transform gates of a single layer at a time. The non-lesioned performance is indicated as a dashed line at the bottom.

the early layers is removed, but layers 15 — 45 seem to have close to no effect on the final performance. About 60% of the layers do not learn to contribute to the final result, likely because MNIST is a simple dataset that does not require much depth.

A different picture emerges for the CIFAR-100 dataset (right) with performance degrading noticeably when removing any of the first ≈ 40 layers. This suggests that for complex problems a Highway network can learn to utilize all of its layers, while for simpler problems like MNIST it will keep many of the unneeded layers idle. Such behavior is desirable for deep networks in general, but appears difficult to obtain using plain networks.

6.5 Comparison of Highway Network Variants

Many possible variants of the Highway network architecture can be constructed as discussed in section 6.2, but all experiments in this chapter so far were performed with the coupled Highway architecture. It is natural to question whether the differences in these variants are substantial enough to lead to significantly different results. In general our understanding of the computations required to solve complex problems is limited, so it is extremely hard to say *a priori* which architecture variant may be more suitable for which types of problems. Therefore, in this section we perform two sets of experiments to compare and contrast the

Table 6.4. Comparison between Highway network variants trained for ImageNet 2012 object recognition. Top-5 classification errors are reported based on 3 runs per network. BN indicates use of Batch Normalization [Ioffe and Szegedy, 2015].

Variant	Top-5 Error
Coupled Highway	10.03 ± 0.17
Coupled Highway-ReL	10.21 ± 0.03
Resnet	9.40 ± 0.18
Coupled Highway + BN	7.53 ± 0.05
Coupled Highway-ReL + BN	7.29 ± 0.11
Resnet + BN	7.17 ± 0.14

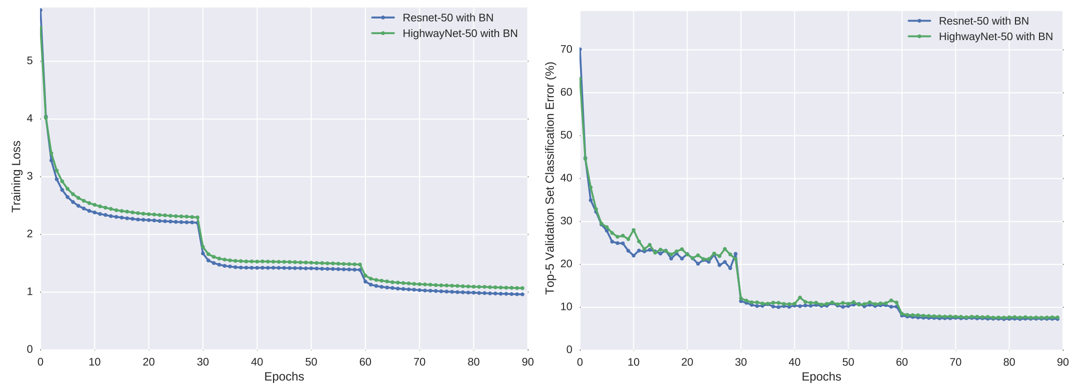
performance of Highway variants in two different domains.

6.5.1 Image Classification

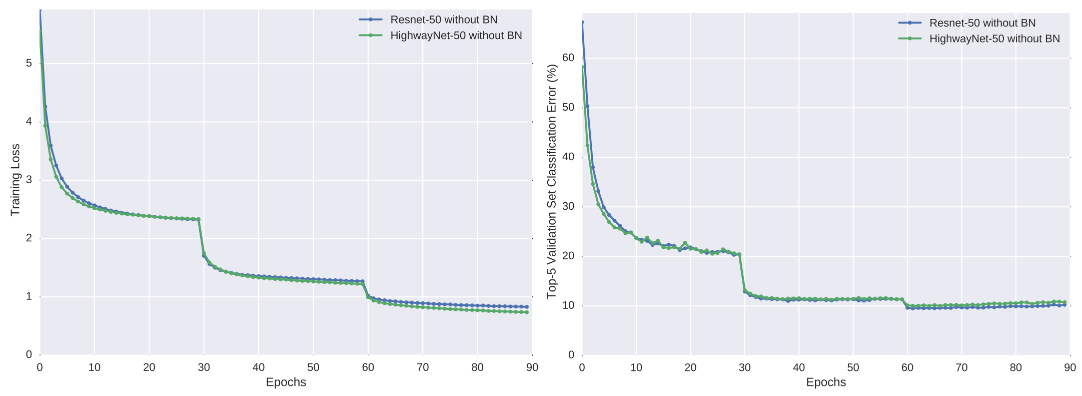
He et al. [2016a] extensively developed and studied the Residual variant of the Highway networks, and used it to outperform all other entries at the 2016 ImageNet challenge. In this set of experiments, the goal is to carefully compare two variants: Residual and coupled Highway. Through these comparisons, it also becomes possible to examine the validity of the following claims regarding coupled Highway networks made in follow-up work [He et al., 2016a,b; Veit et al., 2016]:

1. They are harder to train compared to Residual networks, resulting in stalled training or poor results.
2. They require extensive tuning of the initial bias, and even then produce much worse results compared to Residual networks.
3. They are wasteful in terms of parameters since they utilize extra learned gates, doubling the total parameters *for the same number of units* compared to a Residual networks.

Residual and coupled Highway CNNs with 50 layers each were trained for comparison, following the general model architecture proposed by He et al. [2016a].The design of the two networks were similar (including use of batch normalization (BN) [Ioffe and Szegedy, 2015]), except for the use of coupled Highway blocks instead of Residual blocks, where a block is defined as a chain of following



(a) with batch normalization



(b) without batch normalization

Figure 6.5. Comparing 50-layer Highway vs. Residual networks on ImageNet 2012 classification.

layers: `Convolution-BN-ReLConvolution-BN-ReLConvolution-BN` by He et al. [2016a]. Note that in this notation, the ReL activation function is treated as a layer. To implement the corresponding coupled Highway architecture, two chains of the same operations were used to implement H and T and then combined them using the coupled Highway formulation. An additional coupled Highway CNN called *Highway-ReL* was trained which was slightly different: it used an additional ReL layer at the end of the block in H. The block design for T was `Conv-BN-ReLU-Conv-BN-ReLUConv-BN-Sigmoid` in both Highway networks. Convolution layers in both H and T used the same receptive fields and number of parameters. The transform gate biases were set to -1 at the start of training. For fair comparison, the number of feature maps in convolutional layers throughout the Highway network was reduced such that the total number of parameters is similar to the Residual network. The training algorithm and learning rate schedule were kept the same as those used for the Residual network.

The plots in Figure 6.5a show that the Residual network fits the data better—its final training loss is lower than the coupled Highway network. Table 6.4 shows that the final performance of all networks on the validation set is very similar, with the Residual network producing a slightly better top-5 classification error of 7.17% vs. 7.53% for the *Highway* network. The *Highway-Full* network produces a slightly better mean error of 7.29%. These results contradict claims 1 and 2 above, since the Highway networks could be trained easily without requiring any bias tuning. However, there is some support for claim 3 since the coupled Highway network appears to slightly underfit compared to the Residual network, suggesting lower capacity for the same number of parameters.

Importance of Expressive Gating. The mismatch between the results above and claims 1 and 2 made by He et al. [2016b] can be explained based on the importance of having sufficiently expressive transform gates. For experiments with Highway networks (which they refer to as *Residual networks with exclusive gating*), He et al. [2016b] used 1×1 convolutions for the transform gate, instead of having the same receptive fields for the gates as the plain transformation H. This change in design appears to be the primary cause of instabilities in learning since the gates can no longer function effectively. Therefore, this experiment underscores the importance of using equally expressive transformations for H and T in Highway networks.

Role of Batch Normalization. Since all Highway variants have allow improved credit assignment by design compared to plain networks, it is interesting to inquire as to the necessity of batch normalization for training the above networks. To investigate this, the networks were retrained without batch normalization, resulting in the training curves shown in Figure 6.5b.

Without batch normalization, both networks reach an even lower training error than before but perform worse on the validation set, indicating increased overfitting. Interestingly, the effect is more pronounced for the coupled Highway network, which now fits the data better than the Residual variant. This contradicts claim 3, since a Highway network with the same number of parameters as a Residual network demonstrates slightly higher capacity. On the other hand both networks produce a higher validation error – 10.03% and 9.40% for the Highway and Residual network respectively –indicating a clear case of overfitting. This establishes that batch normalization is not necessary for training these networks and does not speed up learning, but is a powerful regularizer.

Table 6.5. Comparison of various Highway variants for character-aware neural language models [Kim et al., 2015] on the Penn Treebank dataset.

Variant	Functional Form	Perplexity
Plain	$H(\mathbf{x})$	92.60
Residual	$H(\mathbf{x}) + x$	91.32
T-Only	$H(\mathbf{x}) \cdot T(\mathbf{x}) + \mathbf{x}$	82.94
Multiplicative-skip	$\mathbf{x} \cdot C(\mathbf{x})$	79.45
C-Only	$H(\mathbf{x}) + \mathbf{x} \cdot C(\mathbf{x})$	79.15
Coupled	$H(\mathbf{x}) \cdot T(\mathbf{x}) + \mathbf{x} \cdot (1 - T(\mathbf{x}))$	79.13
Full	$H(\mathbf{x}) \cdot T(\mathbf{x}) + \mathbf{x} \cdot C(\mathbf{x})$	79.09

6.5.2 Language Modeling

In this set of experiments, different Highway variants are compared on language modeling. Kim et al. [2015] proposed a novel model for this task and showed that utilizing Highway fully connected layers within it instead of plain layers improved performance for a variety of languages. Their final language model consisted of a stack of convolutional layers followed by Highway layers and then an LSTM layer which predicted the next word in a text dataset based on the words seen so far. Notably, they reported that only two to four Highway layers were necessary to obtain significant modeling improvements, so the central advantage of using Highway layers was not that they eased credit assignment over depth. Instead, it appears that the structure of these layers introduces a favorable modeling bias for this task. Following the study by Kim et al. [2015], similar models that incorporate coupled Highway layers have been used to obtain significant improvements in large-scale language modeling [Jozefowicz et al., 2016] and character level machine translation [Lee et al., 2016].

