IMPROVING ACCURACY OF LEARNING MODELS USING DISJUNCTIVE NORMAL FORM AND SEMISUPERVISED LEARNING

by

Sayed Mehdi Sajjadi Mohammadabadi

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering

The University of Utah

December 2017

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Sayed Mehdi Sajjadi Mohammadabadi**

has been approved by the following supervisory committee members:

Tolga Tasdizen	, Chair	**7/18/2017** Date Approved
Ross Whitaker	, Member	**7/18/2017** Date Approved
Neal Patwari	, Member	**7/18/2017** Date Approved
Behrouz Farhang	, Member	**7/18/2017** Date Approved
Thomas Fletcher	, Member	**7/18/2017** Date Approved

and by **Gianluca Lazzi** , Chair/Dean of

the Department/College/School of **Electrical and Computer Engineering**

and by David B. Kieda, Dean of The Graduate School.

ABSTRACT

The goal of machine learning is to develop efficient algorithms that use training data to create models that generalize well to unseen data. Learning algorithms can use labeled data, unlabeled data or both. Supervised learning algorithms learn a model using labeled data only. Unsupervised learning methods learn the internal structure of a dataset using only unlabeled data. Lastly, semisupervised learning is the task of finding a model using both labeled and unlabeled data. In this research work, we contribute to both supervised and semisupervised learning.

We contribute to supervised learning by proposing an efficient high-dimensional space coverage scheme which is based on the disjunctive normal form. We use conjunctions of a set of half-spaces to create a set of convex polytopes. Disjunction of these polytopes can provide desirable coverage of space. Unlike traditional methods based on neural networks, we do not initialize the model parameters randomly. As a result, our model minimizes the risk of poor local minima and higher learning rates can be used which leads to faster convergence. We contribute to semisupervised learning by proposing 2 unsupervised loss functions that form the basis of a novel semisupervised learning method. The first loss function is called Mutual-Exclusivity. The motivation of this method is the observation that an optimal decision boundary lies between the manifolds of different classes where there are no or very few samples. Decision boundaries can be pushed away from training samples by maximizing their margin and it is not necessary to know the class labels of the samples to maximize the margin. The second loss is named Transformation/Stability and is based on the fact that the prediction of a classifier for a data sample should not change with respect to transformations and perturbations applied to that data sample. In addition, internal variations of a learning system should have little to no effect on the output. The proposed loss minimizes the variation in the prediction of the network for a specific data sample. We also show that the same technique can be used to improve the robustness of a learning model with respect to adversarial examples.

To the memory of my mother.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First, I would like to thank my adviser, Dr. Tolga Tasdizen, for trusting me and giving me the opportunity to join his research group and work on amazing research projects while providing scientific and financial support. He guided me through research with his vast knowledge and patience while giving me the freedom to pursue my research interests. He taught me how to be an independent researcher and always aim high.

I would like to thank the Scientific Computing and Imaging Institute (SCI) for their world class computing resources, facilities and professional staff. Most of our research projects were not possible without cutting-edge servers and clusters available at SCI. I also thank my committee members, Dr. Ross Whitaker, Dr. Tom Fletcher, Dr. Behrouz Farhang and Dr. Neal Patwari, for their helpful comments and their time.

I want to thank Dr. Mojtaba Seyedhosseini who was a great support even before I arrive to the United States and then after that, he helped me to settle down. He was a role model senior PhD student and I learned a lot from him. He was always there when I needed help. I want to thank other members of our research group for helpful discussions and the time that we spent together. Specifically, I would like to thank Mehran Javanmardi, Fitsum Mesadi, Dr. Ting Liu and Nisha Ramesh.

I am grateful for many amazing friends that I had the opportunity to spend time with. I will never forget the moments that we shared together.

I am so grateful of my father Masih Sajjadi and his wife Katayoun Izadapanah for their support, encouragement and love. My deepest gratitude goes to my beautiful sisters, Maryam, Mozhdeh and Golnoush for their emotional and spiritual support. Finally, I am going to thank my mother who is not among us anymore but I always feel her presence, support and love. It is to her that I dedicate this dissertation.

I would also like to acknowledge our funding support. This work was supported by NSF Grant IIS-1149299.

CHAPTER 1

INTRODUCTION

Machine learning is the science of developing data-driven algorithms that create models to learn and describe the underlying distribution and structure of the data. The data are usually in the form of a set of high-dimensional vectors and the goal is to find the model of the underlying probability distribution that generated the data samples. The model can then be used to generate data samples from the same probability distribution or to predict the conditional probability of a given data sample.

In practice, a multidimensional data sample can also contain a *label* dimension. The label determines the category that the data sample belongs to. If the data vector contains a label dimension, then the goal is to learn a model capable of predicting the conditional probability of the label given the rest of the data vector. This task is called *classification* or *supervised learning*. Usually, the data samples belong to 2 sets. In the first set, dimensions of data samples include the label dimension. This set is called the *training set* and the second set is called the *test set*. The data samples belonging to the test set don't have the label dimension and the learned model should be able to predict the probability of the label dimension given the other dimensions. In other words, the goal of supervised learning is to predict the category of a data sample that does not have the label.

If the data samples don't have the label dimension and the goal is to learn the underlying structure of the probability distribution that generated the data, then the task is called *unsupervised learning* [1]. It is also possible to use unlabeled data to improve a supervised learning model. The task of learning a supervised classifier using both labeled and unlabeled is called *semisupervised learning* [2].

The contributions of this dissertation can be broadly divided in 2 parts. In the first part, we mainly focus on supervised learning. We propose a supervised learning model which is based on an artificial neural network. It provides an efficient coverage of space and

a meaningful initialization of the weights. This model provides faster convergence and reduces the risk of ending up in a local minima. Our focus in the second part is mainly on semisupervised learning. We propose 2 unsupervised loss functions that form the basis of a novel and competitive semisupervised learning method.

1.1 Supervised Learning

Supervised learning includes a large number of machine learning algorithms. Many of the real-world challenging problems can be formulated as a supervised learning or classification problem. Support Vector Machines (SVM) [3], Decision Trees [4] and Random Forests [5], the Nearest Neighbor algorithm, Naive Bayes classifier and Artificial Neural Networks are a few examples of supervised learning. Each of these methods can be the optimal choice for general classification purposes depending on the type and size of the data.

However, supervised learning methods based on neural networks gained popularity in recent years mainly because of 2 reasons. First, advances in GPU and CPU technology made it possible to train large-scale neural networks. At the same time, availability of large sets of labeled data helped to create complex models that learn challenging tasks. Second, in recent years, convolutional neural networks (ConvNets) [6] proved to be the state-of-the-art in many computer vision tasks including but not limited to object detection, object recognition [6],[7]-[8] and scene labeling [9]. ConvNets are special types of neural networks with 2 properties: local connectivity and weight sharing. As a result, they can take into account the spatial information of the images. Architecture of a well-known ConvNet is shown in Figure 1.1.

Deep neural networks are also being used for many nonvision tasks including speech recognition [10]. However, in certain applications, traditional classification methods such as SVM and random forests are preferred over neural networks. As an example, when the dimension of data samples are high and there is not enough data samples, a neural network may struggle to find an optimal model. One reason is that during the optimization process of neural networks, the parameters are initialized randomly with noise. Because of that, the optimization process may stop at a poor local minima which is not the best solution. In comparison, SVM is a convex problem and the optimization process will

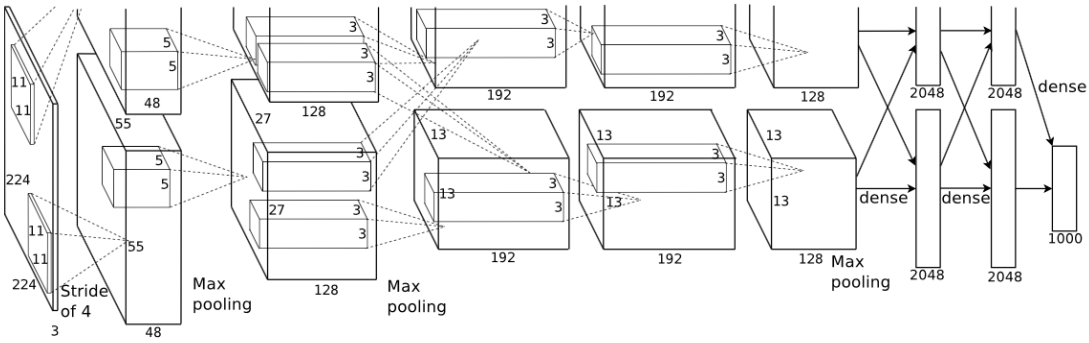


Figure 1.1: A well-known ConvNet structure called AlexNet [6]

always stop at the global minima.

We propose a model based on neural networks that provides an efficient high dimensional space coverage. In this aspect, our work is also related to space covering algorithms such as radial basis function (RBF) networks [11]. Our model is based on the *disjunctive normal form* [12]. We use conjunctions of a set of half-spaces to create a set of convex polytopes. Disjunction of these polytopes can provide desirable coverage of space. As a result of this specific network structure, we can initialize the network parameters in a meaningful manner instead of random noise. Therefore, we introduce an intuitive initialization scheme for the parameters that define the half-spaces. Then, we fine-tune these parameters using backpropagation of the gradient of a cost function. Since the starting point of the optimization process is determined carefully, we can use higher learning rates which leads to faster convergence. Based on this general idea, we propose a classification method named Logistic Disjunctive Normal Networks (LDNN) [13] and a regression method called Logistic Product Basis Network (LPBN) [14].

1.2 Semisupervised Learning

It is possible to improve the accuracy of a supervised classifier with unlabeled data. The methods that utilize unlabeled data to improve the accuracy of a supervised learning model are categorized as semisupervised learning [2]. It is a common belief that the learning process in human beings is more similar to semisupervised learning [15]. Humans are usually given some labeled examples and then the learning is continued by observing

and analyzing other instances which is usually an unsupervised process.

Successful training of a supervised model usually requires a large amount of labeled data. Specifically, ConvNets require large sets of labeled data to achieve their maximum accuracy [6]. However, gathering large sets of labeled data requires a lot of manual effort. On the other hand, unlabeled data is usually cheap to obtain. Therefore, semisupervised learning algorithms can improve the accuracy of a supervised model with relatively low cost.

Another argument for the usefulness of unlabeled data in supervised learning is that even though we don't know the category of unlabeled samples, they can help us to have a better understanding of the underlying distribution of the data. A simple example is shown in Figure 1.2 for illustration. In this example, we have 2 classes of rectangles and triangles. Green circles represent unlabeled data. If we only take labeled samples into account, naturally the optimal decision function would be a straight line placed in the space between samples of 2 classes as it is shown in the left figure. However, we can achieve a better understanding about the formation of the classes by analyzing the unlabeled data. In this simple example, we can observe that unlabeled data are mainly divided into 2 clusters. The first cluster of unlabeled data is closer to samples of triangle class. Therefore, it can be associated to that class. Similarly, the second cluster can be associated with the rectangle class. We can also observe that there is still a low density area between the samples of 2 classes even when we consider the unlabeled data. This low density space is where the decision function should reside. However, we can now observe that the original decision function depicted in the left of Figure 1.2 was not the optimal choice. This simple and intuitive observation is the basic principle of a class of semisupervised learning methods. Label propagation [16] is a well-known example.

In this dissertation, we propose 2 unsupervised loss functions. These loss functions can help a supervised classifier to learn about the distribution of the data using unlabeled samples. These unsupervised loss functions are general and can be used with any learning model which is based on optimization of a loss function using gradient descent.

The first loss function is named mutual-exclusivity [17]. With this loss, our goal is to push the decision boundary away from both labeled and unlabeled training samples in feature space. The motivation of this work is that since the object recognition and

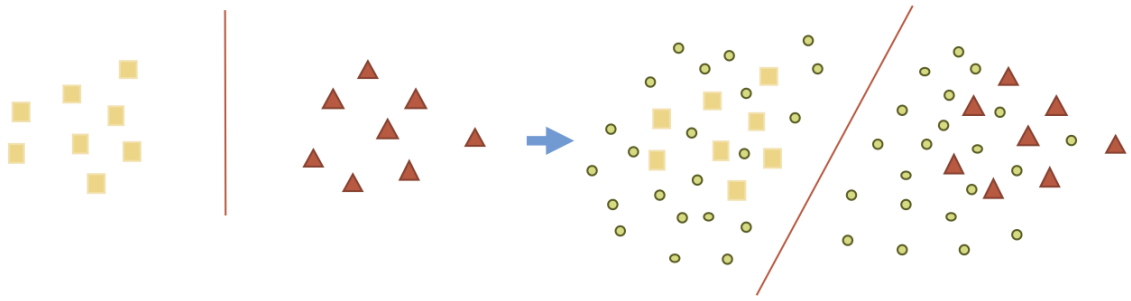


Figure 1.2: A simple illustration of how unlabeled data can improve the accuracy of a supervised classifier.

classification is a relatively easy task for humans, there exists a decision boundary that can separate samples of different classes almost perfectly. Although this decision function is highly complex in pixel space, the existence of such a function shows that the high dimensional manifolds representing each class of data don't intersect with each other and this decision function lies in the space between these manifolds where there is no or very few samples. Given an appropriate feature set, it is possible to push the decision boundary away from training samples by maximizing their margin. For this purpose, we don't need to know the label of training samples. We propose a regularization term based on unlabeled data that pushes the decision boundary towards less dense area of decision space. We use the disjunctive normal form to define this regularization function. In general, it forces the prediction vector of a classifier to be mutually-exclusive for every class. In other words, this loss function pushes the prediction vector to have one nonzero element.

The second loss function is called transformation/stability [18]. Our motivation for this work is based on data augmentation and randomness of learning systems. Data augmentation is an effective technique to artificially increase the size of training data. For example, we can randomly rotate, scale and transform a given image and use it as a new sample with the same label. This transformation usually happens randomly. In addition, most learning models exhibit some degree of randomness in their internal structure. As an example, *dropout* [19] is a well-known regularization technique for neural network approaches. Dropout regularizes a network by randomly turning off a set of hidden nodes

in the neural network structure. Therefore, the internal structure of a network is slightly different in different runs. As a result, even if we freeze the network parameters and pass a sample through the network multiple times, we will get slightly different outputs because of the randomized dropout.

Here we propose an unsupervised loss function that exploits the randomness in the processes mentioned above. The general idea is that the prediction of the network for a specific sample should be the same during multiple passes through the network. In other words, randomness should not change the prediction of a network for a specific sample when we pass that sample multiple times through the network. In summary, we pass a sample through the network n times to obtain n predictions of the same sample. Because of the randomized processes discussed earlier, these n predictions are not exactly the same. Our proposed loss function minimizes the difference between the n predictions. In addition we don't need to know the label of a sample to enforce this loss function. Regardless of the label of a sample, the prediction of the network should be the same during multiple passes.

These 2 unsupervised loss functions form the basis of a novel and state-of-the-art semisupervised learning method. We also show that the transformation/stability loss function can improve the robustness of a network with respect to adversarial examples. Even though neural networks and ConvNets are state-of-the-art for many tasks, it is possible to handcraft data samples that easily fool them. For example, although ConvNets achieved very high accuracy in object detection and recognition, it is possible to create 2 images that are visually identical to human eyes but ConvNet classified them as different classes. We show that we can use transformation/stability loss to improve the robustness of a network with respect to these adversarial examples.

1.3 Software

We made the code of the LDNN classifier and semisupervised learning method publicly available. The LDNN code is written in MATLAB. However, the computationally-intensive optimization part is written in C++. It can be downloaded from:

`http://www.sci.utah.edu/~mehdi/assets/ldnn_v01.tgz`

We made the code of our semisupervised learning method available using 2 publicly available deep learning frameworks: Spatially-sparse convolutional neural networks [20] and Caffe [21]. Both of these frameworks are highly-efficient implementations of ConvNets. We implemented both of our unsupervised loss functions as separate modules in these frameworks. We also provided examples of semisupervised learning networks that use these unsupervised loss functions. The implementations are on GPU using NVIDIA CUDA which provides large-scale processing power that can handle training of networks with millions of parameters and millions of training samples. The code is available at the following address:

`https://github.com/m-sajjadi`

1.4 Overview

Chapter 2 introduces disjunctive normal networks. In this chapter, first we describe the structure of the network and how it compares to RBFs and Multi Layer Perceptrons (MLP). Then we discuss how this specific structure allows for an intuitive initialization scheme. Next, we show the performance of the LDNN classifier on 2 synthetic datasets and compare it to related methods. We show how our initialization method improves the accuracy and convergence speed of the model. Finally, we perform extensive evaluations and comparisons using multiple binary and multiclass benchmark datasets.

In Chapter 3, we introduce our regression model logistic product basis networks (LPBN). First, we explain why conjunction of a set of half-spaces can be considered as basis functions for an effective and flexible feature space that presents data very well. Next, we describe our network which is based on this feature space and introduce our intuitive initialization scheme for the parameters of the network. We compare the accuracy of our model to traditional regression models using multiple benchmark datasets. Finally, using both synthetic and real-world benchmark datasets, we show that our network requires fewer parameters to represent a dataset compared to RBFs because of more efficient and flexible coverage of space.

We introduce mututal-exclusivity loss function in Chapter 4. We initially define a binary indicator function using disjunctive normal form and then replace this binary func-

tion with differentiable approximations. Using synthetic examples, we show that this loss function pushes the decision boundary towards less dense areas of space. Next, we evaluate this loss function on the task of object recognition using ConvNets. We experiment with different sizes of labeled set and unlabeled set and show that adding more unlabeled data improves the overall accuracy of the model.

Chapter 5 describes transformation/stability loss function. We also explain why this loss function is a natural complement of mutually-exclusive loss. Therefore, we introduce our semisupervised learning method based on both loss functions. Finally, we extensively evaluate our method using ConvNets on object recognition benchmarks and show that our method is a competitive and state-of-the-art semisupervised learning model.

CHAPTER 2

DISJUNCTIVE NORMAL NETWORKS

Artificial neural networks are powerful pattern classifiers. They form the basis of the highly successful and popular convolutional neural networks which offer the state-of-the-art performance on several computer visions tasks. However, in many general and nonvision tasks, neural networks are surpassed by methods such as support vector machines and random forests that are also easier to use and faster to train. One reason is that the backpropagation algorithm, which is used to train artificial neural networks, usually starts from a random weight initialization which complicates the optimization process leading to long training times and increases the risk of stopping in a poor local minima. Several initialization schemes and pretraining methods have been proposed to improve the efficiency and performance of training a neural network. However, this problem arises from the architecture of neural networks. We use the disjunctive normal form and approximate the Boolean conjunction operations with products to construct a novel network architecture. The proposed model can be trained by minimizing an error function and it allows an effective and intuitive initialization which avoids poor local minima. We show that the proposed structure provides efficient coverage of the decision space which leads to state-of-the art classification accuracy and fast training times.

2.1 Introduction

An artificial neural network (ANN) consisting of one hidden layer of squashing functions is an universal approximator for continuous functions defined on the unit hypercube [22]-[23]. However, until the introduction of the backpropagation algorithm [24], training such multilayer perceptron (MLP) networks was not possible in practice. The backpropagation algorithm propelled MLPs to be the method of choice for many classification and regression applications. The success of neural networks culminated by convolutional neural networks [25]-[26] (ConvNets), which deliver the current state-of-the

art performance on many computer vision problems including but not limited to classification, detection, localization and scene labeling [27] -[31].

However, MLPs are not always the classifier of choice when it comes to general tasks. In this regard, other techniques such as support vector machines (SVM) [3] and random forests (RF) [5] are the preferred options. The computational cost of training fully connected MLPs can be high yet they don't deliver the best accuracy possible, especially when the size and dimensionality of the dataset grows. An underlying reason is that the optimization process for training MLPs can become more complicated in higher dimensions. These optimization methods usually start from a random starting point and use the gradient descent to find a locally optimal solution. However, this starting point can be anywhere in the high-dimensional space and possibly in the energy well of a poor local optima. Therefore, it may take the gradient descent a lot of iterations to get to a solution which might possibly be significantly suboptimal compared to other local minima. This increases the variation in training times. On the other hand, this random initialization may put the initial point near a local minima and consequently lead the gradient descent to a false local minima solution. Neural networks also suffer from the herd-effect problem [32]. During backpropagation, each hidden unit tries to evolve into a useful feature detector from a random initialization; however, this task is complicated by the fact that all units are changing at the same time without any direct communication between them. Consequently, hidden units cannot effectively subdivide the necessary computational tasks among themselves leading to a complex dance which can take a long time to settle down.

