Regularizing Deep Models for Visual Recognition

by

Elnaz Barshan Tashnizi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
System Design Engineering

Waterloo, Ontario, Canada, 2016

© Elnaz Barshan Tashnizi 2016

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Image understanding is a shared goal in all computer vision problems. This objective includes decomposing the image into a set of primitive components through which one can perform region segmentation, region labeling, object recognition and finally modeling the interactions between recognized objects. However, due to the large intra-class variations in appearance, shape and structure, extracting image primitives is highly challenging. While images come in the form of intensity matrices, in order to cope with this large variations, a high-level abstraction of images is required. Therefore, the main challenge is to bridge the gap between the low-level pixel representation and the high-level abstract image descriptors.

In recent years, we have witnessed a striking popularity of the learned image descriptors using deep networks for visual recognition. The multi-layer architecture of these networks is particularly useful in capturing the hierarchical structure of the image data: simple features are detected at lower layers and fed into higher layers for extracting more complex and abstract representations. Despite the remarkable representational power of deep networks, training these models is computationally expensive. In addition, considering the lack of enough labeled training data in many applications, over-fitting is a serious threat for deep models with large number of free parameters. Also, there are innate issues with the gradient-based optimization procedure used for parameter learning in these models.

This research is aimed at addressing the above issues by leveraging domain knowledge. Particularly, we focus on tailoring deep networks for visual recognition through exploiting the characteristics of the image data. These modifications tend to *regularize* deep models and therefore, improve their generalization performance.

We propose novel ways for incorporation of image-specific domain knowledge into deep networks. As part of this thesis, we show how one can significantly decrease the number of free parameters in fully-connected architectures by exploiting the global characteristics of the image data. For convolutional networks, a new multi-neighborhood architecture is introduced which can capture scale-dependent features. In this architecture, the fine-scale image structures (i.e., appearance features) are captured using a small-sized neighborhood while coarse-scale characteristics (i.e., shape features) are detected by considering a wider range area around each pixel. Besides, we propose an effective regularization method for deep networks in which a frequency parameter is devised to specifically treat the issues of gradient-based optimization for training these models. Finally, we introduce a stage-wise training framework for deep networks in which the learning process is broken down into a number of related sub-tasks completed stage-by-stage, where the learned parameters at each stage acts as a prior for the next stage. This goal is achieved through *gradual* injection of the information presented in the training data so that in the early stages of training, the *coarse-scale* properties of the data are captured while the *finer-scale* characteristics are learned in later stages. The performance of the proposed methods are assessed on a number of image classification data sets. Our comprehensive empirical analysis demonstrates that these *regularized* networks offer a better discrimination and generalization performance compared to their domain-oblivious counterparts.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Paul Fieguth, for his continuous guidance, support, encouragement and patience during this journey. His professionalism and dedication to high-quality research are exemplary. This research would not have been possible without his clever insights, enthusiasm and encouragement. During our weekly meetings, he taught me how best to identify a research problem and how to recognize viable solutions hidden in the thorough analysis of the problem. His critical way of thinking influenced my personality as a researcher to never stop deepening my understanding of scientific phenomena. I am indebted to him for my growth as a researcher.

I am grateful to my thesis committee members, Professor Zohreh Azimifar, Professor Chris Eliasmith and Professor Pascal Poupart, for their valuable comments and suggestions on the final manuscript. I would also like to thank my external examiner Professor Jun Zhang for his compelling questions and suggestions.

Additionally, I would like to thank my friends and colleges in the Vision and Image Processing (VIP) lab. Particularly, a huge thank to Professor Alexander Wong who shared with me his expertise whenever I needed help. I am proud to be a member of such a motivated and outstanding research group.

I am thankful to all of my friends in Waterloo who made my life in this city colorful. Thank you for all of the companionship, joy, support, empathy, kindness and intellectual discussions.

Most importantly, I would like to thank my family for their support. My deepest gratitude goes to my beloved parents, Shahla and Ali, who have nurtured my self-confidence and encouraged me to be ambitious. I am thankful to my brother, Mohammad, who has always been the source of inspiration to me and my sister, Nazanin, for her kindness and constant belief in me. I am especially grateful to my beloved husband and my best friend, Hassan, not only for his patience and support, but for the motivating discussions and the clever insights he gave me during this research. I feel extremely fortunate to have him by my side.

Dedication

To my beloved parents, and in loving memory of my grandmother ...

Table of Contents

List of Tables

List of Figures

List of Symbols

Dataset

$\mathcal{X}$	input domain
$\mathcal{Y}$	output domain
X	set of input samples
Y	set of output samples
x_i	i^{th} input sample
d	dimensionality of the input space
n	number of samples

Network Architecture

L	Number of network layers
l	Layer index
W	Weight matrix
W^l	Weight matrix of layer l
$\mathbf{w}_{:j}$	Vector of connection weights to neuron j
$\mathbf{w}_{j:}$	Vector of connection weights from neuron j
w_{ij}	connection weight between neuron i and neuron j
$\mathbf{b}$	bias vector
$\mathbf{b^l}$	bias vector of layer l
z_j	input to neuron j
a_j	activation of neuron j
f	Convolutional filter

K	Number of the convolutional filters
p	Pooling size
n_l	Number of neurons in layer l

Training Hyper-parameters

γ	Learning rate
λ	Weight decay coefficient
μ	Momentum coefficient
ep	Number of epochs
n_B	Batch size

Network Functions

E	Energy function
Z	Partition function
$\mathcal{C}$	Cost function
$\mathcal{L}$	Loss function
$\mathcal{R}$	Regularization function
φ	Activation function
Ψ	Pooling function

Eigen-RBM

$\mathbf{h}$	set of hidden layer's neurons
h_j	j^{th} hidden neuron
$\mathbf{v}$	set of visible layer's neurons
v_i	i^{th} visible neuron
F	predefined filter bank for eigen-RBM
$\mathbf{f}^k$	k^{th} filter in the filter bank for eigen-RBM
ρ	size of the filter bank for eigen-RBM
α_{kj}	weight of the k^{th} filter in computing $W_{:j}$

Multi-Neighborhood CNN

I^c	Input image at color channel c
I_s	Input image representation at scale s
S	number of scales
w_{gp}	Gaussian photometric weight
w_{sp}	spatial weight
$\mathcal{N}$	Local neighborhood
r_s	Down-sampling factor at scale s
p_s	Pooling size at scale s
V_s	Extracted features at scale s

Periodic Weight Decay

δ_j^l	Back-propagated error (i.e., $\frac{\partial \mathcal{L}}{\partial z_j^l}$) to neuron j
$\hat{W}^{t+1}$	The update weights only based the Loss function
W^{t+1}	The update weights based the overal cost function
β	The decay coefficient for PWD
f	Frequency of the decay in PWD

Stage-wise Training

S	Number of training stages
T_s	Training set at stage s
$\mathcal{X}_s$	input domain at stage s
$\mathcal{Y}_s$	output domain at stage s
W_s^{init}	Network weights initialization at stage s
W_s	Learned weights at stage s
$\mathcal{F}$	Mapping operator

Chapter 1

Introduction

1.1 Motivation

Categorization [1] is a fundamental property of any intelligent agent. In computer vision, object recognition is an essential ingredient in understanding the contents of an image. In the context of everyday object recognition, the main challenge is to cope with the large intra-class variations across different examples of each category. That is, for each class of objects, there are many different possible configurations of shape, appearance and structure. The images are delivered to us as a vast array of fine-scale pixels. However, the intra-class variations ensure that at the pixel level it is nearly meaningless to talk about recognizing "chair" or "cat", since such objects obtain their identity at a much coarser scale. We therefore see the challenge as one of extracting high-level abstract features from low-level pixels.

For more than a decade, computer vision researchers had focused on engineering hand-crafted features to generate higher-level image descriptors (e.g., edges) which could replace pixel intensities [2, 3, 4, 5]. The main problem with engineered features is that they do not generalize well across different tasks. Moreover, designing discriminative features for each specific task requires expert knowledge and thus is an expensive process.

In recent years, the computer vision community has witnessed the success of "learned" image descriptors [6, 7, 8, 9] against engineered hand-crafted features for image classification. Inspired by the visual recognition process in the human cortex [10], these feature learning models generally follow a multi-layer, i.e., "deep", architecture to capture high-level abstract features. A deep network for feature learning consists of a stack of local or non-local feature detectors where simple features (e.g., edges) are detected at lower layers and fed into higher layers for extracting more complex representations (e.g., object parts).

The origin of deep models, and more specifically deep neural networks, dates back to more than twenty years ago [11, 12]. However, training these networks for general use was impractically slow. Therefore, by the beginning of 2000s, they were already overtaken by other approaches such as kernel methods. This was certainly not the end of the story; the significant growth in computational power (particularly in GPUs and distributed computing) and access to large labeled data sets (e.g., ImageNet [13]) paved the way for their return in late 2000s.

The popularity of deep models spiked when it was shown that they outperform traditional methods in a number of important benchmarks [9, 14]. The exceptional performance of deep models can be mainly attributed to their flexibility in representing a rich set of highly non-linear functions [15, 16], as well as the devised methods for efficient training of these powerful networks [7, 17, 18, 19]. Furthermore, employing various regularization techniques [20, 21, 22] ensured that deep models with huge numbers of free parameters are statistically desirable in the sense that they will generalize well to unseen data.

The automatic and generic approach of feature learning in deep models enables one to use them across different applications (e.g., image classification [6, 23], speech recognition [24, 25], language modeling [26, 27] and information retrieval [28, 29]) with relatively little adjustments. Therefore, deep models seem to be *domain-oblivious* in the sense that in order to use them across different applications, only a small amount of domain-specific customizations is required.

Ideally, the domain-obliviousness of deep networks is advantageous, as having access to a universal and generic model reduces the hassles of adapting for new applications. However, despite the remarkable advances in this area, training deep models with a huge number of free parameters is an intricate and ill-posed optimization problem. In fact, based on the No-Free-Lunch theorem [30], no algorithm can work well for all categorization tasks unless it is given some prior knowledge about it. Therefore, incorporating some essential parts of the domain knowledge as a *prior* into the learning procedure is essential to improve the model's generalization performance.

In the case of visual data, effective and intuitive domain knowledge have been devised in different areas of computer vision and image processing; there is a large body of literature regarding the general characteristics of image data (e.g., [31, 32, 33]) or effective image representations for visual recognition (e.g., [34, 35, 36]). The ideas, concepts and methods that were utilized in these works are inspirational and can serve as a concrete form of domain knowledge in image understanding.

Our research aims at tailoring deep networks for visual recognition through leveraging the properties of the image data. This includes a set of new models and techniques which enable us to incorporate domain knowledge into deep networks. These modifications tend to *regularize* deep models and, therefore, enhance their performance for visual recognition. In the following sections, we will elaborate further regarding our objectives and contributions.

1.2 Objectives

Despite the striking representational power of deep networks [15, 16] and the significant advances in training them [17, 37], the use of these models for visual recognition is still challenging. In particular, the huge number of free parameters in these models makes the process of model fitting computationally hard and statistically unstable [38]. As discussed in the previous section, the use of domain-knowledge can serve as a remedy for this issue.

In this dissertation, instead of fully relying on the representational power of deep models in learning discriminative features, we want to leverage image-specific characteristics in order to regularize and eventually improve deep networks. Motivated by this idea, this research aims at addressing the following questions:

1. How can one tailor deep models for visual recognition?
2. How much benefit do we get by these adjustments?

In order to address the first question, we note that domain knowledge about visual recognition can be incorporated into deep models in pursuit of the following objectives:

- **Improving optimization:** In training deep networks, the optimal values for the parameters are found through a gradient-based optimization procedure [39]. Image-specific priors can be utilized to address the known issues of this optimization procedure (e.g., gradient instability) as well as improving the speed of convergence and overall training time.

- **Improving generalization:** Deep networks usually have a huge number of free parameters—much greater than the number of training samples—and thus are subject to over-fitting. Considering this fact, the image characteristics can be exploited in order to restrict the search space for the parameters, and consequently regularize these models. For instance, the parameters of deep networks can be initialized using image priors. Furthermore, domain-specific regularization can be done by aggregating a set of deep models.

For the second question, the effect of incorporating image-specific knowledge in training deep networks should be assessed empirically. Due to the large number of tunable hyper-parameters, systematic analysis of the performance of deep networks is challenging. There are a large number of phenomena interacting and it is difficult to disentangle their effects. Therefore, it is easy to misinterpret the results and draw grand conclusions. In order to prevent this issue, the experimental setup should be simple, controlled and revealing. Moreover, one should not fully rely on the final recognition performance of the model. To disclose the contributing factors in success of a model, careful inspection of the model through a set of analytical experiments is required. We will take these considerations into account in the design and analysis of our experiments.

1.3 Summary of Contributions

In pursuit of the described objectives, this research leads to the following contributions:

- Regularizing Restricted Boltzmann Machines (RBMs) [40]—the building block for Deep Belief Networks (DBNs) [17]—through the use of global characteristics of the input images at different levels of details. We propose Eigen-RBM, a generalization of fully connected RBMs, with an efficient training strategy in which the number of free parameters is independent of the image size. In this method, the number of free parameters is reduced by constraining the network weights to be a linear combination of a set of "predefined" filters based on domain knowledge.

- Improved feature learning using convolutional networks [11] by using a multi-scale representation of the input image. In order to capture image features at different levels of detail, we propose multi-neighborhood convolutional networks in which fine-scale image characteristics (i.e., *appearance* features) are captured through a small-size neighborhood while coarse-scale image characteristics (i.e., *shape* features) are detected by exploring a wider range of dependencies over a larger neighborhood. In addition, we suggest a scalable learning strategy through which one can use an already-trained single-scale network to extract multi-scale image features without increasing the training cost. That is, the learning is performed only on the original scale, but then the learned parameters are applied to an ensemble of networks at different scales.

- Introducing a new general-purpose regularizer for training deep networks. We show that weight decay, as a widely-used regularizer in deep networks, can exacerbate some of the known optimization issues in training these models. In order to address this problem, we propose Periodic Weight Decay (PWD), a generalization for basic weight decay, in which a frequency parameter is devised to specifically treat the issues of gradient-based optimization for training deep networks. Compared to basic weight decay, PWD offers a faster convergence rate and a better generalization performance for deep networks.

- Improved feature learning for visual recognition using a stage-wise training framework for deep networks. In this method the network is steered towards capturing "typical" characteristics of each category—instead of their "unique" features—through a sequence of related learning stages. To that end, during the early stages of training the *coarse-scale* properties of the image are captured while the *fine-scale* characteristics are learned in the subsequent stages. Moreover, the solution found in each stage acts as a prior to regularize the next training stage.

1.4 Thesis Structure

Following the sequence of the mentioned contributions, this thesis is organized in seven chapters. Chapter 2 offers an overview on the deep learning literature with an emphasis on the important changes—in their architecture, training procedure and regularization schemes—which played a pivotal role in their revival. In Chapter 3, we focus on the scalability issue for RBMs and show that how the computational burden of training these fully-connected models can be reduced by exploiting the global characteristics of the input image. Our novel multi-neighborhood convolutional architecture along with its scalable training strategy is presented in Chapter 4. Utilizing the multi-scale representations of the input images, we propose a multi-neighborhood architecture that can effectively capture image features at different levels of details. The focus of Chapter 5 is on improving the gradient-based optimization procedure in training deep models. In this chapter we introduce Periodic Weight Decay (PWD) for regularizing deep networks and show that it leads to a faster convergence and better generalization performance for deep models. Subsequently, in Chapter 6 we introduce our novel stage-wise training framework for deep networks in which the network is guided step-by-step towards a better solution by gradual injection of the presented information in the visual data. Finally, the thesis is concluded in Chapter 7.2.3 with a general discussion over the proposed methods and a set of directions for future research.

Chapter 2

Background and Literature Review

This chapter provides a review over the deep learning literature with the aim of highlighting the pivotal factors in the revival of deep networks. Besides the significant growth in the computational power (particularly in GPU and distributed computing), a set of important changes in the model architecture and training procedure paved the way for the immense popularity of deep networks. We first start by describing the architectural properties of deep networks which distinguish them from their classical shallow counterparts. Then, after giving an overview on the general framework for training neural networks, the challenges of extending this framework to deeper networks is discussed. Finally, we focus on the necessity of regularization for deep networks and review some of the widely-used regularization techniques in training these models.

2.1 Model Architecture

The architectural properties of deep networks have a crucial impact on their success, and set the stage for the efficient training of these models. In contrast to conventional shallow networks, deep networks are especially useful for modeling high-dimensional and structured data where higher level characteristics can be obtained by combining lower-level structures. As it comes from its name, the most distinguished architectural property of these networks is depth. In this section, the role of depth in the representational power of the network is discussed. Then, we show that how the local nature of network layers offers scalability to high-dimensional data and captures the structural properties of the data.

2.1.1 Depth

The amazing representational power of neural network architectures was known since 1989 through the "universal approximation theorem" [41]. Based on this theorem, a *single-hidden-layer* neural network with finite but large enough number of hidden neurons can approximate any continuous function to any desired precision[1]. The quality of this approximation improves by increasing the number of hidden neurons. Given the striking expressive power of a single hidden layer network, one may question the motives for having a deeper architecture.

Despite the fact that single hidden layer architectures can represent any function, their representation may not be efficient for some classes of functions [42]. That is, they might need a huge number of neurons, parameters and consequently training examples. Furthermore, it has been proven that well-known single-level architectures like Gaussian Kernel Machines and decision trees[2] are sensitive to the curse of dimensionality [43, 44].

Recently, it has been theoretically shown that depth of the network can compensate for its width. Sutskever and Hinton showed that in the case of limited width, the depth of a network is the key factor in determining its expressive power [45]. Montufar et al. [15] offered a comparative study between a narrow but deep model and a shallow network with equal number of neurons in terms of the number of response regions. Based on this study, a deep model can generate a larger number of response regions compared to its shallow counterpart and thus, offers a better expressive power. A related result indicates that with a 2-layer network, one needs an exponential number of neurons to represent a simple radial function that is easily expressible with a small three-layer network [16].

On the other hand, many classes of data (e.g., image, speech and text data) have a natural hierarchical structure. Capturing this hierarchical architecture and learning an abstract representation of the data is a primary goal in various machine learning applications. For instance, image data can be described in different levels of abstraction (e.g.,

[1]Mild conditions on the activation function are required.

[2]Note that for decision trees level refers to the *architectural level* and not the level of the tree.

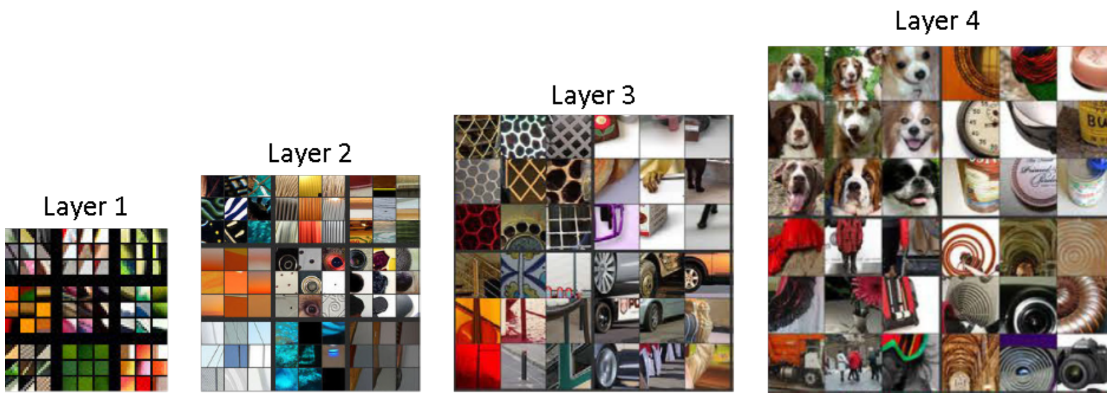


Figure 2.1: Visualization of the learned features using a deep network [49]. This figure shows the image patches that cause high activations in each network layer. Observe the increasing level of abstraction as we move up in the network.

pixel intensities, edges, object parts, etc.). In particular, lower level structures can be combined to create a higher level structure that corresponds to a more abstract concept. This hierarchical nature of the data has led to the emergence of many successful multi-level image models, including wavelet models [34, 46], scale-space models [47, 35] and pyramid representations [36, 48]. Following this idea, the multi-layer architecture of deep networks enables them to capture hierarchical characteristics of image data (see figure 2.1).

Inspiration from the human brain and more specifically the human visual system make the desire for having deep architectures even more intense among AI researchers [10]. Evidence from neuroscience suggests that human brain is similar to a deep network, and the input signals are passed through several layers of neurons where they are encoded in different levels of abstractions [50].

More importantly, multi-layer neural networks offer an impressive performance gain in many challenging learning problems especially in computer vision. Deep models have outperformed their shallow counterparts with a large margin in major visual classification benchmarks [51]. This practical advantage has been obtained through resolving a number of training obstacles that deep models used to have.

The above factors have resulted in a dramatic increase in popularity for deep architectures in recent years. Although convincing, our theoretical understanding of multi-layer networks is still limited and there are a lot of unanswered questions.

2.1.2 Locality

Before the substantial growth in the popularity of deep models, most of the employed neural network architectures were fully-connected [52, 53]. In fully-connected architectures the number of parameters grows almost linearly with the dimensionality of the input. Therefore, the use of fully-connected networks for high-dimensional data is impractical. In addition, these networks ignore structural properties of the data for visual recognition. In order to overcome these drawbacks, in recent deep architectures a set of locality constraints are devised which make multi-layer models particularly useful for high-dimensional structured data [7, 8, 54]. The locality constraints, i.e., architectural regularizers, also include a weight-sharing scheme. This scheme reduces the number of parameters by sharing the same weights between different localities[3]. In addition, the network is equipped with an abstraction mechanism to represent the hierarchical structure of the data when multiple local layers are stacked.

Convolutional Models

In visual recognition, the most widely used models that benefit from local structures are the convolutional networks [11, 7, 55]. These hierarchical models aim at learning a set of local feature detectors or filters at each layer that are shared across the extent of the image. The intuition behind this structure is that if a feature is believed to detect useful information in some part of the image, then it can extract relevant information from other parts of the image as well. Considering this shared nature of the connections and the small size of the convolutional filters, the number of parameters that are needed to be learned and stored reduces greatly. The filters are learned in a supervised mode using classification error back-propagation [9] or in an unsupervised mode employing a generative framework [7]. Figure 2.2 offers a schematic illustration of a single layer feature extractor of this architecture. Given an $N \times N$ input image I, and a set of $M \times M$ learned filters

[3]This idea is similar to having the same filter applied to different localities of the image in image processing.