To empirically compare how variants of Highway networks perform as part of the model proposed, several models, only differing in the Highway layer variant used, were trained on the Penn Treebank dataset (PTB; Marcus et al., 1993) using the setup and code made available by the authors. Specifically, the "LSTM-Char-Large model" configuration was used, only changing the two coupled Highway layers to different variants. The performance of each model variant, measured in terms of the perplexity on the test set, is shown in Table 6.5.

The Full, Coupled, C-Only, and Multiplicative-skip variants have similar performance, which is better than the T-Only variant and significantly better than the Residual variant. The Residual variant results in poor performance, close to that obtained by using a single plain layer, even though four Residual layers were used in order to match the amount of parameters used by other variants. A useful result is that the Multiplicative-skip variant performs almost as well as the coupled variant, since this reduces the number of the parameters and computations in the fully connected layers by half. Overall, the variation in performance across variants on this task highlights that no single variant can be expected to be the best choice for any specific task.

6.6 Discussion

Highway networks are the culmination of a series of ideas proposed in the past to improve credit assignment in NNs. On one hand, like locally competitive networks discussed in the previous chapters and mixtures of experts [Jacobs et al., 1991b], they automatically direct different input patterns to different subnetworks, reducing cross-pattern interference and improving credit assignment. From this perspective they are models of models, similar to locally competitive networks, as described in Chapter 5. On the other hand, their similarity to LSTM RNNs provides resistance to vanishing gradients and enables the training of very deep networks that can not be trained otherwise.

Highway networks also continue a long history of feedforward architectures with skip connections (subsection 3.1.6), but with some important differences. The skip connections used by Highway layers do not connect all units in a lower layer to all units in a higher layer, and do not have learnable weights that remain fixed after training, unlike past work. Instead, they only connect the output of the i^{th} unit in one layer to the output of the i^{th} unit in the next layer, where $i \in \{1, 2, \ldots N\}$, and N is the number of units in each of the layers. These connections are modulated by learned gating units, which produce different outputs for different input patterns.

Due to the ability of unit outputs in Highway networks to skip layers, a possible limitation is that many units might remain unused and the network capacity may be underutilized. However, experiments in this chapter demonstrate that this possibility does not manifest in practice, and the propagation of error signals through the network is sufficient to train the units to perform useful computations. For instance, deep Highway networks with fewer parameters match or exceed the accuracy of shallower maxout networks on test data, which would not be possible if the units were underutilized.

Moreover, the unique structure of Highways layers can be exploited to directly evaluate the contribution of each layer as done in Figure 6.4. Through the same analysis, Highway networks allow us to examine how much computation depth may be needed for a given problem. Such a direct analysis can not be performed with plain NNs since removing any layer completely disrupts their computations.

Simply redesigning NNs to improve credit assignment, as shown in this chapter, has another positive side effect: Highway layers can be used as building blocks in a variety of contexts and problem domains. While only feedforward architectures were used in this chapter, the next chapter explores how they can be used to address an important trainability challenge in RNNs that prevents them from fully benefiting from the potential efficiency of deep transformations.

Chapter 7

Recurrent Highway Networks

Simple RNNs struggle to model long-term dependencies due to fundamental difficulties which result from their design: the vanishing and exploding gradient problems (subsection 3.1.5). LSTM RNNs overcome this difficulty by changing the architecture, but the recurrent transition function in LSTM – which maps the previous state to the next – does not take advantage of depth. Therefore, if there are complex dependencies between data at nearby time-steps, LSTM RNNs may not be efficient. Simple RNNs can be modified to utilize depth in the recurrent transition [Pascanu et al., 2013a], but this makes training them extremely hard and unreliable, even when increasing the depth by a single layer.

In this chapter, we overcome this difficulty with the help of Highway networks, and develop *Recurrent Highway Networks* (RHNs), which remain easy to train even with tens of layers in the recurrent transition. Benefiting from increased modeling power, deeper RHNs obtain better results than shallower ones for the same number of parameters, and set the state-of-the-art on two challenging character prediction benchmarks: `enwik8` and `text8`.

7.1 Recurrence Depth in RNNs

When an RNN processes a sequence with T time-steps, it induces a deep function of depth T. This is the motivation behind the use of backpropagation through time for training RNNs – it can be treated as a FNN with T layers – and makes it valid to say that RNNs are deep "in time".

Like feedforward layers, recurrent layers can also be connected in a linear graph (or *stack*) [Schmidhuber, 1992] to produce a deep RNN. Such an RNN is said to be deep in two dimensions, space (due to stacking) **and** time, and requires proper credit assignment across both of them. Increased depth from stacking layers enables these deep RNNs to be more powerful without substantial increase in computational complexity similar to deep FNNs, and introduces corresponding challenges in training. Since Highway layers are designed to address the difficulties of training as more and more layers are stacked, researchers have recently used variants of them to also train deep stacked RNNs [Zhang et al., 2016; Kalchbrenner et al., 2015].

However, stacking layers is not the only way in which RNNs can profit from increased depth. Consider the depth of the recurrent transition function that computes a simple RNN layer's output $\hat{\mathbf{y}}^{[t+1]}$ at time $t+1$ from the previous output $\hat{\mathbf{y}}^{[t]}$ at time t (Equation 2.6). Even though the computation is deep in time when the RNN processes a long sequence, the depth of the above function, which we call the *recurrence depth*, is still one. This implies that a single layer NN models the relationships between adjacent time-steps of the sequence, which may be extremely complex for real world data.

Pascanu et al. [2013a] proposed to increase the recurrence depth by adding multiple non-linear layers to the recurrent transition, resulting in Deep Transition RNNs (DT-RNNs) and Deep Transition RNNs with Skip connections (DT(S)-RNNs). While being powerful in principle, these architectures are seldom used due to exacerbated gradient propagation issues resulting from extremely long credit assignment paths. In related work, Chung et al. [2015] added extra connections between all states across consecutive time-steps in a stacked RNN, which also increases recurrence depth. However, their model required many extra connections with increasing depth, gave only a fraction of units access to the largest depth, and still faced gradient propagation issues along the longest paths.

A general method to increase the recurrence depth is to let an RNN "tick" for several *micro time-steps* per time step of the sequence [Schmidhuber, 1991; Srivastava et al., 2013b; Graves, 2016], and learn the optimal number of micro time-steps from the data itself. This method can adapt the recurrence depth to the problem, but the RNN has to learn by itself which parameters to use for memories of previous events and which for standard deep nonlinear processing. Moreover, learning the optimal number of micro time-steps is difficult due to non-differentiability. Graves [2016] proposed an approach to tackle the non-differentiability with a smooth approximation and reported improvements on simple algorithmic tasks, but did not obtain performance improvements on real world datasets.

Compared to stacking recurrent layers, increasing the recurrence depth can add significantly higher modeling capacity to an RNN. Figure 7.1 illustrates that stacking d RNN layers allows a maximum credit assignment path length (number of non-linear transformations) of $\mathrm{d}-1+\mathrm{T}$ between hidden states which are T

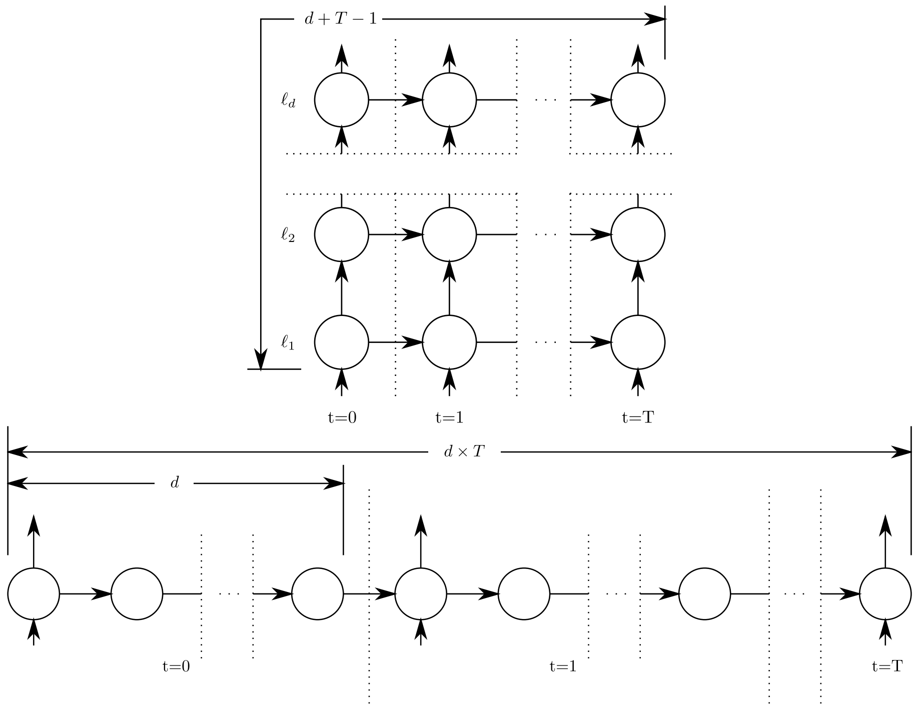


Figure 7.1. Comparison of stacked RNN with depth $d = 2$ (top) and Deep Transition RNN of recurrence depth $d = 2$ (bottom) processing a sequence of length T. Long and dotted vertical lines demarcate time-steps. Within each time-step, each circle denotes a layer. The sequential processing in both RNNs shows that the longest credit assignment path between hidden states T time-steps apart are longer in the latter architecture ($d \times$ T) compared to the first $d +$ T $- 1$.

time-steps apart, while a recurrence depth of d enables a maximum path length of d$\times$T. This allows greater power and efficiency using larger recurrence depths, but it also explains why this makes training extremely difficult compared to stacked RNNs.