Another related classification approach is to partition the decision space or cover the desired parts of space and then treat each partition or part of the covered space accordingly. Radial Basis Functions Networks (RBF) are popular examples of these methods. RBFs are commonly used models which use radial functions such as Gaussians as basis functions [11]. Each radially symmetric Gaussian is tuned to respond to a local region of feature space. One important drawback of such models is that their local coverage can be inefficient when nonlocal coverage of space is needed [14]. In other words, so many local radial basis functions are needed to cover a nonlocal part of the decision space sufficiently. The reason is that each of these radial functions only covers a local part of the space. It is also well known that an RBF network suffers from the curse of dimensionality [33].

As the number of dimensions grow, the number of radial basis functions required grows exponentially.

In this chapter, we introduce a new network architecture that overcomes the difficulties associated with MLPs and backpropagation for supervised learning. Our model also provides an efficient space coverage which can either be local or nonlocal. This is done by designing a set of convex polytopes that unlike RBFs are flexible in shape and adaptive in coverage. Our network consists of one adaptive layer of feature detectors implemented by logistic sigmoid functions followed by 2 fixed layers of logical units that compute conjunctions and disjunctions, respectively. We call the proposed network architecture Logistic Disjunctive Normal Network (LDNN). Unlike MLPs, LDNNs allow for a simple and intuitive initialization of the network weights which avoids the herd-effect. Furthermore, due to the single adaptive layer, it allows larger step sizes in minimizing the error function. Finally, we present results of experiments conducted on 10 binary and 6 multiclass classification problems. We repeated each trial repeatedly and reported the mean, min and max of error rates for each problem in order to consider the variability in the results. LDNNs outperformed MLPs in every case and produced the best accuracy in 11 out of the 16 classification problems in comparison to SVMs and RFs.

2.2 Related Work

Extensive research has been performed to improve the performance of the backpropagation algorithm including batch vs. stochastic learning [34]-[35], squared error vs. cross-entropy [36] and optimal learning rates [37]-[38]. Many other practical choices including normalization of inputs, initialization of weights, stopping criteria, activation functions, target output values that will not saturate the activation functions, shuffling training examples, momentum terms in optimization, and optimization techniques that make use of the second-order derivatives of the error are summarized in [39]. More recently, Hinton *et al.* proposed a Dropout scheme for backpropagation which helps prevent coadaptation of feature detectors [19]. Despite the extensive effort devoted to making learning MLPs as efficient as possible, the fundamental problems outlined in Section 2.1 remain because they arise from the architecture of MLPs. There are several initialization and unsupervised pretraining methods proposed to alleviate the the herd-effect problem. For example, Con-

trastive divergence [40]-[41] can be used to pretrain networks in an unsupervised manner prior to backpropagation. Contrastive divergence has been used successfully to train deep networks. The LDNN model proposed in this chapter can be seen as an architectural alternative for supervised learning of one hidden layer ANNs.

The idea of representing classification functions in disjunctive form has been previously explored in the literature. Fuzzy min-max networks [42] -[44] represent the classification function as the union of axis aligned hypercubes in the feature space. The most important drawback of this model is its limitation to axis aligned decision boundaries which can significantly increase the number of conjunctions necessary for a good approximation. We construct a significantly more efficient approximation by using an union of convex polytopes. Furthermore, fuzzy min-max neural networks employ an adhoc expansion-contraction scheme for learning, whereas we formulate learning as an energy minimization problem. Lu *et al.* [45] proposed a multisieving network that decomposes learning tasks. Lee *et al.* [46] proposed a disjunctive fuzzy network which is based on prototypes; however, it lacks an objective function and is based on an adhoc training procedure. Similarly, the modular network proposed by Lu and Ito [47] removes the axis aligned hypercube restriction from fuzzy min-max networks; however, their network cannot be learned by minimizing a single energy function. Our LDNN model uses differentiable activation functions which makes it possible to optimize the network parameters in an unified manner by minimizing a single energy function. We show that unified training of our classifier results in very significant accuracy advantages over the modular network. Differentiable approximations of min-max functions have been used to construct fuzzy neural network that can be trained using steepest descent [48]- [51], but these have produced results that are significantly less accurate than state-of-the-art classification techniques. A closely related approach to ours is adaptive mixtures of local experts which uses a gating network to stochastically select the output from a set of feedforward networks [52]. The reader is referred to [53] for a survey of mixture of expert methods. The products of experts approach models complex probability distributions by multiplying simpler distributions is also related [54].

Besides the network approaches discussed in the previous paragraph, the idea of partitioning the decision space and learning simpler decision functions in each partition has

been explored. RBFs mentioned in the previous section are related to this approach. The set of radial basis functions are usually Gaussians and their parameters can be obtained by unsupervised clustering of the data or fitting a Gaussian mixture model to our data using the EM algorithm [55]. Then, we can use linear regression to obtain a set of linear weights to combine the responses of the basis functions. It is also possible to learn the RBF parameters in an unified manner by back-propagation of the error using chain rule [56]. Mixture discriminant analysis treats each class as a mixture of Gaussians and learns discriminants between the Gaussians [57]. Subclass discriminant analysis also relies on modeling classes as mixtures of Gaussians prior to learning discriminant [58]. Local linear discriminant analysis clusters the data and learns a linear discriminant in each cluster [59]. In these approaches, partitioning of the space is treated as a step independent from the supervised learning step. Wang and Saligrama proposed a more recent approach that unifies space partitioning and supervised learning [60]. While this method is related in concept to our disjunctive learning, in Section 2.4.3, we show that LDNNs outperform space partitioning by a large margin. Dai *et al.* proposed an approach which places local classifiers close to the global decision boundary [61]. Toussaint and Vijayakumar propose a products-of-sigmoids model for discontinuously switching between local models [62]. Another approach greedily builds a piecewise linear classifier by adding classifiers in regions of error clusters [63]. Local versions of SVMs have also been explored [64]-[65]. A specific type of local classification is based on the idea of pairwise coupling between positive and negative examples or clusters is conceptually close to the initialization we propose for our LDNN model. These methods typically employ a clustering algorithm, learning classifiers between pairs of positive and negative clusters found by clustering, finally followed by a combination scheme such as voting to integrate the pairwise classifiers into a single decision [66]- [72]. The modular network [47] discussed previously also falls into this category.

There are other methods that share similarities with our proposed structure. One recent example proposed by Goodfellow *et al.* is Maxout Networks [31]. Instead of using an activation function over the output of a single node, they take the maximum output of a group of hidden nodes as the output. Here, the Max operator acts as an activation function. Maxout is similar to our model in a sense that it combines the output of a set of

linear functions using Max operator. However, as we explain in Section 2.3, sigmoids are important elements of our model which are not present in Maxout. We also propose a very consistent and intuitive initialization scheme for our model. We provide comparisons with Maxout networks and show that they are outperformed by LDNN. Another work similar to our approach is Sum-Product Networks (SPN) [73]. SPNs are probabilistic models that provide tractable inferences. SPN is based on the notion of a *network polynomial* and represents unnormalized probability distributions. This leads to a deep structure with interleaved layers of sums and products. SPN is similar to our proposed structure because it uses sum units to mix different submodels. It also uses products that combine features of submodels. However, our approach is different. Unlike SPNs, we use logistic sigmoid functions before the product layer to approximate half-spaces. Then, we use products to form convex polytopes. Sigmoid functions are not present in SPNs but as mentioned earlier, they are crucial components of our approach.

2.3 Methods

2.3.1 Network Architecture

Consider the binary classification problem $f : \mathbf{R}^n \rightarrow \mathbf{B}$ where $\mathbf{B} = \{0, 1\}$. Let

$$\Omega^+ = \{\mathbf{x} \in \mathbf{R}^n : f(\mathbf{x}) = 1\} \tag{2.1}$$

Let's approximate Ω^+ as the union of N convex polytopes

$$\tilde{\Omega}^+ = \cup_{i=1}^{N} \mathcal{P}_i \tag{2.2}$$

where the i'th polytope is the intersection $\mathcal{P}_i = \cap_{j=1}^{M_i} \mathcal{H}_{ij}$ of M_i half-spaces

$$\mathcal{H}_{ij} = \{\mathbf{x} \in \mathbf{R}^n : h_{ij}(\mathbf{x}) > 0\} \tag{2.3}$$

We can replace M_i with $M = \max_i M_i$ without loss of generality. H_{ij} is defined in terms of its indicator function

$$h_{ij}(\mathbf{x}) = \begin{cases} 1, & \sum_{k=1}^{n} w_{ijk} x_k + b_{ij} \geq 0 \\ 0, & otherwise \end{cases}, \tag{2.4}$$

where w_{ijk} and b_{ij} are the weights and the bias term. Any Boolean function $b : \mathbf{B}^n \rightarrow \mathbf{B}$ can be written as a disjunction of conjunctions, also known as the disjunctive normal form [12].

Hence, we can construct the function

$$\tilde{f}(\mathbf{x}) = \bigvee_{i=1}^{N} \underbrace{\left(\bigwedge_{j=1}^{M} h_{ij}(\mathbf{x}) \right)}_{b_i(\mathbf{x})} \tag{2.5}$$

such that

$$\tilde{\Omega}^{+} = \{\mathbf{x} \in \mathbf{R}^n : \tilde{f}(\mathbf{x}) = 1\} \tag{2.6}$$

Since $\tilde{\Omega}^{+}$ is an approximation to Ω^{+}, it follows that $\tilde{f}$ is an approximation to f. Our next step is to provide a differentiable approximation to this disjunctive normal form. First, the conjunction of binary variables $\bigwedge_{j=1}^{M} h_{ij}(\mathbf{x})$ can be replaced by the product $\prod_{j=1}^{M} h_{ij}(\mathbf{x})$. Then, using De Morgan's laws [12], we can replace the disjunction of the binary variables $\bigvee_{i=1}^{N} b_i(\mathbf{x})$ with $\neg \bigwedge_{i=1}^{N} \neg b_i(\mathbf{x})$, which in turn can be replaced by the expression

$$1 - \prod_{i=1}^{N} (1 - b_i(\mathbf{x})) \tag{2.7}$$

Finally, we can approximate the perceptrons $h_{ij}(\mathbf{x})$ with the logistic sigmoid functions

$$\sigma_{ij}(\mathbf{x}) = \frac{1}{1 + e^{-\sum_{k=1}^{n} w_{ijk} x_k + b_{ij}}}. \tag{2.8}$$

This yields the differentiable approximation to $\tilde{f}$

$$\hat{f}(\mathbf{x}) = 1 - \prod_{i=1}^{N} (1 - \underbrace{\prod_{j=1}^{M} \sigma_{ij}(\mathbf{x})}_{g_i(\mathbf{x})}), \tag{2.9}$$

which can also be visualized as a network (Figure 2.1). We refer to the proposed network architecture as LDNN. The only adaptive parameters of the LDNN are the weights and biases of the first layer of logistic sigmoid functions. The second layer consists of N soft NAND gates which implement the logical negations of the conjunctions $g_i(\mathbf{x})$ using products. The output layer is a single soft NAND gate which implements the disjunction using De Morgan's law. We will refer to a LDNN classifier which has N NAND gates in the second layer and M discriminants per NAND gate as a $N \times M$ LDNN. Note that other variations of disjunctive normal networks can be constructed by using any classifier that is differentiable with respect to its parameters in place of the logistic sigmoid functions.

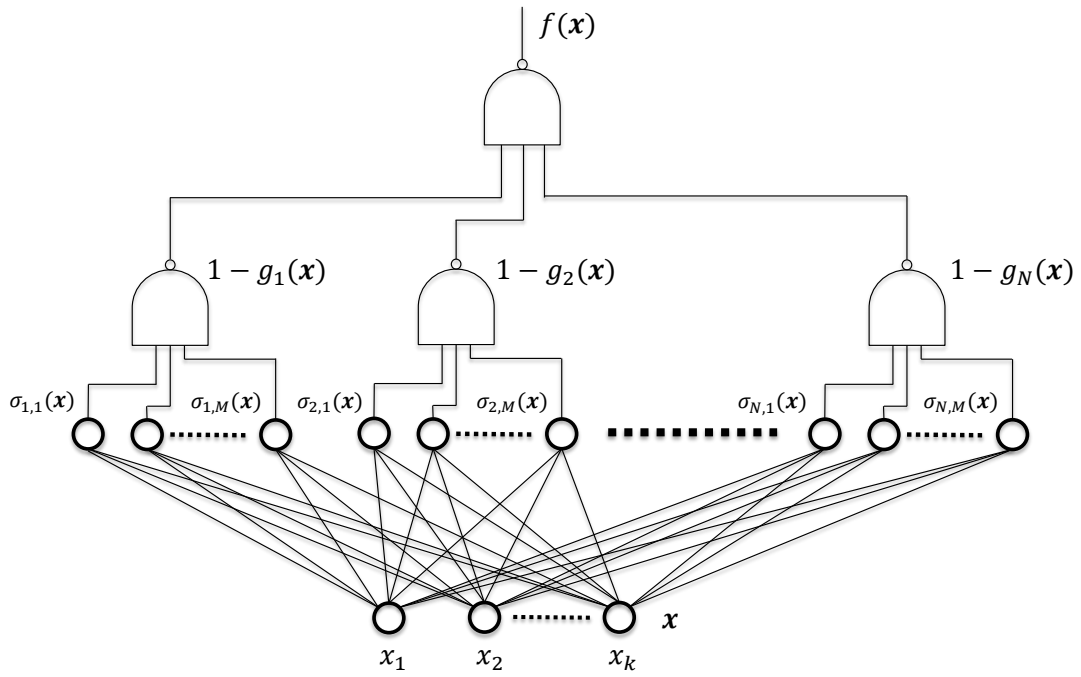


Figure 2.1: LDNN architecture. The first hidden layer is composed of $M \times N$ logistic sigmoid functions. The second hidden layer computes the logical negation of N conjunctions using soft NAND gates. The output layer computes the disjunction. The soft NAND gates are implemented as continuous functions by subtracting the product of their inputs from 1.

2.3.2 Model Initialization

Consider a set of training examples $\Gamma = \{(\mathbf{x}, y(\mathbf{x}))\}$ where $y(\mathbf{x})$ denotes the desired binary class corresponding to $\mathbf{x}$. Let Γ^+ and Γ^- be the subsets of Γ for which $y = 1$ and $y = 0$, respectively. The disjunctive normal form permits a very simple and intuitive initialization of the network weights. To initialize a $N \times M$ LDNN, we first partition Γ^+ and Γ^- into N and M clusters, respectively. Let $\mathbf{v_{ij}} = \mathbf{c}_i^+ - \mathbf{c}_j^-$ where $\mathbf{c}_i^+$ and $\mathbf{c}_j^-$ are the centroids of the i'th positive and j'th negative clusters, respectively. We initialize the weight vectors as $\mathbf{w_{ij}} = \mathbf{v_{ij}}/|\mathbf{v_{ij}}|$. Finally, we initialize the bias terms b_{ij} such that the logistic sigmoid functions $\sigma_{ij}(\mathbf{x})$ take the value 0.5 at the midpoints of the lines connecting the positive and negative cluster centroids. In other words, let

$$b_{ij} = \langle \mathbf{w}_{ij}, 0.5(\mathbf{c}_i^+ + \mathbf{c}_j^-) \rangle \tag{2.10}$$

where $\langle \mathbf{a}, \mathbf{b} \rangle$ denotes the inner product of the vectors $\mathbf{a}$ and $\mathbf{b}$. This procedure initilizes $g_i(\mathbf{x})$, the i'th conjunction in the second hidden layer of the LDNN, to a convex polytope

which aims to separate the training instances in the i'th cluster of Γ^+ from all training instances in Γ^-.

We give an intuitive description of LDNN initialization in the context of the 2 moons dataset. An illustration of this dataset and 3 clusters for each of the 2 classes are shown in (Figure 2.2a). Initial discriminants for the positive clusters taken one at a time are shown in (Figure 2.2b-d). The conjunction of these discriminants form convex polytopes for the positive clusters (Figure 2.2e-g). The disjunction of these conjunctions before and after weight optimization (Section 2.3.3) are illustrated in (Figure 2.2h). This initialization procedure is similar to the modular neural network proposed by Lu and Ito *(12)* as well as to locally linear classification by pairwise coupling *(20)* in general. Each module in Lu and Ito's modular network independently learns a linear classifier between a pair of positive and negative training data clusters. The key difference of our classifier from Lu and Ito's network, as well as from locally linear classification by pairwise coupling in general, is that we learn all the linear discriminants simultaneously by minimizing a single error function. When each module is trained independently, the success of the initial clustering can strongly influence the outcome. In Section 2.4, we show, using both real and artificial datasets, that this important disadvantage can create very significant differences in classification accuracy between modular networks and LDNNs.

2.3.3 Model Optimization

The LDNN model can be trained by choosing the network weights and biases that minimize the quadratic error

$$E(\mathcal{W}, \Gamma) = \sum_{(\mathbf{x},y)\in\Gamma} \left(y - f(\mathbf{x})\right)^2, \tag{2.11}$$

where f is determined by the set of network weights and biases $\mathcal{W}$. Starting from an initialization as described in Section 2.3.2, we minimize (2.11) using gradient descent. To derive the update equations, we need to find the partial derivatives of the error with respect to the network weights and biases. Using the fact that $\partial\sigma_{ij}/\partial w_{pqk}$ is nonzero only when $i = p$ and $j = q$, the derivatives of the error function with respect to the network weights are obtained using the chain rule

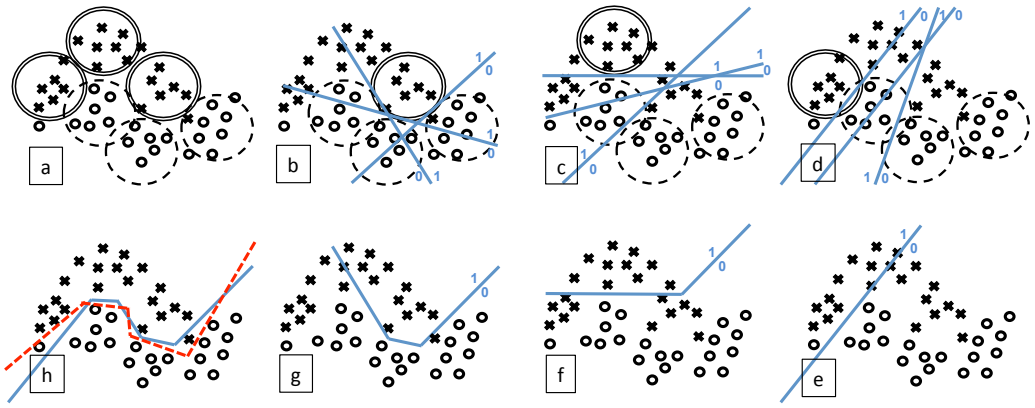


Figure 2.2: A binary classification problem: (a) positive and negative training examples partitioned into 3 clusters each; linear discriminants from each negative cluster to (b) the first positive cluster, (c) the second positive cluster and (d) the third positive cluster; the conjunction of the discriminants for (g) the first positive cluster, (f) the second positive cluster and (e) the third positive cluster; (h) the disjunction of the conjunctions before (blue) and after (red) gradient descent. The 1/0 pair on the sides of the discriminants represent the direction of the discriminant.

$$\begin{aligned}\frac{\partial E}{\partial w_{ijk}} &= \frac{\partial E}{\partial f}\frac{\partial f}{\partial g_i}\frac{\partial g_i}{\partial \sigma_{ij}}\frac{\partial \sigma_{ij}}{\partial w_{ijk}} \\ &= -2(y - f(\mathbf{x}))\left(\prod_{r\neq i}(1 - g_r(\mathbf{x}))\right) \times \\ &\quad \left(\prod_{l\neq j}\sigma_{il}(\mathbf{x})\right)\left(\sigma_{ij}(\mathbf{x})(1 - \sigma_{ij}(\mathbf{x}))x_k\right) \\ &= 2(f(\mathbf{x}) - y)\left(\prod_{r\neq i}(1 - g_r(\mathbf{x}))\right) g_i(\mathbf{x})\left(1 - \sigma_{ij}(\mathbf{x})\right) x_k \end{aligned} \tag{2.12}$$

Similarly, we obtain the derivative of the error function with respect to the network biases as

$$\frac{\partial E}{\partial b_{ij}} = 2(f(\mathbf{x}) - y)\left(\prod_{r\neq i}(1 - g_r(\mathbf{x}))\right) g_i(\mathbf{x})\left(1 - \sigma_{ij}(\mathbf{x})\right) \tag{2.13}$$

We perform stochastic gradient descent after randomly permuting the order of the instances in Γ and updating the model weights and biases according to

$$w_{ijk}^{new} = w_{ijk} - \alpha\frac{\partial E}{\partial w_{ijk}} \tag{2.14}$$

and

$$b_{ij}^{new} = b_{ij} - \alpha\frac{\partial E}{\partial b_{ij}} \tag{2.15}$$

respectively. The constant α is the step size. This constitutes one epoch of training. Multiple epochs are performed until convergence as determined using a separate validation set.