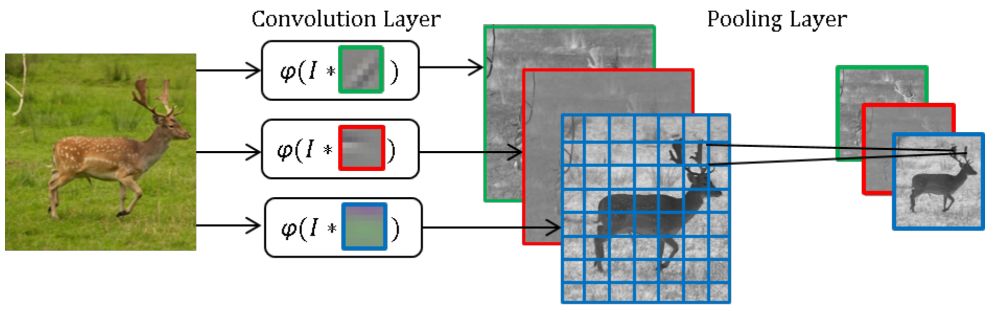


Figure 2.2: Schematic diagram of feature extraction using a single layer convolutional network. The input image is passed through a set of learned filters followed by a nonlinear function φ and a pooling layer. All the filters have the same size.

$\{f_1, f_2, ..., f_K\}$, the output H of the convolutional layer would be K images of size $(N - M + 1) \times (N - M + 1)$ that are computed as:

$$H_{ij}^k = \varphi\big((f_k * I)_{ij} + b_k\big) \tag{2.1}$$

where $\varphi(.)$ is a nonlinear activation function and b_k is the learned bias of the layer. Note that in this architecture, since all of the learned filters are of the same size (i.e., $M \times M$), only a single-size neighborhood centered at each input pixel is considered.

Spatial Pooling

In order to capture abstract representations in a multi-layer network, higher level feature detectors need information from progressively larger regions. Stemming from this fact, the convolutional layer is usually followed by a "spatial pooling" (i.e., sub-sampling) layer which shrinks the output of the convolution layer (see figure 2.2). More formally, a local neighborhood $\mathcal{N}$ of size $p \times p$ is shrunk to a single pooling unit through a many-to-one function $\Psi(.)$. The pooling function $\Psi(.)$ is commutative in the sense that changing the order of the inputs does not change the output. This mechanism makes deep networks more scalable while it allows the representations to be invariant to local transformations (e.g., max-pooling makes the representations invariant to local translations of the input).

Conventional choices for pooling function $\Psi(.)$ are the *max* and *average* functions. However, for visual recognition, the use of average pooling has been discouraged [56, 57].

Average pooling has the drawback of down-weighting strong activations since it considers the effect of all elements, including the very small ones, in the pooling region. Moreover, if the activation function $\varphi(.)$ can generate both positive and negative values (e.g., *tanh*), then applying average pooling causes the strong positive and negative activations to cancel out each other. It has been shown that employing a *stochastic* pooling function acts as a regularizer and can effectively reduce over-fitting [57].

2.2 Training

Among different types of artificial neural networks, feedforward networks are the most extensively investigated models for image classification. The general framework for training deep feedforward neural networks is similar to that of the classical (less deep) models. In particular, stochastic gradient descent and error back-propagation are the essential techniques used for training them. However, for efficiently training deeper networks, some special considerations should be made. In fact, without addressing these considerations, deep networks do not perform better than the shallow ones [58].

In this section, after introducing the notation, we explain the general training framework for feedforward neural networks. For simplicity of further discussion, we assume that no weight-sharing scheme is used. Afterwards, the challenges of extending this framework to training a deep architecture are discussed. Finally, a number of strategies that paved the way for efficient training of deep models are reviewed.

2.2.1 Notation

Given a feedforward neural network with L layers, each layer l is parametrized with a weight vector W^l and a bias vector b^l. The input to each neuron j in the l^{th} layer is denoted by z_j^l and is defined by

$$z_j^l = \sum_k w_{kj}^l a_k^{l-1} + b_j^l \tag{2.2}$$

where a^{l-1} is the output (i.e., activation) vector of the previous layer. Given an activation function φ, the activation of each neurn j is computed as

$$a_j^l = \varphi(z_j^l). \tag{2.3}$$

The input data x_i is fed to the first layer (i.e., input layer) and the activation of the L^{th} layer a_i^L is the estimated target using the network.

2.2.2 Optimization

The goal of training is to find the optimum values for the network parameters W and b such that a loss function $\mathcal{L}$ is minimized on the training set $\{(x_i, y_i)\}_{i=1}^n$. For classification problems, there are a number of real-valued loss functions which are related to the prediction error. Among those, a basic choice would be the Mean Squared Error (MSE):

$$\mathrm{L}_{MSE} = \frac{1}{2n}\sum_i (y_i - a^L(x_i))^2 \tag{2.4}$$

where y_i is the true target and $a^L(x_i)$ is the estimated output for x_i. Another popular loss function is negative log-likelihood which is defined as

$$\mathrm{L}_{NLL} = -\frac{1}{n}\sum_i \ln(a_{y_i}^L(x_i)) \tag{2.5}$$

where $a_{y_i}^L$ denotes the activation of the certain output neuron that corresponds to the true label. Note that the choice of the loss function and the activation function for the output layer are interrelated. MSE is typically matched with a linear activation function in the output layer while negative log-likelihood is used with a soft-max output layer which is computed as

$$a_j^L = -\frac{e^{z_j^L}}{\sum_k e^{z_k^L}}. \tag{2.6}$$

For the hidden layers, commonly a sigmoid activation function (i.e., $\sigma(z) = \frac{1}{1+e^{-z}}$) is used.

The loss function $\mathcal{L}$ is minimized with respect to the network parameters using the Stochastic Gradient Descent (SGD) algorithm [39]. SGD is a generalization of GD (i.e., Gradient Descent) in which the gradient of the loss function is computed on a randomly chosen *subset* of the training data, as an unbiased approximation of the gradient on the entire training set. For each randomly selected training batch SGD takes the following steps:

1. **Forward Pass:** Computes the network output (see figure 2.3a).
2. **Backward Pass:** Calculates the gradient of the loss function with respect to the network parameters using error back-propagation.
3. **Update:** Update the parameters by taking a step in the opposite direction of the gradient.

Error back-propagation [12] is an efficient algorithm for computing the gradient of the loss function in which all of the partial derivatives are computed in a single backward pass. It starts by computing the *error signal* δ_j^L for the neurons of the output layer which is defined as:

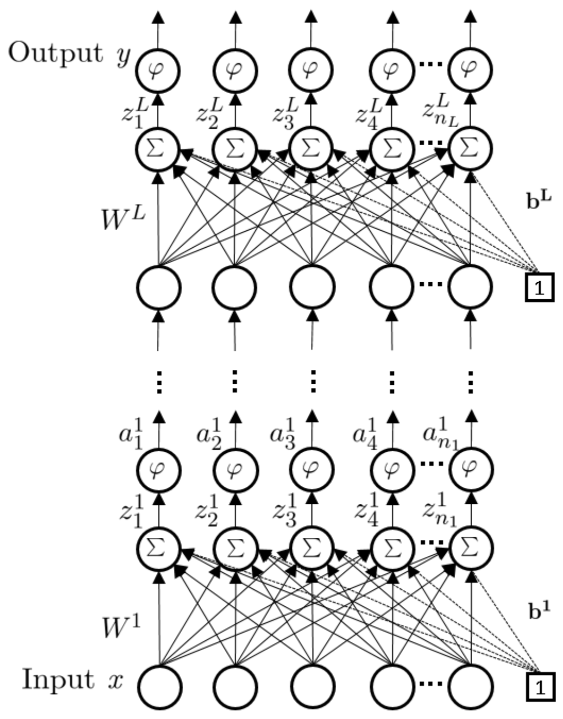
$$\begin{aligned} \delta_j^L &= \frac{\partial \mathcal{L}}{\partial z_j^L} \\ &= \frac{\partial \mathcal{L}}{\partial a_j^L} \varphi'(z_j^L) \end{aligned} \tag{2.7}$$

Then, the error is back-propagated to the earlier layers and interestingly for each layer l, the error vector δ^l can be computed based on δ^{l+1} which has been already calculated:

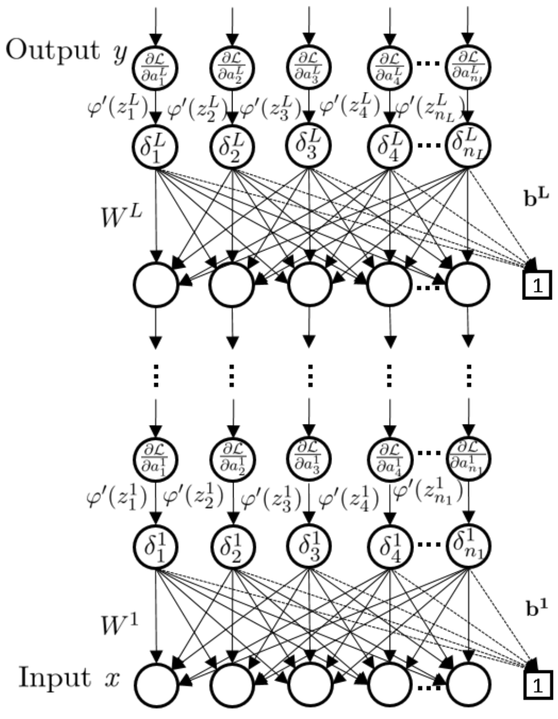
$$\delta_j^l = \sum_k w_{jk}^{l+1} \delta_k^{l+1} \varphi'(z_j^l) \tag{2.8}$$

Applying the chain rule, the partial derivatives of the loss function $\mathcal{L}$ with respect to the network weights at each layer l can be computed in terms of the error at that layer and the activations of its previous layer as follows:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{ij}^l} &= \frac{\partial \mathcal{L}}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{ij}^l} \\ &= \delta_j^l a_i^{l-1} \end{aligned} \tag{2.9}$$



(a) Forward Pass



(b) Backward Pass

Figure 2.3: Schematic diagram of the forward pass and backward pass (error back-propagation) in the training procedure of neural networks.

Similar to the weights, the partial derivatives of the loss function with respect to the biases are calculated as

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b_j^l} &= \frac{\partial \mathcal{L}}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} \\ &= \delta_j^l.\end{aligned} \tag{2.10}$$

After estimating the gradients using error back-propagation, the network parameters are updated as follows.

$$w_{ij}^l = w_{ij}^l - \gamma \frac{\partial \mathcal{L}}{\partial w_{ij}^l} \tag{2.11}$$
$$b_j^l = b_j^l - \gamma \frac{\partial \mathcal{L}}{\partial b_j^l}$$

where γ is the learning rate for SGD.

A schematic diagram for the error back-propagation algorithm is presented in figure 2.3b.

2.2.3 Challenges

Although the standard SGD and error back-propagation are effective methods for training shallow networks, their use for training deeper models is problematic. Therefore, the performance of deep models will not be superior to that of shallow ones if the standard training methods are used [58]. The main reason behind this deficiency is that the speed of learning for different layers are not similar. Having different learning speeds for different layers of the network is a fundamental consequence of using a gradient-base optimization for a multi-layer architecture.

As a toy example, consider a neural network with three hidden layers with a single hidden neuron in each layer (See figure 2.4) [59]. Considering (2.9) and (2.10), δ^l is directly proportional to the size of the partial gradients. Therefore, it can be used as a rough measure for the speed at which the parameters of layer l are learned. Now we compare the learning speed of the output layer with that of the first hidden layer. Based on (2.7), δ^4 becomes

$$\delta^4 = \varphi'(z^4) \frac{\partial \mathcal{L}}{\partial a^4}. \tag{2.12}$$

By repetitive application of (2.8), δ^1 is computed as follows:

$$\begin{aligned} \delta^1 &= \varphi'(z^1) w^2 \varphi'(z^2) w^3 \varphi'(z^3) w^4 \varphi'(z^4) \frac{\partial \mathcal{L}}{\partial a^4} \\ &= w^2 \varphi'(z^1) w^3 \varphi'(z^2) w^4 \varphi'(z^3) \delta^4 \end{aligned} \tag{2.13}$$

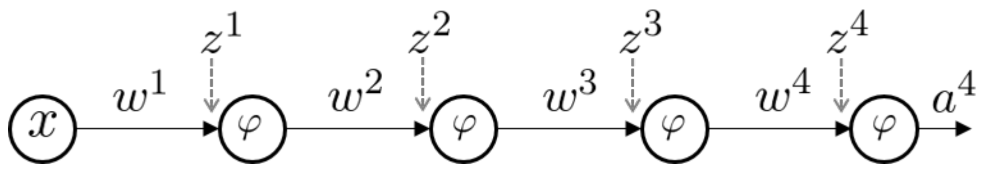


Figure 2.4: A toy network with three hidden layers for illustrating the gradient vanishing problem.

Therefore, the gradient (i.e., the speed of learning) in an earlier layer is a product of $w\varphi'(z)$ terms from all of its upper layers. Now, if a sigmoid activation function is used (i.e., $\varphi(z) = \frac{1}{1+e^{-z}}$), the magnitude of $\varphi'(z^l)$ is always less than 1/4. On the other hand, the weights are usually initialized to small values (i.e., less than one). As a result, the speed of the learning (i.e., the gradient) exponentially decreases (with respect to the layer number) as we move backward from the top to the earlier layers. This phenomenon is known as *gradient vanishing* or diffusion of gradients.

One may argue that the size of the weights grow during training and as a result, the magnitude of $w\varphi'(z)$ might be greater than one. This situation could be equivalently problematic, because the gradient (i.e., the speed of the learning) tend to increase exponentially as one moves backward to the earlier layers. This situation is called *gradient exploding.*

To recap, with the standard SGD, earlier layers of the network will have either progressively smaller speeds of learning, or larger ones. This will cause a slow convergence for the first case, and instability in the later. Therefore, some measures should be taken to overcome these difficulties.

2.2.4 Solutions

A number of methods have been proposed to tackle the difficulties of training deep neural networks. Glorot and Bengio found that sigmoid activation functions are not suitable for deep networks with random initialization [58]. Their analysis demonstrates that using sigmoid activation function in a randomly initialized network drives the top layer neurons to saturation around zero at the start of the training. To alleviate this problem, the use of a different non-linearity such as *softsign* where $\varphi(z) = \frac{1}{1+|z|}$ is recommended. In addition, they proposed a normalized initialization scheme to protect the variance of the back-propagated gradients from vanishing or exploding.

Rectified Linear Units (ReLUs) are arguably the most popular alternative for the problematic sigmoid neurons in deep neural networks [56, 19, 18]. The rectifier activation function $\varphi(z) = max(z, 0)$ prevents gradient vanishing because for active neurons, the activation function behaves linearly. Also, it has been shown that this activation function is biologically plausible and improves sparsity of the learned representations [19]. Superior recognition performance of ReLUs has been demonstrated for many deep learning applications [56, 22].

Using Momentum-based gradient methods [60] in training deep networks improves the speed of convergence and the quality of the found local minimum [61]. The key idea behind momentum-based gradient descent is to update the parameters in directions more effective than steepest descent by incorporating information about how the gradient is changing. To that end, the speed of moving in directions which consistently reduce the cost function is accumulated. More precisely, for each parameter w, a velocity vector v is considered and updated based on the following rule:

$$v = \mu v - \gamma \frac{\partial \mathcal{L}}{\partial w} \tag{2.14}$$

where the momentum coefficient $\mu \in [0, 1)$ controls the contribution degree of previous gradients. Accordingly, the gradient descent update rule (2.11) changes to $w = w + v$. It has been shown that careful scheduling of the momentum coefficient increases the effectiveness of SGD in training deep networks [61].

2.3 Regularization

Deep networks typically have a huge number of free parameters. Although exerting locality and weight-sharing constraints reduces this number, still a large number of parameters should be learned based on the training data. Therefore, the risk of over-fitting (i.e., poor generalization) is high for deep models.

Having access to large labeled data sets (e.g., ImageNet [13]) alleviates this problem to some extent. However, still the large number of parameters is a threat to the model generalization. Therefore, it is necessary to limit the degrees of freedom of the model in the parameter space through employing a regularization technique. Beside the general-purpose regularizers, special-purpose regularization schemes have been developed for deep networks. The use of these regularizers has had a profound impact on the re-emergence of multi-layer neural networks. In the following, we review some of the important regularization techniques used in deep learning.

2.3.1 General-purpose regularizers

Two types of generalization error bounds have been proved for multi-layer neural networks [62]. The first one is based on the *number of parameters* in the network, and suggest that to obtain a good generalization performance, the number of training data should grow linearly with the number of network parameters [63].

In practice however, neural networks perform well even when the number of training samples is considerably less than the number of network parameters (e.g., [64]). Motivated by this observation, a second type of generalization error bound was shown based on the *size of the parameters* [65]. This bound suggests that as long as the size (i.e., norm) of the parameters is small, neural networks can have a good generalization performance—even when the number of parameters is huge.

The successful application of general-purpose regularizers in training deep networks has corroborated the theoretical role of the parameter sizes in the generalization performance of neural network. One simple practice to control the size of the final weights is *early stopping* [66]. Another relevant technique is the popular and widely used weight decay regularizer [67, 68]. In this method, the cost function is augmented with a penalty term $\mathcal{R}(w)$ which is based on the L_2 or L_1 norm of the weights. Therefore, the learner should choose the smallest weight vector that solves the problem. Employing the L_1 norm of the weights as a regularizer promotes sparsity, allowing a few large weights [68]. Minimizing the L_2 norm of the parameters on the other hand pushes all the weights towards zero. A more detailed description of weight decay method can be found in section 5.2.

2.3.2 Pre-training

An important part of the gradient-based optimization methods is the *initialization* procedure, where the starting point for the update process is determined. In *convex* optimization, these initial values might affect the speed of convergence to the optimal solution. In *non-convex* optimization however, the effect of initialization is more significant. In fact, the initial values can change the local optimum that the gradient-based optimization will end up to. This is the case for neural networks and therefore, the "quality" of the optimization outcome depends crucially on the choice of initialization [61, 58]. Particularly, for deep networks random initialization often leads to poor local minimums.

Pre-training methods address this issue by initializing the network parameters near a good solution [17, 14, 20]. In these techniques, the initial values of the weights are determined by layer-wise pre-tranining of the network, starting from the first layer. Layer-wise pre-training is usually an unsupervised procedure where each layer is treated as a Restricted Boltzmann Machine (RBM) [40] or an Auto-Encoder [69]. Pre-training acts as a regularizer (i.e., a good prior) in the sense that it restricts the search-space for the parameters. That is, the final values of the parameters will be close to their initial values determined by an auxiliary objective function [70]. This explanation justifies the better generalization performance of the pre-trained networks [70]. Similarly, transfer learning [21] can be considered as a pre-training method and consequently a regularization scheme for deep networks. More information about this subject can be found in section 6.2.

2.3.3 Dropout

One of the main approaches for preventing a model from over-fitting is *model averaging*. In this approach, the output is computed by taking a weighted average over the predictions using all possible configurations of the model parameters. However, for deep neural networks—which typically have a large number of parameters—this kind of model averaging is computationally intractable, both in training and test phases. One idea to alleviate this problem is to approximate the averaging method by combining the predictions of only a small ensemble of networks. However, this is still problematic, as the test-time computational complexity will still be high, and the small ensemble may not approximate the whole distribution of possible models accurately enough.

Dropout is an effective regularization technique for neural networks that addresses the above concerns [37, 22]. In this method, for each presentation of each training data, a new *thinned* network architecture is used. These different thinned architectures are produced by random removal of neurons from the base architecture with a predefined probability $1 - p$. In each step of training, if a neuron is chosen to be dropped, then all of its outgoing and incoming weights are removed. During the test time, a single network with the base architecture is used in which the learned weights are scaled down by a factor p[4]. The reason for this down-scaling is to make sure that the expected output of each neuron at training time is the same as its actual output at the test time. The good generalization performance of dropout is attributed to the fact that it prevents complex co-adaptation of hidden units. That is, a hidden unit cannot rely on the presence of other hidden units and thus the co-adaptations are weakened.

Drop-connect is also a similar method to dropout with the difference that, instead of hidden neurons, the connections between neurons are randomly dropped [71].

[4]This down-scaling is applied only on the layer regularized with dropout.

Chapter 3

Scalable Learning for Restricted Boltzmann Machines

As mentioned in section 2.3.2, in the unsupervised pre-training of deep neural networks, each layer is treated as a Restricted Boltzamnn Machine (RBM) [40]. RBM is a probabilistic generative model widely used for feature learning in visual recognition [6, 17, 72]. RBM has a large number of free parameters and, therefore, the computational cost of model-fitting is high, which limits its applicability to small-sized images. In this chapter we propose Eigen-RBM, a scalable RBM for visual recognition in which the number of free parameters to learn is independent of the image size. Eigen-RBM exploits the global structure of the image and does not impose any locality or translation-invariance assumptions, and regularizes the network weights to be a linear combination of a set of predefined filters. We show that, compared to basic RBM, Eigen-RBM can achieve similar or better performance in both recognition and sample generation with significantly less training time[1].

[1]The materials in this chapter has been previously published in [73], where the thesis author is the first author.

3.1 Objective

One of the key challenges in visual recognition is the limited number of labeled data which are hardly sufficient to train the discriminative models with a large number of parameters. To cope with this problem, generative models can be used as domain-specific regularizers for discriminative systems. These models impose additional constraints on the parameters to perform well in generation as well as recognition.

Despite the merits of employing generative models in visual recognition, there are many unresolved obstacles. Of these, one of the most important issues is that of high computational complexity. Although considerable advances have been made [7, 17], running these algorithms requires special hardware and software support.

In this chapter, we focus on Restricted Boltzmann Machines (RBMs), the most widely-used generative model for visual recognition, and study the issue of computational complexity. The number of RBM parameters grows roughly quadratically with the size (i.e., the number of pixels) of the input image. Therefore, extending RBMs to high-resolution images is not computationally tractable or desirable, since having large numbers of parameters is a threat to good model generalization. To address this problem, we propose Eigen-RBM, a regularized extension of RBM, for visual recognition in which the number of free parameters to learn is independent of the image size. Eigen-RBM exploits the global structure of the image and does not impose any locality or translation-invariance assumption, and regularizes the network weights to be a linear combination of a set of predefined filters.

3.2 Background

3.2.1 Generative Models for Images

The basis of generative modeling approaches for image applications is the seminal work by Geman and Geman [31]. The heart of any generative approach is modeling the *prior* distribution of the data. However, due to the high-dimensionality and non-Gaussian statistics of natural images [32], defining a generic prior model is quite challenging. Researchers have evolved this model of [31] by investigating richer priors [74, 75, 76, 77].