7.2 Architecture

Recall that the recurrent state transition in a standard RNN is described by $\mathbf{y}^{[t]} = f(\mathbf{W}\mathbf{x}^{[t]} + \mathbf{R}\mathbf{y}^{[t-1]} + \mathbf{b})$. We propose to construct a Recurrent Highway Network (RHN) layer with one or multiple Highway layers in the recurrent state transition (equal to the desired recurrence depth). Formally, let $\mathbf{W}_{H,T,C} \in \mathbb{R}^{n \times m}$ and $\mathbf{R}_{H_\ell,T_\ell,C_\ell} \in \mathbb{R}^{n \times n}$ represent the weights matrices of the H nonlinear transform and the T and C gates at layer $\ell \in \{1, \ldots, L\}$. The biases are denoted by $\mathbf{b}_{H_\ell,T_\ell,C_\ell} \in \mathbb{R}^n$

and let $\mathbf{s}_\ell$ denote the intermediate output at layer ℓ with $\mathbf{s}_0^{[t]} = \mathbf{y}^{[t-1]}$. Then an RHN layer with a recurrence depth of L is described by

$$\mathbf{s}_\ell^{[t]} = H(\mathbf{s}_{\ell-1}^{[t]}) \cdot T(\mathbf{s}_{\ell-1}^{[t]}) + \mathbf{s}_{\ell-1}^{[t]} \cdot C(\mathbf{s}_{\ell-1}^{[t]}), \tag{7.1}$$

where

$$H(\mathbf{s}_{\ell-1}^{[t]}) = tanh(\mathbf{W}_H \mathbf{x}^{[t]} \mathbb{I}_{\{\ell=1\}} + \mathbf{R}_{H_\ell} \mathbf{s}_{\ell-1}^{[t]} + \mathbf{b}_{H_\ell}), \tag{7.2}$$
$$T(\mathbf{s}_{\ell-1}^{[t]}) = \quad \sigma(\mathbf{W}_T \mathbf{x}^{[t]} \mathbb{I}_{\{\ell=1\}} + \mathbf{R}_{T_\ell} \mathbf{s}_{\ell-1}^{[t]} + \mathbf{b}_{T_\ell}), \tag{7.3}$$
$$C(\mathbf{s}_{\ell-1}^{[t]}) = \quad \sigma(\mathbf{W}_C \mathbf{x}^{[t]} \mathbb{I}_{\{\ell=1\}} + \mathbf{R}_{C_\ell} \mathbf{s}_{\ell-1}^{[t]} + \mathbf{b}_{C_\ell}), \tag{7.4}$$

and $\mathbb{I}_{\{\}}$ is the indicator function.

A schematic illustration of the RHN computation graph is shown in Figure 7.2. The output of the RHN layer is the output of the L^{th} Highway layer i.e. $\mathbf{y}^{[t]} = \mathbf{s}_L^{[t]}$. Note that $\mathbf{x}^{[t]}$ is directly transformed only by the first Highway layer ($\ell = 1$) in the recurrent transition[1] and for this layer $\mathbf{s}_{\ell-1}^{[t]}$ is the RHN layer's output of the previous time step. Subsequent Highway layers only process the outputs of the previous layers. Dotted vertical lines in Figure 7.2 separate multiple Highway layers in the recurrent transition.

For conceptual clarity, it is important to observe that an RHN layer with $\text{L} = 1$ is essentially a basic variant of an LSTM layer. Similar to other variants such as GRU [Cho et al., 2014] and those studied by Greff et al. [2017a] and Jozefowicz et al. [2015], it retains the essential components of the LSTM – multiplicative gating units controlling the flow of information through self-connected additive cells. However, an RHN layer naturally extends to $\text{L} > 1$, extending the LSTM to model far more complex state transitions. Similar to Highway and LSTM layers, other variants can be constructed without changing the basic principles, for example by fixing one or both of the gates to always be one.

7.3 Experiments

Experiments were performed on four datasets of discrete symbolic sequences: the JSB Chorales polyphonic music prediction dataset [Boulanger-Lewandowski et al., 2012], the Penn Treebank [Marcus et al., 1993] word-level language modeling, and the Hutter Prize Wikipedia datasets: `text8` and `enwik8` [Hutter, 2012]. All tasks consisted of training models for next step prediction. On JSB Chorales, the goal was to perform an optimization stress test similar to subsection 6.3.2. On

[1]This is not strictly necessary, but simply a convenient choice.

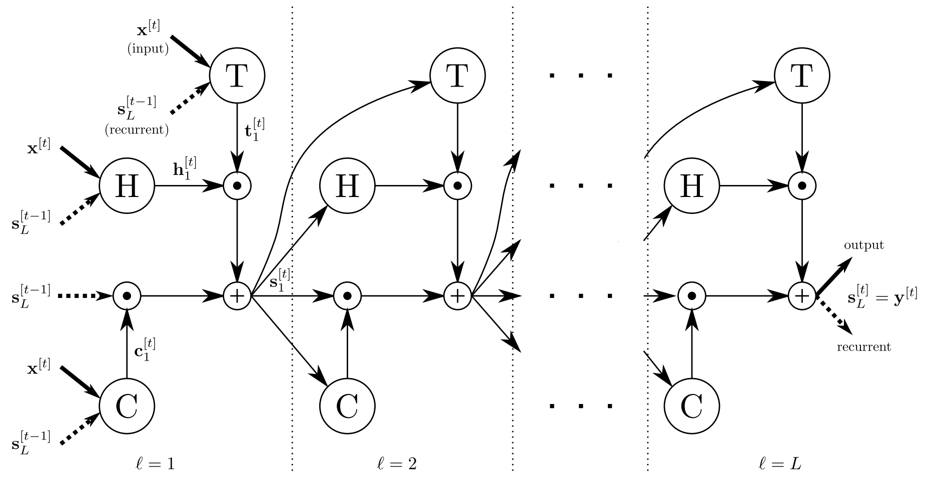


Figure 7.2. Schematic showing computation within an RHN layer inside the recurrent loop. Vertical dashed lines delimit stacked Highway layers. Horizontal dashed lines imply the extension of the recurrence depth by stacking further layers. H, T & C are the transformations described in equations 7.2, 7.3 and 7.4, respectively.

Penn Treebank, the primary goal was to understand the benefits of increasing the recurrence depth while keeping model size constant, i.e, utilizing the same number of paramters for increasing depth instead of width of the layers.

Setup: For all experiments, the carry gate was coupled to the transform gate by setting $C(\cdot) = \mathbf{1} - T(\cdot)$ as in Chapter 6. Note that a similar coupling was also used by the Gated Recurrent Unit (GRU) architecture [Cho et al., 2014], a variant of LSTM. It reduces model size for a fixed number of units and prevents an unbounded blow-up of state values since each Highway layers output $\mathbf{s}_\ell^{[t]}$ is a convex combination of $H(\mathbf{s}_{\ell-1}^{[t]})$ and $\mathbf{s}_{\ell-1}^{[t]}$. This stabilizes training, but imposes a modeling bias which may be sub-optimal for certain tasks. For optimization and Wikipedia experiments, the transform gates were biased towards being closed at the start of training, again similar to the bias initialization used for Highway networks in Chapter 6. All networks used a single RHN layer since we are only interested in studying the influence of recurrence depth, and not of stacking multiple layers. Source code for the experiments in this chapter is available at `https://github.com/julian121266/RecurrentHighwayNetworks`.

Regularization of RHNs: Like all RNNs, we found suitable regularization of RHNs to be essential for obtaining good generalization. The regularization technique proposed by Gal [2015] was used, which is an interpretation of dropout

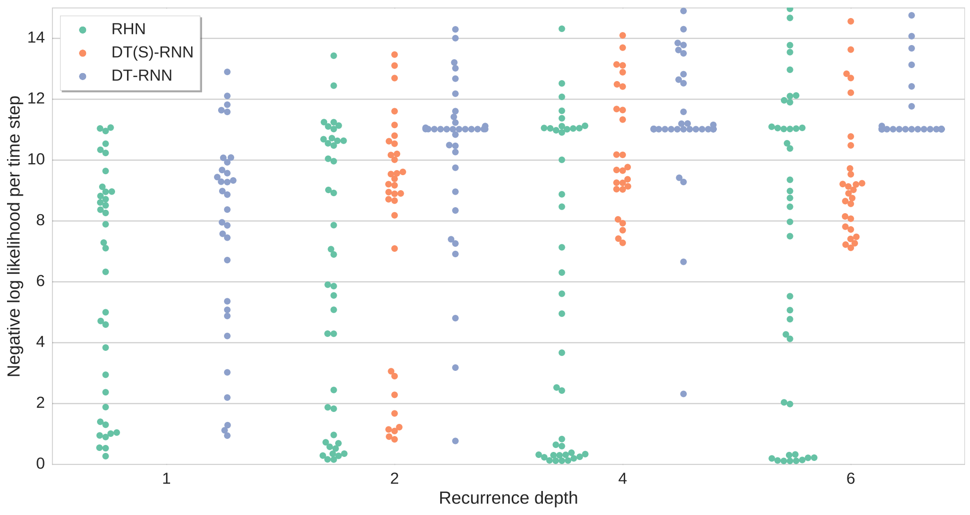


Figure 7.3. Swarm plot of optimization experiment results for various architectures for different depths on next step prediction on the JSB Chorales dataset. Each point is the result of optimization using a random hyperparameter setting. The number of network parameters increases with depth, but is kept the same across architectures for each depth. For architectures other than RHN, the random search was unable to find good hyperparameters when depth increased.

based on approximate variational inference. Hence, RHNs regularized by this technique are referred to as variational RHNs. For the word-level language modeling on the Penn Treebank dataset, results are reported both with and without Weight Tying (WT) regularization: a technique specificially proposed for regularizing language modeling RNNs by sharing the same set of weights between the network's input and output layers Press and Wolf [2016]; Inan and Khosravi [2016].