Notice that it is possible to achieve 0 training error for any finite training set Γ by letting each positive training instance and each negative training instance represent a positive and negative cluster centroid, respectively. However, in practice, this is expected to lead to overfitting and poor generalization and typically a much smaller number of clusters than training instances is used.

2.4 Experiments

2.4.1 Artificial Datasets

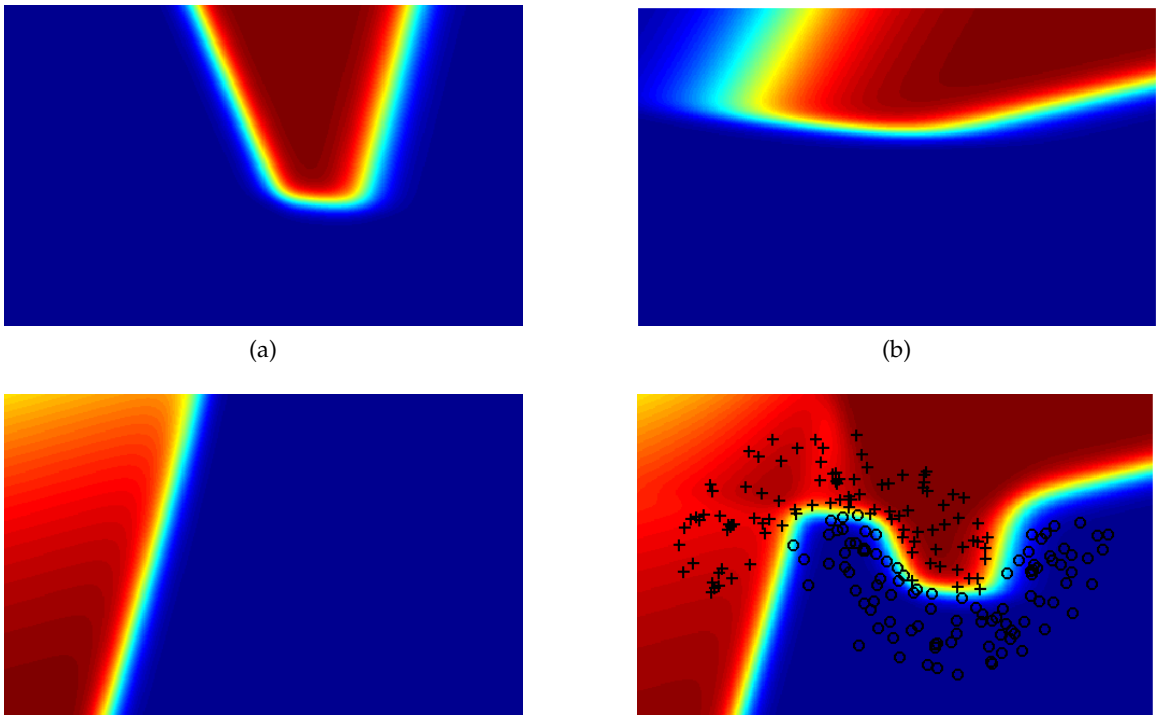
We first experimented with the *2 moons* artificial dataset to evaluate the LDNN algorithm with and without the proposed clustering initialization. We also compare the LDNN model with the modular neural networks(ModN) [47]. To construct the 2 moons dataset, we start by generating random radius and angle pairs (r, θ). For both moons, r is an uniform random variable between $R - W/2$ and $R + W/2$ where R and W are parameters that determine the radius and the width of the moons, respectively. For the top moon, θ is an uniformly distributed random variable between 0 and π. For the bottom moon, θ is an uniformly distributed random variable between π and 2π. The Cartesian coordinates for data points on the top and bottom moons are then generated as $(R\cos\theta, R\sin\theta)$ and $(R\cos\theta - W/2, R\sin\theta - \alpha)$, respectively. The parameter α determines the vertical separation between the 2 moons. We generated a training and a testing dataset by using the parameters $R = 1$, $W = 0.7$, $\alpha = -0.7$ which generates slightly overlapping classes. Both datasets contained 1000 instances on the top moon and 1000 instances on the bottom moon. Then, for each $n \in [1, 7]$, we trained 50 $n \times n$ LDNNs starting from random parameter initializations, 50 $n \times n$ LDNNs initialized from k-means clustering with n clusters per moon and 50 $n \times n$ ModNs initialized from k-means clustering with n clusters per moon. For ModNs, the n^2 linear discriminants are trained independently using data from the n^2 pairs of positive (top moon) and negative (bottom moon) clusters and then combined using min/ma functions. We used stochastic gradient descent with a step size of 0.3, a momentum term weight of 0.1 and 500 epochs for training all models. Testing accuracies were computed over the second dataset which was not used in training. Table 2.1 shows the mean, minimum and maximum testing error over the 50 trials for each of the models. We observe that training the LDNN model starting from a random initialization is successful

Table 2.1: Average, min. and max. testing error percentages over 50 repetitions for LDNN initialized with random parameters, initialized with clustering and ModN [47] initialized with clustering for different model sizes.

n	LDNN random init		LDNN cluster init		ModN cluster init	
	Av.	Range	Av.	Range	Av.	Range
1	15.6	[15.2, 18.6]	15.6	[15.2, 20.2]	15.5	[15.2, 16.3]
2	6.6	[3.0, 15.8]	3.3	[2.9, 3.7]	4.2	[3.6, 5.4]
3	4.1	[1.1, 15.6]	2.3	[1.2, 3.5]	2.7	[1.2, 4.8]
4	3.6	[1.2, 15.6]	2.2	[1.3, 3.5]	3.0	[1.8, 5.2]
5	3.4	[1.2, 15.4]	2.2	[1.2, 4.2]	2.8	[1.4, 5.7]

in general; however, the range of testing error rates varies by a larger amount compared to when a cluster initialization is used resulting in a slightly worse mean testing error. We also note that the LDNN model performs better both on average and when comparing the maximum error rates over the 50 trials than the ModN model. Figure 2.3 illustrates the output of the LDNN model for $n = 3$, which appears to be an appropriate choice based on Table 2.1. The outputs of the 3 conjunctions are also shown separately to give further intuition into the behavior of the LDNN model. Notice the similarity to Figures 2.2(e-h)).

The *2-spirals* dataset is an extremely difficult dataset for the MLP architecture trained with the backpropagation algorithm [32]. The original dataset consists of 194 (x, y) pairs arranged in 2 interlocking spirals that orbit the origin 3 times. The classification task is to determine which spiral any given (x, y) point belongs to. We used the farthest distance clustering algorithm [74] for initialization of both models. The k-means clustering algorithm places most centroids near the origin where the data points are denser and fewer centroids on the spiral arms further from the origin where the data is sparser. On the other hand, the farthest distance clustering algorithm provides more uniformly distributed centroids which leads to better classification results with fewer clusters. We performed clustering with maximum distance thresholds 2.2, 2.0 and 1.5 resulting in 18, 21 and 27 clusters per class, respectively. For each of these, we trained a LDNN and a ModN. Note that the number of parameters in both models is the same for the same number of clusters. We used stochastic gradient descent with a step size of 0.3, a momentum term weight of 0.1 and $2,000$ epochs for training all models. LDNN achieved 0 percent training error in each



(a) (b) (c) (d)

Figure 2.3: Two moons test set: (a)-(c) the 3 conjunctions in the second layer of the network evaluated individually, and (d) the output of the 3×3 LDNN. +/o symbols denote the 2 classes.

of these cases while the ModN's training error was 0.232, 0.062 and 0 percent, respectively. These results suggest that the unified learning framework of LDNN is able to capture the spiral dataset with many fewer parameters than independent, pairwise learning of discriminants as in [47]. Furthermore, it can be seen from Figure 2.4 that LDNN creates a much smoother approximation to the spirals than pairwise learning. Finally, we note that LDNN initialized randomly was not able to find a satisfactory local minimum of the error function via gradient descent. This is similar to the failure of the standard MLP architecture for this dataset. This observation underlines the importance of the existence of an intuitive initialization for the LDNN architecture.

2.4.2 Two-class Problems

We experimented with 10 different binary classification datasets from the *UCI Machine Learning Repository* [75] and the *LIBSVM Tools* webpage [76]. For each dataset, we trained the LDNN, ModN, MLP, SVM, RF, Maxout and RBF classifiers.

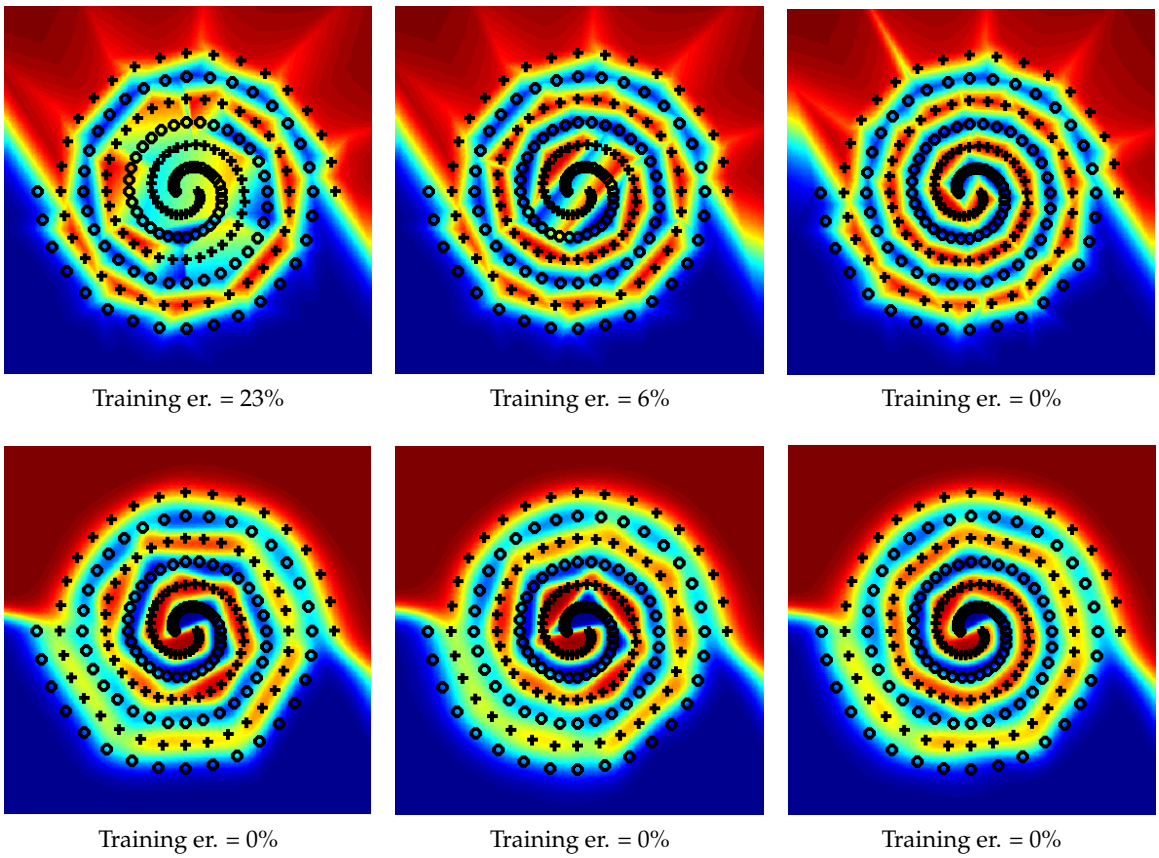


Figure 2.4: Two spirals dataset: ModN (top) and LDNN (bottom). First column: 18 clusters/class, Second column: 21 clusters/class, Third column: 27 clusters/class

2.4.2.1 Dataset normalization, training/testing set split

Datasets were normalized as follows: For LDNN, ModN, MLP, Maxout and RBF training, we applied a whitening transform [74] to datasets with a large number of training instances (Forest cover type and Webspam) since the covariance matrix could be estimated reliably. All other datasets were normalized by centering each dimension of the feature vector at the origin by subtracting its mean and then scaling by dividing it with its standard deviation. For SVM training, each dimension of the feature vector was linearly scaled to the range $[0, 1]$. For RF training, no normalization is necessary.

The IJCNN and COD RNA binary datasets had previously determined training and testing sets. For the rest of the datasets, we randomly picked 2/3's of the instances for training and the rest for testing. The training set was further randomly split into a training (%90) and cross-validation (%10) set for determining the parameters of every method.

2.4.2.2 Model and classifier training parameter selection

For LDNN classifiers, we need to choose the number of NAND gates (N) and the number of discriminants per group (M). These parameters translate into the number of positive and negative clusters, respectively, in the initialization. Various combinations, up to 40 clusters per class, were tried to find the selection that gives the best accuracy on validation set. For any given number of clusters, the k-means algorithm was repeated 50 times and the clustering result with the lowest sum of square distances to nearest cluster centroid was selected to initialize the LDNN weights. We also fine tuned the step size for gradient descent. The number of epochs for training was selected using the cross-validation set and early stopping. For the IJCNN dataset, the cross-validation set was also used in training as in [77].

For MLP training, there are 2 main parameters. The first one is the number of hidden nodes which was varied from 2 to 500 to find the best accuracy on validation set. This was followed by fine tuning the step size for backpropagation. The number of epochs was chosen using the cross-validation set and early stopping. We also trained a second MLP classifier (MLP-m) for which the number of hidden nodes was chosen as $N \times M$ to match the total number of logistic sigmoid functions in the LDNN classifier. This was done to compare LDNN to a MLP with approximately the same degrees of freedom. The optimal parameters for 4 datasets were already matched to LDNN parameters. Similarly, a modular network, which we refer to as Mod-N, with the same number of conjunctions and disjunctions as the LDNN classifier was trained to control for the degrees of freedom.

There are 3 main parameters involved in RF training. The first one is the number of trees. We choose a sufficiently large number of trees to ensure that the out of bag error rate converges. The second parameter is the number of features that will be considered in every node of the tree. We tried a range of numbers around the square root of the number of features [5]. The last parameter is the fraction of total samples that will be used in the construction of each tree. We tried $2/3$, $1/2$, $1/3$, $1/4$ and $1/5$ as possible values for this parameter.

For SVM training, a RBF kernel was used for all datasets except for the MNIST dataset for which a 9th degree polynomial kernel was used [40]. For all datasets except MNIST, we used the grid search tool provided by the Libsvm guide [76] to set the parameters of

the RBF kernel.

For Maxout, we need to find the number of hidden nodes h and the number of linear functions per hidden node m. We tried different combinations and picked the one resulting in the best performance on the validation set. Similar to LDNN and MLP, we fine tuned the step size for gradient descent. The number of epochs for training was also selected using the cross-validation set and early stopping.

RBFs were trained using Netlab [78]. Netlab performs a few steps of k-means to initialize unsupervised learning of a Gaussian Mixture Model using the Expectation-Maximization (EM) algorithm. The predicted value is calculated by linearly combining Gaussian kernels. Linear weights are obtained by least squares fitting. For every dataset, we need to find the number of basis functions. So, we tried up to several hundred basis functions to pick the best using the cross-validation set.

The training and model parameters selected for all models are listed in Table 2.2.

2.4.2.3 Results

All of the classifiers we consider, with the exception of SVM and RBF, are stochastic. Therefore, each experiment with the exception of SVM and RBF was repeated 50 times to obtain mean, minimum and maximum testing errors which are reported in Table 2.2 for all classifiers. The LDNN classifier outperformed MLPs for all datasets. Furthermore, LDNNs also outperform MLP-m in all datasets. All algorithms were run on an *Intel i7-3770* 3.4 Ghz CPU.

Our results signify that the LDNN network architecture and training offers a more accurate alternative to MLPs using backpropagation. The LDNN classifier also outperformed the Mod-N classifier in all datasets including several datasets such as *Forest cover type* and *Wisconsin breast cancer* where the accuracy difference was very large. This emphasizes the importance of training the entire network in an unified manner. Considering all of the classifiers tested, LDNNs had the lowest testing error in 7 out of 10 datasets. LDNNs outperformed SVMs in 8 out of 10 cases, Maxouts in 8 out 10 cases and RFs in 7 out of 10 cases.

Table 2.2: **Column 1**: Binary classification datasets, their source, number of positive / negative training/testing examples and data dimensionality. **Column 2**: Classifier type. **Column 3-4**: Average testing and [min,max] testing error (%) Best average testing errors are shown in bold. **Column 5**: Model and classifier training parameters used. LDNN, Mod-N: $N \times M$ model and ϵ step size. MLP and MLP-m: h number of hidden nodes and ϵ step size. RF: t number of trees, f number of features considered per node and s training instance sampling rate for each tree. SVM: C penalty factor, γ: RBF kernel width. Maxout: h number of hidden nodes, m number of linear discriminants per hidden node and ϵ step size. RBF: h number of radial basis functions.