In the past decade we have witnessed a growth of attention [7, 8, 17] toward using generative models for visual recognition. The problem of visual recognition has some innate challenges, including a lack of labeled data and image occlusion, both of which can be addressed in a generative framework:

- Unsupervised Learning: Labeled data is costly to produce, however generative models are able to learn from unlabeled data.
- Occlusion: Investigating the generative property of a model, ambiguities in the raw sensory inputs can be resolved by means of inferring the missing pixels [7, 8].

3.2.2 Restricted Boltzmann Machines

Restricted Boltzman Machines (RBMs) [40] are the most widely used generative model for feature extraction in visual recognition [17, 54, 72, 78]. RBMs are bipartite undirected graphical models with a set of binary hidden units **h** and a set of visible units **v** (binary or real-valued) arranged in two layers. There are symmetric connections between these two layers represented by weight matrix W (see figure 3.1). In this structure, visible units correspond to input data (e.g., image pixels) and hidden units are the extracted abstract representations. In order to capture more abstract features, RBMs can be stacked to form deep architectures such as Deep Belief Networks [17] and Deep Boltzman Machines (DBMs) [79].

In an RBM, the probability of a configuration $(\mathbf{h}, \mathbf{v})$ is

$$P(\mathbf{h}, \mathbf{v}) = \frac{1}{Z} exp\big(-E(\mathbf{h}, \mathbf{v})\big) \tag{3.1}$$

where Z is the partition function, with energy $E(\mathbf{h}, \mathbf{v})$

$$E(\mathbf{h}, \mathbf{v}) = \frac{1}{2} \sum_{i \in vis} (v_i - c_i)^2 - \sum_{j \in hid} h_j b_j - \sum_{i \in vis, j \in hid} v_i w_{ij} h_j \tag{3.2}$$

where b and c are the biases of the hidden and visible units. Given the joint distribution $P(\mathbf{h}, \mathbf{v})$ (3.1), the probability that an RBM assigns to an input vector **v** can be obtained from the marginal $P(\mathbf{v})$. The parameters of an RBM can be optimized in a purely unsupervised manner, based on maximizing the likelihood of the training data. Since maximizing

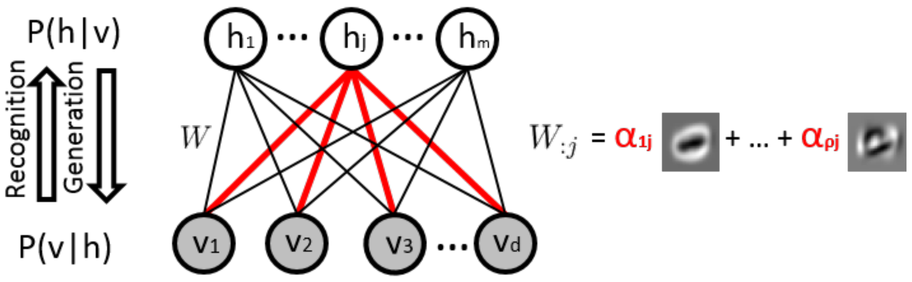


Figure 3.1: Schematic diagram of Eigen-RBM in which filters (i.e., $W_{:j}$) are defined as linear combinations of a set of pre-defined filters.

$P(\mathbf{v})$ with respect to the network weights does not have a closed-form solution, gradient ascent is applied.

One key disadvantage is the large number of parameters to be learned, thus the application of RBMs is limited to small-size images. Furthermore, having a large number of parameters is always a threat to the good generalization of a model. One way of tackling this problem is to reduce the number of parameters using a weight-sharing scheme [7, 8, 54]. Convolutional architectures [7] assume that the network weights (i.e., filters) are local and stationary, however, in practice translation invariance is frequently violated.

3.3 Methodology

3.3.1 Eigen-RBM

Suppose that $\mathcal{X} \in \mathbb{R}^d$ is a set of observed input variables and $\mathcal{Y} \in \mathbb{N}$ is a set of latent output variables drawn jointly from distribution $P(\mathcal{X}, \mathcal{Y})$. Given a set of unlabeled observed samples $X = \{x_1, x_2, \ldots, x_n\}$ as n realizations of $\mathcal{X}$, we are looking for a weight matrix W which maps x to $\mathbf{h}$ such that the likelihood of the observations is maximized. We use $W_{:j}$ to denote the weight vector that connects all of the units of the visible layer to hidden unit h_j.

In order to scale RBMs to realistic-sized images, we propose that the weights $W_{:j}$ to be

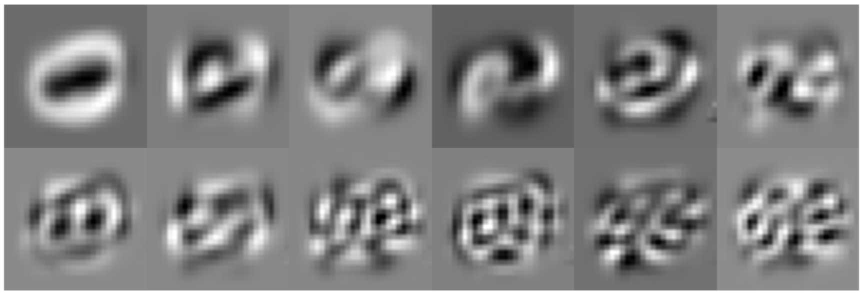


Figure 3.2: Eigendigits filter bank: Eigen-decomposition of the training data. Observe how the Eigendigits capture information at a variety of scales from coarse to fine.

defined as linear combinations of a set of predefined filters (see figure 3.1). That is, given a filter bank $F = \{\mathbf{f}^1, \mathbf{f}^2, ..., \mathbf{f}^\rho\}$, we define weight vector (i.e., filter) $W_{:j}$ as

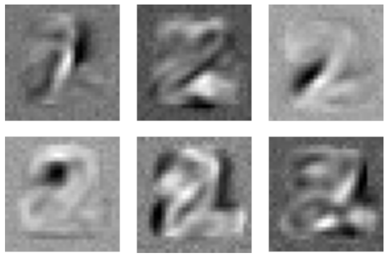
$$W_{:j} := \sum_{k=1}^{\rho} \alpha_{kj} \mathbf{f}^k \tag{3.3}$$

where the size of the filter bank is much smaller than the number of visible units (i.e., $\rho \ll d$). In this way, the number of parameters is independent of the image size and becomes instead a function of the size of the filter bank. In addition, the global structure of the image is exploited and no locality or translation invariance assumption is imposed.

Training an RBM consists of learning the weights α_{kj} of the filters in the filter bank. Performing gradient ascent on the log-likelihood of the training data, the update rule for coefficient α_{kj} is computed as

$$\begin{aligned} \frac{\partial logP(\mathbf{v})}{\partial \alpha_{kj}} &= \sum_{i \in vis} \frac{\partial logP(\mathbf{v})}{\partial w_{ij}} \frac{\partial w_{ij}}{\partial \alpha_{jk}} \\ &= \sum_{i \in vis} (< v_i h_j >_{data} - < v_i h_j >_{model}) f_i^k \end{aligned} \tag{3.4}$$

The freedom of choosing the filter bank enables us to capture different aspects of the image. Since the filters are applied linearly to the image (rather than being convolved), a good choice would be a filter bank that captures global information at different levels of details. A simple choice for the filter bank could thus be the set of eigenvectors corresponding to large eigenvalues of the covariance matrix of the training data. Figure 3.2 shows a sample


(a) Basic RBM

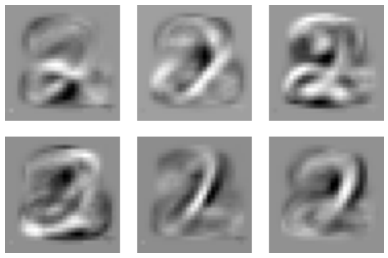

(b) Eigen-RBM

Figure 3.3: The learned filters for single digit training data (digit 2) as computed by Basic RBM and Eigen-RBM. Notice the noisy corners of the learned filters by Basic RBM that comes from the learning procedure.

of this filter bank for hand-written digits of MNIST data set [11]. The RBM with the weights defined as linear combinations of Eigen-filters is Eigen-RBM.

To assess whether Eigen-RBM can produce similar results to basic RBM, which has many more free parameters, both networks are trained with images of handwritten digit **2** taken from the MNIST data set. The filter bank in Eigen-RBM consists of the top 30 eigenvectors. As figure 3.3 shows, with about one twenty-seventh the number of parameters, Eigen-RBM produces similar results to RBM. Although noise-reduction is not the objective, it is interesting to observe the reduction of noise in the corners of the learned Eigen-RBM filters: the Eigen-RBM filters are random combinations of Eigendigits, which are not noisy, whereas the learned filters for basic RBM are initialized with random values.

3.3.2 Connection to PCA

Employing the top eigenvectors of the covariance matrix connects Eigen-RBM to Principal Component Analysis (PCA) [80]. To see this connection, assume a_j to be the input to hidden unit unit h_j. Considering the bipartite structure of RBM, a_j is as follows:

$$a_j \quad = \quad w_{1j}v_1 + w_{2j}v_2 + ... + w_{dj}v_d. \tag{3.5}$$

Defining $W_{:j}$ as a linear combination of a set of existing filters using (3.3), we have:

$$\begin{aligned} a_j &= (\alpha_{1j} f_1^1 + ... + \alpha_{\rho j} f_1^{\rho}) v_1 + (\alpha_{1j} f_2^1 + ... + \alpha_{\rho j} f_2^{\rho}) v_2 \\ &+ ... + (\alpha_{1j} f_d^1 + ... + \alpha_{\rho j} f_d^{\rho}) v_d \\ &= \alpha_{1j}(\mathbf{f}^1 \cdot \mathbf{v}) + \alpha_{2j}(\mathbf{f}^2 \cdot \mathbf{v}) + ... + \alpha_{\rho j}(\mathbf{f}^{\rho} \cdot \mathbf{v}) \end{aligned} \tag{3.6}$$

where "$\cdot$" denotes the inner product. Consequently, the idea of linear combination is equivalent to defining RBM on the linear projection of the data:

$$a_j = \alpha_{1j}\hat{v}_1 + \alpha_{2j}\hat{v}_2 + ... + w_{\rho j}\hat{v}_{\rho} \tag{3.7}$$

where $\hat{\mathbf{v}} = \{\hat{v}_1, \hat{v}_2, ..., \hat{v}_P\}$ is the embedded data into the new lower-dimensional space. Therefore, when $\mathbf{f}^i$s are the top eigenvectors of the covariance matrix, as in Eigen-RBM, it is equivalent to applying PCA on the images and then defining the visible units to be the dimension-reduced data.

3.4 Results and Discussion

We conducted a set of experiments on a small version of MNIST[2] data set of handwritten digits to study the effectiveness of Eigen-RBM compared to that of RBM. The feature learning procedure consists of two stages. In the first stage, the generative weights are learned in a purely unsupervised manner. In the next phase, the learned generative weights are fine-tuned using error backpropagation. The extracted representations are used to classify the test set using a one-nearest-neighbor classifier [81].

In order to determine the optimum number of Eigendigits for Eigen-RBM, we use the Akaike Information Criterion (AIC) [82]. AIC is a model selection criterion to moderate the trade-off between the model complexity and the goodness-of-fit of the model. If the underlying error is independently and normally distributed, AIC is defined as

$$AIC = 2\rho + n \ln(Error_{avg}) \tag{3.8}$$

where ρ is the number of parameters (e.g., here Eigendigits), n is the number of samples and $Error_{avg}$ denotes the average error rate over the sample set [83]. As (3.8) indicates,

[2]A subset of 600 and 100 samples from each digit class is chosen randomly for training and testing, respectively.

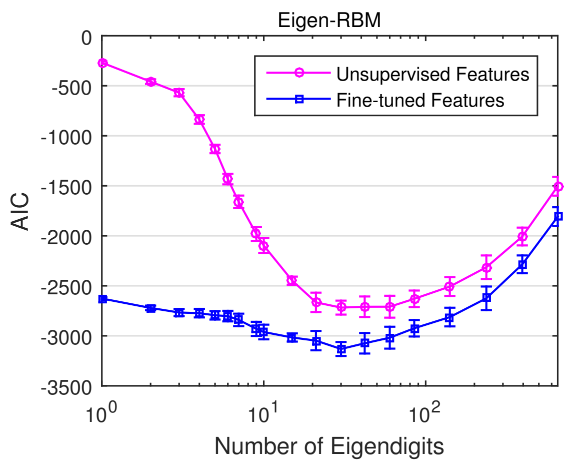
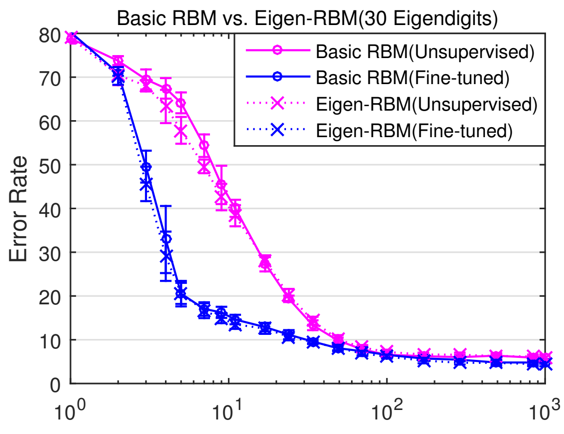


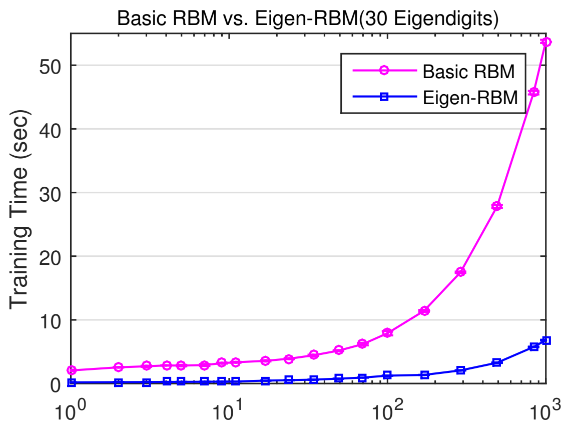
Figure 3.4: The Akaike Information Criterion (AIC) [82] for Eigen-RBM averaged over ten randomly selected subsets of MNIST data sets (6000 training samples and 1000 test samples). Note that for both unsupervised and fine-tuned features, the optimum number of Eigendigits, found by minimizing the AIC, are similar.

AIC rewards a high goodness of fit while it penalizes a large number of free parameters to be estimated. Given a set of models, the model with the minimum AIC is preferred. Figure 3.4 presents the computed AIC for Eigen-RBM for different numbers of Eigendigits, illustrating that the required number of Eigendigits for both unsupervised and fine-tuned models are similar. Based on this criterion, in Eigen-RBM we use the top 30 Eigendigits of the data. Therefore, in the case of MNIST data set with 784-dimensional inputs, the number of free parameters for Eigen-RBM is $30 \times |h|$ while in Basic RBM we have $784 \times |h|$ parameters to learn. As a result, the number of free parameters in Eigen-RBM is near one twenty-sixth the number of the parameters of basic RBM.

Figure 3.5 proposes a comparison between basic RBM and Eigen-RBM in terms of recognition rate and running time. To ensure a fair comparison, both RBM and Eigen-RBM are trained with the same number of epochs. This result demonstrates that with near one twenty-sixth the number of the parameters of basic RBM, Eigen-RBM performs similar to and even better than basic RBM for small number of hidden neurons. As illustrated in figure 3.5 (b), for a large number of hidden neurons—which improves the recognition



(a) Classification Error Rate



(b) Training Time

Figure 3.5: Comparing the basic RBM and Eigen-RBM in terms of classification error rate and training time. For both methods equal number of training epochs has been used. Training time has been computed for 10 epochs of unsupervised training. Observe that the Eigen-RBM has a similar error performance relative to basic RBM (a), but with much less training time (b).

performance—the difference between the training time of basic RBM and Eigen-RBM becomes significant.

It is important to know that learning overcomplete representations[3], such as RBM features, runs the risk of learning trivial solutions [7]. To avoid this problem, one can encourage the features to be sparse so that for each input only a small fraction of hidden neurons becomes active [84, 85]. Without imposing any sparsity penalty, a side-effect of the new weight-learning algorithm is to learn sparse representations. The histogram in Figure 3.6 shows how often the hidden neurons are active for the training images[4]. Evidently, compared to basic RBM, in Eigen-RBM there are many more hidden neurons that are never active or active for a small fraction of time.

For the next step of this comparative study, we take a look at the mind of the network

[3]More sources of features than observations.

[4]This way of plotting the histogram of activations has been previously used in [86] for analyzing sparsity.

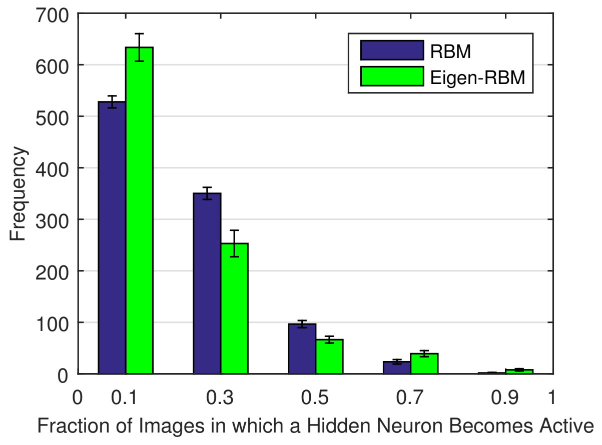


Figure 3.6: A histogram of active hidden neurons averaged over ten random draws of small MNIST data set. The histogram shows the number of hidden neurons that are active for some fraction of the images. The left-most bin shows the number of neurons that are active for 10% of the images, allowing us to conclude that Eigen-RBM representations are sparser than those of basic RBM.

to see what the network believes in. For each class of digits on the complete MNIST data set, an RBM and an Eigen-RBM both with 50 hidden units are trained. After training, starting from a random binary input for the hidden layer, we run alternate Gibbs sampling for 500 iterations. Figure 3.8 shows the generated samples for each class at different Gibbs sampling iterations. As this figure shows, compared to basic RBM, Eigen-RBM, with fewer free parameters, produces similar or better samples (e.g., digit 6, 7 and 8). In our last experiment, the average number of active hidden units is measured on 1000 samples drawn from each class of digits. As figure 3.7 illustrates, for all digit classes, Eigen-RBM generates similar or better samples with more sparse representations than that of basic RBM.

3.5 Summary

The focus of this chapter was on regularizing RBMs with fully connected architectures through providing some prior knowledge about the global characteristics of the images

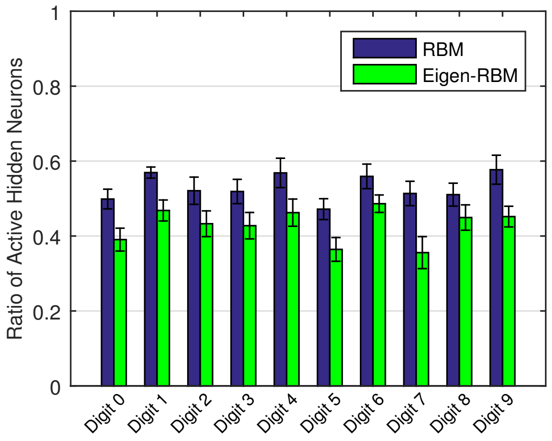
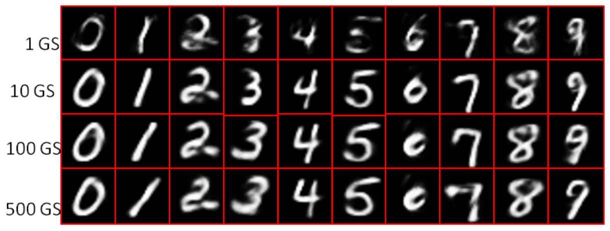
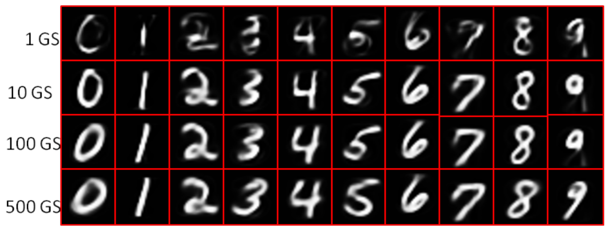


Figure 3.7: Fraction of hidden units which are active for the generated samples of each class using basic RBM and Eigen-RBM. For all digit classes Eigen-RBM generates similar or better samples with more sparse representations than basic RBM.



(a) Basic RBM



(b) Eigen-RBM

Figure 3.8: Generated samples using basic RBM and Eigen-RBM trained on single digit classes. The rows show the sample evolution as a function of Gibbs sampling iterations. Eigen-RBM, with fewer free parameters, produces similar or better samples (e.g., digits 6, 7 and 8).

at different levels of details. We presented Eigen-RBM, with a scalable weight-learning algorithm in which the number of free parameters is independent of the image size. Compared to basic RBM, Eigen-RBM has similar or better performance in both recognition and sample generation, with much less training time. Without imposing any specific sparsity regularization, the new weight learning algorithm leads to more sparse representations, the subject of future work.

It is possible to extend this regularization scheme to convolutional architectures as well. For Convolutional RBMs (CRBMs) [7], since the filters are convolved with the local regions in the image, the filter bank should be able to capture *local* information at different levels of details. In this way, one choice would be the eigenvectors corresponding to the largest eigenvalues of the covariance matrix of the local patches of the training data. Driving a convolutional formulation for PCA, this filter bank for Eigen-CRBM can be generated. In the next chapter, we will follow a related direction and show how to learn image features at different levels of details through multi-neighborhood convolutional architectures in which neighborhoods of different sizes around each pixel are explored.

Chapter 4

Multi-Neighborhood Convolutional Network

In this chapter we explore the role of scale for improved feature learning in convolutional networks. We propose multi-neighborhood convolutional networks which are designed to learn image features at different levels of detail. Utilizing nonlinear scale-space models, the proposed multi-neighborhood model can effectively capture fine-scale image characteristics (i.e., appearance) using a small-size neighborhood, while coarse-scale image structures (i.e., shape) are detected through a larger neighborhood. In addition, we introduce a scalable learning method for the proposed multi-neighborhood architecture and show how one can use an already-trained single-scale network to extract image features at multiple levels of detail. The experimental results demonstrate the superior performance of the proposed multi-scale multi-neighborhood models over their single-scale counterparts without an increase in training cost[1].

[1]The materials in this chapter has been previously published in [87], where the thesis author is the first author.

4.1 Objective

As discussed in section 2.1.2, most of the deep networks used for visual recognition follow a convolutional architecture for feature learning. The conventional convolutional networks used for image classification [54, 56, 88] are *single-neighborhood*, in the sense that for each pixel only a single-size neighborhood centered on that pixel is considered, and *single-scale*, in the sense that only the input image at a single level of detail (i.e., original input representation) is used for extracting features.