7.3.1 Optimization Stress Test

RHN is an architecture designed to enable the optimization of recurrent networks with deep transitions. Therefore, the primary experimental verification required is whether RHNs with high recurrence depths are easier to optimize compared to other alternatives, preferably using conventional gradient based methods. In this section, optimization of RHNs is compared to DT-RNNs and DT(S)-RNNs [Pascanu et al., 2013a].

Networks with recurrence depth of 1, 2, 4 and 6 were trained for next step prediction on the JSB Chorales dataset. The number of Highway units in each

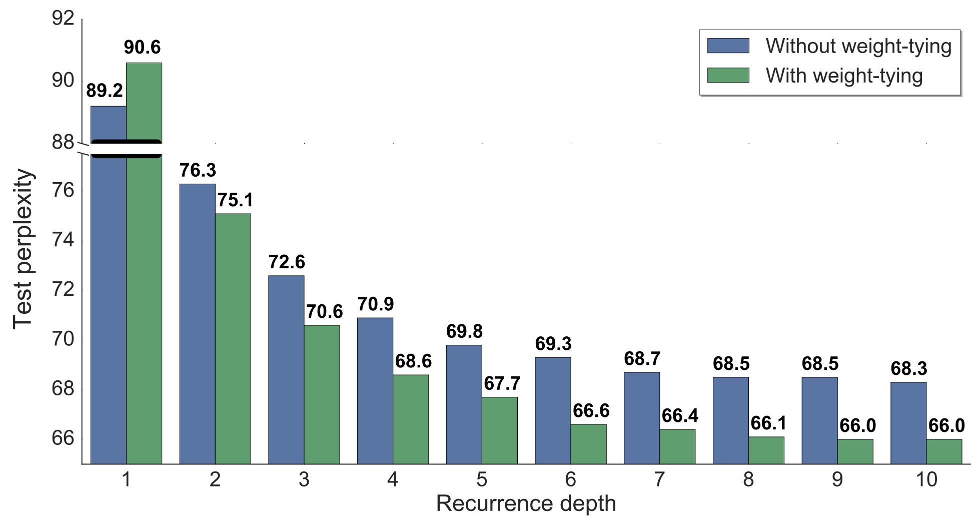


Figure 7.4. Test set perplexity on Penn Treebank word-level language modeling using RHNs with fixed number of total parameter and increasing recurrence depth. Increasing the depth improved performance up to 9 layers.

layer in the recurrent transition was equal and set to $\{1.5 \times 10^5, 3 \times 10^5, 6 \times 10^5, 9 \times 10^5\}$. Optimization was performed using batch gradient descent with momentum equal to 0.9. The batch size was set to 32 and training for a maximum of 1000 epochs was performed, stopping early if the loss did not improve for 100 epochs. $tanh(\cdot)$ was used as the activation function for the nonlinear layers. 60 random hyperparameter settings sampled uniformly were then evaluated for each architecture and depth. The initial transform gate bias was sampled from $\{0, -1, -2, -3\}$, the initial learning rate from $[10^0, 10^{-4}]$ using the logarithmic scale. Finally, all weights were initialized using a Gaussian distribution with standard deviation sampled uniformly on the logarithmic scale from $[10^{-2}, 10^{-8}]$.

In general, larger networks should reach the minimum objective function value on the training set, or at least a lower value compared to smaller networks. However, the swarm plot in Figure 7.3 shows that both DT-RNN and DT(S)-RNN become considerably harder to optimize with increasing depth, similar to plain feedforward networks. It also shows that the optimization of RHNs did not become more difficult as depth increased, indicating that RHNs are a viable choice for constructing powerful and efficient deep sequence models that can be trained reliably.

Table 7.1. Validation and test set perplexities of the best RHNs obtained, compared to those reported for state of the art word-level language models on the Penn Treebank dataset. The model from Kim et al. [2015] uses feed-forward Highway layers to transform a character-aware word representation before feeding it into LSTM layers. *dropout* indicates the regularization used by Zaremba et al. [2014] which was applied to only the input and output of recurrent layers. *Variational* refers to the dropout regularization from Gal [2015] based on approximate variational inference. WT indicates Weight Tying regularization Press and Wolf [2016]; Inan and Khosravi [2016]. RHNs with large recurrence depth achieve highly competitive results and are highlighted in bold.

Model	**Size**	**Best Val.**	**Test**
RNN-LDA + KN-5 + cache [1]	9 M	–	92.0
Conv.+Highway+LSTM+dropout [2]	19 M	–	78.9
LSTM+dropout [3]	66 M	82.2	78.4
Variational LSTM [4]	66 M	77.3	75.0
Variational LSTM + WT [5]	51 M	75.8	73.2
Pointer Sentinel-LSTM [6]	21 M	72.4	70.9
Variational LSTM + WT + augmented loss [7]	51 M	71.1	68.5
Variational RHN	**32 M**	**71.2**	**68.5**
Neural Architecture Search with base 8 [8]	32 M	–	67.9
Variational RHN + WT	**23 M**	**67.9**	**65.4**
Neural Architecture Search with base 8 + WT [8]	25 M	–	64.0
Neural Architecture Search with base 8 + WT [8]	54 M	–	62.4

[1] [Mikolov and Zweig, 2012]
[2] [Kim et al., 2015]
[3] [Zaremba et al., 2014]
[4] [Gal, 2015]
[5] [Press and Wolf, 2016]
[6] [Merity et al., 2016]
[7] [Inan et al., 2016]
[8] [Zoph and Le, 2016]

7.3.2 Sequence Modeling

Penn Treebank

To evaluate whether increasing recurrence depth can result in more powerful models without increasing model size, RHNs with fixed total number of parameters (32 M) and recurrence depths ranging from 1 to 10 were trained for word-level language modeling on the Penn TreeBank dataset [Marcus et al., 1993]. This resulted in RHNs with number of units per Highway layer ranging from 1275 to 830 units. Each network had a single linear layer that reduced the dimension of the inputs, followed by a single hidden RHN layer and a softmax output layer with 10 K outputs (the vocabulary size). All sentences were truncated to a length of 35 as done by Zaremba et al. [2014]. Batch size for gradient descent was fixed to 20. The learning rate started at a value of 0.2, and was decayed by a factor of 0.98 starting at 20 epochs. L2 regularization with coefficient 10^{-7} was added to the objective function. During training, the gradients were rescaled whenever their norm exceeded a value of 10 [Pascanu et al., 2013b]. Dropout rates were chosen to be 0.25 for the linear layer, 0.75 for the input to the RHN gates, 0.25 for the hidden units and 0.75 for the outputs. All weights were initialized from a uniform distribution on $[-0.04, 0.04]$. It was found that for the best 10-layer model obtained, lowering the weight decay to 10^{-9} further improved results.

For each depth, the test set perplexity of the best model based on performance on the validation set is shown in Figure 7.4. The results for each model trained again with WT regularization [Inan and Khosravi, 2016] are also shown, which further reduces the total number of parameters. In both cases, the test score improves as the recurrence depth increases from 1 to 10, dramatically at first, then leveling out at nine to ten layers. Note that as the recurrence depth increased from 1 to 10 layers, the “width" of the network decreased since the number of parameters was kept fixed. Thus, these results demonstrate that even for small datasets utilizing parameters to increase depth can yield much greater benefits than increasing width. Table 7.1 compares the obtained result to the best published results on this dataset. RHNs outperform most single models as well as all previous ensembles, and also benefit from the recently proposed WT regularization similar to LSTMs. The only model that outperforms RHNs was automatically discovered through a large scale architecture learning technique by Zoph and Le [2016].

Table 7.2. Entropy in Bits Per Character (BPC) on the `enwik8` test set (results under 1.5 BPC & without dynamic evaluation). LN refers to the use of layer normalization [Ba et al., 2016].

Model	**BPC**	**Size**
Grid-LSTM [Kalchbrenner et al., 2015]	1.47	17 M
MI-LSTM [Wu et al., 2016]	1.44	17 M
mLSTM [Krause et al., 2016]	1.42	21 M
LN HyperNetworks [Ha et al., 2016]	1.34	27 M
LN HM-LSTM [Chung et al., 2016]	1.32	35 M
RHN - Rec. depth 5	**1.31**	**23** M
RHN - Rec. depth 10	**1.30**	**21** M
Large RHN - Rec. depth 10	**1.27**	**46** M

Table 7.3. Entropy in Bits Per Character (BPC) on the `text8` test set (results under 1.5 BPC & without dynamic evaluation). LN refers to the use of layer normalization [Ba et al., 2016].