Dataset	Classifier	Av. test error	Test error range	Model Parameters
Adult	LDNN	15.25	[15.14, 15.41]	7×4, $\epsilon = 0.007$
UCI	MLP	15.37	[15.17, 15.74]	$h = 20$, $\epsilon = 0.005$
Train:	RF	**14.14**	[13.97, 14.30]	$t = 300$, $f = 3$, $s = 2/3$
7,508+ / 22,654-	SVM	15.57	—	$C = 32768$, $\gamma = 0.007812$
Test:	Mod-N	17.39	[16.32, 20.51]	7×4, $\epsilon = 0.007$
3,700+ / 11,306-	MLP-m	15.43	[15.14, 15.81]	$h = 28$, $\epsilon = 0.007$
Dim = 14	Maxout	15.44	[15.14, 15.69]	$h = 5$, $m = 4$, $\epsilon = 0.005$
	RBF	15.67	—	$h = 125$
Wisconsin cancer	LDNN	**0.80**	[0.52, 1.58]	2×1, $\epsilon = 0.05$
UCI	MLP	1.37	[0.52, 2.64]	$h = 15$, $\epsilon = 0.05$
Train:	RF	1.79	[1.58, 2.11]	$t = 300$, $f = 10$, $s = 2/3$
142+ / 238-	SVM	1.59	—	$C = 2048$, $\gamma = 0.000488$
Test:	Mod-N	14.58	[7.93, 24.33]	2×1, $\epsilon = 0.05$
70+ / 119-	MLP-m	1.59	[0.52, 2.11]	$h = 2$, $\epsilon = 0.05$
Dim = 30	Maxout	1.58	[0.52, 3.17]	$h = 2$, $m = 3$, $\epsilon = 0.05$
	RBF	1.58	—	$h = 14$
PIMA diabetes	LDNN	**17.92**	[17.25, 19.60]	6×10, $\epsilon = 0.02$
UCI	MLP	19.34	[16.86, 23.13]	$h = 100$, $\epsilon = 0.01$
Train:	RF	20.81	[20.39, 21.56]	$t = 150$, $f = 2$, $s = 1/5$
179+ / 334-	SVM	21.57	—	$C = 32$, $\gamma = 0.125$
Test:	Mod-N	24.29	[19.60, 27.05]	6×10, $\epsilon = 0.02$
89+ / 166-	MLP-m	19.56	[17.64, 22.35]	$h = 60$, $\epsilon = 0.02$
Dim = 8	Maxout	20.75	[17.64, 23.92]	$h = 6$, $m = 10$, $\epsilon = 0.005$
	RBF	18.82	—	$h = 12$
Australian credit	LDNN	**12.93**	[12.22, 13.53]	5×4, $\epsilon = 0.02$
UCI	MLP	13.90	[12.22, 15.28]	$h = 20$, $\epsilon = 0.01$
Train:	RF	12.95	[12.22, 13.10]	$t = 150$, $f = 1$, $s = 1/5$
205+ / 256-	SVM	16.59	—	$C = 0.03125$, $\gamma = 0.5$
Test:	Mod-N	14.62	[12.22, 17.46]	5×4, $\epsilon = 0.02$
1012+/127-	MLP-m	14.03	[11.79, 16.15]	$h = 20$, $\epsilon = 0.02$
Dim = 14	Maxout	14.15	[12.22, 16.59]	$h = 5$, $m = 3$, $\epsilon = 0.005$
	RBF	13.53	—	$h = 3$
Ionosphere	LDNN	**3.40**	[2.56, 4.27]	1×36, $\epsilon = 0.05$
UCI	MLP	8.66	[6.83, 13.67]	$h = 40$, $\epsilon = 0.025$
Train:	RF	5.38	[5.12, 5.98]	$t = 200$, $f = 5$, $s = 1/5$

Table 2.2 – continued

Dataset	Classifier	Av. test error	Test error range	Model Parameters
150+ / 84-	SVM	4.27	—	$C = 2, \gamma = 2$
Test:	Mod-N	5.98	[4.27, 8.54]	$1 \times 36, \epsilon = 0.05$
75+ / 42-	MLP-m	8.73	[6.83, 11.11]	$h = 36, \epsilon = 0.05$
Dim = 33	Maxout	6.37	[2.56, 17.94]	$h = 1, m = 36, \epsilon = 0.05$
	RBF	4.27	—	$h = 21$
German credit	LDNN	**22.58**	[21.02, 23,42]	$6 \times 1, \epsilon = 0.05$
UCI	MLP	23.96	[22.52, 26.12]	$h = 20, \epsilon = 0.01$
Train:	RF	24.28	[23.42, 24.92]	$t = 250, f = 4, s = 2/3$
200+ / 467-	SVM	25.83	—	$C = 8, \gamma = 0.125$
Test:	Mod-N	30.03	[30.03, 30.03]	$6 \times 1, \epsilon = 0.05$
100+ / 233-	MLP-m	24.51	[23.12, 26.12]	$h = 6, \epsilon = 0.05$
Dim = 24	Maxout	25.09	[22.22, 28.22]	$h = 1, m = 2, \epsilon = 0.05$
	RBF	23.72	—	$h = 26$
Forest cover type	LDNN	8.87	[8.09, 9.96]	$20 \times 10, \epsilon = 0.1$
UCI	MLP	9.00	[7.73, 13.26]	$h = 200, \epsilon = 0.1$
Train:	RF	**3.90**	[3.84, 3.94]	$t = 150, f = 15, s = 2/3$
188,868+ / 198,474-	SVM	6.91	—	$C = 32, \gamma = 8$
Test:	Mod-N	25.68	[24.40, 26.77]	$20 \times 10, \epsilon = 0.1$
94,443+ / 99,237-	Maxout	7.07	[6.63, 8.13]	$h = 20, m = 15, \epsilon = 0.01$
Dim = 54	RBF	24.53	—	$h = 100$
IJCNN challenge	LDNN	**1.28**	[1.02, 1.58]	$10 \times 8, \epsilon = 0.25$
Libsvm	MLP	1.80	[1.41, 2.27]	$h = 80, \epsilon = 0.1$
Train:	RF	2.00	[1.91, 2.09]	$t = 250, f = 3, s = 2/3$
3,415+ 31,585-	SVM	1.41	—	$C = 32, \gamma = 8$
Test:	Mod-N	5.01	[4.13, 7.95]	$10 \times 8, \epsilon = 0.25$
8,712+ / 82,889-	Maxout	1.39	[1.03, 3.28]	$h = 10, m = 10, \epsilon = 0.1$
Dim = 22	RBF	7.81	—	$h = 90$
COD-RNA	LDNN	**3.36**	[3.30, 3.46]	$8 \times 8, \epsilon = 0.05$
Libsvm	MLP	3.50	[3.37, 3.73]	$h = 64, \epsilon = 0.05$
Train:	RF	3.37	[3.34, 3.39]	$t = 200, f = 3, s = 2/3$
19,845+ / 39,690-	SVM	3.67	—	$C = 512, \gamma = 8$
Test:	Mod-N	4.15	[2.82, 4.72]	$8 \times 8, \epsilon = 0.05$
90,539+ / 181,07-	Maxout	3.47	[3.35, 3.67]	$h = 5, m = 10, \epsilon = 0.05$
Dim = 8	RBF	4.10	—	$h = 37$
Webspam	LDNN	1.21	[1.12, 1.27]	$15 \times 15, \epsilon = 0.1$
Libsvm	MLP	1.25	[1.13, 1.41]	$h = 225, \epsilon = 0.1$
Train:	RF	1.17	[1.13, 1.19]	$t = 100, f = 11, s = 2/3$
141,460+ / 91,874-	SVM	**0.78**	—	$C = 8, \gamma = 8$
Test:	Mod-N	4.57	[3.89, 5.17]	$15 \times 15, \epsilon = 0.1$
70,729+ / 45,937-	Maxout	0.97	[0.9, 1.08]	$h = 12, m = 17, \epsilon = 0.02$
Dim = 138	RBF	7.73	—	$h = 150$

In 5 out of 10 cases, the mean LDNN error was lower than the minimum RF error. The RF mean error was lower than the LDNN minimum error in only 2 out of 10 cases. LDNNs never severely overfit the data, whereas RFs has significant accuracy differences between training and testing sets for several datasets including *Adult*, *PIMA Indian diabetes*, *German credit* and *Forest cover type*. LDNNs outperformed RBFs in all the cases. This can be explained by the way RBF and LDNN cover the decision space. RBFs use simple radial functions to cover desired parts of space. This type of coverage can be inefficient because the basis functions are not flexible and cover only local parts of space. In many cases, a nonlocal coverage of space is needed which leads to picking too many radial functions in order to cover the space adequately [14]. On the other hand, LDNN combines a set of convex polytopes that are flexible in shape and coverage and can also be local or nonlocal depending on the type of coverage which is needed.

2.4.3 Multiclass Problems

We also experimented with 6 multiclass datasets from the *UCI Machine Learning Repository* [75]. Each dataset was first normalized in the same way as described in Section 2.4.2.1. For each dataset, we trained the LDNN, RF and SVM classifiers with the exception of the MNIST dataset for which the SVM results are reported from [40]. In that paper, a SVM is trained on a feature set generated by a deep belief network. We used one-vs-all to generalize LDNN to multiclass problems. The model and classifier training parameters were chosen as described in Section 2.4.2.2 and are reported in Table 2.3. LDNN and RF experiments were repeated 20 times to obtain mean, minimum and maximum testing errors which are reported in Table 2.3. The LDNN classifier is also related to the idea of space partitioning [60] which combines partitioning of the space and learning a local classifier for each partition into a global objective function for supervised learning. All space partitioning classifier results are reported from [60]. LDNNs had the best accuracy in 4 out of 6 datasets. Note that the minimum and maximum testing errors for LDNNs were equal for MNIST.

Table 2.3: **Column 1**: Multiclass datasets, their source, number of training/testing examples and data dimensionality. **Column 2**: Classifier type. **Column 3-4**: Average testing, and [min,max] testing error (%). Best average testing errors are shown in bold. **Column 5**: Model and classifier training parameters used. LDNN: $N \times M$ model and ϵ step size. RF: t number of trees, f number of features considered per node and s training instance sampling rate for each tree. SVM: C penalty factor, γ: RBF kernel width. The space partitioning (SP) results are from [60].

Dataset	Classifier	Av. test err	Test err range	Model Parameters
Isolet	LDNN	4.17	[3.65, 4.49]	4×4, $\epsilon = 0.01$
Train: 6,238	RF	5.61	[5.25, 5.90]	$t = 200$, $f = 30$, $s = 2/3$
Test: 1,559	SVM	**3.21**	—	$C = 8$, $\gamma = 0.03125$
Classes=26	SP-LDA	5.58	—	Results taken from [60]
Dim=617				
Landsat	LDNN	**7.98**	[7.65, 8.25]	9×9, $\epsilon = 0.1$
Train: 4,435	RF	9.15	[8.65, 9.55]	$t = 200$, $f = 6$, $s = 2/3$
Test: 2,000	SVM	8.15	—	$C = 2$, $\gamma = 8$
Classes=6	SP-LDA	13.95	—	Results taken from [60]
Dim=36				
Letter	LDNN	**2.32**	[2.12, 2.72]	20×20, $\epsilon = 0.4$
Train: 16,000	RF	3.89	[3.65, 4.02]	$t = 500$, $f = 3$, $s = 2/3$
Test: 4,000	SVM	2.35	—	$C = 8$, $\gamma = 8$
Classes=26	SP-LR	13.08	—	Results taken from [60]
Dim=16				
Optdigit	LDNN	2.29	[2.00, 2.67]	5×5, $\epsilon = 0.1$
Train: 3,823	RF	2.89	[2.50, 3.11]	$t = 200$, $f = 7$, $s = 2/3$
Test: 1,797	SVM	**1.56**	—	$C = 8$, $\gamma = 0.125$
Classes=10	SP-P	4.23	—	Results taken from [60]
Dim=62				
Pendigit	LDNN	**1.80**	[1.68, 1.94]	8×8, $\epsilon = 0.005$
Train: 7,494	RF	3.64	[3.40, 3.83]	$t = 250$, $f = 4$, $s = 2/3$
Test: 3,498	SVM	1.86	—	$C = 8$, $\gamma = 2$
Classes=10	SP-P	4.32	—	Results taken from [60]
Dim=16				
MNIST	LDNN	**1.23**	[1.23, 1.23]	30×30, $\epsilon = 0.45$
Train: 60,000	RF	3.00	[2.88, 3.14]	$t = 500$, $f = 26$, $s = 2/3$
Test: 10,000	SVM	1.40	—	Results taken from [40]
Classes=10				
Dim=717				

2.5 Conclusion

We believe that the LDNN network architecture and training can become a favorable alternative to MLPs with one hidden layer. The LDNN classifier has several advantages over MLPs: First, LDNNs allow for a simple and intuitive initialization of the network weights before supervised learning. It is guaranteed that this initialization scheme puts the initial point close to a desired local minima. This makes the optimization process more predictable and leads to more stable and reliable performance. Similarly, LDNN structure along with proposed initialization helps to avoid the herd-effect problem. Second, due to the single adaptive layer, learning can use larger step-sizes in gradient descent. We demonstrated empirically that LDNNs are more accurate than MLPs. Similar to MLPs, the LDNN classifier also requires the choice of model complexity. The number of conjunctions (number of positive training clusters) and the number of logistic sigmoid functions per conjunction (number of negative training clusters) need to be chosen. However, the complexity of the model could be chosen automatically by either using a validation set, as commonly done for SVM training, or by initializing the LDNN in different ways. For instance, sequential covering algorithms can be used to generate a set of rules [79]. Each rule is a conjunction and the final classification is a disjunction of these conjunctions which can easily be converted to a LDNN classifier and fine tuned using gradient descent. LDNNs outperform RBFs significantly which is a proof that LDNNs provide more efficient coverage of the decision space.

While LDNNs are similar in architecture to modular neural networks [47], they are significantly more accurate owing to the unified training of the network that we introduced. LDNNs outperformed RFs in 13 of the 16 datasets and outperformed SVMs in 12 of the 16 datasets. Based on these results and observations, we believe that LDNNs should be considered as a state-of-the art classifier that provides a viable alternative to RFs and SVMs. Further improvements in accuracy can be possible by using cross-entropy instead of the square error criterion or by using adaptive step sizes for training LDNNs. Finally, another possibility is to use more powerful nonlinear discriminants such as conic sections in (2.8).

CHAPTER 3

LOGISTIC PRODUCT BASIS NETWORKS

We introduce a novel general regression model that is based on a linear combination of a new set of nonlocal basis functions that forms an effective feature space. We propose a training algorithm that learns all the model parameters simultaneously and offer an initialization scheme for parameters of the basis functions. We show through several experiments that the proposed method offers better coverage for high-dimensional space compared to local Gaussian basis functions and provides competitive performance in comparison to other state-of-the-art regression methods.

3.1 Introduction

Linear combination of a set of basis functions is one of the most commonly used methods for regression. By choosing a proper set of basis functions, we can project the data into a feature space that provides effective representation of the data distribution [80]. Common basis functions include but are not limited to polynomials, natural splines, radial basis functions (RBF) and wavelet bases [33]. The choice of the basis functions determines the type of regression method. RBFs are important mathematical models and use radial functions such as Gaussians as basis functions [11]. Each radially symmetric Gaussian is tuned to respond to a local region of feature space. Therefore, RBFs require a sufficiently large number of basis functions to cover the space.

In general, some regression methods originate from popular classification methods such as Random Forests [5], Support Vector Machines and Artificial Neural Networks. For example, random forest regression uses ensembles of regression trees instead of classification trees to make the final prediction [81]. In this chapter, however, our attention is on a broad class of methods that work in *weight-space* according to [82]. The most commonly used example is linear regression. There have been many works trying to improve this linear model. For example, *Robust Regression* [83] was proposed to reduce the effect of

outliers in training data. In this chapter, we introduce a new structure which is based on a new set of nonlocal basis functions and provides efficient coverage of high-dimensional spaces. We refer to the proposed structure as Logistic Product Basis Network (LPBN). We compare LPBNs with state-of-the-art regression methods such as Random Forests, Support Vector Regression [84] and RBFs.

3.2 Related Work

Consider the problem of finding a function $f : \mathbf{R}^n \rightarrow \mathbf{R}$ for fitting a set of training data

$$\Gamma = \{\mathbf{x}_i \in \mathbf{R}^n, y_i \in \mathbf{R}\}_{i=1}^{p} \tag{3.1}$$

such that $f(\mathbf{x}_i) = y_i$. Let's specify f as a linear combination of N basis functions

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \phi_i(\mathbf{x}) \tag{3.2}$$

with $\phi_i(\mathbf{x})$ the basis functions that map the data points to the feature space. This mapping is performed in order to obtain a better representation of the data and overcome the limited expressiveness of a linear model. The model is reduced to simple linear regression by setting $\phi_i(\mathbf{x}) = \mathbf{x}$. α_i are the weights of the linear combination. There is a broad range of regression models that are based on this form. Support Vector Regression (SVR) can be considered as an example of linear methods. In SVR, α_i are determined by minimizing the ℓ_2-norm of the weight vector α. The condition of this minimization is that the maximum deviation of the predicted value $f(\mathbf{x}_i)$ from y_i shouldn't exceed ϵ which is called the *margin of tolerance* [85]. The basis functions or mappings are implicitly determined by the type of kernel being used.

A more recent approach is the *Extreme Learning Machine (ELM)* [86] for regression. They map the training data to ELM feature space. The feature mapping layer in ELM need not be tuned. It is also possible to use kernels if the mapping function is unknown.

RBF regression is another example of the methods working in weight-space. The set of basis functions $\phi_i(\mathbf{x})$ are Gaussians and their parameters can be obtained by unsupervised clustering of the data or fitting a Gaussian mixture model to our data using the EM algorithm [55]. Then, we can obtain the linear weights α_i using linear regression. It is also possible to learn the RBF parameters in an unified manner by back-propagation of the error using chain rule [56]. RBFs are popular for modeling, regression and interpolation.

Therefore, there are many works on improvement of the training and performance of these models. One approach is to use regularization in order to improve the generalization ability of the model and avoid overfitting (e.g., [87]). However, most regularization schemes are general techniques and can be applied to many other methods including our proposed structure. It is well known that an RBF network suffers from the curse of dimensionality [88]. In other words, as the number of dimensions grow, the number of radial basis functions required grows exponentially.

Another work similar to our approach is Sum-Product Networks (SPN) [73]. SPNs are probabilistic models that provide tractable inferences. SPN is based on the notion of *network polynomial* and represents unnormalized probability distributions. This leads to a deep structure with interleaved layers of sums and products. SPN is similar to our proposed structure because it uses sum units to mix different submodels with mixing weights corresponding to α_i. It also uses products that combine features of submodels. However, our approach is different. Our proposed structure is based on a flexible set of basis functions. Unlike SPNs, we use logistic sigmoid functions before the product layer to approximate half-spaces. Then, we use products to form basis functions that are convex polytopes. Sigmoid functions are not present in SPNs but are crucial components of our approach. These basis functions are able to provide both local and nonlocal coverage of high-dimensional space which leads to more efficient representation of data space.

3.3 Methods

3.3.1 Network Architecture

In our proposed structure, we build the basis functions $\phi_i(\mathbf{x})$ by the intersection $\cap_{j=1}^{M_i} \mathcal{H}_{ij}$ of M_i half-spaces. $\mathcal{H}_{ij}$ is defined in terms of its indicator function

$$h_{ij}(\mathbf{x}) = \begin{cases} 1, & \sum_{k=1}^{n} w_{ijk} x_k + b_{ij} \geq 0 \\ 0, & otherwise \end{cases} \tag{3.3}$$

where w_{ijk} and b_{ij} are the weights and the bias term. Intersection of the half-spaces forms a convex polytope that covers desired parts of a multidimensional space. However, unlike RBFs, this is not necessarily a local coverage and we can cover the space more efficiently using flexible basis functions.

The indicator functions are binary variables. Therefore, it is possible to rewrite the intersection of half-spaces as the conjunction of the indicator functions $\bigwedge_{j=1}^{M_i} h_{ij}(\mathbf{x})$. Our

next step is to find a differentiable approximation for this general form. We replace the conjunctions by the product $\prod_{j=1}^{M_i} h_{ij}(\mathbf{x})$. This product can be considered as a soft AND gate that implements the conjunction. $h_{ij}(\mathbf{x})$ can be approximated with logistic sigmoid functions

$$\sigma_{ij}(\mathbf{x}) = \frac{1}{1 + e^{-\sum_{k=1}^{n} w_{ijk} x_k + b_{ij}}}, \tag{3.4}$$

We can replace M_i with $M = \max_i M_i$. Based on our experiments, this replacement does not hurt the performance by overfitting. Hence, our basis functions will be in the form $\phi_i(\mathbf{x}) = \prod_{j=1}^{M} \sigma_{ij}(\mathbf{x})$ and we can construct the final fitting function as

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \left(\prod_{j=1}^{M} \sigma_{ij}(\mathbf{x}) \right) \tag{3.5}$$

This function can be visualized as a network shown in Figure 3.1. We refer to the proposed network architecture as LPBN. In this structure, N is the number of the basis functions which is the same as the number of soft AND gates and M is the number of the logistic sigmoid functions per basis function. This structure is referred to as a $N \times M$ LPBN.

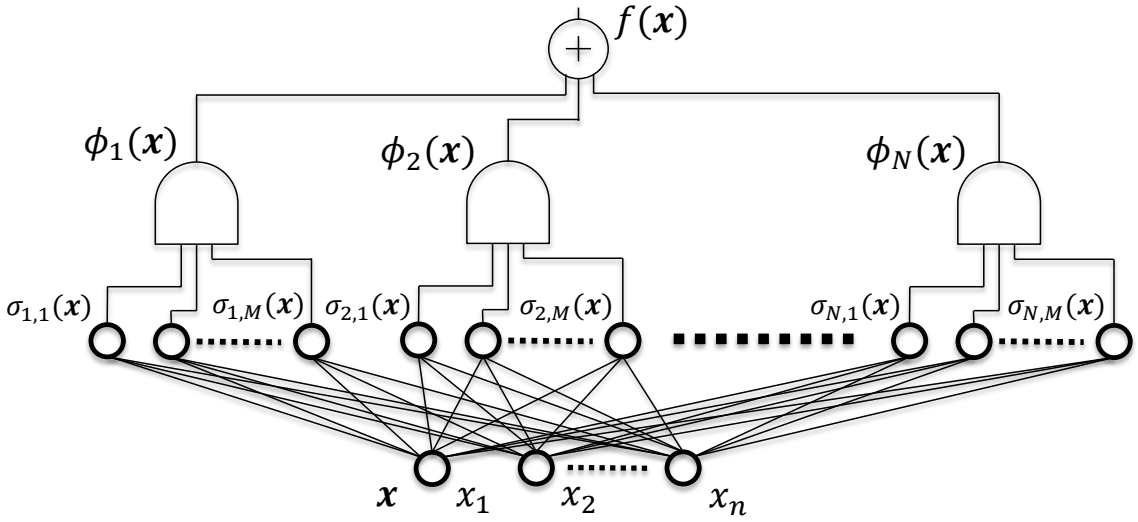


Figure 3.1: LPBN architecture. The first hidden layer is composed of $M \times N$ logistic sigmoid functions. The second hidden layer computes N conjunctions using soft AND gates. The output layer is a linear combination of the soft gates. The soft AND gates are implemented as continuous functions by product of their inputs.