However, to be sure, scale-dependent image features significantly predate the emergence of deep learning, and there exist many effective image representation models with strong theoretical and experimental justifications that have been employed in feature extraction. In particular, there is a whole class of multi-scale image models, including wavelet models [34, 46, 89], scale-space models [35, 47, 90], and pyramid representations [36, 48, 91], among many others.

We are motivated to study the intersection of these two fields – multi-scale image representations and multi-layer networks. Multi-layer networks have achieved astonishing learning performance [9, 37, 92], but are for computational reasons limited to learning filters on rather small patches. Since multi-scale image features are sensitive to image structure on all scales and can, in principle, involve relatively large regions of support in their operations, we would like to explore the feature complementarity between the two strategies, in which popular deep convolutional architectures are steered or guided to reflect extracted scale-dependent attributes.

The idea of increasing the size of neighborhood (i.e., learning larger filters) to extract more complex features was examined by Coates et al. [55] and they found that despite the increase in the number of learned parameters, the recognition accuracy is not improved. In this chapter, we show that increasing the neighborhood size improves the recognition accuracy if we use a *multi*-neighborhood architecture. Ciresan et al. [93] introduced the use of different image representations in a multi-column deep neural network for image classification. Although successful, training the large number of parameters associated with the multi-column architecture is computationally expensive. A multi-scale convolutional neural network was introduced by Farabet et al. [23] specifically for scene labeling. Training this model requires the representation of the images at different scales which makes the

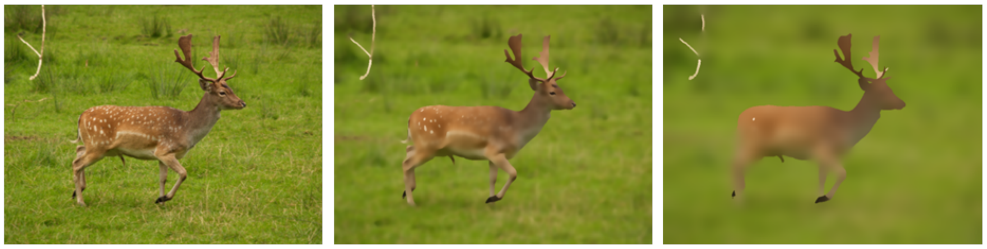


Figure 4.1: Scale space image representations from fine to coarse scale using iterative bilateral scale space [94] with $\mathcal{N} = 4$, $\sigma_{gp} = 0.1$ and $\sigma_{sp} = 3$. Observe the changes in the level of details and how coarse-scale structures are preserved sharp while fine-scale details are filtered away.

learning both computationally and conceptually complex. Furthermore, they extract an equal number of features at different scales. However, there are fewer degrees of freedom presented at coarse-scale image representations and therefore, a fewer number of features should be extracted from those representations.

In this chapter we present an extension of deep convolutional models which can more explicitly capture features at different levels of detail through the use of *nonlinear* scale-space models [94], whereby we show how such a multi-scale model can be modified to a multi-neighborhood architecture. Our proposed multi-neighborhood model captures fine-scale image characteristics (i.e., *appearance* features) through a small-size neighborhood and coarse-scale image characteristics (i.e., *shape* features) by exploring a wider range of dependencies over a larger neighborhood. Finally, we propose a scalable learning strategy for this multi-neighborhood architecture; we will demonstrate that with the same number of parameters as for a single-neighborhood convolutional architecture, we can have a multi-neighborhood model with higher recognition accuracy.

4.2 Background

Originally motivated by biological vision [35], scale-space models [35, 47, 90] are used to perceive or extract image structures at multiple scales. Specifically, a scale-space representation of an image is a set of *transformed* images, each showing structures at a different scale from fine to coarse, normally accomplished via a smoothing-like transformation. Using a linear transformation (i.e., linear scale-space) all of the image regions, including sharp edges, are blurred uniformly, normally an undesirable behavior, certainly in the context of object recognition and shape analysis. In order to keep large-scale regions intact, specifically a preservation of sharp edges, nonlinear scale-space models [94, 95, 96] have been developed, such as the example in figure 4.1.

In this work we employ an iterative bilateral scale-space (IBSS) approach [94] as a nonlinear multi-scale representation. The IBSS approach facilitates the decomposition of details in an image based on a combination of both spatial and photometric differences. As a result, we are able to obtain a nonlinear multi-scale representation where details are well separated and well localized at their respective scales.

Suppose we have a color image I^c, where c represents the channel in what is here an $\{R, G, B\}$ color space. Letting j index the pixel location, then the IBSS multi-scale image representation over S scales can be expressed as

$$\mathcal{I}^c(j) = \left\{ I_0^c(j),\ I_1^c(j),\ I_2^c(j), \ldots, I_{S-1}^c(j) \right\} \tag{4.1}$$

where subscript $0 \leq s \leq S$ represents the scale. The scale set is initialized at the finest scale (i.e., $I_0^c \equiv I^c$) and then recursively expressed over scale:

$$I_s^c(j) = \frac{\sum\limits_{q \in \mathcal{N}} w_{gp}\big(j, \mathcal{N}_q\big) w_{sp}\big(j, \mathcal{N}_q\big) I_{s-1}^c\left(\mathcal{N}_q\right)}{\sum\limits_{q \in \mathcal{N}} w_{gp}\big(j, \mathcal{N}_q\big) w_{sp}\big(j, \mathcal{N}_q\big)}. \tag{4.2}$$

Here $\mathcal{N}_q$ refers to the pixel location within a local neighborhood $\mathcal{N}$, and w_{gp} and w_{sp} are the Gaussian photometric and spatial weights, respectively, on j:

$$w_{gp}\big(j, \mathcal{N}_q\big) = \exp\left[-\frac{1}{2} \left(\frac{\left\| I_{s-1}^c(j) - I_{s-1}^c(\mathcal{N}_q) \right\|}{\sigma_{gp}} \right)^2 \right], \tag{4.3}$$

$$w_{sp}\big(j, \mathcal{N}_q\big) = \exp\left[-\frac{1}{2} \left(\frac{\|j - \mathcal{N}_q\|}{\sigma_{sp}} \right)^2 \right]. \tag{4.4}$$

The parameters σ_{gp} and σ_{sp} denote the scaling constants controlling the weights associated with photometric locality and spatial locality constraints. Therefore, in higher scale representation of an image the value of each pixel is a weighted average of its local neighborhood in the previous scale. From the neighboring pixels, those with closer location and intensity values contribute more in the new value of the target pixel.

4.3 Methodology

As figure 4.1 shows, the finer-scale representations of the image have more information regarding details, such as texture, whereas coarse-scale representations are more suitable for reflecting high-level structures such as shape. It is clear that a wider context than texture and pixel details are required for capturing shape features, therefore coarser scale representations should be further investigated. In order to capture a richer set of features using a multi-scale convolutional architecture, it will be essential to have neighborhoods of different sizes, consequently we need a multi-neighborhood architecture with small/large neighborhoods at fine/coarse scales, respectively. An obvious challenge, however, in the growth of neighborhood size is the increase in the number of model parameters that must be learned, making even more complex an already expensive training procedure.

On the basis of this motivation and computational objection, we introduce two different multi-scale multi-neighborhood architectures. Based on the nonlinear scale-space models discussed in section 4.2, our purpose is to design a multi-scale convolutional architecture with feature detectors sensitive over a range of scales.

Given the input image $I \equiv I_0$, let $I_0, I_1, ..., I_{S-1}$ be the representation of I on the S scales without sub-sampling, that is

$$I_s \in \mathbf{R}^{N\times N} \ \text{ for all } \ s \in \{0, ..., S-1\}. \tag{4.5}$$

Our baseline solution, the *Uniform Multi-Scale* architecture, presented in figure 4.2, is a set of parallel feature extractors each with the same architecture[2]: K filters of size $M \times M$ and pooling size p, applied at each scale I_s. The final feature vector V is obtained by concatenating the extracted features V_s over scales. Essentially this architecture constructs a fixed neighborhood around each pixel at multiple scales.

As equation 4.2 shows, a coarse-scale image representation I_s is basically obtained by eroding small structures from a finer scale I_{s-1}. Keeping this in mind, the coarser the

[2]The architectural details of a basic convolutional layer can be found in section 2.1.2.

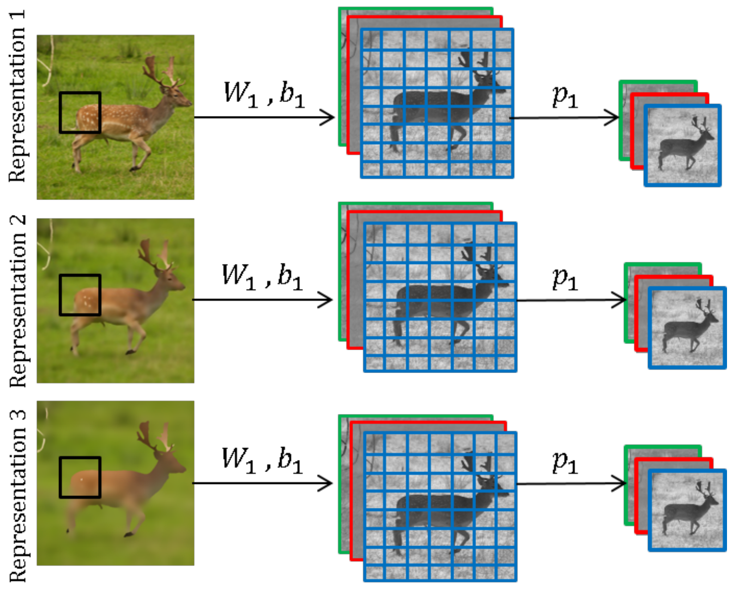


Figure 4.2: Schematic diagram of the uniform multi-scale architecture. Scale-space representations of the input image are passed through a set of parallel networks with uniform architectures.

image scale clearly the less fine structures are present, therefore a greater degree of down-sampling can be applied without losing salient information. Based on this fact, we propose a multi-scale architecture called *Multi-neighborhood in Input*, shown in figure 4.3, in which the input representation at each scale is down-sampled with a different ratio. That is, before feeding I_s to its corresponding network, the image is down-sampled by a factor r_s, where $r_{s-1} < r_s$. Note that except the input, the rest of the network architecture is left unchanged (i.e., K filters of size $M \times M$ and pooling size p at each scale s).

A final strategy is to introduce aspects of scale in pooling. The *Multi-neighborhood in Pooling* architecture is shown in figure 4.4. In this model, similar to the uniform multi-scale architecture, all of the input images I_s are of the same $\mathbf{R}^{N \times N}$ size, and are fed into S parallel networks each with K filters of size $M \times M$; the difference appears in the differing pooling sizes where $p_0 < p_1 < ... < p_{S-1}$.

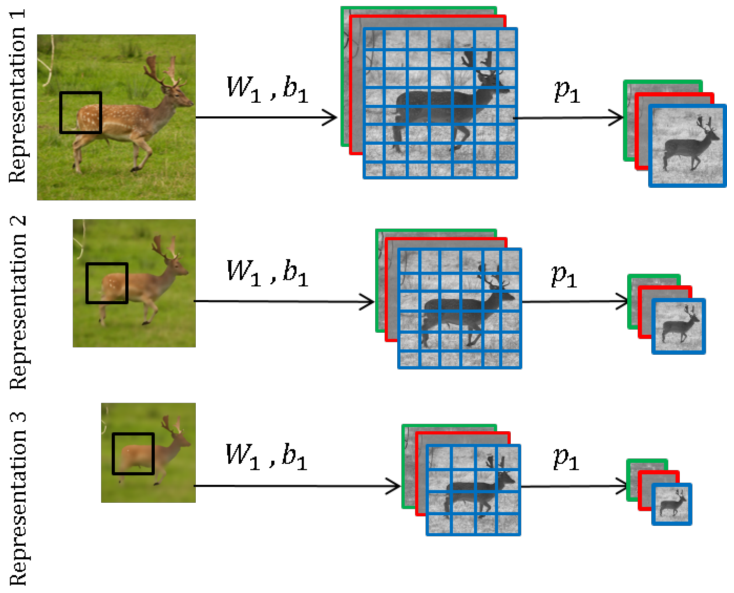


Figure 4.3: Schematic diagram of the multi-neighborhood in input architecture. Scale-space representations of the input image are passed through a set of parallel networks with larger neighborhood size for coarser scales by sub-sampling the input image.

4.4 Results and Discussion

In this section we discuss the process of efficiently training the three proposed multi-scale architectures for feature extraction and then examine their performance, compared to each other and to an equivalent single-scale architecture. For the experiments the standard CIFAR10 data set [6] is used, consisting of colored images of size 32×32 for ten classes of objects, each having 5000 training samples and 1000 test samples. Employing IBSS, the representations of images at five different scales are generated (i.e., $S = 5$). More explicitly, considering a local neighborhood of size of 4 ($\mathcal{N}$ in equation 4.2) and $\sigma_{gp} = 0.1$ (equation 4.3) and $\sigma_{sp} = 3$ (equation 4.4), the representations of image at scales $0, 3, 5, 7, 13$ are used, the scales chosen as being representative of the original images at a variety of meaningfully–different levels of detail from fine to coarse.

The three proposed multi-scale architectures are analyzed for both unsupervised and

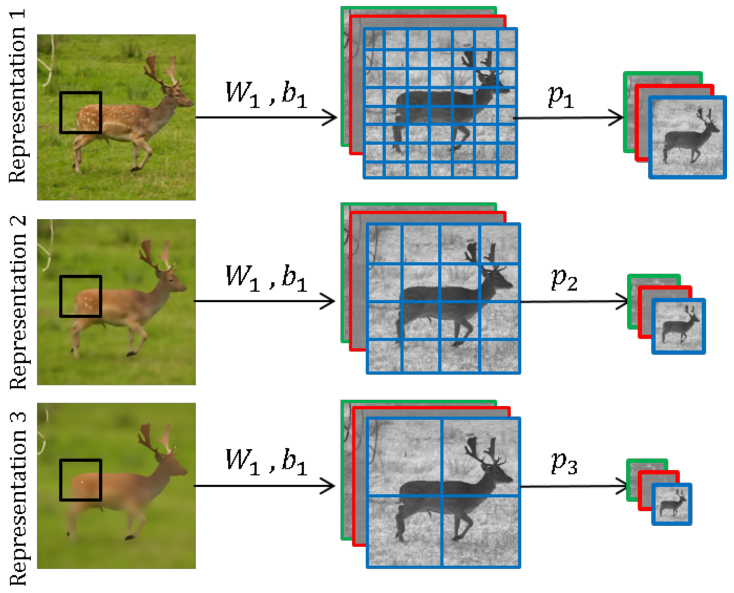
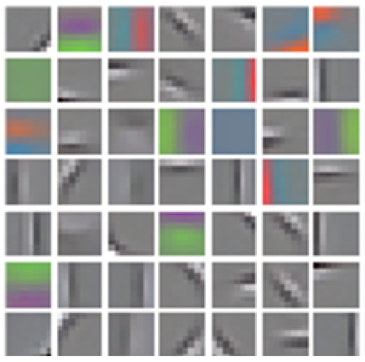
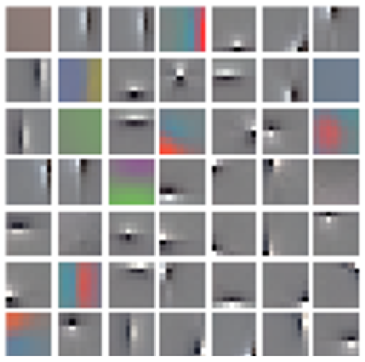


Figure 4.4: Schematic diagram of the multi-neighborhood in pooling architecture. Scale-space representations of the input image are passed through a set of parallel networks with a larger pooling size for coarser scales.

supervised feature learning. In unsupervised feature learning, similar to Coates et al [55], K-means clustering is used to learn a filter bank of size 8, which is then used in a single-layer convolutional model with max-rectifying nonlinearity and max-pooling of size 4×4 with stride 4. For supervised feature learning, at each scale a CNN with two layers of convolution is used. Each convolutional layer uses 64 filters of size 5×5, max-rectifying nonlinearity and pooling of size 3×3 with stride 2. The pooling function for the first and second layer are max and average, respectively. Note that for multi-layer feature learning architectures, down-sampling of the input is only applied to the first layer and larger pooling of the output is only applied to the last layer. After training each architecture, the extracted features are passed to an L2-SVM classifier with the regularization parameter determined by cross-validation.


(a) Uniform
Scale 0


(b) Uniform
Scale 5

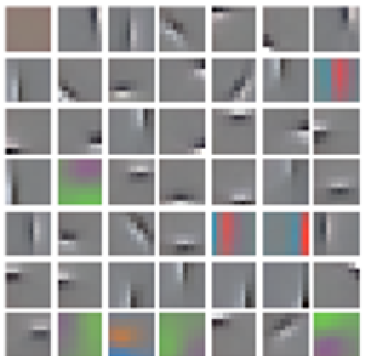

(c) Sub-sampled Input
Scale 5

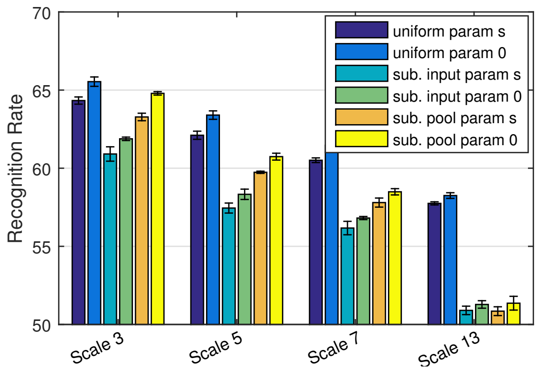
Figure 4.5: Visualization of the learned filters on CIFAR10 using single-scale networks across different scales. We observe that input sub-sampling (c) leads to learned edge-detectors with a wider range of angles compared to learning without sub-sampling (b). The learned filters in (c) are similar to those in (a), primarily differing in size.

4.4.1 Scalable Learning

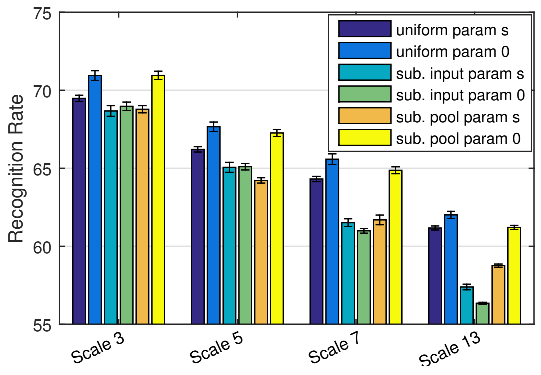
As discussed in section 4.3, the proposed multi-scale architectures are composed of a set of parallel feature extractors applied to the representations of the input image at an ensemble of scales. The remaining question, then, is how to train these parallel single-scale networks.

A straightforward solution is to train each of these single-scale networks using the image representation at the corresponding scale. Figure 4.5 shows the unsupervised learned filters on CIFAR10 using the original image representation (i.e., scale 0) versus a coarse scale input representation (scale 5) with and without input down-sampling. Two immediate observations are:

1. For the coarse-scale input representation (figure 4.5b and 4.5c), input sub-sampling (i.e., a larger neighborhood) leads to learned edge-detectors with a wider range of angles compared to those learned without sub-sampling.
2. The learned filters using a coarse-scale input representation with a larger neighborhood (figure 4.5c) are similar to those of scale 0 (figure 4.5a), primarily differing in size.



(a) Unsupervised Feature Learning



(b) Supervised Feature Learning

Figure 4.6: Comparison of the learned features on CIFAR10 for single-scale networks of each architecture across different scales using two training methods: 1) learning the parameters of each single-scale network using the input representation at the corresponding scale (noted by "param s" in the legend), and 2) reusing the learned parameters at the original image representation (noted by "param 0" in the legend). Notice the comparable or superior performance of the latter strategy for most of the cases except for supervised training with sub-sampling in input at scales 7 and 13.

On that basis, one clear strategy to train a set of parallel single-scale networks is to train the network for scale 0 (i.e., the original input representation) and then to reuse its learned parameters for all of the single-scale networks. Figures 4.6 offers a comparative study of two training methods for feature learning using a single-scale network:

1. Preserving those learned parameters from the original image representation. In this case, 100 filters are learned in total.
2. Learning the parameters of each single-scale network using the input representation at the corresponding scale. In this case, 100 filters are learned at each scale.

We observe that for these single-scale networks with different architectures, the preservation of the learned parameters at scale 0 leads to comparable or superior performance for all of the 12 cases in unsupervised learning and for 10 out of 12 cases in supervised learning. For supervised training with sub-sampling in input, the learned parameters at scales 7 and 13 produce better recognition rates compared to the learned filters at scale 0.

As a result, we have a plausible approach for scalable learning for the proposed multi-scale architectures. We do not, in fact, need to train each single-scale block independently. Instead, learning takes place at only scale 0 and then is propagated to the remaining scales. Therefore, the computational complexity of training a multi-scale architecture is unchanged from single-scale networks. Note that at the test time we still need to generate the multi-scale representations of the data and pass the input image through multiple parallel networks. Therefore, the test time complexity increases compared to a single-scale network. In particular, the computational complexity at the test time increases almost linearly with the number of scales.

4.4.2 Classification Results

Given the scalable learning algorithm in section 4.4.1, we evaluate the effectiveness of the proposed architectures for image classification. First we focus on unsupervised feature learning. Figure 4.7 shows the recognition performance of different architectures on the CIFAR10 data set as a function of the number of learned filters. Observe that, with the same training cost, the three multi-scale architectures usually outperform the single-scale counterpart, clearly demonstrating the advantage of extracting scale-dependent or detail-dependent image representations. Among the multi-scale models, both of the multi-neighborhood architectures (i.e., in input and in pooling) offer a better performance than the uniform model, especially for larger numbers of filters. Therefore, these multi-neighborhood architectures create the opportunity to avoid learning a larger number of filters for a single-scale network, and instead obtaining the same or better classification rate with a smaller number of filters.

A second test examines the proposed multi-scale architectures for supervised feature learning. Table 4.1 illustrates the positive effect of extracting features at multiple scales. Among the proposed multi-scale models, the multi-neighborhood architectures generate more discriminative features due to exploring a wider range of dependencies over a larger neighborhood. Also, for multi-scale models observe how the scalable learning strategy substantially reduces the training cost (i.e., the number of learned filters) while the recognition

CNN Architecture	No. of Scales	Learned Filters	Accuracy %
Single Scale: scale 0	1	128	74.98±0.29
Uniform Multi-scale*	5	128	75.45±0.11
Uniform Multi-scale	5	128× 5	76.08±0.43
Multi-neigh. Input*	5	128	76.20±0.33
Multi-neigh. Input	5	128× 5	76.74±0.21
Multi-neigh. Pooling*	5	128	76.01±0.27
Multi-neigh. Pooling	5	128× 5	76.43±0.09

Table 4.1: Recognition rate for the CIFAR10 data set using different two-layer CNN architectures. For the starred architectures the CNN filters are learned only at the original scale and then applied to all scales.