Model	**BPC**	**Size**
MI-LSTM [Wu et al., 2016]	1.44	17 M
mLSTM [Krause et al., 2016]	1.40	10 M
BN LSTM [Cooijmans et al., 2016]	1.36	16 M
HM-LSTM [Chung et al., 2016]	1.32	35 M
LN HM-LSTM [Chung et al., 2016]	1.29	35 M
RHN - Rec. depth 10	**1.29**	**20** M
Large RHN - Rec. depth 10	**1.27**	**45** M

`enwik8` & `text8` Datasets

This experiment evaluated RHNs for the next symbol prediction task on the challenging Hutter Prize Wikipedia datasets: `text8` and `enwik8` [Hutter, 2012]. These text datasets have 27 and 205 unicode symbols in total, respectively. Due to their size (100 M characters in total) and complexity (inclusion of Latin/non-Latin alphabets, XML markup and various special characters for `enwik8`) these datasets stress the learning and generalization capacity of RHNs.

Variational RHNs with recurrence depths of 5 or 10, and 1000 or 1500 units per hidden layer, were trained on both datasets. Similar to experiments on Penn Treebank, gradient scaling and L2 regularization with a coefficient of 10^{-7} was used, and the network consisted of a linear layer to project the inputs down to fewer dimensions, followed by a single RHN layer, followed by a softmax output layer. The size of the linear layer was the same as the size of the symbol set. For batch gradient descent, an initial learning rate of 0.2 and a learning rate decay factor of 0.96 was used. The learning rate was multiplied by the decay factor, after each epoch, starting after 5 epochs of trainings. For the largest models with 10 stacked layers and 1500 units the decay factor was 0.97. Weights were initialized uniformly from the range $[-0.04, 0.04]$ and an initial bias value of -4 was set for the transform gates to facilitate learning early in training. Dropout probabilities were selected using a grid search over values in $\{0.05, 0.1\}$ for the linear layer and the RHN hidden units, and $\{0.3, 0.4, 0.5\}$ for the output layer and RHN inputs. The performance of the models was measured in terms of entropy of its predictions on the test set in Bits Per Character (BPC).

On `text8` a validation/test set score of **1.19/1.27** BPC for a model with 1500 units and recurrence depth 10 is achieved. Similarly, on `enwik8` a validation/test set score of **1.26/1.27** BPC is achieved for the same model and hyperparameters. Table 7.2 and Table 7.3 show that RHNs outperform all published models on `text8` and `enwik8` with significantly fewer total parameters.

7.4 Analysis

RHNs are built using Highway layers, so an analysis of trained networks similar to that in section 6.4, was performed examine their behavior and understand the difference, if any, from their behavior in the feedforward case.

For the RHN with a recurrence depth of six trained on the JSB Chorales dataset in subsection 7.3.1, Figure 7.5 shows the mean transform gate activity in each layer over time for four example sequences (A—D), each 50 time-steps long. Note that although the gates were biased towards zero (white) before training by setting their biases to negative values, the network has learned to activate gates in all the layers through training. The gate activity in the first layer of the recurrent transition is typically high on average, indicating that at least one layer in the transition is almost always utilized. Gates in other layers have varied behavior, dynamically switching their activity over time in a different way for each sequence and time-step. This indicates that the RHN utilizes varying amount of modeling power over time.

Next, the lesioning experimental setup from section 6.4 was used to compare the contributions of the Highway layers in a trained network toward its performance. That is, for one layer at a time, all the gates were pushed towards carry behavior by setting the bias to a large negative value, and the resulting loss was

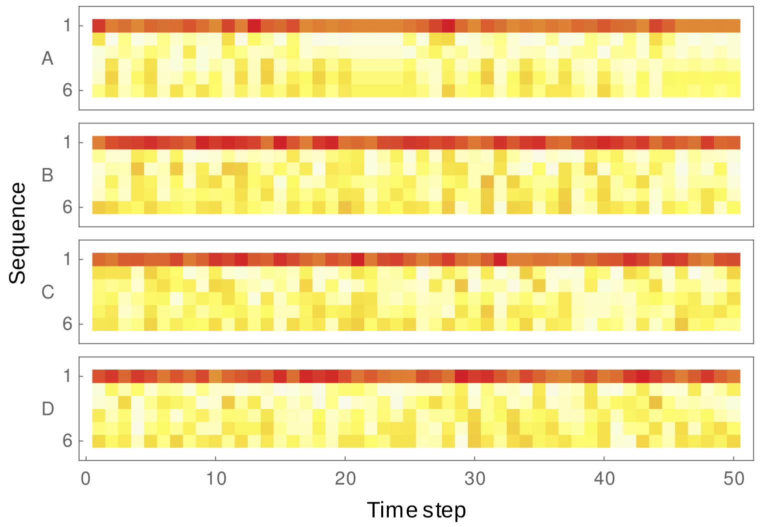


Figure 7.5. Mean activations of the transform (T) gates at different depths (1—6) in a trained RHN for four sequences (A—D) from the JSB Chorales dataset, on which it was trained in subsection 7.3.1. Each sequence is 50 time-steps long. An active transform gate indicates that the recurrence layer is used to process input at a particular time step, as opposed to passing it to the next layer.

measured. Figure 7.6 shows the change in loss due to the lesioning of each layer, and hence its contribution to the network performance. We find that the first layer contributes several times more to the overall performance compared to others. However, unlike the 50-layer lesioned feedforward Highway networks in section 6.4, it is notable that removing any layer hurts the performance substantially. This is likely due to the recurrent nature of the network. Since the same six layers are activated at each time step of each sequence, each layer is likely to have at least a few units that learn useful computations for some subsequences in the dataset.

7.5 Discussion

The work of Pascanu et al. [2013a] showed that RNNs with deep recurrent transitions fall in the category of models that can perform very well on sequence processing tasks, but can not be trained reliably. This limitation was effectively removed by employing Highway networks instead of plain layers in the transition, resulting in the RHN architecture. The extensive experiments in this chapter on challenging benchmarks show that increased recurrence depth can substantially

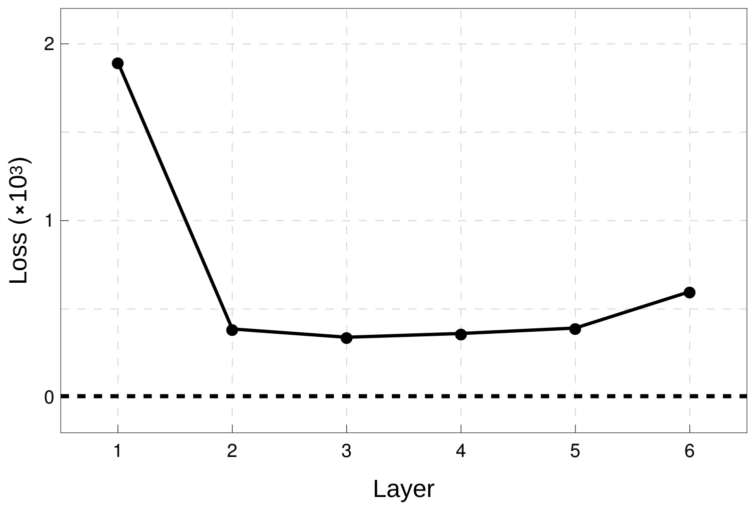


Figure 7.6. Changes in loss when the Highway layers in a trained RHN are biased towards carry behavior (i.e., effectively removed), one layer at a time. The recurrence of the RHN was six, and it was trained on the JSB Chorales dataset for next step prediction (subsection 7.3.1).

improve sequence models without taking a hit in terms of number of parameters.

It should be noted that although increasing depth while decreasing layer width results in improved performance, RHNs with high recurrence depth benefit less from massively parallel computing hardware such as Graphics Processing Units. This is because parallel computations get traded for sequential computations as depth increases. However, this trade-off affects all NN architectures and is in fact desirable in situations where the amount of parallel processing resources are limited.

Similar to the feedforward case, the Highway layers in RHNs perform adaptive computation, i.e., the effective amount of transformation is dynamically adjusted for each sequence and time step. The maximum depth of computation at each time step is limited to the recurrence depth of the RHN layer. In principle, it would be more attractive to not have this limitation and simply learn to repeatedly use an LSTM layer for multiple micro-ticks as needed, but efforts to accomplish this behavior are still in their early stages [Graves, 2016].

Chapter 8

Conclusion

This thesis contributes neural network architectures that permit efficient and reliable training of large and deep NNs. Chapter 3 summarized a long line of previous research that was inspired by similar motivations of improving the trainability of neural networks. Standing on the shoulders of giants, in chapters 4 to 7, a series of network architectures were proposed to overcome two significant hurdles in training powerful NNs: cross-pattern interference and vanishing gradients.

Below, we summarize the proposed architectures, with references to the corresponding publications in which they first appeared. Finally, the thesis concludes with a discussion of future work.

LOCAL WINNER-TAKE-ALL NETWORKS (LWTA) [Srivastava et al., 2013a, 2015b]

LWTA networks represent a unique and effective solution to the problem of cross-pattern interference in NNs. Instead of utilizing per-unit non-linearities, these networks derive their power from simple local competition among groups of units in each layer.

Experiments on digit classification and sentiment analysis tasks demonstrated their strong performance on benchmarks. More importantly, analysis of LWTA revealed implicit local gating of input patterns to specialized sets of computation units, thereby mitigating cross-pattern interference and making training efficient.

While it had been noted previously that the ReL activation function directs input patterns through different linear subnetworks within a network, the extent and utility of this phenomenon had never been investigated. Building upon this observation, we formulated and tested the model of models hypothesis to understand the superior performance of not just networks with the ReL activation function, but also LWTA and maxout networks, and studied their common properties.