3.3.2 Model Initialization

The specific choice of the basis functions allows for a simple and intuitive initialization of the weights and biases of the logistic sigmoid functions. We can partition training data points $\{\mathbf{x}_i \in \mathbf{R}^n\}_{i=1}^p$ into N clusters. For the i'th cluster, we let $\mathbf{v}_{ij} = \mathbf{c}_i - \mathbf{c}_j$ where $\mathbf{c}$ are the cluster centroids and $j \neq i$. We initialize the weight vectors as $\mathbf{w}_{ij} = \mathbf{v}_{ij}/|\mathbf{v}_{ij}|$ for $i \neq j$. Finally, we initialize the bias terms b_{ij} such that the logistic sigmoid functions $\sigma_{ij}(\mathbf{x})$ take the value 0.5 at the midpoints of the lines connecting $\mathbf{c}_i$ and $\mathbf{c}_j$. In other words, let

$$b_{ij} = \langle \mathbf{w}_{ij}, 0.5(\mathbf{c}_i + \mathbf{c}_j) \rangle \tag{3.6}$$

where $\langle \mathbf{a}, \mathbf{b} \rangle$ denotes the inner product of the vectors $\mathbf{a}$ and $\mathbf{b}$. This procedure initilizes $\phi_i(\mathbf{x})$, the i'th basis function, to a convex polytope that separates the training instances in the i'th cluster from all other training data. This creates a $N \times N - 1$ LPBN.

3.3.3 Model Optimization

The LPBN model can be trained by choosing the parameters $\mathcal{W} = \{\alpha_i, w_{ijk}, b_{ij}\}$ that minimize the quadratic error

$$E(\mathcal{W}, \Gamma) = (y(\mathbf{x}) - f(\mathbf{x}))^2, \quad \mathbf{x} \in \Gamma \tag{3.7}$$

Starting from initialization described in Section 3.3.2 for logistic functions and random initialization for α_i, we minimize (3.7) using gradient descent. In order to get the update equations, we need to find the partial derivatives of the error with respect to all the network weights and biases. It is done via chain rule

$$\begin{aligned} \frac{\partial E}{\partial w_{ijk}} &= \frac{\partial E}{\partial f} \frac{\partial f}{\partial \phi_i} \frac{\partial \phi_i}{\partial \sigma_{ij}} \frac{\partial \sigma_{ij}}{\partial w_{ijk}} \\ &= -2(y - f(\mathbf{x}))\alpha_i \Big(\prod_{l \neq j} \sigma_{il}(\mathbf{x})\Big)\sigma_{ij}(\mathbf{x})(1 - \sigma_{ij}(\mathbf{x}))x_k \\ &= 2(f(\mathbf{x}) - y)\alpha_i \phi_i(\mathbf{x}) \left(1 - \sigma_{ij}(\mathbf{x})\right) x_k \end{aligned} \tag{3.8}$$

Similarly, we obtain the derivative of the error function with respect to the network biases as

$$\frac{\partial E}{\partial b_{ij}} = 2(f(\mathbf{x}) - y)\alpha_i \phi_i(\mathbf{x}) \left(1 - \sigma_{ij}(\mathbf{x})\right) \tag{3.9}$$

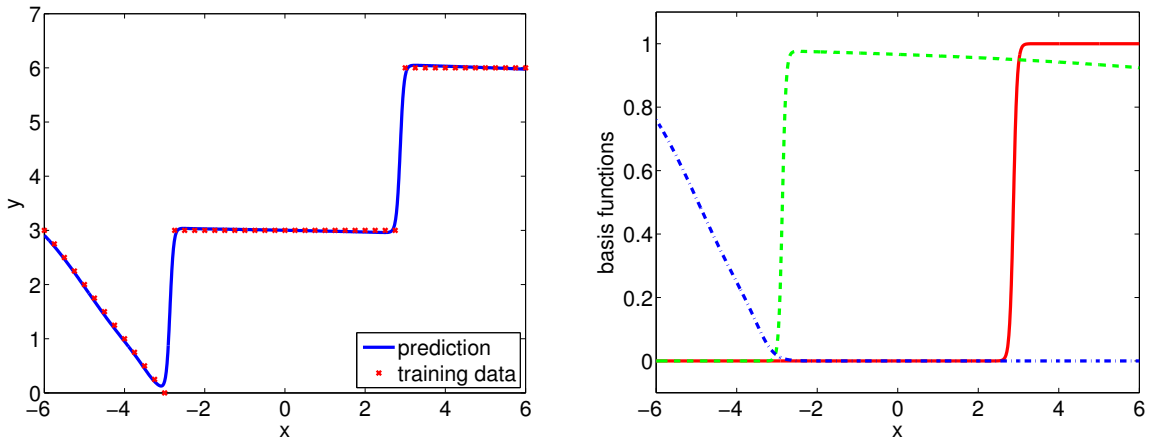
Partial derivatives with respect to α_i are

$$\frac{\partial E}{\partial \alpha_i} = \phi_i(\mathbf{x}) \tag{3.10}$$

3.4 Experiments

First, we compare RBF and LPBN using a set of synthetic 1-dimensional data points shown in Figures 3.2a and 3.3a. We trained both networks with 3 basis functions. Network predictions are shown in the same figures using solid lines. We can see that RBF failed to provide a good prediction for this data. On the other hand, LPBN fits the data efficiently. The basis functions learned by the networks are shown in Figures 3.2b and 3.3b. We observe that every basis function of the LPBN corresponds to a specific part of our data and they are not necessarily local. We repeated the experiment for RBF using 10 basis functions. The network predictions and basis functions are shown in Figures 3.3c and 3.3d, respectively. We can see that RBF still doesn't fit the data very well.

Next, we experimented with 8 regression datasets from the *UCI Machine Learning Repository* [75] and the LIBSVM Tools webpage [76]. We tried to include datasets with different sizes and dimensions. However, datasets with too few or extremely high dimensions were excluded. We also avoided datasets with missing values. The *Energy Efficiency* dataset has 2 different targets. We compared LPBN with Random Forests, ϵ-SVR, Neural Networks with one hidden layer, ELM and RBFs.



(a) (b)

Figure 3.2: LPBN performance on synthetic data using 3 basis functions: (a) training data and LPBN predictions, and (b) basis functions after training.

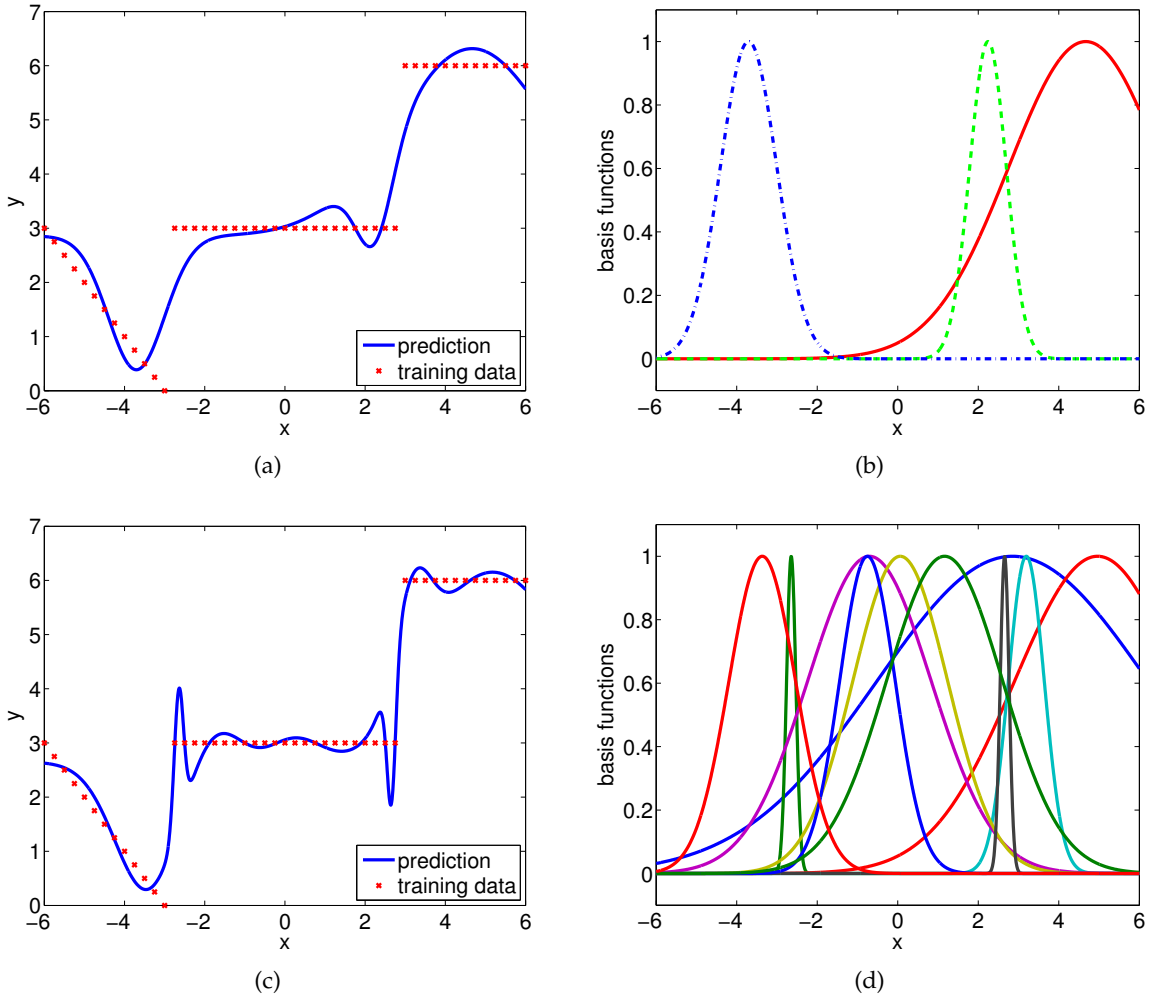


Figure 3.3: RBF performance on synthetic data: training data and RBF predictions with (a) 3 kernels and (c) 10 kernels, and the Gaussian kernels after training with (b) 3 kernels and (d) 10 kernels.

3.4.1 Dataset Normalization, Training/Testing Set Split

Datasets were normalized as follows: For LPBN, Neural Networks, ELM and RBF, all the datasets were normalized by centering each dimension of the feature vector at the origin by subtracting its mean and then scaling by dividing it with its standard deviation. For ϵ-SVR training, each dimension of the feature vector was linearly scaled to the range $[0, 1]$. For all the datasets with the exception of *MSD*, we randomly picked half of the instances for training and the other half for testing. We used (10%) of the training data as a cross-validation set to find the optimal training parameters.

3.4.2 Model and Classifier Training Parameter Selection

For LPBN, we need to choose the number of basis functions (N), which is also the number of clusters for initialization process. For every dataset, we tried different numbers of basis functions in the range of 2 to 20 in order to find the selection that gives the best accuracy on the cross-validation set. Similarly, we found the gradient descent parameters (i.e., step size and number of training epochs) using the cross-validation set. For Neural Network, the number of hidden nodes and parameters of gradient descent were found using the cross-validation set.

We performed our experiments on Random Forest using the code available in [89]. For RF training, the first parameter is the number of trees. We choose a sufficiently large number of trees to ensure that the out of bag error rate converges. The second parameter is the number of features that will be considered in every node of the tree. We tried a range of numbers around the square root of the number of features [5]. The last parameter is the fraction of total samples that will be used in the construction of each tree. We tried $3/4$, $2/3$, $1/2$, $1/3$, $1/4$ and $1/5$ as possible values for this parameter.

For ϵ-SVR training, a RBF kernel was used for all the datasets. The main parameters are γ of RBF kernel, penalty coefficient (C) and ϵ [90]. We performed a grid search to find these parameters using the cross-validation set. For the first 6 datasets, we trained ELMs with Gaussian kernels. We found the kernel width (γ) and regularization parameter (C) using a grid search similar to ϵ-SVR. It was not possible to use kernel-based ELM for the last 2 datasets because of their size. Therefore, we used ELMs with Gaussian basis functions. The main parameters are the number of basis functions (N) and regularization parameter (C). We obtained these parameters by performing a grid search on the cross-validation set.

RBFs were trained using Netlab [78]. Netlab performs a few steps of k-means to initialize unsupervised learning of a Gaussian Mixture Model using an Expectation-Maximization (EM) algorithm. The predicted value is calculated by linearly combining Gaussian kernels. Linear weights are obtained by least squares fitting. For every dataset, we need to find the number of basis functions. Therefore, we tried up to 50 basis functions to pick the best using the cross-validation set. We evaluate the regression methods using the coefficient of determination (R^2 value). The training and model parameters selected for all models are listed in Table 3.1.

Table 3.1: **Column 1**: Regression datasets, their source, number of training/testing examples and data dimensionality. **Column 2**: Method of regression. **Column 3-4**: Average testing over 50 rounds, [min,max] testing R^2. Best average testing results are shown in bold. **Column 5**: Model and classifier training parameters used. Ep: Epochs. LPBN: N number of basis functions and ϵ step size. RF: t number of trees, f number of features considered per node and s training instance sampling rate for each tree. ϵ-SVR: C penalty factor, γ RBF kernel width and ϵ margin of tolerance. NN: N number of hidden nodes and ϵ step size. ELM: C regularization parameter, γ kernel width and N number of basis functions. k: Kernel

Dataset	Method	Av. test R^2	Test R^2 range	Model Parameters
Concrete	LPBN	0.8750	[0.8598, 0.8901]	$N = 7, \epsilon = 0.001$, Ep $= 300$
strength	RBF	0.7663	[0.7531, 0.7804]	No of RBF functions $= 30$
UCI	RF	0.8485	[0.8429, 0.8553]	$t = 100, f = 5, s = 2/3$
Train: 515	ϵ-SVR	0.8456	—	$C = 150, \gamma = 2.85, \epsilon = 0.1$
Test: 515	NN	**0.8840**	[0.8641, 0.9051]	$N = 42, \epsilon = 0.001$, Ep $= 300$
Dim = 8	ELM	0.8629	—	$C = 45, \gamma = 12$, k = RBF
Energy efficiency	LPBN	0.9886	[0.9784, 0.9932]	$N = 6, \epsilon = 0.01$, Ep $= 100$
Heating Load	RBF	0.9898	[0.9867, 0.9922]	No of RBF functions $= 45$
UCI	RF	0.9940	[0.9932, 0.9948]	$t = 100, f = 6, s = 2/3$
Train: 384	ϵ-SVR	0.9892	—	$C = 600, \gamma = 2.25, \epsilon = 0.1$
Test: 384	NN	0.9842	[0.9813, 0.9878]	$N = 30, \epsilon = 0.01$, Ep $= 700$
Dim = 8	ELM	**0.9963**	—	$C = 55000, \gamma = 24$, k = RBF
Energy efficiency	LPBN	**0.9708**	[0.9399, 0.9839]	$N = 12, \epsilon = 0.02$, Ep $= 150$
Cooling Load	RBF	0.9653	[0.9642, 0.9663]	No of RBF functions $= 45$
UCI	RF	0.9605	[0.9584, 0.9629]	$t = 100, f = 6, s = 2/3$
Train: 384	ϵ-SVR	0.9577	—	$C = 390, \gamma = 2.2, \epsilon = 0.2$
Test: 384	NN	0.9673	[0.9403, 0.9818]	$N = 132, \epsilon = 0.02$, Ep $= 150$
Dim = 8	ELM	0.9675	—	$C = 1500, \gamma = 12.5$, k = RBF
Abalone	LPBN	**0.5825**	[0.5734, 0.5904]	$N = 5, \epsilon = 0.001$, Ep $= 100$
Libsvm	RBF	0.5411	[0.5181, 0.5551]	No of RBF functions $= 12$
Train: 2088	RF	0.5542	[0.5511, 0.5575]	$t = 400, f = 6, s = 1/5$
Test: 2088	ϵ-SVR	0.5511	—	$C = 10, \gamma = 2.1, \epsilon = 0.1$
Dim = 8	NN	0.5816	[0.5660, 0.5912]	$N = 20, \epsilon = 0.001$, Ep $= 100$
	ELM	0.5797	—	$C = 20, \gamma = 18$, k = RBF
Cpusmall	LPBN	0.9728	[0.9710, 0.9739]	$N = 15, \epsilon = 0.0005$, Ep $= 150$
Libsvm	RBF	0.9615	[0.9590, 0.9631]	No of RBF functions $= 30$
Train: 4096	RF	**0.9740**	[0.9733, 0.9741]	$t = 200, f = 8, s = 2/3$
Test: 4096	ϵ-SVR	0.9732	—	$C = 250, \gamma = 2.2, \epsilon = 0.1$
Dim = 12	NN	0.9719	[0.9649, 0.9732]	$N = 210, \epsilon = 0005$, Ep $= 150$
	ELM	0.9700	—	$C = 1500, \gamma = 100$, k = RBF
MG	LPBN	0.7195	[0.7085, 0.7289]	$N = 10, \epsilon = 0.02$, Ep $= 500$
Libsvm	RBF	0.7032	[0.6934, 0.7120]	No of RBF functions $= 40$
Train: 692	RF	**0.7270**	[0.7197, 0.7309]	$t = 200, f = 3, s = 2/3$
Test: 692	ϵ-SVR	0.7033	—	$C = 17, \gamma = 3, \epsilon = 0.1$
Dim = 6	NN	0.6315	[-0.0340, 0.7174]	$N = 90, \epsilon = 0.02$, Ep $= 500$