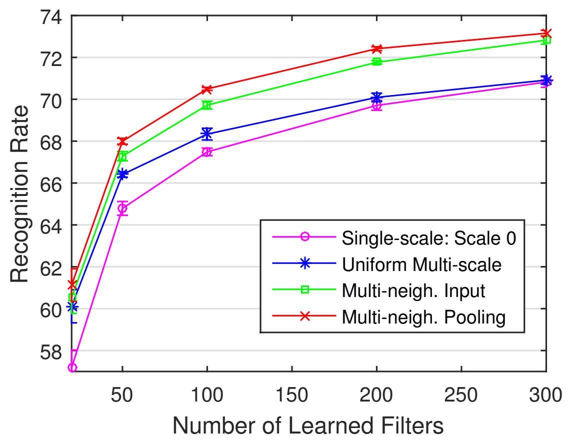


Figure 4.7: Performance of unsupervised feature learning on the CIFAR10 data set using different architectures as a function of the number of hidden neurons. Observe how utilizing scale-scale representations and multi-neighborhood architectures improves the recognition performance.

rate is still superior to that of their single-scale counterpart.

Finally, we evaluated the robustness of the proposed architectures to additive Gaussian noise. As figure 4.8 shows, all of the three proposed multi-scale architectures are more

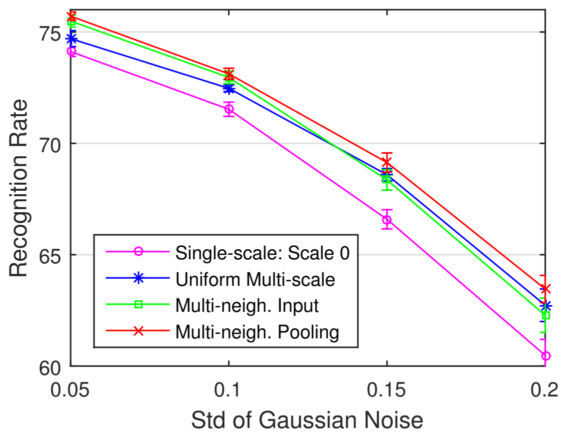


Figure 4.8: Performance of supervised feature learning on the noisy CIFAR10 data set as a function of the standard deviation of the Gaussian noise. Notice the superior performance of multi-scale architectures in dealing with noise.

robust to noise than the classic single-scale strategy, particularly for higher levels of noise.

4.5 Summary

This chapter was focused on improved feature learning using convolutional networks through providing some prior knowledge about the multi-scale representation of the input image. We presented three multi-scale convolutional architectures, designed to learn image features corresponding to different levels of detail. Utilizing nonlinear scale-space models, we showed that a multi-neighborhood architecture can effectively capture fine-scale and coarse-scale image characteristics. Our proposed scalable learning method allows a multi-scale architecture to be based on an already-trained network. That is, the learning is performed only on the original scale, but then the learned parameters are applied to an ensemble of networks at different scales. Experiments on the CIFAR10 dataset illustrated that multi-scale multi-neighborhood models outperform their single-scale counterparts without learning new feature detectors for the higher scale representations of the data.

The concentration of this work was on fusing the extracted *features* in each scale for improved recognition. As a future direction, one can study the performance boost when the *decisions* of the classifiers for each scale are combined as apposed to fusing the features. Also, we believe that the real power of the proposed architectures emerges on much larger input images where more detailed information is presented and thus the image can be represented in more (different) levels of abstraction, the subject of future work.

Employing a multi-neighborhood architecture, similar to other ensemble learning schemes for deep networks, increases the required amount of memory and computation at the test time. One way to tackle this problem is to transfer the learned knowledge by the ensemble into a smaller network [97, 98]. Another strategy would be a sequential training procedure. That is, the found solution by a trained network using the coarse-scale properties of the data can be used to regularize the subsequent fine-scale learning process. In this sequential training procedure, it is crucial to attain a good generalization in each stage to keep the subsequent stages away from over-fitting. Motivated by this necessity, before introducing our stage-wise training framework in Chapter 6, a new regularization scheme for improving the generalization performance of deep networks will be presented in the next chapter.

Chapter 5

Periodic Regularization for Training Deep Models

Weight decay [67, 68] is arguably the most widely used general-purpose regularizer in training neural networks. Despite its popularity, we show that applying this penalty function on a network layer decreases the speed of learning in its previous layers. However, in a deep network, earlier layers already suffer from slow-paced learning due to the so called problem of *diffusion of gradients*. This issue becomes more critical when we notice practitioners usually apply a larger weight decay to higher layers to avoid over-fitting. Motivated by this problem, we propose Periodic Weight Decay (PWD) in which one can determine *how hard* and *how frequently* the weights should be shrunken. In contrast to basic weight decay, PWD is not a part of the objective function and thus it avoids shrinking the parameters at every single update step. The experimental results on a number of image classification data sets authenticate the faster convergence and better generalization of PWD over basic weight decay.

5.1 Objective

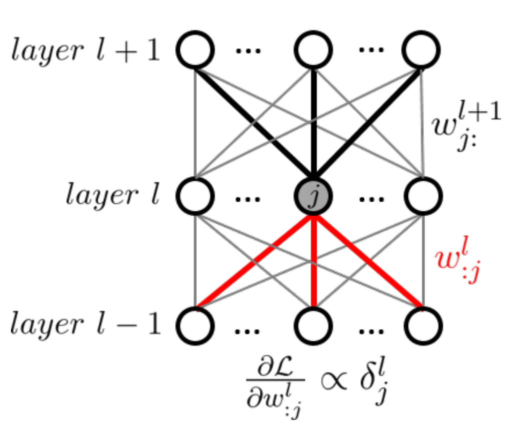
The remarkable success of deep learning in computer vision applications [6, 9, 23] can be attributed to two types of factors. The first type includes the *general* properties of deep models as a powerful off-the-shelf framework for learning: hierarchical architecture [10], flexible structure with outstanding representational power (e.g., different types of layers [56], units [18, 92] and connectivity [11]), efficient training using parallel processing and distributed computing, and desirable statistical properties when used along with regularization techniques (i.e., avoid over-fitting despite the huge degrees of freedom) [37, 71]. The second set of factors are the *domain specific* adjustments that are made to deep models in order to customize them for computer vision tasks. These arrangements include the use of pooling [7, 56, 57], convolutional weight-sharing [11, 54, 99], data augmentation [9, 69, 100], and multi-stage training [17, 21, 101].

In addition to the above adjustments, as discussed in section 2.3, in supervised training of a deep network with a large number of parameters, it is important to limit the degrees of freedom in the parameter space to avoid over-fitting to the training data. In order to address this problem, various general-purpose regularizers (e.g., Tikhonov regularization [67], lasso [68] and dropout [37]) and domain specific priors (e.g., convolutional weight-sharing [11], transformation invariance [7] and robustness to partial corruption of the input [69]) have been proposed.

Among the general-purpose regularizers, weight decay [67, 68] is arguably the dominant method in the statistics community. In neural networks, it is commonly used [102, 103, 104] as an important ingredient in addition to other regularization techniques [37, 71]. While the statistical advantages of weight decay are promising [65], one needs to be cautious about the computational side-effects of its use. As these regularizers are convex, they can safely be used in convex optimization problems [105]. However, this is not necessarily the case in non-convex problems such as the training of neural networks.

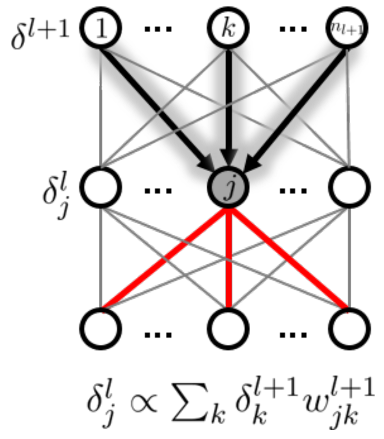
The optimization method used in neural networks is (stochastic) gradient descent [106] in which the parameters are updated by *back-propagating* the error from the output layer to the earlier layers of the network[1]. An important issue with this method — especially for deep networks — is the so called *diffusion of gradients* phenomenon, in which the error is progressively vanishing as it is back-propagated to the earlier layers of the network [38]. As a result, the gradient of the objective function with respect to the weights of the earlier

[1]SGD is the most widely used method in training deep neural networks since it learns good parameters much more quickly compared to far more elaborate optimization techniques [107].



layer $l+1$
layer l
layer $l-1$
j
$w_{j:}^{l+1}$
$w_{:j}^{l}$
$\frac{\partial \mathcal{L}}{\partial w_{:j}^{l}} \propto \delta_j^l$

(a)



(b)

Figure 5.1: Schematic diagram for diffusion of gradients. As figure (a) shows, the amount of update (i.e., learning) for $w_{:j}^{l}$ is proportional to the size of the error received by neuron j in layer l, δ_j^l. Figure (b) demonstrates that this error is directly proportional to the weighted summation of the errors in the above layer back-propagated through $w_{j:}^{l+1}$. Therefore, small size for $w_{j:}^{l+1}$, encouraged by weight decay, leads to slow learning in $w_{:j}^{l}$.

layers becomes small and a minor update takes place for those layers[2]. This problem could become more severe when a weight decay penalty is employed. As figure 5.1 shows, the amount of error that is back-propagated from a layer to its previous layer is proportional to the size of its weights[3]. On the other hand, a weight decay regularizer—which will be formally described in section 5.2—makes sure that the sizes of the parameters (i.e., their norm) are small. Consequently, weight decay contributes to the diffusion of gradients. This issue becomes more critical when we notice practitioners usually use larger weight decay penalties for higher layers to over-fitting.

To put it in a nutshell, in training deep networks there is a regularization-optimization trade-off with respect to the weights' sizes: on the one hand the weights should be kept small to avoid over-fitting, on the other hand, having small sized weights contributes to

[2]See section 2.2.3 for a detailed description of the diffusion of the gradients issue

[3]For example, if all of the weights in a layer are zero, then the error is not back-propagated to the previous layers.

the issue of gradient vanishing.

Motivated by this problem, we propose a new weight decay framework for regularizing deep neural networks, called Periodic Weight Decay (PWD), in which the regularizer is no longer a part of the objective function. Instead, it acts as a second update step which is applied periodically during the optimization. We can therefore decide *how hard* and *how frequently* the weights should be penalized. This flexibility is especially useful for the stochastic gradient descent method; given a decay frequency, the weights are regularized in *some* of the update steps. Therefore, during the rest of the update steps they are allowed to grow large enough to pass the gradient properly. Experimental results on a number of well-known image classification data sets authenticate the faster convergence and better generalization of PWD over basic weight decay.

5.2 Background

Training a feedforward neural network is a supervised learning procedure in which the optimum value for the network parameters W is estimated through minimizing a non-convex cost function $\mathcal{C}(W)$. Given a loss function $\mathcal{L}(W)$ and a regularization term $\mathcal{R}(W)$, $\mathcal{C}(W)$ is defined as

$$\mathcal{C}(W) = \mathcal{L}(W) + \mathcal{R}(W). \tag{5.1}$$

To prevent the network from over-fitting, one of the most commonly used regularization techniques is weight decay or $L2$ regularization which is defined as

$$\mathcal{R}(W) = \frac{\lambda}{2} \|W\|_2^2 \tag{5.2}$$

where $\|.\|_2$ denotes the $L2$ norm and the regularization coefficient λ determines how greatly the parameters should be penalized. Weight decay pushes the weights toward zero and so, as (5.1) shows, the cost function has to compromise between minimizing the network loss and finding small weights. As a result, the learner chooses the smallest weight vector that solves the problem and hence irrelevant components of the weight vector are suppressed [108]. Moody et al. [108] theoretically showed that having small weights prevents the network from learning the presented noise in the data and hence improves generalization. In other words, it smooths out the behaviour of the network in response to small changes in the input. Besides, there is a generalization error bound for neural networks based on the size of the parameters which advocates the use of small weights [65].

To locally minimize the non-convex cost function (5.1) regularized with weight decay, stochastic gradient descent (SGD) [39] is used and the parameters are updated based on the following rule:

$$\begin{aligned} w^{t+1} &= w^t - \gamma^t \frac{\partial \mathcal{L}}{\partial w} - \gamma^t \lambda w^t \\ &= (1 - \gamma^t \lambda) w^t - \gamma^t \frac{\partial \mathcal{L}}{\partial w} \end{aligned} \tag{5.3}$$

where γ^t is the learning rate for the update step t. As (5.3) shows, before subtracting the gradient, the weights are rescaled by a shrinkage coefficient of $(1 - \gamma^t \lambda)$ which justifies its appellation, *weight decay*. Also, note that this shrinkage coefficient becomes less significant (i.e., the penalty becomes more relaxed) during training since as a common practice the learning rate γ^t follows an annealing schedule and decreases during learning. The effect of this decrease in the dominance of the regularizer is studied in section 5.4.2.

In the basic framework for weight decay, since the regularizer is part of the objective function, the parameters are downscaled at every single update step. This continuous suppression of the weights can exacerbate the known issue of diffusion of gradients [38] in training deep networks. Assume that w^l_{ij} connects neuron i in layer $l-1$ to neuron j in layer l. Considering the gradient descent update rule (5.3), the amount of update (i.e., learning) for w^l_{ij} depends on the gradient of the loss function with respect to this weight. This gradient is proportional to the amount of error[4] *received* by neuron j in layer l, δ^l_j, in error back-propagation. Therefore, when the magnitude of this received error is small, w^l_{ij} learns slowly. On the other hand, the amount of the error received by neuron j is directly proportional to the size of weights which connects neuron j in l to all of the neurons in its above layer (i.e., $w^{l+1}_{j:}$). More formally

$$\frac{\partial \mathcal{L}}{\partial w^l_{ij}} \propto \delta^l_j \propto \sum_k \delta^{l+1}_k w^{l+1}_{jk}. \tag{5.4}$$

[4]The difference between the true and the estimated target that is defined based on the loss function $\mathcal{L}$. See section 2.2.2 for more details.

According to (5.4), a small value for $w_{:j}^{l+1}$ reduces the amount of error received by neuron j in layer $l+1$ and consequently the gradient of the loss function with respect to w_{ij}^{l}. Based on this result, applying weight decay on the weights of a layer reduces the gradient of the loss function with respect to the parameters of its lower layer. In other words, decaying the weights in a layer reduces the speed of learning in its earlier layers, especially when this suppression occurs at each single update step. Considering the fact that in deep networks earlier layers already suffer from slow-paced learning due to diffusion of gradients (i.e., lower layers receive smaller gradient compared to the top layers of a network), exacerbating this issue by applying a continuous penalty is not desirable.

5.3 Methodology

5.3.1 Periodic Weight Decay

Periodic weight decay (PWD) is a generalization of basic weight decay in which one can determine *how frequently* the weights should be shrunken. This flexibility in deciding the frequency of the decay is designed to address the issue of diffusion of gradients in basic weight decay. The main idea behind PWD is to avoid shrinking the parameters at every single update. Instead, the weights are allowed to grow for certain number of update steps until a (more severe) penalty is applied. Therefore, during the initial update steps the gradient is not affected by the regularizer and is properly passed backward to the earlier layers. Then, when the parameters at the lower layers have grown larger and started moving towards a local minimum, a relatively harsher penalty can be applied without reducing the speed of learning.

Since the penalty is not applied in all of the update steps, the regularizer should be disentangled from the loss minimizer in the objective function. Therefore, each update step is divided into two stages. In the first stage, the weights are updated merely based on the gradient of the loss function:

$$\hat{W}^{t+1} = W^t - \gamma^t \nabla_w \mathcal{L} \tag{5.5}$$

The second stage is focused on regularizing the parameters and solves a minimization problem to find a small-sized weight vector W^{t+1} which is close to $\hat{W}^{t+1}$, the solution of the first stage:

$$W^{t+1} = \operatorname{argmin} \frac{1}{2} \left\| W^{t+1} - \hat{W}^{t+1} \right\|_2^2 + \frac{\lambda}{2} \left\| W^{t+1} \right\|_2^2 \tag{5.6}$$

This is a simple optimization problem and has a closed-form solution. Setting the gradient to zero and substituting $\hat{W}^{t+1}$ from (5.5), the updated value for the weight vector W becomes

$$\begin{aligned} W^{t+1} &= \beta^t \hat{W}^{t+1} \\ &= \beta^t \left[W^t - \gamma^t \nabla_w \mathcal{L} \right], \end{aligned} \tag{5.7}$$

where β^t is defined as

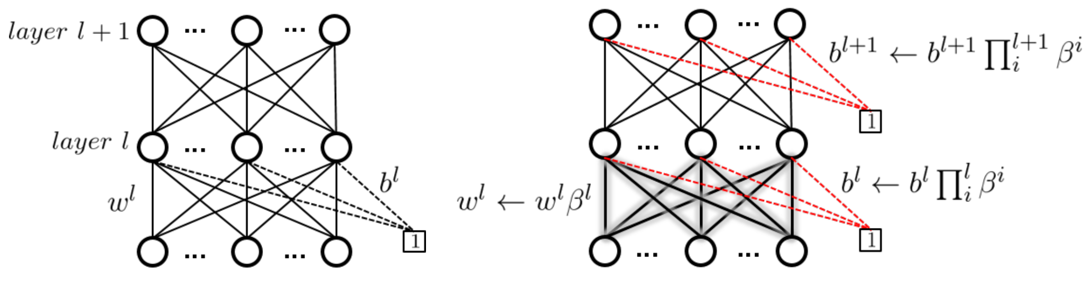
$$\beta^t = \begin{cases} \frac{1}{1+\lambda} & \text{if } t \bmod \left\lfloor \dfrac{n_B}{f} \right\rfloor = 0 \\ 1 & \text{otherwise.} \end{cases} \tag{5.8}$$

Here, f is the frequency of applying the regularizer in each training epoch and n_B is the total number of update steps (i.e., training batches) per epoch. That is, PWD pushes the weights towards zero with the decay coefficient β and the frequency of f during each training epoch. Also, note that, in contrast to basic weight decay, since the decay coefficient β is not a function of the learning rate, the dominance of the penalty does not necessarily decrease during training.

5.3.2 Extension to Multiple Layers

As demonstrated in (5.5) and (5.6), when PWD is applied, the parameters are updated during two sub-steps. The first update is in favor of minimizing the loss while the second update is focused on penalizing the solution of the first stage. Taking into account this decoupling, it is important for the regularization step to not to disrupt the effect of the loss minimization step. In order to fulfill this requirement, one way is to make sure that applying the regularization step does not change the network's predicted labels.

Interestingly, for neural networks with rectified linear units (ReLu) [18]— the most-widely used activation function in deep learning literature—PWD can be applied to any



$layer\ l+1$
$layer\ l$
w^l
b^l
1
(a)
$b^{l+1} \leftarrow b^{l+1} \prod_i^{l+1} \beta^i$
$b^l \leftarrow b^l \prod_i^l \beta^i$
$w^l \leftarrow w^l \beta^l$
1
(b)

Figure 5.2: Schematic diagram of applying PWD on layer l of the network. The state of the network before and after applying the decay has been depicted in (a) and (b), respectively. The solid lines are used to show the weights (denoted by w^l) and the dashed lines indicate the biases (denoted by b^l). Notice the difference between the update rule for the weights and the biases. Decaying the weights of each layer (i.e., shaded lines) requires decaying the biases of that layer and all of its upper layers (i.e., red lines).

of the network layers without changing the predicted labels[5]. Assume that β^l is the PWD coefficient for regularizing layer l. After computing $\hat{W}^{l,t+1}$ and $\hat{b}^{l,t+1}$ using (5.5), PWD is applied to the weights as

$$W^{l,t+1} = \beta^{l,t}\hat{W}^{l,t+1}. \tag{5.9}$$

With shrinking $\hat{b}^{l,t+1}$ in the same way, the input to layer l is scaled down by factor $\beta^{l,t}$. If this shrinkage is transferred linearly to all of the above layers, the final output will be scaled down linearly and thus the network's prediction will not be affected. To achieve this, the biases should be penalized[6] as

$$b^{l,t+1} = \prod_{i=1}^{l} \beta^{i,t}\hat{b}^{l,t+1}. \tag{5.10}$$

Informally, decaying the weights of layer l requires decaying the biases of layer l and all of its above layers (see figure 5.2).

[5]This property approximately holds for sigmoid networks if downscaling of the weights does not change the status of the neurons (i.e., linear, saturated or inactive).

[6]In the case of basic weight decay, usually the biases are not penalized [109, 110].

Algorithm 1 Backward Pass in SGD Training with PWD

Input: parameters $[W, b]$, layer index l, total number of layers L, weight decay magnitude β, weight decay frequency f, training epoch index *epoch*, batch index *batch*, total number of batches n_B, learning rate γ.

Output: update parameters $[W_{new}, b_{new}]$.

```
 1: Update Weights: $W_{new} \leftarrow W - \nabla_w \mathcal{L}$
 2: Update Biases: $b_{new} \leftarrow b - \nabla_b \mathcal{L}$
 3: Step size: $s \leftarrow \left\lfloor \frac{n_B}{f} \right\rfloor$
 4: Current Step: $t \leftarrow (epoch - 1) \times n_B + batch$
 5: if $t \mod s$ is 0 then
 6:     Regularizing Weights: $W^l_{new} \leftarrow \beta^l W^l_{new}$
        Regularizing Biases:
 7:     for all layer $i \in [l, L]$ do
 8:         $b^i_{new} \leftarrow \beta^l b^i_{new}$
 9:     end for
10: end if
```

The PWD procedure is summarized in Algorithm 1.

5.3.3 Related Work

In the optimization literature, there is a body of work on disentangling the regularization term from the objective function and applying the penalty in a separate step. Among them one can name projected gradient methods [111, 112], truncated gradient [113] and forward-backward splitting [105]. These methods are designed for convex loss and regularization functions and are mostly focused on promoting sparsity for batch settings and online learning. In contrast, PWD is motivated by the issue of diffusion of gradients in training of multi-layer neural networks where the objective function is non-convex. Furthermore, PWD is distinguished with its periodic nature; it does not suppress the weights at every update step. That is, to tackle the gradient vanishing problem (see equation (5.4)), a frequency parameter is introduced in the penalty function so that the weights are decayed only in *some* of the update steps. This periodic scheme can be applied to max-norm regularizers [114] and other weight constraints as well.