A battery of experiments on MNIST, CIFAR-10, CIFAR-100 and ImageNet 2012 datasets confirmed the validity of our hypothesis, underscoring the significance of implicit local gating present in all three types of networks studied.

HIGHWAY NETWORKS [Srivastava et al., 2015a]
Highway networks retain the property of avoiding cross-pattern interference among units in the same layer through the use of explicit gating units, and further enable credit assignment across a large number of layers through skip connections.

Image classification experiments on MNIST, CIFAR-10, and CIFAR-100 with Highway networks demonstrated that training them reliably remained straightforward even as they became increasingly more powerful due to the addition of more layers, while in the past NN became increasingly cumbersome to train in this case. Therefore, they represent a significant advance in the ability to train large and deep NNs.

RECURRENT HIGHWAY NETWORKS (RHNS) [Zilly and Srivastava et al., 2017]
RHNs are an extension of Highway networks to the sequential setting, and successfully address another identified challenge in the field: training of RNN with deep recurrent transitions for modeling complex temporal dependencies efficiently.

Experiments performed with RHNs consistently demonstrated the benefits of addressing the challenge of increased depth in time. On the Penn Treebank dataset for language modeling, increasingly better performance was obtained by RHNs with progressively deeper transitions while the number of parameters remained the same. On the challenging `enwik8` and `text8` benchmarks, the same trend was observed and state of the art results were obtained in terms of both performance and parameter efficiency.

8.1 Future Work

The developments presented in this thesis overcome many hurdles in the training of NNs, but they also open up several new lines of inquiry:

What are the inductive biases of the new architectures?
This question pertains to developing an understanding of the implicit assumptions a practitioner makes when choosing a particular network architecture for a new task. As discussed in section 3.2, choosing an architecture amounts to selecting a prior set of assumptions that affect learning and generalization. This is because

even if the network is capable of representation a huge variety of function types, the information baked into the architecture of a network (and relatedly, its initialization) biases it towards learning certain types of functions before others. For example, we would like to know the types of functions that specific variants of Highway networks are biased towards learning. This knowledge would be both *prescriptive*, since it would enable the selection of the most suitable variant based on an understanding of the task characteristics, and *descriptive*, since it would explain why a particular variant performs the best on one task but not another. We have already made some initial progress in this direction in our own work [Greff et al., 2017b].

What is the effect of these architectures on optimization?
The architectures developed in this thesis were designed to improve credit assignment among units and avoid vanishing gradients during backpropagation. While the results demonstrate that these goals were successfully achieved, a proper understanding of the full loss landscape of NNs is still lacking. In particular, an important topic for investigation is: how does the addition of elements like local competition, gating and skip connections modify the loss landscape of the network? A satisfactory answer to this question would a) explain the success of the proposed architectures, and b) suggest other methods of effecting the same modifications. For recent work in this direction, see Orhan and Pitkow [2017].

How to best utilize their benefits?
Experimental results presented in this thesis already demonstrate substantial improvements in performance or efficiency on many benchmark problems, but an important open question is: which new problems can now be attacked with the help of an expanded set of tools that we could not tackle before? There is also an equally important secondary question: which other techniques can we develop specifically keeping in mind these new developments to solve existing problems even better, or even new set of problems? For example, are there regularization techniques that work particularly well with LWTA or Highway networks? The answers to this questions will be the key to further broadening the impact of the work presented here.

We believe that continued investigations along these lines will lead to further advancements in understanding and successful applications of NNs.

Publications during the PhD program

R. K. Srivastava, B. R. Steunebrink, M. Stollenga, and J. Schmidhuber. Continually adding self-invented problems to the repertoire: First experiments with PowerPlay. In *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference On*, pages 1–6. IEEE, 2012b.

R. K. Srivastava, J. Schmidhuber, and F. Gomez. Generalized Compressed Network Search. In *Parallel Problem Solving from Nature-PPSN XII*, pages 337–346. Springer, 2012a.

R. K. Srivastava, B. R. Steunebrink, and J. Schmidhuber. First Experiments with PowerPlay. *Neural Networks*, 41:130–136, 2013b.

R. K. Srivastava, J. Masci, S. Kazerounian, F. Gomez, and J. Schmidhuber. Compete to compute. In *Advances in Neural Information Processing Systems*, pages 2310–2318, 2013a.

H. Fang*, S. Gupta*, F. Iandola*, **R. K. Srivastava***, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J. Platt, C. L. Zitnick, and G. Zweig. From Captions to Visual Concepts and Back. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

R. K. Srivastava, J. Masci, F. Gomez, and J. Schmidhuber. Understanding Locally Competitive Networks. In *International Conference on Learning Representations*, 2015b.

R. K. Srivastava, K. Greff, and J. Schmidhuber. Training Very Deep Networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2377–2385. Curran Associates, Inc., 2015a.

K. Greff, **R. K. Srivastava**, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 2017a.

K. Greff, **R. K. Srivastava**, and J. Schmidhuber. Highway and residual networks learn unrolled iterative estimation. In *International Conference on Learning Representations*, 2017b.

J. G. Zilly*, **R. K. Srivastava***, J. Koutník, and J. Schmidhuber. Recurrent Highway Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 4189–4198, July 2017.

Bibliography

S. C. Ahalt, A. K. Krishnamurthy, P. Chen, and D. E. Melton. Competitive learning algorithms for vector quantization. *Neural networks*, 3(3):277–290, 1990.

P. Anderson, G. N. Gross, T. Lømo, and O. Sveen. Participation of inhibitory and excitatory interneurones in the control of hippocampal cortical output. In M. A. Brazier, editor, *The Interneuron*, volume 11. University of California Press, Los Angeles, 1969.

D. Angluin and C. H. Smith. Inductive inference: Theory and methods. *ACM Computing Surveys (CSUR)*, 15(3):237–269, 1983.

M. Anthony and P. L. Bartlett. *Neural Network Learning: Theoretical Foundations*. cambridge university press, 2009.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

P. L. Bartlett. The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network. *IEEE transactions on Information Theory*, 44(2):525–536, 1998.

P. L. Bartlett and S. Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov): 463–482, 2002.

S. Becker and Y. LeCun. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 29–37, 1988.

Y. Bengio and O. Delalleau. On the expressive power of deep architectures. In *Algorithmic Learning Theory*, pages 18–36. Springer, 2011.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Y. Bengio, O. Delalleau, and N. L. Roux. The curse of highly variable functions for local kernel machines. In *Advances in Neural Information Processing Systems*, pages 107–114, 2006.

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

M. Bianchini and F. Scarselli. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE Transactions on Neural Networks*, 2014.

C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford university press, 1995.

C. M. Bishop. *Pattern Recognition and Machine Learning*. springer, 2006.

J. Blitzer, M. Dredze, and F. Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. *Annual Meeting-ACL*, 2007. URL `http://acl.ldc.upenn.edu/P/P07/P07-1056.pdf`.

A. Blum and R. L. Rivest. Training a 3-node neural network is NP-complete. In *Advances in Neural Information Processing Systems*, pages 494–501, 1989.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information processing letters*, 24(6):377–380, 1987.

N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *ArXiv e-prints*, June 2012.

K. P. Burnham and D. R. Anderson. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer Science & Business Media, 2003.

G. A. Carpenter and S. Grossberg. The art of adaptive pattern recognition by a self-organising neural network. *Computer*, 21(3):77–88, 1988.

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Gated feedback recurrent neural networks. In *International Conference on Machine Learning*, 2015.

J. Chung, S. Ahn, and Y. Bengio. Hierarchical Multiscale Recurrent Neural Networks. *ArXiv e-prints*, Sept. 2016.

D. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, High Performance Convolutional Neural Networks for Image Classification. In *International Joint Conference on Artificial Intelligence*, pages 1237–1242, 2011.

D. C. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber. Deep neural networks segment neuronal membranes in Electron Microscopy images. In *Advances in Neural Information Processing Systems*, 2012.

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

N. Cohen, O. Sharir, and A. Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pages 698–728, 2016.

R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

T. Cooijmans, N. Ballas, C. Laurent, Ç. Gülçehre, and A. Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

G. Dahl, T. Sainath, and G. Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *International Conference on Acoustics, Speech, and Signal Processing*, 2013. URL `http://www.cs.toronto.edu/gdahl/papers/reluDropoutBN_icassp2013.pdf`.

J. Deng, A. Berg, S. Satheesh, S. Hao, A. Khosla, and F. Li. ImageNet large scale visual recognition competition 2012 (ILSVRC2012). http://www.image-net.org/challenges/LSVRC/2012/, 2012. URL `http://www.image-net.org/challenges/LSVRC/2012/`.

L. Deng and J. Chen. Sequence classification using the High-Level features extracted from deep neural networks. In *International Conference on Acoustics, Speech, and Signal Processing*, 2014. URL `http://research.microsoft.com/pubs/215610/Revised-ICASSP14-DengChen_ChenEdited.docx`.

G. Desjardins, K. Simonyan, and R. Pascanu. Natural neural networks. In *Advances in Neural Information Processing Systems*, pages 2071–2079, 2015.

T. G. Dietterich, B. London, K. Clarkson, and G. Dromey. Learning and inductive inference. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1982.

J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. DeCAF: a deep convolutional activation feature for generic visual recognition. *arXiv:1310.1531 [cs]*, Oct. 2013. URL `http://arxiv.org/abs/1310.1531`.