Table 3.1 – continued

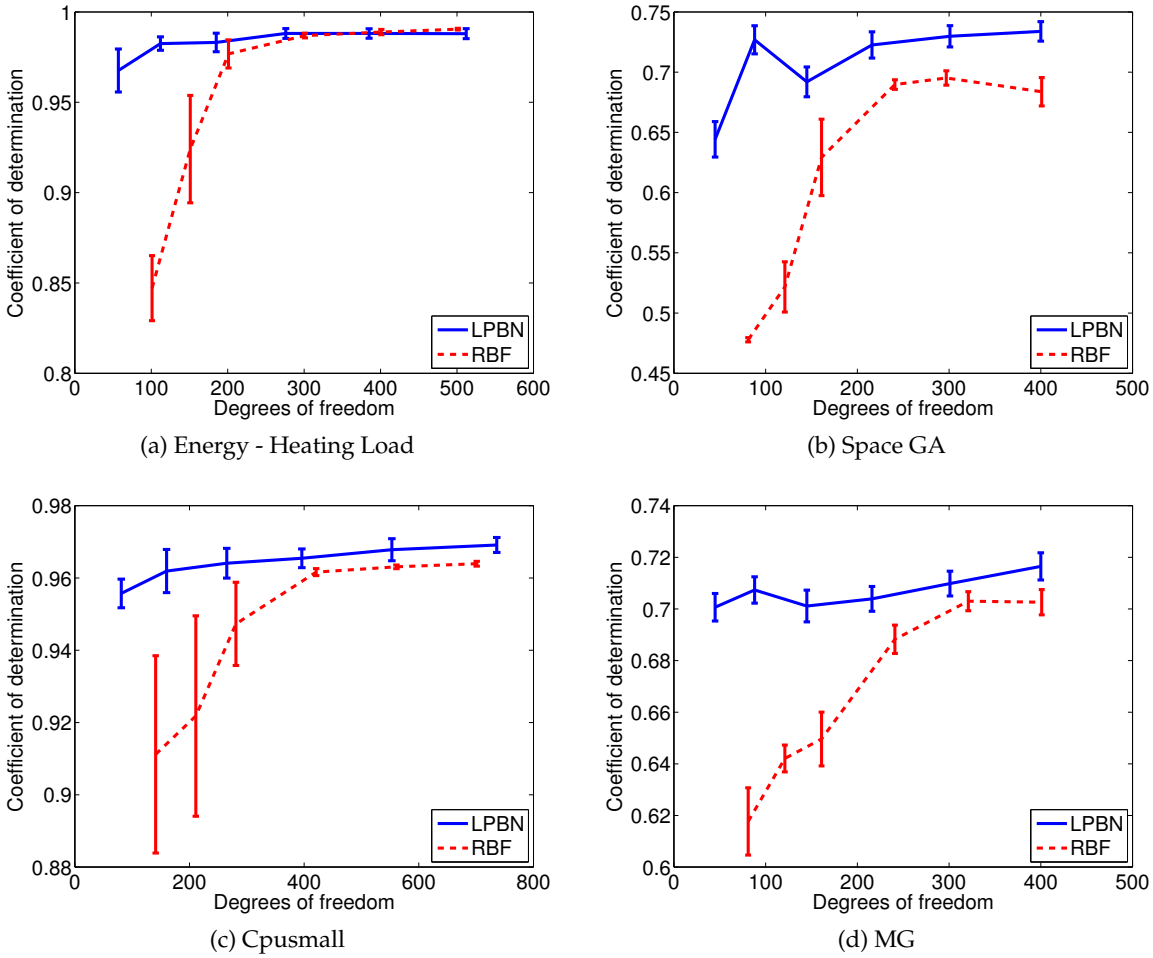
Dataset	Method	Av. test R^2	Test R^2 range	Model Parameters
	ELM	0.7169	—	$C = 15$, $\gamma = 5$, k = RBF
Space GA	LPBN	**0.7341**	[0.7090, 0.7450]	$N = 8$, $\epsilon = 0.1$, Ep = 300
Libsvm	RBF	0.6949	[0.6769, 0.7065]	No of RBF functions = 37
Train: 1553	RF	0.6137	[0.6102, 0.6165]	$t = 200$, $f = 5$, $s = 3/4$
Test: 1553	ϵ-SVR	0.7105	—	$C = 21$, $\gamma = 3.2$, $\epsilon = 0.06$
Dim = 6	NN	0.6470	[0.4607, 0.7150]	$N = 56$, $\epsilon = 0.1$, Ep = 300
	ELM	0.6986	—	$C = 370$, $\gamma = 41$, k = RBF
Million Song	LPBN	**0.3138**	—	$N = 4$, $\epsilon = 10^{-7}$, Ep = 1500
Dataset (MSD)	RBF	0.2554	—	No of RBF functions = 145
UCI	RF	0.2693	—	$t = 100$, $f = 30$, $s = 4/5$
Train: 463715	ϵ-SVR	0.2827	—	$C = 20$, $\gamma = 60$, $\epsilon = 0.1$
Test: 51630				(1/2 of training data)
Dim = 90	NN	0.2888	—	$N = 12$, $\epsilon = 10^{-7}$, Ep = 1500
	ELM	0.2972	—	$C = 10^5$, $N = 5000$
				(basis functions = Gaussian)
Relative location	LPBN	0.9942	—	$N = 17$, $\epsilon = 2 \times 10^{-6}$
of CT slices				Ep = 7000
UCI	RBF	0.8617	—	No of RBF functions = 360
Train: 26750	RF	0.9955	—	$t = 200$, $f = 40$, $s = 4/5$
Test: 26750	ϵ-SVR	0.9953	—	$C = 10$, $\gamma = 0.1$, $\epsilon = 0.1$
Dim = 385	NN	**0.9970**	—	$N = 272$, $\epsilon = 2 \times 10^{-6}$
				Ep = 7000
	ELM	0.9574	—	$C = 10^7$, $N = 15000$
				(basis functions = Gaussian)

3.4.3 Results

All of the regression methods we consider, with the exception of ϵ-SVR and ELM with kernels, are stochastic. Therefore, each experiment on the first 6 datasets was repeated 50 times for stochastic methods in order to obtain mean, minimum and maximum of R^2 on test sets which are reported in Table 3.1 for all the methods. For the last 2 datasets, it was not possible to repeat experiments 50 times mainly because of training times. It was also infeasible to train ϵ-SVR on all training samples of *MSD* dataset. It can be seen that LPBNs performed better than other methods in 4 out of 9 regression problems and performed close to the best on the rest. LPBNs also outperform RBFs in 8 out of 9 problems.

We also compared LPBN with RBF in terms of number of learning parameters. We trained various LPBNs with $N = [3, 8]$ and RBFs with $N = [10, 50]$ for the number of basis functions. For every setting, we repeated the experiments 50 times and computed the average of the R^2 value for the test set for each model. The results are plotted against the degrees of freedom (total number of parameters) in Figure 3.4 for different datasets. The variance of different experiments for every model is shown by error bars. We can see that LPBNs provide better performance with fewer degrees of freedom. This is more obvious when both methods have a small number of learning parameters. This is mainly because LPBNs provide nonlocal basis functions which provide better coverage of the space compared to local basis functions of RBFs. LPBNs also offer better coverage in higher dimensions. Consider the results on the last 2 datasets in Table 3.1. These datasets are relatively large with high dimensions. We can see that LPBNs perform better than RBFs with significantly less basis functions. This is mainly because RBFs require exponentially more basis functions as the number of dimensions grow. On the other hand, the basis functions of LPBNs are nonlocal which can allow for a more efficient representation in higher dimensions.

However, it must be noted that unlike RF, our method is not designed for very high dimensional data. Finally, we also tried using back-propagation to fine-tune the RBF networks further [56], but this resulted in only minor accuracy improvements.



(a) Energy - Heating Load

(b) Space GA

(c) Cpusmall

(d) MG

Figure 3.4: Coefficients of determination vs. degrees of freedom for the LPBN and RBF models: These graphs compare LPBN and RBF in terms of number of learning parameters for 4 datasets. Every experiments is repeated 50 times and the mean of R^2 is shown versus the number of free parameters. Variance of the experiments is shown by error bars.

3.5 Conclusion

We introduced LPBN as a general method of regression and proposed a training algorithm and an effective initialization scheme. The training algorithm learns all the weights simultaneously. We showed that LPBNs provide competitive results compared to popular regression methods. Especially, we showed that LPBNs provide more efficient space coverage compared to RBFs because of a more flexible set of nonlocal basis functions that leads to a feature space that represents the data distribution very well.

CHAPTER 4

MUTUAL EXCLUSIVITY LOSS FOR SEMISUPERVISED DEEP LEARNING

In this chapter, we consider the problem of semisupervised learning with deep Convolutional Neural Networks (ConvNets). Semisupervised learning is motivated on the observation that unlabeled data is cheap and can be used to improve the accuracy of classifiers. We propose an unsupervised regularization term that explicitly forces the classifier's prediction for multiple classes to be mutually-exclusive and effectively guides the decision boundary to lie on the low density space between the manifolds corresponding to different classes of data. Our proposed approach is general and can be used with any backpropagation-based learning method. We show through different experiments that our method can improve the object recognition performance of ConvNets using unlabeled data.

4.1 Introduction

Training high accuracy classifiers often requires a very large amount of labeled training data. Recently, ConvNets [25]-[26] have shown impressive results on many vision tasks including but not limited to classification, detection, localization and scene labeling [28]. However, ConvNets work best when a large amount of labeled data is available for supervised training. For example, the state-of-the-art results for large the 1000-category 'ImageNet' [91] dataset was significantly improved using ConvNets [6],[92]. Unfortunately, building large labeled datasets is a costly and time consuming process. For example, the ImageNet dataset is the result of significant manual effort. On the other hand, unlabeled data is easy to obtain. For example, there are plenty of online resources for images and video sequences of different types. There has been an increased interest in exploiting the readily available unlabeled data to improve the performance of classifiers.

Several works have tried to use unlabeled data for training ConvNets. Convolutional

deep belief networks [93] is a generative model for natural images which is based on deep belief networks [40] and trained using unlabeled data. Unlabeled data has also been used for pretraining of convolutional layers in a ConvNet [94]-[95] in an effort to reduce the amount of labeled data required during supervised training. One example is Predictive Sparse Decomposition (PSD) [96] for learning the filter coefficients in the filter bank layer. However, many recent supervised models trained on large datasets usually start from a random initialization of the filter weights which shows that these solutions are not computationally justified. Ladder networks [97] and region embedding [98] are 2 more recent examples of semisupervised learning in ConvNets.

There are different approaches to semisupervised learning in general [2],[15]. The classical approaches include self-training, cotraining and in general multiview learning [99]-[100]. In these methods, the strong predictions of a single classifier or multiple classifiers will be added to the training set of the same classifier or other classifiers. Another class of methods for semisupervised learning is called generative models. There are different methods in this category which are based on Gaussian Mixture Models (GMM) and Hidden Markov Models (HMM) [101]. These generative models generally include the unlabeled data in modeling the probability distribution of the training data and labels. Another approach to semisupervised learning is Transductive SVM (TSVM) [102] or S3VM [103]. The goal of these methods is to maximize the classification margin by using the unlabeled data. A large body of semisupervised approaches are graph-based methods. These methods are generally based on the similarities between labeled and unlabeled samples [104]-[105]. These similarities are encoded in the edges of a graph.

In this chapter, we propose a semisupervised learning method that makes use of unlabeled data and pushes the decision boundary of Convolutional Neural Networks (CNN) to the less dense areas of decision space and provides better generalization on the test data.

4.1.1 Motivation

In many visual classification tasks, it is easy for a human to classify the training samples perfectly; however, the decision boundary is highly nonlinear in the space of pixel intensities. Therefore, we can argue that the data corresponding to every class lies on a highly nonlinear manifold in the high-dimensional space of pixel intensities and these manifolds

don't intersect with each other. An optimal decision boundary lies between the manifolds of different classes where there are no or very few samples. Decision boundaries can be pushed away from training samples by maximizing their margin. Furthermore, it is not necessary to know the class labels of the samples to maximize the margin of a classifier as in TSVMs. However, finding a classifier with a large margin is only possible if the feature set is chosen or found appropriately. For TSVMs, the burden is on the kernel of choice. On the other hand, since ConvNets are feature generators without their final fully connected classification layer, if there is a feature space that allows a large margin classifier, they should be capable of finding it in theory. Our argument then is that since object recognition is a relatively easy task for a human, there must be such a feature space that ConvNets can generate with a large margin.

Motivated by this argument, we propose a regularization term that uses unlabeled data to encourage the classification layer of a ConvNet to have a large margin. In other words, we propose a regularization term which makes use of unlabeled data and pushes the decision boundary to a less dense area of decision space and forces the set of predictions for a multiclass dataset to be mutually-exclusive.

4.2 Unsupervised Regularization Function

Let's assume that $\mathcal{L} = \{(x_i, y_i)\}_{i=1}^{i=L}$ is the set of labeled training data and $\mathcal{U} = \{(x_i)\}_{i=L+1}^{i=N}$ is the set of unlabeled training data. We also assume that y_i belongs to one of the K classes $\{c_1, c_2, \ldots, c_K\}$. Consider $\mathbf{f}(\mathbf{w}, \mathbf{x})$ to be the output vector of a general classifier with learning parameters $\mathbf{w}$ and input vector $\mathbf{x}$. We define $l_{\mathcal{L}}(\mathbf{f}(\mathbf{w}, \mathbf{x}_i), y_i)$ to be the loss function defined for the classifier which is calculated based on labeled data of $\mathcal{L}$. This loss function can be quadratic error, cross-entropy or any other form of loss function. We can assume that in ideal case, the output vector $\mathbf{f}(\mathbf{w}, \mathbf{x})$ is a multidimensional binary indicator function $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{B}^K$ where $\mathbb{B} = \{0, 1\}$ and n is the dimension of input data. If sample $\mathbf{x}_i$ belongs to class c_k, then this binary function is in the following form:

$$f_j(\mathbf{w}, \mathbf{x}_i) = \begin{cases} 1, & j = k \\ 0, & j = 1 \ldots K,\ j \neq k \end{cases}, \tag{4.1}$$

It can be seen that this indicator function cannot take any arbitrary vector of the space $\mathbb{B}^K$. In fact, it belongs to a very specific subset of this multidimensional space which has only one nonzero element. We call this subset $\mathbb{B}_s^K$. We define another binary indicator function

$I(\mathbf{f}(\mathbf{x}, \mathbf{w}))$ which determines if a binary vector $\mathbf{f} \in \mathbb{B}^K$ also belongs to $\mathbb{B}_s^K$ or not. We define this Boolean function $I : \mathbb{B}^K \rightarrow \mathbb{B}$ using disjunction of conjunctions, also known as the disjunctive normal form [12]:

$$I(\mathbf{f}) = \bigvee_{j=1}^{K} \left(\bigwedge_{k=1, k\neq j}^{K} \neg f_k \bigwedge f_j \right) \tag{4.2}$$

The output of this indicator function for a valid prediction $\mathbf{f}(\mathbf{x}, \mathbf{w})$ of an ideal classifier should be one. This is typically achieved indirectly in a supervised learning setting by one-vs.-rest classification which assigns a target value of 1 to the correct class and a target value of 0 to the other classes. However, the labels of the samples are not required if we directly enforce Eq. (4.2). In this chapter, we enforce this condition in form of a regularization term. To this end, we approximate the binary $I(\mathbf{f})$ with a differentiable function that can be optimized with gradient descent. We replace the conjunction of a set of binary variables $\bigwedge_{i=1}^{K} x_i$ by their product $\prod_{i=1}^{K} x_i$. We also approximate *not* operation of a binary variable $\neg x_i$ with $1 - x_i$. Finally, we substitute the disjunction of the binary variables $\bigvee_{i=1}^{K} x_i$ with their sum $\sum_{i=1}^{K} x_i$. Now relaxing f to be the output of classifiers which are not binary but continuous valued between 0 and 1, $I(\mathbf{f})$ becomes a differentiable function between 0 and 1. By applying the above-mentioned approximations, we define the following unsupervised loss function which is calculated using both samples of $\mathcal{L}$ and $\mathcal{U}$:

$$l_{\mathcal{U}}(\mathbf{f}(\mathbf{w}, \mathbf{x}_i)) = -\sum_{j=1}^{K} f_j(\mathbf{w}, \mathbf{x}_i) \prod_{k=1, k\neq j}^{K} (1 - f_k(\mathbf{w}, \mathbf{x}_i)) \tag{4.3}$$

It must be noted that our goal was to maximize $I(\mathbf{f})$. Therefore, we needed to add the minus sign in Eq. (4.3) when we define it as a loss function to be minimized. Total loss functions to be minimized are defined as follows:

$$l_{\text{tot}} = l_{\mathcal{L}} + \lambda l_{\mathcal{U}} \tag{4.4}$$

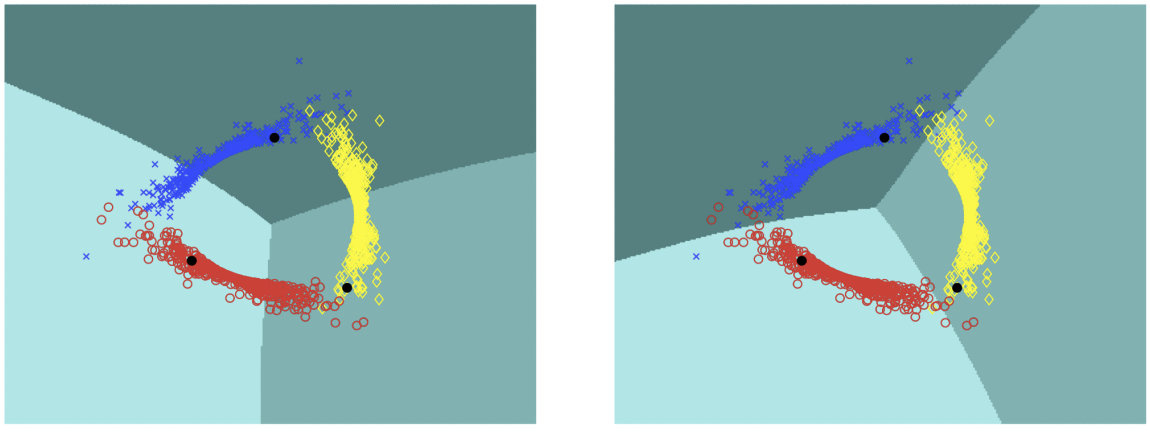
This unsupervised loss function $l_{\mathcal{U}}$ can be combined with any other loss function and can be used with any backpropagation-based learning.

Intuitively, this loss function forces the classifier's prediction to be mutually-exclusive for every class. In addition, it can be observed that this regularization term forces the decision boundary to be as far as possible from any data sample and as a result, it will

be placed in a less dense area of decision space. We show this by an example. Figure 4.1 shows a synthetic dataset with 3 classes of diamonds, circles and crosses. Labeled samples are shown with black circles. We trained a simple 2 layer neural network on this dataset. Decision areas of the neural networks are shown with different colors. Figure 4.1 (a) shows the result of the network trained without unsupervised regularization and Figure 4.1 (b) is the result of the network with proposed unsupervised regularization. We can see that unsupervised regularization places the decision boundary in a less dense area of space. On the other hand, the network trained using only the labeled data without regularization provides poor decision boundaries. A number of studies [106]-[107] show that unlabeled data can be more informative if the classes overlap less. A measure for class overlap is conditional entropy $H(X|Y)$. The empirical conditional entropy can be defined as:

$$H(Y|X) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} P(c_k|\mathbf{x}_i)\log P(c_k|\mathbf{x}_i) \tag{4.5}$$

Note that $P(c_k|\mathbf{x}_i)$ can be estimated with $f_k(\mathbf{w}, \mathbf{x}_i)$. It is shown in [108] that this entropy minimization can be used to form a regularization term based on unlabeled data. Similar to our proposed method, they try to minimize a loss function which is based on labeled samples and also use a regularization term which is based on Eq. (4.5) and calculated using unlabeled data. However, in multiclass problems, our regularization term explicitly forces the classifier's prediction for different classes to be mutually-exclusive. We experimen-



(a) (b)

Figure 4.1: Example showing that unsupervised regularization moves the decision boundary to a less dense area. (a) Without and (b) With unsupervised regularization.

tally show that our proposed regularization term generally performs better than entropy minimization which is based on Eq. (4.5) on some datasets, especially when we have few labeled examples.

4.3 Experiments

In this section, we present the results of applying our regularization term for object recognition using ConvNets. We show extensive results on MNIST [26], CIFAR10 [109], NORB [110] and SVHN [111] datasets. We also show some preliminary results on ILSVRC 2012 [91] using the AlexNet model [6].

In general, we divide the training data of each dataset into 2 sets and consider one set to be the labeled data $\mathcal{L}$ and the other set to be unlabeled data $\mathcal{U}$. In this section, we mainly compare the performance of a ConvNet trained using only labeled data and a ConvNet trained using labeled data and our regularization term calculated using both labeled and unlabeled data. The entropy minimization regularization of [108] is also used for training ConvNets in comparison. For all the datasets with the exception of MNIST and ImageNet, we also trained ConvNets with entropy minimization regularization and compared the results with our proposed regularization scheme. It must be noted that the entropy regularization has not been previously used with ConvNets to the best of our knowledge.

For every dataset, we train the ConvNet using different ratio of labeled and unlabeled data. In separate experiments, we randomly pick 1%, 10%, 50% and 100% of training data as the labeled set and the rest is reserved for the unlabeled set. Then for each setting, we evaluate the improvement obtained by using unlabeled data. We repeat each experiment 5 times for each setting and report the average and standard deviation over error rates of different experiments. The update for model parameters is constituted of 2 parts. The first part is based on labeled data and the second part is from unlabeled data. These 2 are combined with parameter λ according to Eq. (4.4). However, in our experiments, the labeled set is usually smaller than the unlabeled set. Therefore, at each epoch, we use every labeled sample multiple times in order to compensate for the difference in size of labeled and unlabeled datasets. In most of our experiments, λ is fixed to 1. We experimentally observed that the performance of our regularization method is not overly sensitive to λ.

We incorporated our unsupervised regularization term into cuda-convnet which is a GPU implementation of ConvNet and publicly available at [112]. With the exception of MNIST, all other experiments were performed using this GPU implementation. Our setup for all datasets except MNIST and ImageNet consists of 2 convolutional layers followed by 2 locally connected layers. There are 64 maps in each convolutional layer and 32 maps in each locally connected layer. Filters are 5×5 in convolutional layers and 3×3 in locally connected layers (the same as `'layers-conv-local-13pct.cfg'` of [112] for CIFAR10). We added a fully-connected layer of the size 256 before the output. In all the experiments, we found the number of epochs and learning rates using cross-validation on a small portion of training data and repeat the training on all training data.