5.4 Results and Discussion

In this section the efficiency of PWD in regularizing neural networks for image classification is examined. First, the performance of PWD for various parameter configurations is analyzed. Then, a comparative study with basic weight decay is conducted where their speed of convergence and generalization performance are explored.

For the experiments, three different image classification data sets are used: CIFAR10 [6], STL10 [115] and Street View House Numbers (SVHN) [116]. CIFAR10 consists of 32×32 colored images of ten classes of objects, each having 50000 training samples and 10000 test samples. STL10 is similar to CIFAR10 but with much fewer labeled training data, 500 training and 800 testing samples per class[7]. Here, the down-sampled version of STL (i.e., 32×32) is used. For SVHN data set, the goal is to recognize the digit at the center of each 32×32 RGB image and it includes 73257 training images and 26032 testing images[8].

The convolutional neural networks used for the experiments have similar hyper-parameters to [117]: convolutional layers with filters of size 5×5 and max-rectifying non-linearity followed by pooling of size 3×3 with stride 2. In the first layer max-pooling is used, whereas in the other layers average-pooling is adopted. In order to specify the number of layers, filter for convolutional layers and neurons in fully-connected layers a convention is used. For instance, 64C-64C-10F stands for a CNN with two convolutional layers, each with 64 filters, and a fully connected layer of size 10 neurons as the last layer. For these experiments PWD is applied only to the fully connected layers which are more prone to over-fitting. However, it can be used for convotional layers as well.

5.4.1 Analysis

As discussed in section 5.2, there is a direct relationship between the issue of gradient diffusion and frequent decay of the weights. A follow-up question would be whether a

[7]For STL10 the unlabeled training set is not used for training.

[8]Note that for SVHN, the extra training images are not used.

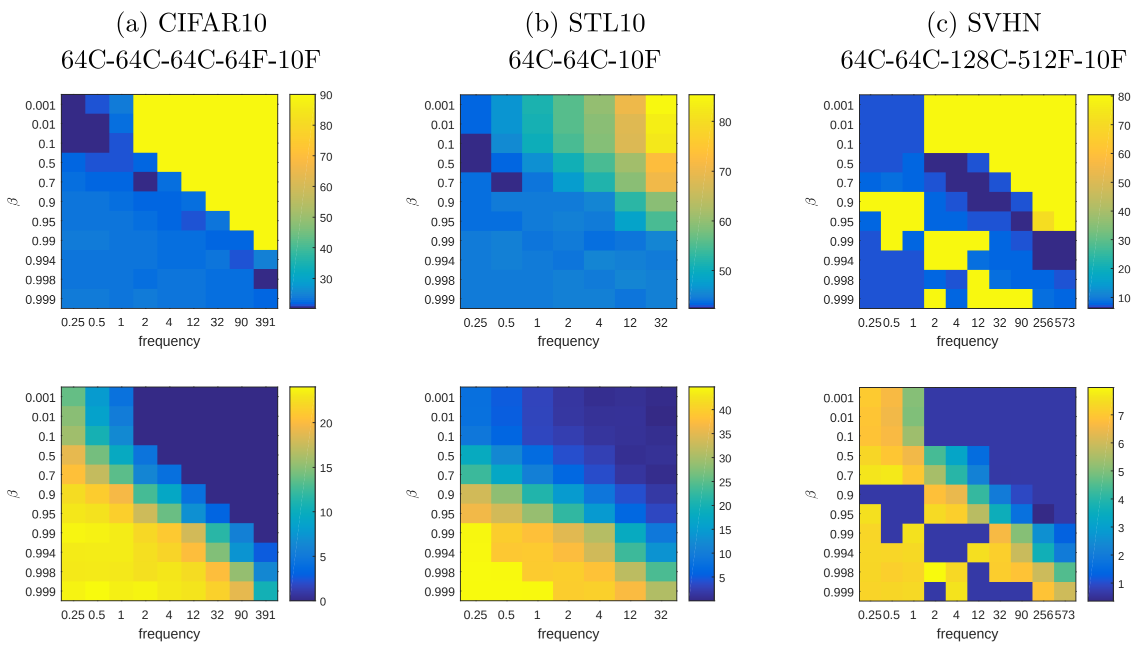


Figure 5.3: Classification performance for various values of PWD parameters for different data sets and network architectures. The top row shows the classification error rate and the bottom row displays the difference between train and test error as a measure of generalization. Note that in all cases the superior performance (i.e., dark blue pixels) is observed around the main diagonal. Also, these plots illustrate that the parameter configurations above the main diagonal lead to under-fitting while those below the main diagonal are more prone to over-fitting and explosion of the gradient (see column (c)).

greater but less frequent penalty can compensate for a smaller but more frequent one. Table 5.1 displays the classification performance of a 64C-64C-10F CNN on CIFAR10 data set as a function of the PWD parameters: the decay coefficient β and the decay frequency during each training epoch. As one moves down through the table rows, the decay coefficient becomes less significant (a larger beta corresponds to a less significant decay). Also, the first column of the table stands for applying PWD once per four epochs while the last column corresponds to applying PWD at every single update at each epoch. Considering the performance in each column, less significant penalty (i.e., moving down through the rows) results in a larger difference between the test and train error, i.e.,

Table 5.1: CIFAR10 classification performance for various values of PWD parameters (i.e., the decay coefficient β, and the decay frequency during each training epoch). The first and last columns correspond to applying the weight decay once per four epochs and at every single update step, respectively. The table shows the test and the train error for a $64C - 64C - 10F$ network architecture and in each row the cell with lowest error rate is shaded. The results for this data set advocates the use of a greater but less frequent penalty.

β \ Freq.	0.25	0.5	1	2	4	12	32	90	391
0.001	20.94 (14.49)	20.92 (15.43)	21.28 (16.45)	22.98 (19.65)	25.22 (22.79)	29.99 (28.38)	36.65 (35.94)	46.08 (46.25)	68.22 (68.47)
0.01	20.93 (14.41)	20.90 (15.36)	21.22 (16.44)	22.93 (19.64)	25.28 (22.61)	29.96 (28.43)	36.87 (35.97)	45.90 (46.04)	68.10 (68.56)
0.1	20.75 (13.65)	20.70 (14.72)	21.10 (15.94)	22.85 (19.52)	25.01 (22.41)	29.49 (27.98)	35.86 (34.88)	44.94 (44.94)	64.14 (64.76)
0.5	21.37 (9.00)	20.86 (10.49)	20.88 (13.77)	21.87 (17.27)	23.41 (20.12)	26.96 (25.07)	30.89 (30.07)	44.80 (44.46)	64.99 (65.14)
0.7	22.63 (6.22)	21.56 (8.70)	21.23 (11.86)	21.31 (15.69)	22.34 (17.83)	25.20 (22.69)	28.36 (27.32)	37.24 (36.80)	56.91 (56.97)
0.9	24.46 (5.77)	23.78 (7.57)	23.04 (8.87)	21.33 (11.79)	21.24 (14.23)	22.33 (17.72)	24.65 (22.08)	28.61 (27.56)	40.37 (39.43)
0.95	24.92 (4.68)	24.71 (6.03)	24.26 (7.24)	22.28 (10.41)	21.41 (11.26)	21.58 (15.32)	22.40 (18.81)	25.95 (23.88)	33.08 (32.03)
0.99	25.12 (0.19)	25.12 (0.45)	25.46 (4.42)	24.79 (6.90)	23.49 (7.40)	21.51 (10.02)	21.39 (12.44)	21.74 (16.97)	25.00 (22.63)
0.994	25.03 (0.16)	25.16 (0.19)	25.38 (0.63)	24.85 (4.83)	24.37 (6.05)	23.03 (8.14)	22.22 (11.59)	21.79 (15.11)	23.79 (20.59)
0.998	25.45 (0.14)	25.32 (0.15)	25.26 (0.19)	25.25 (0.65)	25.27 (1.59)	24.69 (6.06)	23.07 (7.63)	22.07 (11.46)	21.67 (16.08)
0.999	25.48 (0.13)	25.36 (0.15)	25.32 (0.16)	25.32 (0.20)	25.25 (0.42)	25.44 (5.51)	23.82 (5.77)	22.66 (9.05)	21.63 (13.34)

over-fitting. On the other hand, increasing the frequency of the decay (i.e., moving right through the columns) leads to an improved generalization, i.e., smaller difference between the test and train error. Therefore, as the table shows, the best performance in each row belongs to the parameter configurations around the main diagonal elements, where $\beta^{freq.}$ is approximately constant. This means that a less significant penalty should be applied more frequently. In this case, the total number of update step during each epoch is a limit in the sense that it determines the maximum possible decay frequency. Furthermore, applying a large but less frequent penalty instead of a small but more frequent one reduces the average number of operations per update step. Considering the superior performances in each row (i.e., the shaded cells), this table suggests that applying a larger but less frequent penalty offers a better performance compared to a less significant decay applied at every update step.

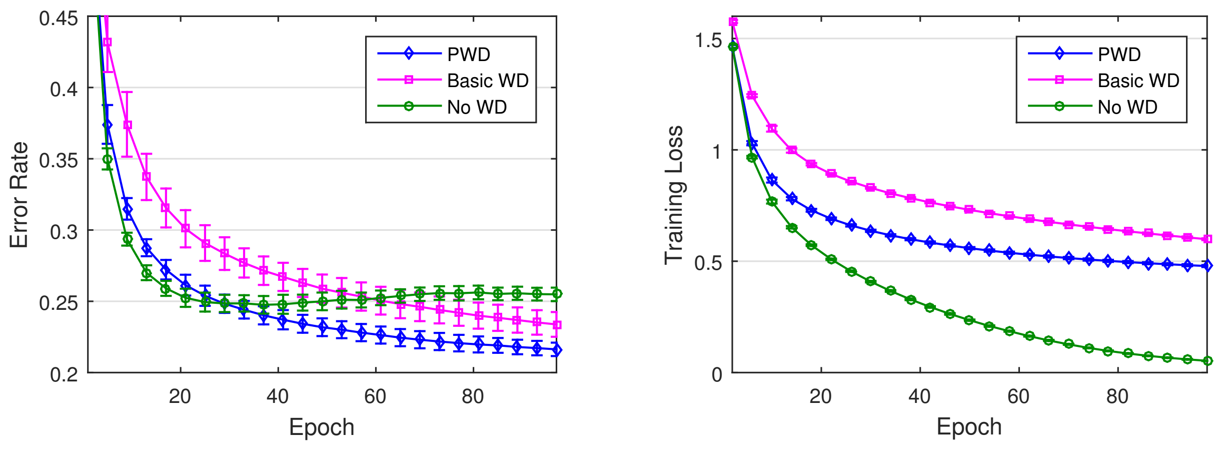
To examine the universality of the above results, the experiment is repeated for different data sets with different network architectures. As the top row in figure 5.3 shows, for all of the three cases, the lowest test error rates (i.e., dark blue pixels) are obtained using the parameter configurations around the main diagonal entries[9]. For the case of SVHN data set, the optimization diverges for small and low frequency penalties due to the explosion of the gradient (see the lower left corner of figure 5.3 (c))[10]. Therefore, when a low-frequency weight-decay is applied, the penalty should be large enough to avoid the explosion of the gradient. The second row of figure 5.3 illustrates the difference between the train and test error as a measure of generalization. The diagonal pattern in these plots acknowledges the fact that increasing the size and the frequency of the penalty improves generalization.

5.4.2 Comparative Study

This section begins with inspecting the performance of a CNN (64C-64C-10F) trained in three different ways: 1) without weight decay 2) regularized with basic weight decay 3) regularized with PWD. Figure 5.4 shows the progression of train and test error for these

[9]For the sake of readability, the parameter configuration for PWD is the same for all of the fully connected layers.

[10]See section 2.2.3 for an explanation about the explosion of the gradient. Also, note that the poor performance of the network in the upper right corner of the plot is due to a great and high-frequency regularization which doesn't let the network weights to grow large enough



(a) (b)

Figure 5.4: CIFAR10 classification performance throughout training for basic weight decay, PWD and without weight decay in terms of (a) test error and (b) training loss. Notice the faster convergence of PWD compared to basic WD especially at the start of the training (i.e., epochs 1 to 50).

three approaches over 100 training epochs averaged over 10 different random initializations. The optimum value of the parameters for PWD and basic weight decay are chosen using cross-validation[11]. Based on this figure, it is evident that applying a form of weight decay penalty is necessary to avoid over-fitting. Considering the progression of the performances especially at the beginning of the training (e.g., before the epoch = 50), applying PWD results in a faster converges than that of basic weight decay. This higher speed of convergence in PWD is due to its low decay frequency (i.e, once per epoch in contrast to once at every update step) which doesn't suppress the back-propagated gradient in every single update step.

As discussed in section 5.2, the decay coefficient in basic weight decay is a function of the learning rate. On the other hand, In practice, the learning rate is decayed throughout training (e.g., to $1/100th$ of its original value) so that the gradient descent takes shorter steps and settles into a local minimum [57, 37]. As a result of this decay in the learning

[11]For PWD $\beta = 0.1$ with the decay frequency of once per two epochs and for basic weight decay $\lambda = 0.06$

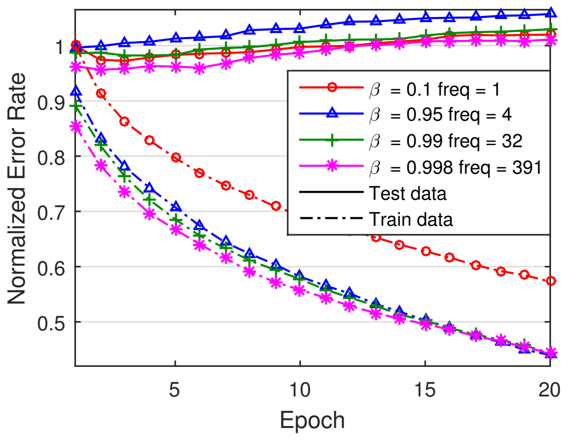


Figure 5.5: Effect of omitting PWD regularization (i.e, setting β to 1) for the last twenty epochs of training on the classification performance of CIFAR10. For each choice of β and f, the results after removing the decay are normalized to the respective case *with* the penalty. The increased gap between train and test error for different configurations of PWD parameters demonstrates that eliminating the regularizer even close to the end of training leads to a significant drop in the generalization performance of the network.

rate, in basic weight decay the penalty becomes less and less significant during training. In contrast, the decay coefficient β in PWD is independent of the learning rate and thus the dominance of the penalty doesn't decreases along the training epochs. To study the effect of this difference, the performance of a network regularized with PWD is compared to the case where the penalty is removed for a couple of final training epochs. Figure 5.5 illustrates the normalized performance of the network when the weight decay penalty is removed for the last twenty epochs. Evidently, for all of the PWD parameter configurations, omitting the penalty even close to the end of the training leads to a significant drop in the generalization performance of the network. This result advocates against the decrease of weight decay penalty throughout the training epochs.

The classification performance of PWD compared to basic weight decay for different data sets and architectures is summarized in table 5.2. These results suggest that the classification error of PWD is lower than basic weight decay. In the case of SVHN data set, the difference between the produced error using PWD and basic weight decay is not statistically significant. The reason is that, for this data set the optimum frequency parameter in PWD, found by cross-validation, is 573—which means applying the penalty at every single update step. This choice of the frequency parameter explains the similar test error produced by PWD and basic weight decay for this data set. Considering the difference between the training and test error for these two regularizers, PWD offers a superior generalization performance.

This improved generalization using PWD is important in two aspects. Firstly, as we will see in section 6.4, in a stage-wise training framework [101] this better generalization is crucial since it prevents the network from being over-fitted to the training data. Secondly, the network has a higher potential to obtain improved performance by further training of the network. We designed an experiment to examine the validity of this hypothesis. For the corresponding networks of table 5.2, we continue training the network for an additional 50 epochs. In order to be able to further reduce the training loss, we removed the penalty but with a sharp reduction in the learning rate by a factor of 0.01. According to the literature [9, 71, 117], multiple sharp decreases of the learning rate close to the end of training is a common practice to obtain state of the art results. Table 5.3 shows the test error and the percentage of improvement obtained by this additional training for PWD and basic weight decay[12]. Based on these results, the amount of improvement obtained for PWD is larger than basic weight decay. This smaller improvement for basic weight decay indicates that it is more prune to over-fitting compared to PWD.

5.5 Summary

The focus of this chapter was on weight decay as a widely-used general-purpose regularizer for deep networks. We showed that the continuous decay of the weights during every single update step can exacerbate the known issue of diffusion of gradients in training deep networks. To address this problem, we proposed PWD as a generalization for basic weight decay in which a frequency parameter is devised to avoid shrinking the weights at every single update step. That is, the weights are allowed to grow large enough to properly

[12]This experiment is conducted for the last three rows of table 5.2 in which the produced training error using PWD is significantly larger than that of basic weight decay.

Table 5.2: Classification error rate for CIFAR10, STL10 and SVHN using PWD and basic weight decay averaged over 10 random initializations. For each cell, the number in the parenthesis shows the training error. For each data set, the number marked in bold is the lowest test error (statistically significant according to paired t-test with $p < 0.05$). For the SVHN data set, the difference between the test error of PWD and basic WD is not statistically significant.

Data set	**Architecture**	**Basic WD**	**PWD**
CIFAR10	64C-64C-10F	21.21 (13.17)	**20.26** (14.03)
CIFAR10	64C-64C-64C-10F	20.99 (4.48)	**20.00** (7.70)
STL10	64C-64C-10F	43.92 (18.50)	**43.06** (25.36)
SVHN	64C-64C-128C-512F-10F	6.28 (2.32)	6.27 (4.12)

Table 5.3: Classification error rate after 50 epochs of additional training with reduced learning rate ($\times 0.01$). The numbers in the parentheses show the percentage of improvement obtained after this additional training. For each data set, the number marked in bold is the lowest test error (statistically significant according to paired t-test with $p < 0.05$). The smaller improvement for basic weight decay indicates that the trained networks using this method is more close to be over-fitted to the training data compared to PWD.

Data set	**Basic WD**	**PWD**
CIFAR10	20.49 (2.38%)	**19.19** (4.05%)
STL10	41.60 (5.28%)	**40.29** (6.43%)
SVHN	5.97 (4.94%)	**5.68** (9.41%)

pass the gradients for certain number of update steps until a (more severe) penalty is applied. The experimental results on a number of image classification data sets illustrates the improved generalization performance using PWD compared to basic weight decay. The analysis suggests that applying a larger but less frequent penalty instead of a less significant decay at every single update step leads to the faster convergence of the network especially at the start of the training.

In contrast to basic weight decay, the decay coefficient in PWD is independent of the learning rate and thus the dominance of the penalty does not decreases along the training epochs. This property contributes to the better generalization performance of PWD compared to basic weight decay.

This improved generalization using PWD sets the stage for more complex training procedures. In the next chapter, we introduce a stage-wise training strategy for deep networks where the network is steered towards a good solution through a sequence of related learning stages. In this sequential training framework, obtaining a good generalization in each stage using PWD is vital to protect the subsequent stages from over-fitting.

Chapter 6

Stage-wise Training: An Improved Feature Learning Strategy for Deep Models

In this chapter we show that network training performance can be improved using a stage-wise learning strategy, in which the learning process is broken down into a number of related sub-tasks that are completed stage-by-stage. The idea is to inject the information to the network *gradually* so that in the early stages of training the *coarse-scale* properties of the data are captured while the *finer-scale* characteristics are learned in later stages. Moreover, the solution found in each stage serves as a prior to the next stage, which produces a regularization effect and enhances the generalization of the learned representations. We show that decoupling the classifier layer from the feature extraction layers of the network is necessary, as it alleviates the diffusion of gradients and over-fitting problems. Experimental results in the context of image classification support these claims[1].

[1]The materials in this chapter has been previously published in [101], where the thesis author is the first author.

6.1 Objective

As discussed in Chapter 2, there are two major obstacles in training deep nets:

Over-fitting stems from the very large number of parameters present in deep networks. The key, then, is to employ regularization techniques to limit the degrees of freedom in the parameter space. Among the classic regularizers for neural networks we find early-stopping [118], Tikhonov regularization (i.e., L2 weight decay) [67] and lasso (i.e., L1 regularization) [68]. Specialized regularizers for deep neural networks include convolutional weight-sharing [11, 7], dropout [37] and drop-connect [71].

The diffusion of gradients is caused by the progressive vanishing of the error as it is back-propagated to earlier layers of the network. The most widely used methods to address this problem are layer-wise pre-training [17, 20], piecewise linear activation functions [18] and transfer learning [119, 21].

Despite remarkable advances in this area, the training of deep networks remains challenging and continues to motivate further developments. In particular, our research described in this chapter focuses on the issues of over-fitting and gradient-diffusion into two questions of interest:

1. How should computational resources be distributed between learning the classifier and the feature extractors?
2. Should all of the information (i.e., constraints) be fed to the network at once? Or is it preferable for the network to be guided step-by-step towards learning more discriminative features through a gradual, strategic presentation of the information?

One of the structural properties of deep networks is that all of the feature extraction layers and the classifier layer are trained at the same time. Recently, [120] showed that the feature extractors at successive layers are co-adapted and there is a complicated interaction between them. Therefore, training of feature extractor layers can not be factorized and it is important to learn them "at the same time". However the same limitation does not necessarily apply to the classifier layer, leading to a question whether the classifier and feature extraction layers should be trained at the same time. Although in general coupled training of the feature extractors and classifier results in a better performance [121, 122], it is not clear to what extent these two steps should be co-adapted. In this research we show that for deep nets, training indeed benefits from treating the classifier layer separately, and that the problem of diffusion of gradient can be addressed by preventing the complex co-adaptation of the feature extraction layers with the classification layer.

The second question that we raise in this chapter is whether we can increase the generalization of the learned features by the gradual injection of information to the network during training. In particular, given an image classification problem, our goal is to develop stage-wise learning, whereby the network is first steered in the direction of capturing discriminative information related to coarse-scale structures (i.e., shape features). Then, by gradually increasing the level of detail presented at the input, the learned feature extractors are fine-tuned to grasp the discriminative information related to the fine-scale image characteristics (i.e., appearance features).

Motivated by these questions, we propose a stage-wise training framework for representation learning using deep networks in which over-fitting can be avoided through stage-wise evolution of the information fed to the network, and whereby gradient diffusion can be addressed by decoupling the feature extraction layers from the classifier layer across successive training stages.

6.2 Background

6.2.1 Layer-wise Pre-training

Layer-wise Pre-training [17] played a significant role in revitalizing deep nets. The main idea behind this method is to train only one layer of the network at a time, starting from the first layer. After training each layer, its computed output is considered as the input for training the next layer. This layer-wise pre-training strategy is usually performed in an unsupervised way because of two reasons: 1) cheap access to abundant unlabeled data, 2) avoiding over-fitting due to the large number of parameters per layer. The pre-trained weights are used to initialize the network for a fine-tuning stage where all of the layers are trained together.