S. E. Dreyfus. The computational solution of optimal control problems with time lag. *IEEE Transactions on Automatic Control*, 18(4):383–385, 1973.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

J. C. Eccles, M. Ito, and J. Szentágothai. *The cerebellum as a neuronal machine*. Springer-Verlag New York, 1967.

J. Elman. Finding structure in time. *Cognitive science*, 211:1–28, 1990. URL `http://www.sciencedirect.com/science/article/pii/036402139090002E`.

S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems*, pages 524–532, 1990.

K. Fukushima. Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36 (4):193–202, 1980a.

K. Fukushima. Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36 (4):193–202, 1980b.

Y. Gal. A theoretically grounded application of dropout in recurrent neural networks. *arXiv preprint arXiv:1512.05287*, 2015.

F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. *Neural computation*, 12(10):2451–2471, 2000.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, page 249–256, 2010. URL `http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS2010_GlorotB10.pdf`.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP*, volume 15, pages 315–323, 2011.

K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

Y. Gong, S. Kumar, H. A. Rowley, and S. Lazebnik. Learning binary codes for high-dimensional data using bilinear projections. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, page 484–491. IEEE, 2013. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6618913`.

I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1319–1327, 2013a. URL `http://jmlr.org/proceedings/papers/v28/goodfellow13.html`.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT press, 2016.

I. J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio. Pylearn2: a machine learning research library. *arXiv:1308.4214 [cs, stat]*, Aug. 2013b. URL `http://arxiv.org/abs/1308.4214`.

I. J. Goodfellow, M. Mirza, X. Da, A. Courville, and Y. Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. In *International Conference on Learning Representations*, 2014. URL `http://arxiv.org/abs/1312.6211`.

G. J. Goodhill and H. G. Barrow. The role of weight normalization in competitive learning. *Neural Computation*, 6(2):255–269, 1994.

B. Graham. Spatially-sparse convolutional neural networks. *arXiv:1409.6070 [cs]*, Sept. 2014. URL `http://arxiv.org/abs/1409.6070`. arXiv: 1409.6070.

K. Grauman and R. Fergus. Learning binary hash codes for large-scale image search. In *Machine Learning for Computer Vision*, page 49–87. Springer, 2013. URL http://link.springer.com/chapter/10.1007/978-3-642-28661-2_3.

A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-24796-5. URL http://www.springerlink.com/index/10.1007/978-3-642-24797-2.

A. Graves. Adaptive Computation Time for Recurrent Neural Networks. *ArXiv e-prints*, Mar. 2016.

A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *International Conference on Acoustics, Speech, and Signal Processing*, 2013. URL http://arxiv.org/abs/1303.5778.

K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 2017a.

K. Greff, R. K. Srivastava, and J. Schmidhuber. Highway and residual networks learn unrolled iterative estimation. In *arXiv Preprint arXiv:1612.07771*, 2017b.

A. Griewank. *Automatic Differentiation*. Princeton Companion to Applied Mathematics, Nicolas Higham Ed., Princeton University Press /, 2014.

S. Grossberg. Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors. *Biological cybernetics*, 23 (3):121–134, 1976.

R. Grosse and R. Salakhudinov. Scaling up natural gradient by sparsely factorizing the inverse fisher matrix. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2304–2313, 2015.

D. Ha, A. Dai, and Q. V. Le. HyperNetworks. *ArXiv e-prints*, Sept. 2016.

B. Hammer and T. Villmann. Mathematical Aspects of Neural Networks. In *ESANN*, pages 59–72, 2003.

J. B. Hampshire and A. Waibel. The meta-pi network: Building distributed knowledge representations for robust multisource pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(7):751–769, 1992.

J. Håstad. *Computational limitations of small-depth circuits*. MIT press, 1987. URL http://dl.acm.org/citation.cfm?id=SERIES9056.27031.

J. Håstad and M. Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1(2):113–129, 1991. URL http://link.springer.com/article/10.1007/BF01272517.

K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852 [cs]*, Feb. 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016a.

K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016b.

D. O. Hebb. *The Organization of Behavior*. Wiley, New York, 1949.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, and T. N. Sainath. 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.

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

G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580 [cs]*, July 2012b.

S. Hochreiter. *Untersuchungen zu dynamischen neuronalen Netzen*. Masters thesis, Technische Universität München, München, 1991.

S. Hochreiter and J. Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997a.

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

S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the diffculty of learning Long-Term dependencies. 2001. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.24.7321&rep=rep1&type=pdf`.

L. Holmström and P. Koistinen. Using additive noise in back-propagation training. *IEEE Transactions on Neural Networks*, 3(1):24–38, 1992.

L. Holmström, P. Koistinen, and R. J. Ilmoniemi. *Classification of Unaveraged Evoked Cortical Magnetic Fields*. Rolf Nevanlinna Inst., University of Helsinki, 1989.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. URL `http://www.sciencedirect.com/science/article/pii/0893608089900208`.

M. Hutter. The human knowledge compression contest. http://prize.hutter1.net/, 2012.

H. Inan and K. Khosravi. Improved learning through augmenting the loss, 2016.

H. Inan, K. Khosravi, and R. Socher. Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling. *arXiv preprint arXiv:1611.01462*, 2016.

S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, Feb. 2015.

A. G. Ivakhnenko. Polynomial theory of complex systems. *IEEE Transactions on Systems, Man and Cybernetics*, (4):364–378, 1971.

R. A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307, 1988.

R. A. Jacobs, M. I. Jordan, and A. G. Barto. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive Science*, 15(2):219–250, 1991a.

R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991b. URL `http://www.mitpressjournals.org/doi/abs/10.1162/neco.1991.3.1.79`.

K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *Proc. of the ICCV*, pages 2146–2153, 2009.

Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093 [cs]*, June 2014. URL `http://arxiv.org/abs/1408.5093`. arXiv: 1408.5093.

R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. 2015.

R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

J. S. Judd. *Neural Network Design and the Complexity of Learning*. MIT press, 1990.

N. Kalchbrenner, I. Danihelka, and A. Graves. Grid long short-term memory. *CoRR*, abs/1507.01526, 2015. URL `http://arxiv.org/abs/1507.01526`.

B. L. Kalman and S. C. Kwasny. Why tanh: Choosing a sigmoidal function. In *Neural Networks, 1992. IJCNN., International Joint Conference On*, volume 4, pages 578–581. IEEE, 1992.

B. L. Kalman and S. C. Kwasny. High Performance Training of Feedforward & Simple Recurrent Networks. 1994.

B. L. Kalman, S. C. Kwasny, and A. Abella. Decomposing input patterns to facilitate training. In *Proceedings of the World Congress on Neural Networks*, volume 3, pages 503–506, 1993.

Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*, 2015.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-Normalizing Neural Networks. *arXiv preprint arXiv:1706.02515*, 2017.

T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69, 1982.

B. Krause, L. Lu, I. Murray, and S. Renals. Multiplicative LSTM for sequence modelling. *ArXiv e-prints*, Sept. 2016.

A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, Computer Science Department, University of Toronto, 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012. URL `http://books.nips.cc/papers/files/nips25/NIPS2012_0534.pdf`.

K. J. Lang and M. J. Witbrock. Learning to tell two spirals apart. In *Proc. of 1988 Connectionist Models Summer School*, 1988.

K. J. Lang, A. H. Waibel, and G. E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural networks*, 3(1):23–43, 1990.

Y. LeCun. Generalization and network design strategies. *Connectionism in perspective*, pages 143–155, 1989.

Y. LeCun, I. Kanter, and S. A. Solla. Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters*, 66(18):2396, 1991.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=726791`.

Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, May 2015. ISSN 1476-4687. doi: 10.1038/nature14539.

C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. pages 562–570, 2015. URL `http://jmlr.org/proceedings/papers/v38/lee15a.html`.

J. Lee, K. Cho, and T. Hofmann. Fully Character-Level Neural Machine Translation without Explicit Segmentation. *arXiv preprint arXiv:1610.03017*, 2016.

S. E. Lee and B. R. Holt. Regression analysis of spectroscopic process data using a combined architecture of linear and nonlinear artificial neural networks. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 4, pages 549–554 vol.4, June 1992. doi: 10.1109/IJCNN.1992.227262.

M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv:1312.4400*, 2014. URL `http://arxiv.org/abs/1312.4400`.

S. Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. Master's thesis, Univ. Helsinki, 1970.

S. Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160, 1976. ISSN 1572-9125.

A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*, 2013. URL `http://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf`.

W. Maass. Neural computation with winner-take-all as the only nonlinear operation. In *Proceedings of NIPS*, volume 12, 1999. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.2941&rep=rep1&type=pdf`.

W. Maass. On the computational power of winner-take-all. *Neural Computation*, 12:2519–2535, 2000. URL `http://www.mitpressjournals.org/doi/pdf/10.1162/089976600300014827`.

J. Malik. What led computer vision to deep learning?: Technical perspective. *Communications of the ACM*, 60(6):82–83, May 2017. ISSN 00010782. doi: 10.1145/3065384.

M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993. ISSN 0891-2017.

J. Martens. Deep learning via Hessian-free optimization. pages 735–742. Omnipress, June 2010. URL `http://www.icml2010.org/papers/458.pdf`.

J. Martens. New insights and perspectives on the natural gradient method. Dec. 2014.

J. Masci, A. M. Bronstein, M. M. Bronstein, and J. Schmidhuber. Multimodal Similarity-Preserving Hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4), 2014a.

J. Masci, A. M. Bronstein, M. M. Bronstein, P. Sprechmann, and G. Sapiro. Sparse similarity-preserving hashing. In *International Conference on Learning Representations*, 2014b. URL `http://arxiv.org/abs/1312.5479`. arXiv: 1312.5479.

M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, 24:109–164, 1989.