4.3.1 MNIST

MNIST is the most frequently used dataset in the area of digit classification. It contains 60000 training and 10000 test samples of size 28×28 pixels. We trained MNIST using a ConvNet with 2 convolutional layers. The first layer uses 7×7 filters and produces 20 maps. The second layer also uses 7×7 filters but produces 15 maps. A hidden layer with 256 nodes was added before the final layer. No preprocessing was performed on this dataset. We did not use annealing or momentum. For MNIST, we also trained a model with only 80 labeled samples (8 per class). This is equal to 0.13% of labeled data. The results are given in Table 4.1. We can see that when there is a small number of labeled data available (0.13% and 1% in Table 4.1), the proposed unsupervised regularization term significantly improves the accuracy.

Table 4.1: Performance comparison on test data for MNIST dataset. Error rates: average (%) ± std. dev

	semisupervised	labeled data only
0.13%	14.82 ± 0.89	20.96 ± 2.19
1%	1.77 ± 0.07	4.09 ± 0.22
10%	1.28 ± 0.10	1.54 ± 0.08
50%	0.87 ± 0.06	0.86 ± 0.08
100%	0.79 ± 0.05	0.85 ± 0.06

4.3.2 NORB

NORB is a collection of stereo images in 6 classes. The training set of NORB contains 10 folds of 29160 images. It is common practice to use only the first 2 folds for training. The test set contains 2 folds totalizing 58320. The original images are 108×108. However, we scaled them down to 48×48 similar to [29]. Data translation was used during training. Image translation was obtained by randomly cropping the training images to 44×44. The results are given in Table 4.2. We can see that in the case with 1% of labeled data, our supervised term performs better than entropy regularization. The reason is that in this case, 1% of labeled data is not sufficient to guarantee mutual exclusiveness of the predictions of the entropy regularization method. However, our method explicitly forces mutual exclusiveness.

In another set of experiments, we fixed the labeled set to be 1% of total training samples. Then, we increased the size of unlabeled set in 4 steps. We used 25%, 50%, 75% and 100% of training data as the unlabeled set in separate experiments. The results are given in Table 4.3. It can be seen that by adding more unlabeled data, we can improve the performance of classifier.

4.3.3 SVHN

SVHN is another digit classification task similar to MNIST. SVHN contains around 70000 images for training and more than 500000 easier images for validation. We did not use the validation set at all. The test set contains 26032 images, which are RGB images of size 32 × 32. Generally, SVHN is a more difficult task than MNIST because of the large variations in the images. We did not do any kind of preprocessing for this dataset. We sim-

Table 4.2: Semisupervised performance comparison on test data for NORB dataset. Error rates: Average (%) ± std. dev

	proposed method	labeled data only	entropy minimization
1%	19.02 ± 1.56	26.77 ± 1.60	21.12 ± 0.73
10%	6.55 ± 0.37	8.42 ± 0.62	7.14 ± 0.70
50%	4.58 ± 0.25	4.98 ± 0.35	4.91 ± 0.21
100%	4.38 ± 0.24	4.58 ± 0.31	4.24 ± 0.17

Table 4.3: Performance comparison on test data for NORB dataset with fixed labeled set and variable unlabeled set.

size of unlabeled set	error rates: average (%) $\pm$ std. dev
labeled data only	26.77 ± 1.60
25% of training set	21.81 ± 1.24
50% of training set	21.48 ± 2.16
75% of training set	20.75 ± 1.21
100% of training set	19.02 ± 1.56

ply converted the color images to grayscale by removing hue and saturation information. The results are given in Table 4.4.

Similar to NORB, we performed a set of experiments by fixing the size of the labeled set and changing the size of unlabeled data. We increased the size of the unlabeled set in 4 steps. The results are shown in Table 4.5. Here again, we observe that by increasing the size of unlabeled data, we can actually improve the classification performance.

4.3.4 CIFAR10

CIFAR10 is a collection of 60000 tiny 32×32 images of 10 categories (50000 for training and 10000 for test). We augmented the training data using image translations, which is done by taking 24×24 cropped versions of the original images at random locations. A common preprocessing for this dataset is to subtract the per pixel mean of the training set from every image. The results are given in Table 4.6.

Similar to NORB and SVHN, we fixed the labeled set at 1% of training data and increased the size of the unlabeled set in 4 steps. The results are given in Table 4.7. We can

Table 4.4: Semisupervised performance comparison on test data for SVHN dataset. Error rates: Average (%) $\pm$ std. dev

	proposed method	labeled data only	entropy minimization
1%	15.30 ± 0.74	21.53 ± 0.53	15.23 ± 0.36
10%	7.94 ± 0.15	9.88 ± 0.21	7.93 ± 0.17
50%	5.65 ± 0.11	6.17 ± 0.19	5.55 ± 0.22
100%	4.60 ± 0.02	5.08 ± 0.07	4.68 ± 0.09

Table 4.5: Performance comparison on test data for SVHN dataset with fixed labeled set and variable unlabeled set.

size of unlabeled set	error rates: average (%) $\pm$ std. dev
labeled data only	21.53 ± 0.53
25% of training set	16.15 ± 0.27
50% of training set	15.55 ± 0.14
75% of training set	15.57 ± 0.23
100% of training set	15.30 ± 0.74

Table 4.6: Semisupervised performance comparison on test data for CIFAR10. Error rates: Average (%) $\pm$ std. dev

	proposed method	labeled data only	entropy minimization
1%	54.48 ± 0.42	57.61 ± 0.71	55.06 ± 0.65
10%	27.71 ± 0.32	31.48 ± 0.27	27.54 ± 0.34
50%	16.87 ± 0.28	18.31 ± 0.31	17.02 ± 0.26
100%	13.67 ± 0.17	14.11 ± 0.23	13.64 ± 0.23

Table 4.7: Performance comparison on test data for CIFAR10 dataset with fixed labeled set and variable unlabeled set.

size of unlabeled set	error rates: average (%) $\pm$ std. dev
labeled data only	57.61 ± 0.71
25% of training set	55.34 ± 0.69
50% of training set	55.09 ± 0.32
75% of training set	54.40 ± 0.65
100% of training set	54.63 ± 0.66

see that the performance keeps improving as we add more unlabeled data.

4.3.5 ImageNet

We performed preliminary experiments with ILSVRC 2012 which has 1000 classes. We randomly picked 10% of each class from training data as the labeled set and the rest was used for the unlabeled set. We applied our regularization term to the AlexNet model [6]. Using our method, we achieved an error rate of 42.90%. If we don't use the regularization term, the error rate is 45.63%. This shows that our model can be effective even when we

have large number of classes.

4.4 Discussion

We performed an extensive set of experiments to evaluate the performance of our unsupervised regularization term. In all of the experiments, we observed performance improvement by using unsupervised regularization. It must be noted that even for the case in which the labeled set and unlabeled set are the same (the experiments with 100% of training data used as labeled data), we still observe improvement in performance. This is interesting because in these cases, the computational cost of adding our unsupervised term is negligible. The state-of-the-art accuracy on all the datasets that we used in this chapter was obtained by ConvNet-based methods that can benefit from our unsupervised regularization. Therefore, we expect an improvement in their accuracy by using our regularization term. Our regularization term is also scalable to large number of classes. We successfully applied our method to the ImageNet dataset with 1000 classes. Based on our experiments, we can see that for the cases with very few labeled samples, the advantage of using unsupervised regularization term is more significant when the classification task is simpler. For example, CIFAR10 is a more challenging dataset compared to MNIST, NORB and SVHN and benefits less from using unsupervised term. In simpler tasks, ConvNet is able to create a feature space with less dense areas and provides better discriminative features for the classifier of final layer. This means that even for more challenging tasks, if more unsupervised training data becomes available, then large ConvNets can be trained which might be able to create the feature spaces that will have less dense areas and larger margins. This means that if we provide the classifier with better discriminative features, then more challenging tasks could potentially benefit more from unsupervised regularization term.

4.5 Conclusion

We introduced an unsupervised regularization term that forces classifier predictions to be mutually-exclusive for different classes and moves the decision boundary to a less dense area of decision space. We showed that our method can be applied successfully to ConvNets to improve the classification accuracy using both labeled and unlabeled data.

We showed that it is possible to improve classification accuracy by adding more unlabeled data. We also showed that entropy regularization can be applied to ConvNets successfully.

CHAPTER 5

TRANSFORMATION/STABILITY LOSS FOR SEMISUPERVISED LEARNING

Effective convolutional neural networks are trained on large sets of labeled data. However, creating large labeled datasets is a very costly and time-consuming task. Semisupervised learning uses unlabeled data to train a model with higher accuracy when there is a limited set of labeled data available. In this chapter, we consider the problem of semisupervised learning with convolutional neural networks. Techniques such as randomized data augmentation, dropout and random max-pooling provide better generalization and stability for classifiers that are trained using gradient descent. Multiple passes of an individual sample through the network might lead to different predictions due to the nondeterministic behavior of these techniques. We propose an unsupervised loss function that takes advantage of the stochastic nature of these methods and minimizes the difference between the predictions of multiple passes of a training sample through the network. We evaluate the proposed method on several benchmark datasets. We also show that proposed loss function can be used to improve the robustness of a learning model with respect to adversarial examples.

5.1 Introduction

Convolutional neural networks (ConvNets) [25]-[26] achieve state-of-the-art accuracy on a variety of computer vision tasks, including classification, object localization, detection, recognition and scene labeling [28],[9]. The advantage of ConvNets partially originates from their complexity (large number of parameters), but this can result in overfitting without a large amount of training data. However, creating a large labeled dataset is very costly. A notable example is the ImageNet [91] dataset with 1000 category and more than 1 million training images. The state-of-the-art accuracy of this dataset is improved every year using ConvNet-based methods (e.g., [92],[6]). This dataset is the result of significant

manual effort. However, with around 1000 images per category, it barely contains enough training samples to prevent the ConvNet from overfitting [6]. On the other hand, unlabeled data is cheap to collect. For example, there are numerous online resources for images and video sequences of different types. Therefore, there has been an increased interest in exploiting the readily available unlabeled data to improve the performance of ConvNets.

Randomization plays an important role in the majority of learning systems. Stochastic gradient descent, dropout [19], randomized data transformation and augmentation [29] and many other training techniques that are essential for fast convergence and effective generalization of the learning functions introduce some nondeterministic behavior to the learning system. Due to these uncertainties, passing a single data sample through a learning system multiple times might lead to different predictions. Based on this observation, we introduce an unsupervised loss function optimized by gradient descent that takes advantage of this randomization effect and minimizes the difference in predictions of multiple passes of a data sample through the network during the training phase, which leads to better generalization in testing time. The proposed unsupervised loss function specifically regularizes the network based on the variations caused by randomized data augmentation, dropout and randomized max-pooling schemes. This loss function can be combined with any supervised loss function. In this chapter, we apply the proposed unsupervised loss function to ConvNets as a state-of-the-art supervised classifier. We show through numerous experiments that this combination leads to a competitive semisupervised learning method.

5.2 Related Work

There are many approaches to semisupervised learning in general. Self-training and cotraining [99]-[100] are 2 well-known classic examples. Another set of approaches is based on generative models, for example, methods based on Gaussian Mixture Models (GMM) and Hidden Markov Models (HMM) [101]. These generative models generally try to use unlabeled data in modeling the joint probability distribution of the training data and labels. Transductive SVM (TSVM) [102] and S3VM [103] are another semisupervised learning approach that tries to find a decision boundary with a maximum margin on both labeled and unlabeled data. A large group of semisupervised methods is based on graphs

and the similarities between the samples [104]-[105]. For example, if a labeled sample is similar to an unlabeled sample, its label is assigned to that unlabeled sample. In these methods, the similarities are encoded in the edges of a graph. Label propagation [16] is an example of these methods in which the goal is to minimize the difference between model predictions of 2 samples with large weighted edge. In other words, similar samples tend to get similar predictions.

In this chapter, our focus is on semisupervised deep learning. There has always been interest in exploiting unlabeled data to improve the performance of ConvNets. One approach is to use unlabeled data to pretrain the filters of ConvNet [94]-[95]. The goal is to reduce the number of training epochs required to converge and improve the accuracy compared to a model trained by random initialization. Predictive sparse decomposition (PSD) [96] is one example of these methods used for learning the weights in the filter bank layer. The works presented in [113] and [114] are 2 recent examples of learning features by pretraining ConvNets using unlabeled data. In these approaches, an auxiliary target is defined for a pair of unlabeled images [113] or a pair of patches from a single unlabeled image [114]. Then a pair of ConvNets is trained to learn descriptive features from unlabeled images. These features can be fine-tuned for a specific task with a limited set of labeled data. However, many recent ConvNet models with state-of-the-art accuracy start from randomly initialized weights using techniques such as Xavier's method [115],[92]. Therefore, approaches that make better use of unlabeled data during training instead of just pretraining are more desired.

Another example of semisupervised learning with ConvNets is region embedding [98], which is used for text categorization. The work in [116] is also a deep semisupervised learning method based on embedding techniques. Unlabeled video frames are also being used to train ConvNets [117]-[118]. The target of the ConvNet is calculated based on the correlations between video frames. Another notable example is semisupervised learning with ladder networks [97] in which the sums of supervised and unsupervised loss functions are simultaneously minimized by backpropagation. In this method, a feedforward model is assumed to be an encoder. The proposed network consists of a noisy encoder path and a clean one. A decoder is added to each layer of the noisy path. This decoder is supposed to reconstruct a clean activation of each layer. The unsupervised loss function

is the difference between the output of each layer in clean path and its corresponding reconstruction from the noisy path.

Another approach by [119] is to take a random unlabeled sample and generate multiple instances by randomly transforming that sample multiple times. The resulting set of images forms a surrogate class. Multiple surrogate classes are produced and a ConvNet is trained on them. One disadvantage of this method is that it does not scale well with the number of unlabeled examples because a separate class is needed for every training sample during unsupervised training. In [17], the authors propose a mutual-exclusivity loss function that forces the set of predictions for a multiclass dataset to be mutually-exclusive. In other words, it forces the classifier's prediction to be close to one only for one class and zero for the others. It is shown that this loss function makes use of unlabeled data and pushes the decision boundary to a less dense area of decision space.

Another set of works related to our approach try to restrict the variations of the prediction function. Tangent distance and tangent propagation proposed by [120] enforce local classification invariance with respect to the transformations of input images. Here, we propose a simpler method that additionally minimizes the internal variations of the network caused by dropout and randomized pooling and leads to state-of-the-art results on MNIST (with 100 labeled samples), CIFAR10 and CIFAR100. Another example is Slow Feature Analysis (SFA) (e.g., [121] and [122]) that encourages the representations of temporally close data to exhibit small differences.

5.3 Method

Given any training sample, a model's prediction should be the same under any random transformation of the data and perturbations to the model. The transformations can be any linear and nonlinear data augmentation being used to extend the training data. The disturbances include dropout techniques and randomized pooling schemes. In each pass, each sample can be randomly transformed or the hidden nodes can be randomly activated. As a result, the network's prediction can be different for multiple passes of the same training sample. However, we know that each sample is assigned to only one class. Therefore, the network's prediction is expected to be the same despite transformations and disturbances. We introduce an unsupervised loss function that minimizes the mean

squared differences between different passes of an individual training sample through the network. Note that we do not need to know the label of a training sample in order to enforce this loss. Therefore, the proposed loss function is completely unsupervised and can be used along with supervised training as a semisupervised learning method. Even if we don't have a separate unlabeled set, we can apply the proposed loss function on samples of a labeled set to enforce stability.

Here, we formally define the proposed unsupervised loss function. We start with a dataset with N training samples and C classes. Let us assume that $\mathbf{f}^j(\mathbf{x}_i)$ is the classifier's prediction vector on the i'th training sample during the j'th pass through the network. We assume that each training sample is passed n times through the network. We define the $T^j(\mathbf{x}_i)$ to be a random linear or nonlinear transformation on the training sample $\mathbf{x}_i$ before the j'th pass through the network. The proposed loss function for each data sample is:

$$l_{\mathcal{U}}^{\text{TS}} = \sum_{i=1}^{N} \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} \|\mathbf{f}^j(T^j(\mathbf{x}_i)) - \mathbf{f}^k(T^k(\mathbf{x}_i))\|_2^2 \tag{5.1}$$

where 'TS' stands for transformation/stability. We pass a training sample through the network n times. In each pass, the transformation $T^j(\mathbf{x}_i)$ produces a different input to the network from the original training sample. In addition, each time, the randomness inside the network, which can be caused by dropout or randomized pooling schemes, leads to a different prediction output. We minimize the sum of squared differences between each possible pair of predictions. We can minimize this objective function using gradient descent. This process is illustrated in Figure 5.1. In Figure 5.1a, we transform a sample multiple times and pass it through the same network and minimize the difference in predictions. In Figure 5.1b, we pass the same sample multiple times through the network and minimize the variation in prediction caused by dropout.

Although Eq. 5.1 is quadratically dependent on the number of augmented versions of the data (n), calculation of loss and gradient is only based on the prediction vectors. Therefore, the computing cost is negligible even for large n. Note that recent neural-network-based methods are optimized on batches of training samples instead of a single sample (batch vs. online training). We can design batches to contain replications of training samples so we can easily optimize this transformation/stability loss function. If we use data augmentation, we put different transformed versions of an unlabeled data in the

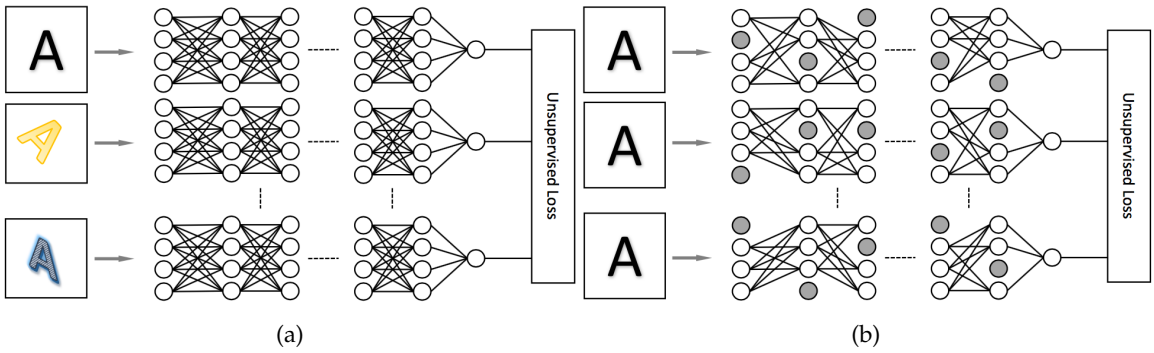


Figure 5.1: Illustration of our loss function. (a) Minimizing the variations caused by data augmentation (b) Minimizing the variations caused by internal network structure.

mini-batch instead of replication. This unsupervised loss function can be used with any backpropagation-based algorithm. Even though every mini-batch contains replications of a training sample, these are used to calculate a single backpropagation signal avoiding gradient bias and not adversely affecting convergence. It is also possible to combine this loss with any supervised loss function. We reserve part of the mini-batch for labeled data which are not replicated.