The optimization explanation for the effectiveness of layer-wise pre-training is that this method initializes the network at a location where it is more likely to converge to a good local minimum [20]. In addition, initializing the network parameters acts as a regularizer in the sense that it restricts the parameters to the regions corresponding to the input distribution [70].

Due to its layer-wise nature, this initialization method does not take into account the interactions between successive layers of the network.

6.2.2 Transfer Learning

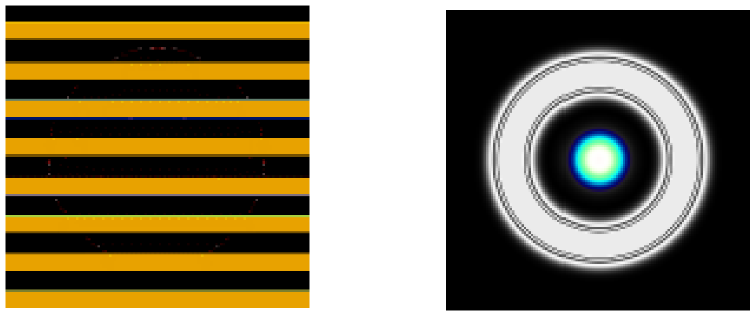
The goal of transfer learning for deep nets [21] is to use the related information in a *base* data set to initialize the parameters of a model on a *target* data set. Assume that our target task is to learn a deep net N_T on the target data set D_T. Given a base data set D_B with similar general properties to D_T, one can train a deep net N_B on that and *transfer* the learned representations from N_B to N_T. The transferred features are used to initialize some layers of N_T which can be kept frozen or fine-tuned using D_T, depending on the size of D_B and D_T and their common characteristics. Since the specificity of the learned features to the target task increases as we move towards the top layers of the network, this method is used to initialize the early layers of the network[2]. Transfer learning is used to prevent over-fitting specially when D_B is much larger than D_T. A thorough study of this method for deep nets can be found in [120].

It is important to note that the applicability of transfer learning is limited to tasks that are supported with a related and large base data set.

6.3 Methodology

Neural networks are discriminative models focused on minimizing the prediction error with respect to some architectural priors, such as the number of layers, the number of neurons at each layer, and the type of the activation function. To distinguish between

[2]Another reason is that early layers suffer more from the diffusion of gradients problem (see section 2.2.3).



(a) Classified as School Bus (b) Classified as Projector

Figure 6.1: Predicted classes using a deep neural network trained on ImageNet data set for a set of produced images by evolutionary algorithms [123]. Note that the prediction confidence for these images is high and around 99%. These counter-intuitve predictions stem from the fact neural networks aim at learning features that are *unique* to each class, as opposed to the *typical* features of a class.

different classes, these networks aim at learning features that are *unique* to each class, as opposed to the *typical* features of a class [123]. However, this training strategy can produce counter-intuitive results, for example how a deep network trained on the ImageNet data set mistakenly classified a black and yellow striped pattern as a school bus [123] (see figure 6.1). The reason for such a mistake is that in decision making no priority is assigned to the coarse properties of the data compared relative to its detailed characteristics.

To address this problem, we propose a stage-wise training framework in which the information in the training data is presented to the network *gradually*. In this way, at the early stages of training the network has access to only a subset of the data, specifically the coarse-scale properties of the data, such that the network learns to perform prediction by extracting features at that coarse scale. Then, during the following stages, finer information is provided to the network and the learned feature extractors from previous stages are permitted to evolve to perform better predictions. In other words, the learned feature extractors at each stage act as a prior for feature leaning at the next stage (see figure 6.2).

6.3.1 Definition of a Training Stage

Let $s \in \{1, 2, ..., S\}$ be the current stage of training. The training set in stage s is denoted by $T_s = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n_s} \subseteq \mathcal{X}_s \times \mathcal{Y}_s$, where $\mathcal{X}_s \subset \mathbb{R}^{p_s}$ and $\mathcal{Y}_s \subset \{0, 1\}^c$ are the input and output domains[3]. At stage s, the learning algorithm $A(., .)$ takes the training set and the initial value of the parameters $W^{s,0}$ as input, and outputs the learned parameters $W^{s,\infty}$:

$$W^{s,\infty} = A(T_s, W^{s,0}), \tag{6.1}$$

where A is, in principle, any state of the art learning strategy currently established in the research literature.

6.3.2 Connection between Successive Stages

Assuming that the network is randomly initialized at the first stage,

$$\mathbf{W}^{1,0} := \text{random} \tag{6.2}$$

a natural way of connecting successive stages would be as follows:

$$\mathbf{W}^{s,0} := \mathbf{W}^{s-1,\infty} \;\; \text{for all} \;\; s \in \{2, ..., S\}. \tag{6.3}$$

However, considering the multi-layer structure of the network and the properties of the gradient-based learning, some layers of the network require a special treatment. In particular, the gradient-based strategy for training multi-layer networks suffers from the *diffusion of gradients* problem: the back-propagated gradients vanish quickly as the depth of the network increases. Consequently, the top layers learn faster than the more distant earlier layers of the network. Furthermore the last layer of the network alone, the classifier layer, has sufficient degrees of freedom (free parameters) to model the entire labeled data by itself. Consequently, the classifier layer is more prone to over-fitting than the feature extraction layers.

It follows, then, that in any sort of stage-wise training framework it is important to use the previous stage to initialize the feature extraction layers, and not the classifier layer, to avoid over-fitting at the classifier layer.

[3]Note that training instances of different stages can be from different domains.

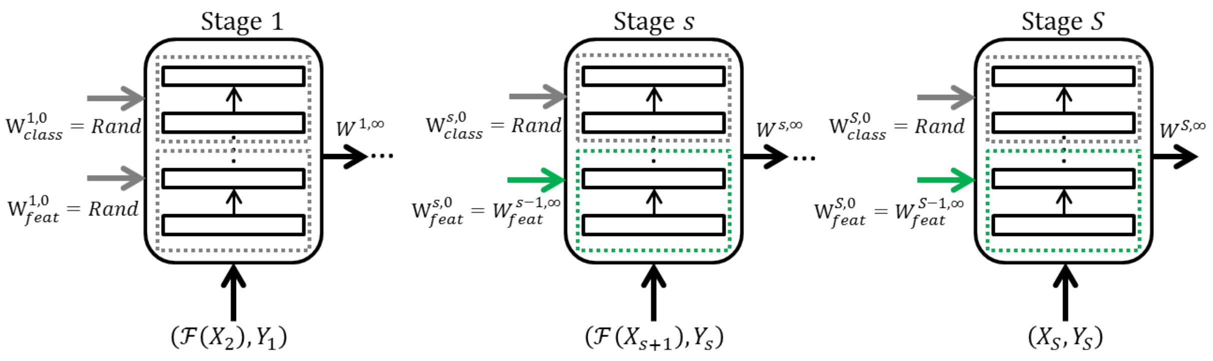


Figure 6.2: Schematic diagram of stage-wise training with information evolution. Note that only the feature extractor layers (and not the classifier) are initialized using the previous stage.

6.3.3 Stage-wise Information Evolution

In the proposed stage-wise framework, the training information passed to each stage should be evolved gradually. Considering the definition of a training stage, this objective can be met in a number of ways:

1. The evolution of the input domain $\mathcal{X}_s$,
2. The evolution of the output domain $\mathcal{Y}_s$, or
3. The evolution of the training set T_s.

The focus of this chapter is on the evolution of the input domain. A discussion on the other information evolution schemes will be presented in Chapter 7.2.3 as a direction for future research.

Assume that X_S is the original representation of the input data; that is, our target is to arrive at the usual training data at the final stage S. We therefore require a stage-to-stage mapping operator $\mathcal{F}$ which projects X_s, the input data at stage s, to lower-dimensional X_{s-1}:

$$X_{s-1} := \mathcal{F}(X_s) \tag{6.4}$$
$$\mathcal{F} : \mathcal{X}_s \mapsto \mathcal{X}_{s-1} \quad \text{where} \quad p_s > p_{s-1}$$

It is important to note that this information evolution method should be used with a *convolutional* weight-sharing scheme for all of feature extraction layers so that the number of parameters becomes independent of the input dimension. The parameter count independence is essential to allow learned parameters to be associated and connected between successive stages using the method discussed in section 6.3.2.

In the case of image data, a choice for mapping $\mathcal{F}$ is the sub-sampling operation and a stage-wise evolution of the input image can be obtained through sub-sampling the original input samples with an increasing ratio. Therefore, at the early stages of training, the network is focused on capturing coarse-scale image characteristics (i.e., *shape* features) through considering a wider context around each pixel. As we move forward toward the final stages, given more detailed information, the learned feature extractors are fine-tuned to detect discriminative information in the fine-scale image structures (i.e., *appearance* features).

6.4 Results and Discussion

We first conduct a set of experiments to understand how the proposed stage-wise training framework for multi-layer neural network improves feature extraction. Then, we examine the performance of a multi-stage trained network for image classification compared to an equivalent single-stage trained counterpart.

For the analytical experiments the standard CIFAR10 data set [6] is used. This data set consists of 32×32 colored images of ten classes of objects, each having 50000 training samples and 10000 test samples. For the purpose of information evolution, each training image is sub-sampled with 5 different increasing ratios (i.e., $S = 5$)[4]. The classification performance of the stage-wise training are evaluated for STL10 [115] and Street View House Numbers (SVHN) [116] data sets, as well. We used the labeled set in STL10 which includes 500 training and 800 testing samples per class. The STL10 is similar to CIFAR10 but with much fewer training images of a larger size (i.e., 96×96). For SVHN data set,

[4]The size of the input training image at stages 1, 2, ..., 5 are 14×14, 16×16, 20×20, 23×23 and 32×32, respectively.

Confidently	Confidently Recog. at				
Recogn. at	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
Stage 1	1	0.957±0.002	0.956±0.003	0.957±0.000	0.954±0.003
Stage 2	0.608±0.004	1	0.953±0.003	0.949±0.002	0.938±0.003
Stage 3	0.477±0.004	0.748±0.004	1	0.950±0.002	0.930±0.004
Stage 4	0.426±0.004	0.665±0.004	0.848±0.003	1	0.929±0.004
Stage 5	0.386±0.003	0.597±0.005	0.754±0.003	0.844±0.003	1

Table 6.1: Connection between the learned models at different stages with evolved inputs. The entry at row i and column j indicates "what ratio of the test samples that are correctly classified (with high confidence) at stage i, are classified correctly (with high confidence) at stage j?". Note that despite the differences in the input data, the models learned at successive stages are closely connected.

the goal is to recognize the digit at the center of each 32×32 RGB image and it includes 73257 training images and 26032 testing images[5].

For the stage-wise training, the architecture of the network is kept unchanged during training stages. We use a convolutional neural network (CNN) with two feature extraction layers (i.e., hidden layers) before the classification layer. The hyper-parameters of this CNN are similar to that of [117] for CIFAR10: 64 filters of size 5×5 at each convolutional layer followed by max-rectifying nonlinearity and pooling of size 3×3 with stride 2. The max and average pooling functions are used for the first and second layer, respectively. To obtain a good generalization at each training stage, we use Periodic Weight Decay (PWD) (proposed in Chapter 5) to regularize the learning process in each stage.

6.4.1 Analysis

Given the fact that different input data is used at different stages, we study how the classification problem solved at each training stage is related to that of the other stages. Table 6.1 shows the connection between the learned models at different stages with evolved inputs. In this table, entry at row i and column j refers the to "what ratio of the test

[5]Note that for SVHN, the extra training images are not used.

samples that are correctly classified (with high confidence) at stage i are classified correctly (with high confidence) at stage j?"[6]. Considering the entries above the main diagonal, most (i.e., more than 96%) of the samples that are recognized confidently at each scale are also classified correctly in the subsequent stages. The entries below the main diagonal show that the learned network at each stage is an improved version of the learned model at its previous stage. This result is interesting in the sense that, despite the difference in the size of the input data at each stage and consequently, different width of the context used for feature extraction, there is a close relationship between the problems that we are solving at successive stages.

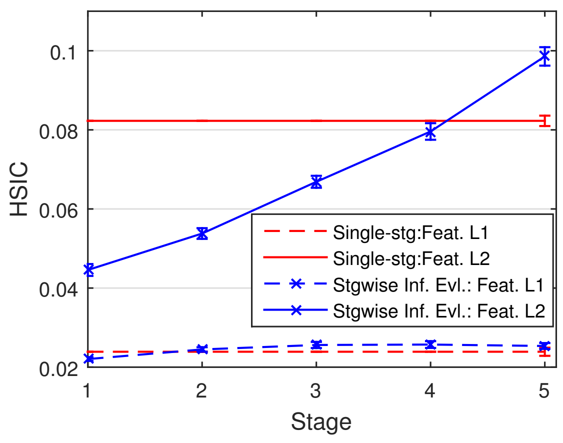
In the next experiment, we factorize the effect of the learned feature extractors from the classifier. In other words, we directly compare the statistical dependence between the learned features (at each layer of the network) and the output variable for different training strategies. In order to measure this dependence, we use the Hilbert-Schmidt Independence Criterion (HSIC) [124]. Assume that we are given n samples belonging to c different classes. Each sample is represented by a d-dimensional vector, which are the extracted features at a specific layer of the network. Using HSIC, we can compute the dependence between the extracted features $Z_{d\times n}$ and their corresponding labels $Y_{c\times n}$ as

$$HSIC(Z, Y) := (n-1)^{-2}\mathbf{tr}(HKHL) \tag{6.5}$$

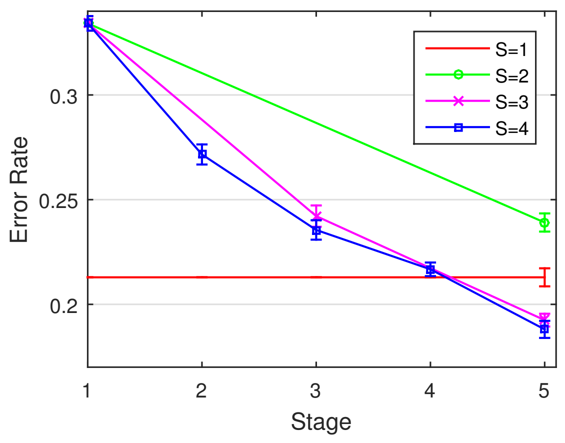
where K is a kernel of Z (e.g., $Z^T Z$), L is a kernel of Y, and H is the centering matrix—$H = I - n^{-1}\mathbf{e}\mathbf{e}^T$ where $\mathbf{e}$ is a vector of all ones. In this experiment we use a linear kernel for both Z and Y and normalize HSIC to have a maximum value of 1 (i.e., multiplying (6.5) by $\frac{(n-1)^2}{\|K\|_F\|L\|_F}$). Figure 6.3a shows the computed HSIC for the extracted features from the test set using the trained network at different stages. This figure indicates that during stage-wise training, the dependence between the learned features and the class labels increases. Moreover, compared to singe-stage training, stage-wise training of the network improves the quality of the extracted representations at all of the network layers in terms of their dependence to the class labels.

Finally, we study the effect of the *speed* of the information evolution on the performance of the learned features. We wonder how many training stages is required for a certain

[6]The confidence of a prediction comes from the output of the soft-max layer for the predicted class.



(a)

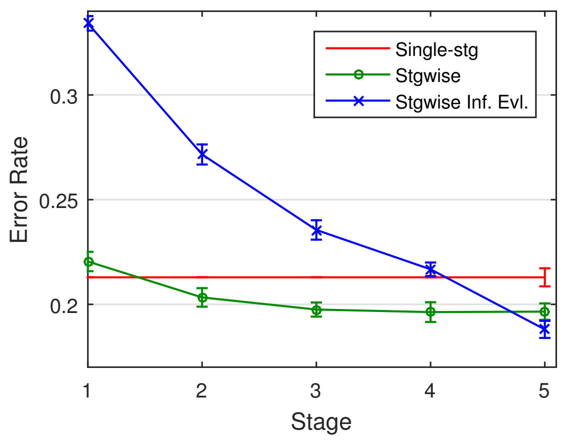


(b)

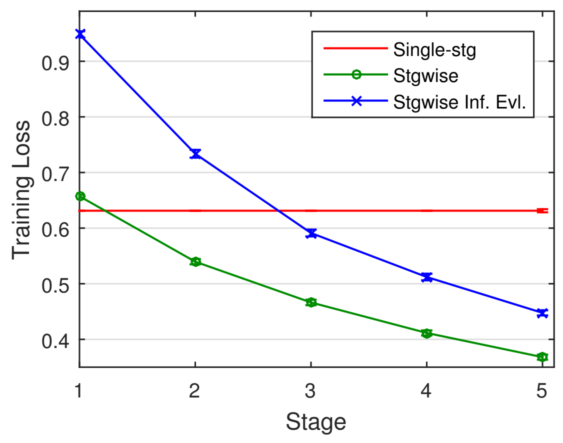
Figure 6.3: (a) Dependency of the extracted features at each layer of the network to the true class label across different stages of training. Observe how this dependency increase during stage-wise training for the extracted features at both layers of the network. (b) Performance of the learned features as a function of the level of information presented to the network during stage-wise training. Notice how the gradual information evolution during multiple training stages leads to a smaller error rate.

amount of information evolution to take place. As explained in the beginning of this section, each input image is represented in 5 different levels of details such that level 5 corresponds to the original input representation. In this experiment we compare and contrast three different training scenarios:

1. Single-stage training using the entire information (e.g., level 5)
2. Stage-wise training with a high speed of information evolution (e.g., jumping from stage 1 to 5)
3. Stage-wise training with a slow speed of information evolution (e.g., stage 1 to 2, ..., 4 to 5).



(a)



(b)

Figure 6.4: Classification performance of three methods (i.e., single-stage, stage-wise without information evolution and stage-wise with information evolution) in terms of (a) test error (b) training loss. Observe the superior performance of the stage-wise training strategy on test data despite the better performance of its counterparts on the training loss, which demonstrates the regularization effect of stage-wise training.

As figure 6.3b shows, it is important to increase the amount of training information fed to the network gradually and during multiple stages.

6.4.2 Classification Results

In this section, we evaluate the classification performance of the learned representations using the proposed stage-wise training framework. Three different training strategies are compared:

1. Conventional single-stage training
2. Stage-wise training without information evolution (i.e., the level of details in training images are the same for all stages)

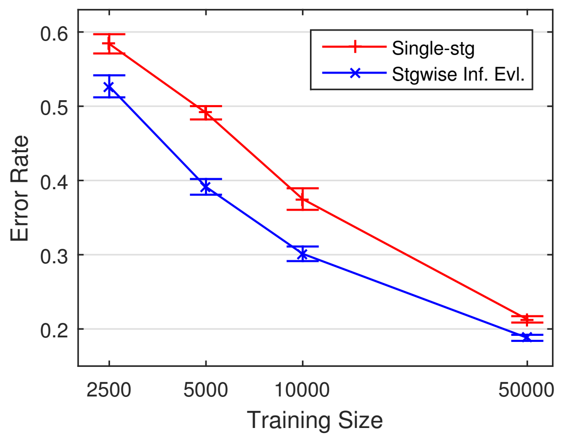


Figure 6.5: Classification performance of single-stage and stage-wise with information evolution methods as a function of the number of training samples. Note that the gap between the performance of the stage-wise framework and single-stage network appears to be more significant for smaller training sizes.

3. Stage-wise training with information evolution (i.e., the level of details in training images increases at successive stages).

The architecture of the trained networks using all of these three methods are the same and is as discussed at the beginning of this section. Figure 6.4a demonstrates the superior performance of the stage-wise training strategy on test data classification compared to the conventional single-stage training. Furthermore, considering the training loss of these methods shown in figure 6.4b, evolution of information during stage-wise training acts as a regularizer and improves the generalization of the learned features. Note that these results are obtained using a simple base-line two-layer network and can be applied to the state of the art models without loss of generality.

A good regularizer should improve the generalization performance of the learned model on small data sets. Considering this fact, we explore the performance of the model as a function of the number of training samples. The result of this experiment is depicted in figure 6.5. It can be observed that the gap between the performance of the stage-wise model and its single-stage counterpart is more significant for small training sets, confirming the fact that stage-wise model performs a more effective regularization.

Finally, to assess the universality of the above results, the classification performance of stage-wise training is examined for different data sets. Table 6.2 shows the classification error rate for stage-wise training with information evolution compared to the conventional single-stage training. These results confirm the improved recognition rate using a stage-wise training where the information is gradually injected to the network. Comparing the results for the CIFAR10 and STL10 data sets, the greater improvement for STL10 is due to the smaller number of training samples for STL10 (see section 6.4), revealing the regularization power of stage-wise training. Moreover, we observe a larger boost of performance for STL10 with large-sized images, which confirms the importance of capturing coarse-scale image structures through considering larger neighborhoods at the early stages of training.

It is important to note that in the stage-wise training framework, obtaining a good generalization performance (i.e., small difference between train and test error) in each stage is essential to protect the subsequent stages from over-fitting. Considering this fact, for all of the experiments of this chapter, we use Periodic Weight Decay (PWD)—proposed in Chapter 5—to regularize the network. The reason for the choice of PWD over basic weight decay is that, as demonstrated in section 5.4, regularizing the network using PWD offers a much smaller difference between train and test error. To demonstrate this effect, we compared the performance of these two regularizers for stage-wise training of STL10 data set[7] during 10 stages. As figure 6.6 shows, using basic weight decay the network over-fits after six stages of training. In contrast, regularizing the network using PWD avoids over-fitting and the test error consistently decreases during all of the training stages and finally leads to a superior test performance compared to basic weight decay.

6.5 Summary

In this chapter we proposed a stage-wise training framework with information evolution for feature extraction using deep neural networks. In this framework, the network is steered towards a good solution through a sequence of related learning stages, where the amount of information provided to the network is gradually increased during these stages. More

[7]According to table 6.2, stage-wise training produces the highest performance boost for this data set.

Table 6.2: Classification error rate for CIFAR10, STL10 and SVHN using stage-wise and conventional single-stage training. The network architecture for all of the data sets is the same and as described in the beginning of the section (i.e., 64C-64C-10F). Observe the superior recognition performance of stage-wise training compared to its single-stage counterpart.