S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer Sentinel Mixture Models. *ArXiv e-prints*, Sept. 2016.

T. Mikolov and G. Zweig. Context dependent recurrent neural network language model. *SLT*, 12:234–239, 2012.

M. Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

T. M. Mitchell. *The Need for Biases in Learning Generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ. New Jersey, 1980.

V. Mnih. CUDAMat: a CUDA-based matrix class for Python. *Department of Computer Science, University of Toronto, Tech. Rep. UTML TR*, 4, 2009.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, page 807–814, 2010. URL `http://machinelearning.wustl.edu/mlpapers/paper_files/icml2010_NairH10.pdf`.

Y. Nesterov. A method of solving a convex programming problem with convergence rate O (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

A. Newell. The chess machine: An example of dealing with a complex task by adaptation. In *Proceedings of the March 1-3, 1955, Western Joint Computer Conference*, pages 101–108. ACM, 1955.

B. Neyshabur, R. Tomioka, and N. Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pages 1376–1401, 2015.

A. E. Orhan and X. Pitkow. Skip connections eliminate singularities. *arXiv preprint arXiv:1701.09175*, 2017.

R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. How to construct deep recurrent neural networks. *ArXiv e-prints*, Dec. 2013a.

R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1310–1318, 2013b. URL `http://jmlr.org/proceedings/papers/v28/pascanu13.html`.

J. Pearl. On the connection between the complexity and credibility of inferred models. *International Journal of General System*, 4(4):255–264, 1978.

D. C. Plaut, S. J. Nowlan, and G. E. Hinton. Experiments on Learning by Back Propagation. Technical Report Technical Report CMU–CS–86–126, Carnegie–Mellon University, Pittsburgh, PA, 1987.

J. Pollack. Cascaded back-propagation on dynamic connectionist networks. In *Proceedings of the Ninth Annual Conference of the Cognilive Science Sociely*, pages 391–404, 1987.

B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

O. Press and L. Wolf. Using the Output Embedding to Improve Language Models. *ArXiv e-prints*, Aug. 2016.

N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

T. Raiko, H. Valpola, and Y. LeCun. Deep learning made easier by linear transformations in perceptrons. In *International Conference on Artificial Intelligence and Statistics*, pages 924–932, 2012. URL `http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS2012_RaikoVL12.pdf`.

M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In *Proceedings of NIPS*, 2007. URL `http://www.iro.umontreal.ca/~lisa/seminaires/14-02-2007-2.pdf`.

M. B. Ring. *Continual Learning in Reinforcement Environments*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, Austin, Texas 78712, August 1994.

A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department, 1987.

A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. FitNets: Hints for thin deep nets. *arXiv:1412.6550 [cs]*, Dec. 2014. URL `http://arxiv.org/abs/1412.6550`.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

N. L. Roux, P.-A. Manzagol, and Y. Bengio. Topmoumoute online natural gradient algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856, 2008.

D. E. Rumelhart and D. Zipser. Feature discovery by competitive learning. *Cognitive science*, 9(1):75–112, 1985.

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

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, and M. Bernstein. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

R. Salakhutdinov and G. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, July 2009. doi: 10.1016/j.ijar.2008.11.006. URL `http://linkinghub.elsevier.com/retrieve/pii/S0888613X08001813`.

T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959. ISSN 0018-8646. doi: 10.1147/rd.33.0210.

J. Schmidhuber. A local learning algorithm for dynamic feedforward and recurrent networks. *Connection Science*, 1(4):403–412, 1989. URL `http://www.tandfonline.com/doi/abs/10.1080/09540098908915650`.

J. Schmidhuber. Reinforcement learning in markovian and non-markovian environments. In *Advances in Neural Information Processing Systems 3*. Morgan-Kaufmann, 1991.

J. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

N. N. Schraudolph. Centering Neural Network Gradient Factors. In *Neural Networks: Tricks of the Trade*, Lecture Notes in Computer Science, pages 207–226. Springer, Berlin, Heidelberg, 1998a. ISBN 978-3-540-65311-0 978-3-540-49430-0. doi: 10.1007/3-540-49430-8_11.

N. N. Schraudolph. Slope centering: Making shortcut weights effective. In *Proceedings of the 8th International Conference on Artificial Neural Networks, Perspectives in Neural Computing*, pages 523–528, 1998b.

H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150, 1995.

P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *International Conference on Document Analysis and Recognition (ICDAR)*, 2003. URL `http://www.cse.salford.ac.uk/prima/ICDAR2003/Papers/0176_689_patrice_p.pdfhttp://www.csc.liv.ac.uk/~prima/ICDAR2003/Papers/0176_689_patrice_p.pdf`.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556 [cs]*, Sept. 2014. URL `http://arxiv.org/abs/1409.1556`.

J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv:1412.6806 [cs]*, Dec. 2014. URL `http://arxiv.org/abs/1412.6806`.

N. Srebro, J. Rennie, and T. S. Jaakkola. Maximum-margin matrix factorization. In *Advances in Neural Information Processing Systems*, pages 1329–1336, 2005.

R. K. Srivastava, J. Masci, S. Kazerounian, F. Gomez, and J. Schmidhuber. Compete to compute. In *Advances in Neural Information Processing Systems*, pages 2310–2318, 2013a.

R. K. Srivastava, B. R. Steunebrink, and J. Schmidhuber. First experiments with powerplay. *Neural Networks*, 2013b.

R. K. Srivastava, K. Greff, and J. Schmidhuber. Training Very Deep Networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2377–2385. Curran Associates, Inc., 2015a.

R. K. Srivastava, J. Masci, F. Gomez, and J. Schmidhuber. Understanding Locally Competitive Networks. In *International Conference on Learning Representations*, 2015b.

C. Stefanis. Interneuronal mechanisms in the cortex. In M. A. Brazier, editor, *The Interneuron*, volume 11. University of California Press, Los Angeles, 1969.

M. F. Stollenga, J. Masci, F. Gomez, and J. Schmidhuber. Deep networks with internal selective attention through feedback connections. In *NIPS*. 2014.

C. Strecha, A. M. Bronstein, M. M. Bronstein, and P. Fua. LDAHash: Improved Matching with Smaller Descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1), 2012.

R. S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, 1984.

R. S. Sutton. Two Problems with Backpropagation and Other Steepest-Descent Learning Procedures for Networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum, 1986.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*, volume 1. MIT press Cambridge, 1998.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv:1409.4842 [cs]*, Sept. 2014. URL `http://arxiv.org/abs/1409.4842`.

D. Terman and D. Wang. Global competition and local cooperation in a network of neural oscillators. *Physica D: Nonlinear Phenomena*, 81(1-2):148–176, 1995.

T. Tieleman. Gnumpy: an easy way to use GPU boards in Python. *Department of Computer Science, University of Toronto*, 2010.

T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4:2, 2012.

L. Van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11), 2008.

P. Van Der Smagt and G. Hirzinger. Solving the ill-conditioning in neural network learning. In *Neural Networks: Tricks of the Trade*, pages 193–206. Springer, 1998.

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Science & Business Media, 2013.

A. Veit, M. Wilber, and S. Belongie. Residual Networks are Exponential Ensembles of Relatively Shallow Networks. *arXiv:1605.06431 [cs]*, May 2016.

C. von der Malsburg. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14(2):85–100, Dec. 1973. ISSN 0023-5946.

L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

L. Wang. On competitive learning. *IEEE Transactions on Neural Networks*, 8(5): 1214–1217, 1997.

Q. Wang and J. JaJa. From maxout to channel-out: Encoding information on sparse pathways. In *International Conference on Artificial Neural Networks*, pages 273–280. Springer, 2014.

J. Weng, N. Ahuja, and T. S. Huang. Cresceptron: A self-organizing neural network which grows adaptively. In *Neural Networks, 1992. IJCNN., International Joint Conference On*, volume 1, pages 576–581. IEEE, 1992.

P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

P. J. Werbos. Applications of advances in nonlinear sensitivity analysis. In *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*, pages 762–770, 1981.

P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

P. J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

P. J. Werbos. Backwards differentiation in AD and neural nets: Past links and new opportunities. In *Automatic Differentiation: Applications, Theory, and Implementations*, pages 15–34. Springer, 2006.

R. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, pages 1–10, 1989. URL `http://www.mitpressjournals.org/doi/abs/10.1162/neco.1989.1.2.270`.

Y. Wu, S. Zhang, Y. Zhang, Y. Bengio, and R. Salakhutdinov. On Multiplicative Integration with Recurrent Neural Networks. *ArXiv e-prints*, June 2016.

D. Yu, M. L. Seltzer, J. Li, J.-T. Huang, and F. Seide. Feature learning in deep neural networks-studies on speech recognition tasks. *arXiv preprint arXiv:1301.3605*, 2013. URL `http://arxiv.org/abs/1301.3605`.

W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent Neural Network Regularization. *ArXiv e-prints*, Sept. 2014.

M. D. Zeiler. ADADELTA: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

M. D. Zeiler and R. Fergus. Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. In *Proceedings of the ICLR*, 2013.

M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, and J. Dean. On rectified linear units for speech processing. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, page 3517–3521. IEEE, 2013. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6638312`.

Y. Zhang, G. Chen, D. Yu, K. Yao, S. Khudanpur, and J. Glass. Highway long short-term memory RNNS for distant speech recognition. In *2016 IEEE, ICASSP*, 2016.

Y. T. Zhou and R. Chellappa. Computation of optical flow using a neural network. In *IEEE 1988 International Conference on Neural Networks*, pages 71–78 vol.2, July 1988. doi: 10.1109/ICNN.1988.23914.

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.