Data set	**Image Size**	**No. of Stages**	**Single-stage**	**Stage-wise**
CIFAR10	32×32	5	20.70±0.46	18.72±0.49
SVHN	32×32	5	8.41±0.15	6.74±0.08
STL10-small	32×32	5	42.74±0.97	38.54±0.79
STL10-large	96×96	10	39.85±1.21	34.03±1.42

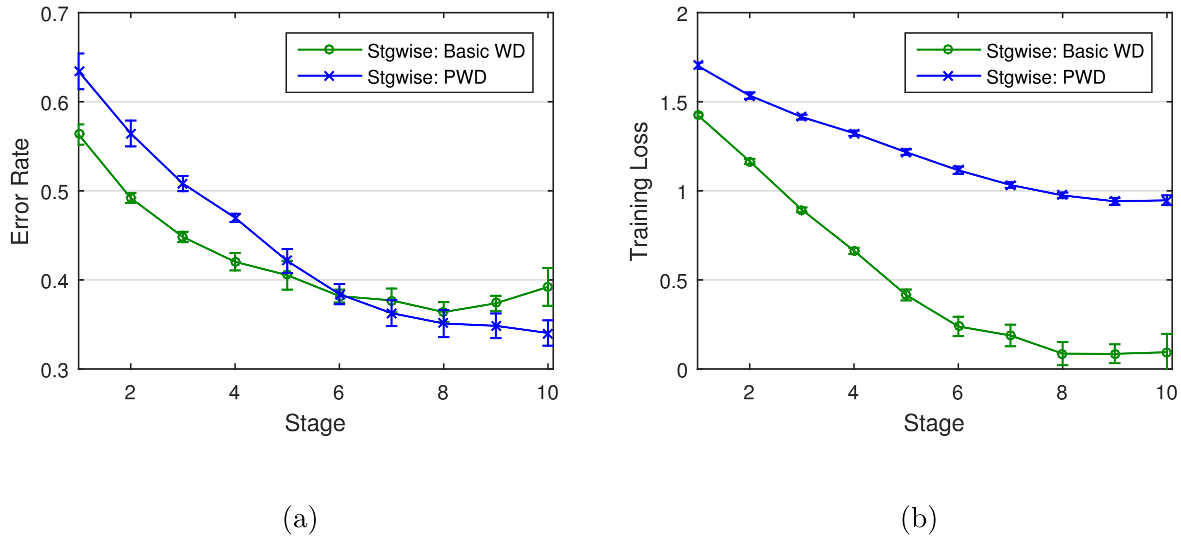


Figure 6.6: Classification performance of stage-wise training for STL10 data set regularized with two different methods (i.e., PWD and basic weight decay) in terms of (a) test error (b) training loss. Observe how employing PWD prevents the stage-wise training from over-fitting and leads to a smaller test error rate at the end of training.

specifically, at the early stages only coarse-scale properties were provided to the network; then, using the solution found at the previous stage, as a prior for the next learning stage, fine-scale learning takes place at the successive stages. In principle, stage-wise training acts as a regularizer.

Moreover, we showed that the classifier layer of the network requires a special treatment. In fact, due to the problem of diffusion of gradients in deep models the classifier can overfit while the early layers are not still learned. This problem is alleviated in a stage-wise framework, where the feature extraction layers are initialized using the previous stage while the classifier layer is randomly initialized at the beginning of each stage.

The experimental analysis showed that the proposed framework improves the accuracy of image classification for a number of data sets. The most important reason for such an improvement is the regularization effect of the stage-wise training. In addition, it was shown that the statistical dependence between the learned features and the class labels increases stage-by-stage, eventually becoming larger than that of single-stage training. This shows that the extracted features using stage-wise model are more discriminative. Nevertheless, we believe that the real power of the proposed method emerges on higher-dimensional inputs (e.g., larger images), as suggested by the great improvement for STL10 data set with large-sized images.

Chapter 7

Conclusion

This research was aimed at tailoring deep networks for visual recognition through exploiting the characteristics of the image data. Incorporating domain knowledge into these networks enabled us to address some of the innate challenges in the optimization procedure of these models. This led to learning image descriptors with improved generalization performance. This chapter summarizes the main contributions presented in this thesis and offers some promising directions for future research.

7.1 Contribution Highlights

In recent years, we have witnessed a rapid growth in the application of deep networks for different tasks and across various domains. In spite of the unique characteristics of each domain, the adjustments that are made to these models to adapt them to the corresponding domain is small. For instance, the architectures of deep networks used in visual recognition tasks surprisingly resemble those used in speech recognition. Also, the algorithm used for learning the parameters is almost the same. This research was built upon the fundamental question that how much benefit one can get from incorporating the essential part of the domain knowledge into these powerful models. Despite the remarkable advances in this area, training deep networks is still computationally expensive. In addition, considering the lack of enough labeled training data in many applications, over-fitting is a serious threat for deep models with a large number of free parameters. Motivated by these concerns, we introduced new methods for embedding image-specific priors into deep networks. Moreover, our comprehensive empirical analysis demonstrated that these *regularized* networks offer a better discrimination and generalization performance compared to their domain-oblivious counterparts. The main contributions of this thesis can be summarized as follows:

- **Eigen-RBM:** We addressed the computational burden of training fully-connected Restricted Boltzmann Machines (RBMs) for visual recognition through utilizing some prior knowledge about the global characteristics of the input images. We proposed a generalization of RBMs, called Eigen-RBM, in which the number of free parameters is independent of the image size. The key idea behind this reduction in the number of free parameters is that Eigen-RBM regularizes the weights to be a linear combination of a set of predefined filters obtained from exploiting the global structure of the training images. Our empirical evaluations illustrated that compared to basic RBM, Eigen-RBM can achieve similar or better performance—both for recognition and sample generation—with much less training time.

- **Multi-Neighborhood Convolutional Networks:** Conventional convolutional architectures are *single-neighborhood*, in the sense that they consider a single-size neighbhorhood around each pixel for feature learning. We proposed multi-neighborhood convolutional networks for visual recognition in which the fine-scale image structures (i.e., appearance features) are captured using a small-sized neighborhood while coarse-scale characteristics (i.e., shape features) are detected by considering a wider range neighborhood around each pixel. Moreover, we devised a scalable learning strategy for multi-neighborhood architectures in which the parameters of an already trained single-neighborhood network are exploited for multi-scale feature feature extraction. The experimental results of visual recognition indicated that the learned features using multi-neighborhood convolutional architecture are more discriminative than that of their single-scale counterparts without increasing the training cost.

- **Periodic Weight Decay (PWD):** We detected an issue with the widely-used weight decay regularizer in training deep networks. We showed that this popular regularizer in fact contributes to the problem of diffusion of gradient in training deep networks as a result of decaying the weights at every single update step. In order to resolve this issue, we proposed PWD, a periodic generalization of weight decay, where one can determine *how hard* and *how frequently* the weights should be penalized. The analysis of our experiment results revealed that the use of a larger but less frequent penalty leads to faster convergence of the optimization process, compared to the case where a small decay that is applied at every update step. Furthermore, in contrast to basic weight decay, the decay coefficient in PWD is independent of the learning rate and thus, the dominance of the penalty does not decreases during the training epochs. The empirical evaluations showed that this constant dominance of the penalty is essential in the superior generalization performance of PWD compared to basic weight decay.

- **Stage-wise Training:** Inspired by the multi-scale characteristics of images, we proposed a stage-wise training framework for training deep networks. In this framework, the network is "guided" through the optimization procedure; it first learns to capture generic features of each category using the coarse-scale representations of the training images. Then, it fine-tunes the learned features at the later stages by providing the detailed representation of the input images to the network. Moreover, the learned representations at each stage acts as a prior for regularizing the parameter learning in its subsequent stage. As a result, the network is guided towards capturing the "typical" characteristics of each class instead of their "unique" features, which is a problematic issue with the conventional training frameworks [123]. In contrast to the ensemble learning schemes used for improved feature learning [87, 93], stage-wise training does not increase the computational time and memory at the test time. The experiments showed that stage-wise training improves the quality of the learned features in terms of their dependence to the corresponding class label. Furthermore, the trained networks using a stage-wise procedure outperform their single-stage counterparts. The larger improvement in the case of small training sets promotes the promising regularization effect of stage-wise learning.

7.2 Future Work

The presented work in this thesis opens up several future research directions, a selection of which is presented in this section.

7.2.1 Convolutional Eigen-RBM

For computational reasons, the use of convolutional architectures (e.g., convolutional RBMs) for feature learning is limited to filters with rather small image patches. Therefore, these methods find small-sized convolutional filters that only exploit a narrow range of dependencies around each pixel. This problem can be addressed by extending the presented idea in Chapter 3—which was designed for regularizing fully-connected RBMs—to convolutional RBMs. That is, the convolutional filters should be constrained to be a linear combination of a set of pre-defined filters. However, in this case, since the filters are applied convolutionally, the pre-defined filters should be able to capture *local* characteristics of the input images at different levels of details. This filter bank can be created by computing the top eigenvectors of the covariance matrix of the local image patches. More precisely, we need a convolutional extension for PCA in which instead of dot product, the eigenvectors are *convolved* with the input images[1]. These eigenvectors that constitute the pre-defined filter bank can be extracted from image patches of larger size to explore a wider range of context around each pixel for the subsequent feature learning. Utilizing this idea one can build a convolutional Eigen-RBM, as a generalization for convolutional RBMs, in which larger-sized filters can be learned without increasing the number of free parameters.

7.2.2 Stage-wise Training with Output Evolution

The proposed stage-wise training framework in Chapter 6 was based on evolving the presented information in the *input space* gradually during the training stages. Considering the definition of a training stage in section 6.3.1, another way of evolving the presented information to the network could be through the *output space*. That is, the labeling of the training data can change during different stages of training. At early stages of training,

[1]We have already derived a convolution extension for Supervised PCA [125] with closed-form solution.

we try to estimate a more “general” or a “smoother” labeling function—i.e., we let the network miss-classify some of the training data point. Then, we fine-tune the learned parameters by providing a modified labeling which is more “ragged” and closer to the true labels of training data in the subsequent stages. In order to construct this set of evolving labeling functions, one can utilize the *dark knowledge* idea presented in [97, 98]. These works suggest that using the true labeling of the data for training the network parameters does not necessarily lead to a better solution. In particular, they showed that a shallow network can do as good as a deep network, if it is provided with the “softened” training labels that the deep network generates. Inspired by this idea, one can build a set of evolving labeling functions by using the geometric average of the true labels and the smoothed labels for stage-wise training[2].

7.2.3 Regularization-based Knowledge Transfer for Stage-wise Training

In stage-wise training (Chapter 6), the transfer of knowledge across subsequent stages was carried out by initializing the parameters of each stage to those learned in the previous stage. Therefore, the learning process at each stage acted as a pre-training step for its subsequent stage. A more direct way of transferring the learned knowledge would be through the general-purpose regularizers. That is, the solution at each stage can be regularized to be as close as possible to the solution found in its previous stage. Therefore, the cost function in each stage has to compromise between minimizing the network loss and finding a parameter configuration close to its previous stage. It is similar to $L1$ or $L2$ norm regularizer with the difference that instead of weights themselves, the *difference between the learned parameters in two subsequent stages* is pushed toward zero. This regulariztion-based transfer learning scheme could be implemented using the proposed periodic framework in Chapter 5.

[2]We have obtained some promising preliminary results using this method.

References

[1] Henri Cohen and Claire Lefebvre. *Handbook of categorization in cognitive science*, volume 4. Elsevier Amsterdam:, 2005.

[2] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[3] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006*, pages 404–417. Springer, 2006.

[5] Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Kernel descriptors for visual recognition. In *NIPS*, volume 1, page 3, 2010.

[6] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 1(4):7, 2009.

[7] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML*, pages 609–616. ACM, 2009.

[8] M Ranzato, Joshua Susskind, Volodymyr Mnih, and Geoffrey Hinton. On deep generative models with applications to recognition. In *CVPR*, pages 2857–2864. IEEE, 2011.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[10] Yann LeCun. Learning invariant feature hierarchies. In *ECCV Workshops*, pages 496–505. Springer, 2012.

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

[12] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. IEEE, 2009.

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

[15] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.

[16] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. *COLT*, 2016.

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

[18] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[19] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier networks. In *AISTATS*, volume 15, pages 315–323, 2011.

[20] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[21] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. *Unsupervised and Transfer Learning Challenges in Machine Learning*, 7:19, 2012.

[22] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8609–8613. IEEE, 2013.

[23] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1915–1929, 2013.

[24] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42, 2012.

[25] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8599–8603. IEEE, 2013.

[26] Ebru Arisoy, Tara N Sainath, Brian Kingsbury, and Bhuvana Ramabhadran. Deep neural network language models. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pages 20–28. Association for Computational Linguistics, 2012.

[27] Tomáš Mikolov. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 2012.

[28] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2333–2338. ACM, 2013.

[29] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[30] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[31] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):721–741, 1984.

[32] Song Chun Zhu and David Mumford. Prior learning and gibbs reaction-diffusion. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(11):1236–1250, 1997.

[33] Uwe Schmidt, Qi Gao, and Stefan Roth. A generative perspective on mrfs in low-level vision. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1751–1758. IEEE, 2010.

[34] Stephane G Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *PAMI*, 11(7):674–693, 1989.

[35] Tony Lindeberg. *Scale-space theory in computer vision*. Springer Science & Business Media, 1993.

[36] Peter J Burt and Edward H Adelson. The laplacian pyramid as a compact image code. *Communications, IEEE Transactions*, 31(4):532–540, 1983.

[37] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[38] Y Bengio, Paolo Frasconi, and P Simard. The problem of learning long-term dependencies in recurrent networks. In *Neural Networks, 1993., IEEE International Conference on*, pages 1183–1188. IEEE, 1993.

[39] Olivier Bousquet and Léon Bottou. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168, 2008.

[40] Geoffrey Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[41] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[42] Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5), 2007.

[43] Yoshua Bengio, Olivier Delalleau, and Nicolas L Roux. The curse of highly variable functions for local kernel machines. In *Advances in neural information processing systems*, pages 107–114, 2005.

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

[45] Ilya Sutskever and Geoffrey E Hinton. Deep, narrow sigmoid belief networks are universal approximators. *Neural Computation*, 20(11):2629–2636, 2008.

[46] Ke Huang and Selin Aviyente. Wavelet feature selection for image classification. *Image Processing*, 17(9):1709–1720, 2008.

[47] Andrew P Witkin. Scale-space filtering: A new approach to multi-scale description. In *ICASSP*, volume 9, pages 150–153. IEEE, 1984.

[48] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *PAMI*, 20(11):1254–1259, 1998.

[49] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

[50] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[51] Rodrigo Benenson. Classification data sets results. `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html`. [Online; accessed July-2016].

[52] Henry A Rowley, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *IEEE Transactions on pattern analysis and machine intelligence*, 20(1):23–38, 1998.

[53] Valery A Petrushin. Emotion recognition in speech signal: experimental study, development, and application. *studies*, 3:4, 2000.

[54] Gary B Huang, Honglak Lee, and Erik Learned-Miller. Learning hierarchical representations for face verification with convolutional deep belief networks. In *CVPR*, pages 2518–2525. IEEE, 2012.

[55] Adam Coates, Andrew Y Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, pages 215–223, 2011.

[56] Kevin Jarrett, Koray Kavukcuoglu, M Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *ICCV*, pages 2146–2153. IEEE, 2009.

[57] Matthew D Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

[58] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[59] Michael A Nielsen. Neural networks and deep learning. *URL: http://neuralnetworksanddeeplearning. com/*, 2015.

[60] Yurii 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.

[61] Ilya Sutskever, James Martens, George E Dahl, and Geoffrey E Hinton. On the importance of initialization and momentum in deep learning. *ICML (3)*, 28:1139–1147, 2013.

[62] Martin Anthony and Peter L Bartlett. *Neural network learning: Theoretical foundations*. cambridge university press, 2009.

[63] David Haussler. Decision theoretic generalizations of the pac model for neural net and other learning applications. *Information and computation*, 100(1):78–150, 1992.

[64] Steve Lawrence, C Lee Giles, and Ah Chung Tsoi. What size neural network gives optimal generalization? convergence properties of backpropagation. 1998.

[65] Peter 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. *Information Theory, IEEE Transactions on*, 44(2):525–536, 1998.

[66] Shun-ichi Amari, Noboru Murata, K-R Muller, Michael Finke, and Howard Hua Yang. Asymptotic statistical theory of overtraining and cross-validation. *IEEE Transactions on Neural Networks*, 8(5):985–996, 1997.

[67] Andrey Nikolayevich Tikhonov. On the stability of inverse problems. In *Dokl. Akad. Nauk SSSR*, volume 39, pages 195–198, 1943.

[68] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[69] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, pages 1096–1103. ACM, 2008.

[70] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, 2010.

[71] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

[72] Uwe Schmidt and Stefan Roth. Learning rotation-aware features: From invariant priors to equivariant descriptors. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2050–2057. IEEE, 2012.

[73] Elnaz Barshan and Paul Fieguth. Scalable learning for restricted boltzmann machines. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 2754–2758. IEEE, 2014.

[74] Max Welling, Simon Osindero, and Geoffrey E Hinton. Learning sparse topographic representations with products of student-t distributions. In *Advances in neural information processing systems*, pages 1359–1366, 2002.

[75] Stefan Roth and Michael J Black. Fields of experts: A framework for learning image priors. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 860–867. IEEE, 2005.

[76] Yair Weiss and William T Freeman. What makes a good model of natural images? In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[77] MarcAurelio Ranzato, Volodymyr Mnih, and Geoffrey E Hinton. Generating more realistic images using gated mrf's. In *Advances in Neural Information Processing Systems*, pages 2002–2010, 2010.

[78] Yichuan Tang, Ruslan Salakhutdinov, and Geoffrey Hinton. Robust boltzmann machines for recognition and denoising. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2264–2271. IEEE, 2012.

[79] Ruslan Salakhutdinov and Geoffrey E Hinton. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 448–455, 2009.

[80] Ian T Jolliffe. *Principal component analysis*, volume 487. Springer-Verlag New York, 1986.

[81] C.M. Bishop et al. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

[82] Hirotugu Akaike. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6):716–723, 1974.

[83] Kenneth P Burnham and David R Anderson. *Model selection and multi-model inference: a practical information-theoretic approach.* Springer Verlag, 2002.

[84] Honglak Lee, Chaitanya Ekanadham, and Andrew Ng. Sparse deep belief net model for visual area v2. In *NIPS*, pages 873–880, 2007.

[85] Kihyuk Sohn, Dae Yon Jung, Honglak Lee, and Alfred O Hero. Efficient learning of sparse, distributed, convolutional feature representations for object recognition. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2643–2650. IEEE, 2011.

[86] Nitish Srivastava. *Improving neural networks with dropout.* PhD thesis, University of Toronto, 2013.

[87] Elnaz Barshan, Paul Fieguth, and Alexander Wong. Scalable multi-neighborhood learning for convolutional networks. In *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2015.

[88] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *NIPS*, pages 766–774, 2014.

[89] Tianhorng Chang and C-CJ Kuo. Texture analysis and classification with tree-structured wavelet transform. *Image Processing, IEEE Transactions on*, 2(4):429–441, 1993.

[90] Angelos Tzotsos, Konstantinos Karantzalos, and Demetre Argialas. Object-based image analysis through nonlinear scale-space filtering. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(1):2–16, 2011.

[91] Shengye Yan, Xinxing Xu, Dong Xu, Stephen Lin, and Xuelong Li. Beyond spatial pyramids: A new feature extraction framework with dense spatial sampling for image classification. In *ECCV*, pages 473–487. Springer, 2012.

[92] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C Courville, and Yoshua Bengio. Maxout networks. *ICML (3)*, 28:1319–1327, 2013.

[93] Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *CVPR*, pages 3642–3649. IEEE, 2012.

[94] Alexander Wong, Akshaya Mishra, Justin Yates, Paul Fieguth, David A Clausi, and Jack P Callaghan. Intervertebral disc segmentation and volumetric reconstruction from peripheral quantitative computed tomography imaging. *Biomedical Engineering, IEEE Transactions on*, 56(11):2748–2751, 2009.

[95] Fernand Meyer and Petros Maragos. Nonlinear scale-space representation with morphological levelings. *Journal of Visual Communication and Image Representation*, 11(2):245–265, 2000.

[96] Joachim Weickert and Brahim Benhamouda. A semidiscrete nonlinear scale-space theory and its relation to the peronamalik paradox. In *Advances in computer vision*, pages 1–10. Springer, 1997.

[97] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, pages 2654–2662, 2014.

[98] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[99] Jiquan Ngiam, Zhenghao Chen, Daniel Chia, Pang W Koh, Quoc V Le, and Andrew Y Ng. Tiled convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1279–1287, 2010.

[100] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[101] Elnaz Barshan and Paul Fieguth. Stage-wise training: An improved feature learning strategy for deep models. In *Journal of Machine Learning Research, Proceedings of The 1st International Workshop on Feature Extraction: Modern Questions and Challenges, NIPS*, pages 49–59, 2015.

[102] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture

for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[103] Andrea Vedaldi and Karel Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 689–692. ACM, 2015.

[104] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

[105] Yoram Singer and John C Duchi. Efficient learning using forward-backward splitting. In *Advances in Neural Information Processing Systems*, pages 495–503, 2009.

[106] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[107] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[108] J Moody, S Hanson, Anders Krogh, and John A Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4:950–957, 1995.

[109] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012.

[110] Alex Krizhevsky. cuda-convnet: High-performance c++/cuda implementation of convolutional neural networks. `https://code.google.com/p/cuda-convnet/`, 2012.

[111] Dimitri P Bertsekas. Nonlinear programming. 1999.

[112] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l 1-ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 272–279. ACM, 2008.

[113] John Langford, Lihong Li, and Tong Zhang. Sparse online learning via truncated gradient. In *Advances in neural information processing systems*, pages 905–912, 2009.

[114] Nathan Srebro and Adi Shraibman. Rank, trace-norm and max-norm. In *International Conference on Computational Learning Theory*, pages 545–560. Springer, 2005.

[115] Adam Coates, Honglak Lee, and Andrew Y Ng. An analysis of single-layer networks in unsupervised feature learning. *Ann Arbor*, 1001(48109):2, 2010.

[116] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[117] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[118] Rich Caruana Steve Lawrence Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference*, volume 13, page 402. MIT Press, 2001.

[119] Rich Caruana. Learning many related tasks at the same time with backpropagation. *Advances in neural information processing systems*, pages 657–664, 1995.

[120] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, pages 3320–3328, 2014.

[121] Mehmet Gönen. Coupled dimensionality reduction and classification for supervised and semi-supervised multilabel learning. *Pattern recognition letters*, 38:132–141, 2014.

[122] Dmitry Storcheus, Mehryar Mohri, and Afshin Rostamizadeh. Foundations of coupled nonlinear dimensionality reduction. *arXiv preprint arXiv:1509.08880*, 2015.

[123] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *arXiv preprint arXiv:1412.1897*, 2014.

[124] Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *Algorithmic learning theory*, pages 63–77. Springer, 2005.

[125] Elnaz Barshan, Ali Ghodsi, Zohreh Azimifar, and Mansoor Zolghadri Jahromi. Supervised principal component analysis: Visualization, classification and regression on subspaces and submanifolds. *Pattern Recognition*, 44(7):1357–1371, 2011.