As mentioned in Section 5.2, mutual-exclusivity loss function of [17] forces the classifier's prediction vector to have only one nonzero element. This loss function naturally complements the transformation/stability loss function. In supervised learning, each element of the prediction vector is pushed towards zero or one depending on the corresponding element in label vector. The proposed loss minimizes the l_2-norm of the difference between predictions of multiple transformed versions of a sample, but it does not impose any restrictions on the individual elements of a single prediction vector. As a result, each prediction vector might be a trivial solution instead of a valid prediction due to lack of labels. Mutual-exclusivity loss function forces each prediction vector to be valid and prevents trivial solutions. This loss function for the training sample $\mathbf{x}_i$ is defined as follows:

$$l_{\mathcal{U}}^{\mathrm{ME}} = \sum_{i=1}^{N} \sum_{j=1}^{n} \left(- \sum_{k=1}^{C} f_k^j(\mathbf{x}_i) \prod_{l=1, l \neq k}^{C} (1 - f_l^j(\mathbf{x}_i)) \right) \tag{5.2}$$

where 'ME' stands for mutual-exclusivity. $f_k^j(\mathbf{x}_i)$ is the k-th element of prediction vector $\mathbf{f}^j(\mathbf{x}_i)$. In the experiments, we show that the combination of both loss functions leads to further improvements in the accuracy of the models. We define the combination of both

loss functions as transformation/stability plus mutual-exclusivity loss function:

$$l_{\mathcal{U}} = \lambda_1 l_{\mathcal{U}}^{\mathrm{ME}} + \lambda_2 l_{\mathcal{U}}^{\mathrm{TS}} \tag{5.3}$$

5.4 Experiments

We show the effect of the proposed unsupervised loss functions using ConvNets on MNIST [26], CIFAR10 and CIFAR100 [109], SVHN [111], NORB [110] and ILSVRC 2012 challenge [91]. We use 2 frameworks to implement and evaluate the proposed loss function. The first one is cuda-convnet [112], which is the original implementation of the well-known AlexNet model. The second framework is the sparse convolutional networks [20] with fractional max-pooling [123], which is a more recent implementation of ConvNets achieving state-of-the-art accuracy on CIFAR10 and CIFAR100 datasets. We show through different experiments that by using the proposed loss function, we can improve the accuracy of the models trained on a few labeled samples on both implementations. In Eq. 5.1, we set n to be 4 for experiments conducted using cuda-convnet and 5 for experiments performed using sparse convolutional networks. Sparse convolutional network allows for any arbitrary batch sizes. As a result, we tried different options for n and $n = 5$ is the optimal choice. However, cuda-convnet allows for mini-batches of size 128. Therefore, it is not possible to use $n = 5$. Instead, we decided to use $n = 4$. In practice, the difference is insignificant. We used MNIST to find the optimal n. We tried different n up to 10 and did not observe improvements for n larger than 5. It must be noted that replicating a training sample 4 or 5 times does not necessarily increase the computational complexity with the same factor. Based on the experiments, with higher n, fewer training epochs are required for the models to converge. We perform multiple experiments for each dataset. We use the available training data of each dataset to create 2 sets: labeled and unlabeled. We do not use the labels of the unlabeled set during training. It must be noted that for the experiments with data augmentation, we apply data augmentation to both labeled and unlabeled set. We compare models that are trained only on the labeled set with models that are trained on both the labeled set and the unlabeled set using the unsupervised loss function. We show that by using the unsupervised loss function, we can improve the accuracy of classifiers on benchmark datasets. For experiments performed using the sparse convolutional network, we describe the network parameters using the format adopted from the original paper

[123]:

$$(10kC2 - FMP\sqrt{2})_5 - C2 - C1$$

In the above example network, $10k$ is the number of maps in the k'th convolutional layer. In this example, $k = 1, 2, ..., 5$. $C2$ specifies that convolutions use a kernel size of 2. $FMP\sqrt{2}$ indicates that convolutional layers are followed by a fractional max-pooling (FMP) layer [123] that reduces the size of feature maps by a factor of $\sqrt{2}$. As mentioned earlier, the mutual-exclusivity loss function of [17] complements the transformation/stability loss function. We implement that loss function in both cuda-convnet and sparse convolutional networks as well. We experimentally choose λ_1 and λ_2 in Eq. 5.3. However, the performance of the models is not overly sensitive to these parameters, and in most of the experiments, it is fixed to $\lambda_1 = 0.1$ and $\lambda_2 = 1$.

5.4.1 MNIST

MNIST is the most frequently used dataset in the area of digit classification. It contains 60000 training and 10000 test samples of size 28×28 pixels. A few examples of the MNIST images are shown in Figure 5.2.

We perform experiments on MNIST using a sparse convolutional network with the following architecture: $(32kC2 - FMP\sqrt{2})_6 - C2 - C1$. We use dropout to regularize the network. The ratio of dropout gradually increases from the first layer to the last layer. We do not use any data augmentation for this task. In other words, $T^j(\mathbf{x}_i)$ of Eq. 5.1 is identity function for this dataset. In this case, we take advantage of the random effects of dropout and fractional max-pooling using the unsupervised loss function. We randomly select 10 samples from each class (total of 100 labeled samples). We use all available training data as the unlabeled set. First, we train a model based on this labeled set only.

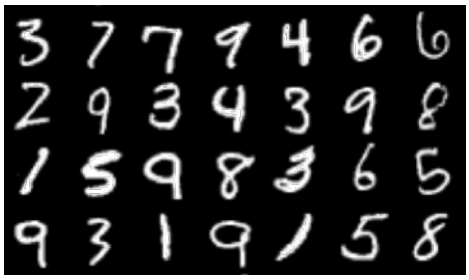


Figure 5.2: Sample images from MNIST dataset.

Then, we train models by adding unsupervised loss functions. In separate experiments, we add transformation/stability loss function, mutual-exclusivity loss function and the combination of both. Each experiment is repeated 5 times with a different random subset of training samples. We repeat the same set of experiments using 100% of MNIST training samples. The results are given in Table 5.1. We can see that the proposed loss significantly improves the accuracy on test data. We also compare the results with ladder networks [97]. Combination of both loss functions reduces the error rate to 0.55% $\pm$ 0.16 which is the state-of-the-art for the task of MNIST with 100 labeled samples to the best of our knowledge. The state-of-the-art error rate on MNIST using all training data without data augmentation is 0.24% [124]. It can be seen that we can achieve a close accuracy by using only 100 labeled samples.

Figure 5.3 shows a few examples of the MNIST test set classified correctly by our semisupervised learning method trained using 100 labeled samples but classified incorrectly using a supervised model trained on the same set of 100 labeled samples.

5.4.2 SVHN and NORB

SVHN is another digit classification task similar to MNIST. This dataset contains about 70000 images for training and more than 500000 easier images [111] for validation. We do not use the validation set. The test set contains 26032 images, which are RGB images of size 32×32. Generally, SVHN is a more difficult task compared to MNIST because of the large variations in the images. We do not perform any preprocessing for this dataset. We simply convert the color images to grayscale by removing hue and saturation information. NORB is a collection of stereo images in 6 classes. The training set contains 10 folds of

Table 5.1: Error rates (%) on test set for MNIST (mean % $\pm$ std).

	100 samples	all training samples
labeled data only	5.44 $\pm$ 1.48	0.32 $\pm$ 0.02
transformation / stability loss [17]	0.76 $\pm$ 0.61	0.29 $\pm$ 0.02
mutual-exclusivity loss	3.92 $\pm$ 1.12	0.30 $\pm$ 0.03
both losses	**0.55 $\pm$ 0.16**	**0.27 $\pm$ 0.02**
ladder networks [97]	0.89 $\pm$ 0.50	-
ladder networks baseline [97]	6.43 $\pm$ 0.84	0.36

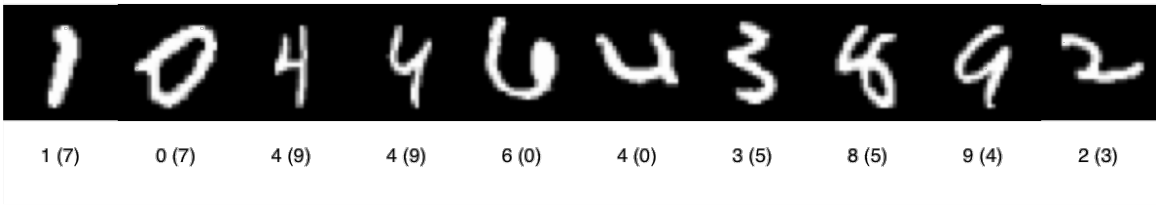


Figure 5.3: Samples of MNIST test set classified correctly by our semisupervised method but classified incorrectly using supervised model. The value in parenthesis shows the wrong prediction of supervised classifier.

29160 images. It is common practice to use only the first 2 folds for training. The test set contains 2 folds, totaling 58320. The original images are 108×108. However, we scale them down to 48×48 similar to [29]. We perform experiments on these 2 datasets using both cuda-convnet and sparse convolutional network implementations of the unsupervised loss function. Examples of SVHN and NORB images are shown in Figure 5.4.

In the first set of experiments, we use cuda-convnet to train models with different ratios of labeled and unlabeled data. We randomly choose 1%, 5%, 10%, 20% and 100% of training samples as labeled data. All of the training samples are used as the unlabeled set. For each labeled set, we train 4 models using cuda-convnet. The first model uses the labeled set only. The second model is trained on the unlabeled set using mutual-exclusivity

(a) (b)

Figure 5.4: Samples of (a) SVHN and (b) NORB datasets.

loss function in addition to the labeled set. The third model is trained on the unlabeled set using the transformation/stability loss function in addition to the labeled set. The last model is also trained on both sets but combines 2 unsupervised loss functions. Each experiment is repeated 5 times. For each repetition, we use a different subset of training samples as labeled data. The cuda-convnet model consists of 2 convolutional layers with 64 maps and kernel size of 5, 2 locally connected layers with 32 maps and kernel size 3. Each convolutional layer is followed by a max-pooling layer. A fully connected layer with 256 nodes is added before the last layer. We use data augmentation for these experiments. $T^j(\mathbf{x}_i)$ of Eq. 5.1 crops every training sample to 28×28 for SVHN and 44×44 for NORB at random locations. $T^j(\mathbf{x}_i)$ also randomly rotates training samples up to $\pm 20^\circ$. These transformations are applied to both labeled and unlabeled sets. The results are shown in Figure 5.5 for SVHN and Figure 5.6 for NORB. Each point in the graph is the mean error rate of 5 repetitions. The error bars show the standard deviation of these 5 repetitions. As expected, we can see that in all experiments, the classification accuracy is improved as we add more labeled data. However, we observe that for each set of labeled data, we can improve the results by using the proposed unsupervised loss functions. We can also see that when the number of labeled samples is small, the improvement is more significant. For example, when we use only 1% of labeled data, we gain an improvement in accuracy of about 2.5 times by using unsupervised loss functions. As we add more labeled samples, the difference in accuracy between semisupervised and supervised approaches becomes smaller. Note that the combination of transformation/stability loss function and mutual-exclusivity loss function improves the accuracy even further. As mentioned earlier, these 2 unsupervised loss functions complement each other. Therefore, in most of the experiments, we use the combination of 2 unsupervised loss functions.

We perform another set of experiments on these 2 datasets using sparse convolutional networks as a state-of-the-art classifier. We create 5 sets of labeled data. For each set, we randomly pick a different 1% subset of training samples as the labeled set and all training data as the unlabeled set. We train 2 models: the first trained only on labeled data, and the second using the labeled set and a combination of both unsupervised losses. Similarly, we train models using all available training data as both the labeled set and unlabeled set. We do not use data augmentation for any of these experiments. In other words, $T^j(\mathbf{x}_i)$ of Eq.

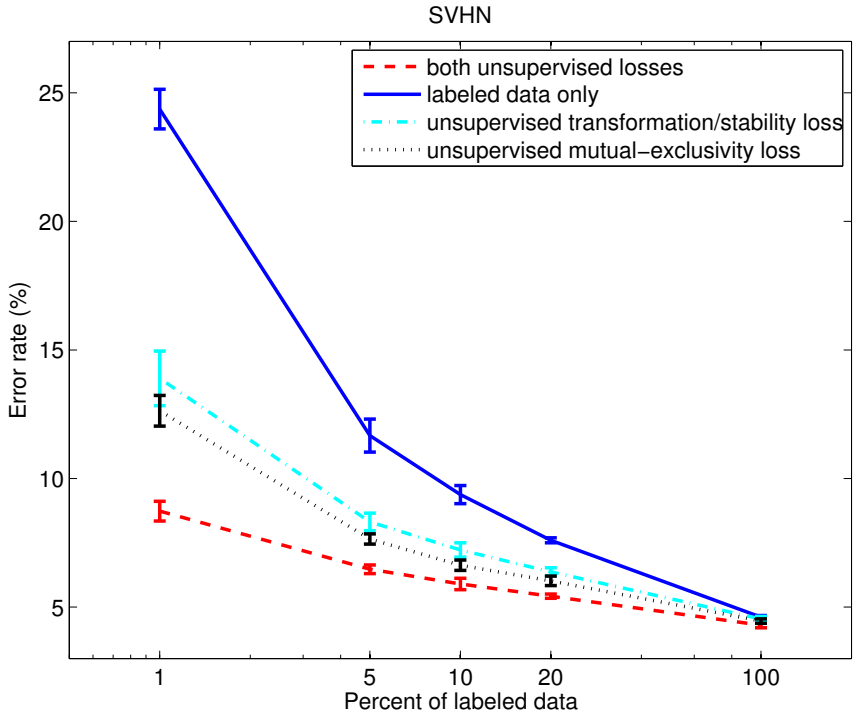


Figure 5.5: SVHN dataset: semisupervised learning vs. training with labeled data only.

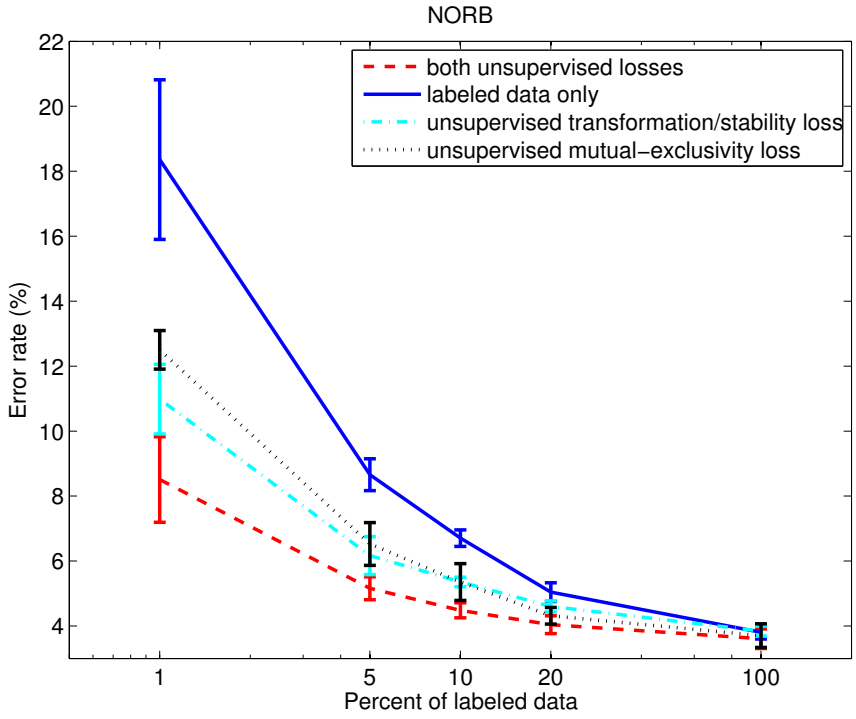


Figure 5.6: NORB dataset: semisupervised learning vs. training with labeled data only.

5.1 is identity function. As a result, dropout and random max-pooling are the only sources of variation in this case. We use the following model: $(32kC2 - FMP\sqrt[3]{2})_{12} - C2 - C1$. Similar to MNIST, we use dropout to regularize the network. Again, the ratio of dropout gradually increases from the first layer to the last layer. The results (average of 5 error rates) are shown in Table 5.2. Here, we can see that by using unsupervised loss functions, we can significantly improve the accuracy of the classifier by trying to minimize the variation in prediction of the network. In addition, for the NORB dataset, we can observe that by using only 1% of labeled data and applying unsupervised loss functions, we can achieve accuracy that is close to the case when we use 100% of labeled data.

Figure 5.7 shows a few examples of the SVHN test set classified correctly by our semisupervised learning method trained on 1% of labeled data but classified incorrectly using a supervised model trained on the same set of labeled samples.

5.4.3 CIFAR10

CIFAR10 is a collection of 60000 tiny 32×32 images of 10 categories (50000 for training and 10000 for test). A few examples of CIFAR10 images are shown in Figure 5.8. We use sparse convolutional networks to perform experiments on this dataset. For this dataset, we create 10 labeled sets. Each set contains 4000 samples that are randomly picked from the training set. All 50000 training samples are used as unlabeled set. We train 2 sets of models on these data. The first set of models is trained on labeled data only, and the other set of models is trained on the unlabeled set using a combination of both unsupervised loss functions in addition to the labeled set. For this dataset, we do not perform separate experiments for 2 unsupervised loss functions because of time constraints. However, based on the results from MNIST, SVHN and NORB, we deduce that the combination of both

Table 5.2: Error on test data for SVHN and NORB with 1% and 100% of data (mean % $\pm$ std).

		labeled data only:	semisupervised:
SVHN	1% of data	12.25 ± 0.80	6.03 ± 0.62
	100% of data	2.28 ± 0.05	2.22 ± 0.04
NORB	1% of data	10.01 ± 0.81	2.15 ± 0.37
	100% of data	1.63 ± 0.12	1.63 ± 0.07

Figure 5.7: Samples of SVHN test set classified correctly by our semisupervised method but classified incorrectly using supervised model. The value in parenthesis shows the wrong prediction of supervised classifier.

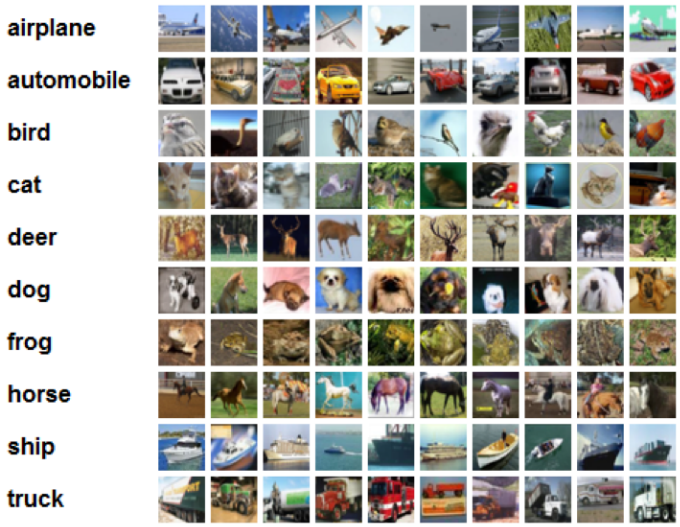


Figure 5.8: Sample images from CIFAR10 dataset.

unsupervised losses provides improved accuracy. We use data augmentation for these experiments. Similar to [123], we perform affine transformations, including randomized mix of translations, rotations, flipping, stretching and shearing operations by $T^j(\mathbf{x}_i)$ of Eq. 5.1. Similar to [123], we train the network without transformations for the last 10 epochs. We use the following parameters for the models: $(32kC2 - FMP\sqrt[3]{2})_{12} - C2 - C1$. We use dropout, and its ratio gradually increases from the first layer to the last layer. The results are given in Table 5.3. We also compare the results to ladder networks [97]. The model in [97] does not use data augmentation. We can see that the combination of unsupervised loss functions on unlabeled data improves the accuracy of the models.

Figure 5.9 shows a few examples of the CIFAR10 test set classified correctly by our semisupervised learning method trained using 4000 labeled samples but classified incorrectly using a supervised model trained on the same set of 4000 labeled samples.

Table 5.3: Error rates on test data for CIFAR10 with 4000 labeled samples (mean % $\pm$ std).

	labeled data only:	semisupervised:
transformation/stability+mutual-exclusivity	13.60 ± 0.24	11.29 ± 0.24
ladder networks [97]	23.33 ± 0.61	20.40 ± 0.47

Figure 5.9: Samples of CIFAR10 test set classified correctly by our semisupervised method but classified incorrectly using supervised model. Wrong predictions of supervised classifier are given in parenthesis.

In another set of experiments, we use all available training data as both labeled and unlabeled sets. We train a network with the following parameters: $(96kC2 - FMP\sqrt[3]{2})_{12} - C2 - C1$. We use affine transformations for this task too. Here again, we use transformation/stability plus the mutual-exclusivity loss function. We repeat this experiments 5 times and achieve **3.18%** $\pm$ **0.1** mean and standard deviation error rate. The state-of-the-art error rate for this dataset is 3.47%, achieved by the fractional max-pooling method [123] but obtained with a larger model ($160n$ vs. $96n$). We perform a single run experiment with $160n$ model and achieve the error rate of **3.00%**. Similar to [123], we perform 100 passes during test time. Here, we surpass state-of-the-art accuracy by adding unsupervised loss functions.

5.4.4 CIFAR100

CIFAR100 is also a collection of 60000 tiny images of size 32×32. This dataset is similar to CIFAR10. However, it contains images of 100 categories compared to 10. Therefore, we have a smaller number of training samples per category. Similar to CIFAR10, we perform experiments on this dataset using sparse convolutional networks. We use all available training data as both labeled and unlabeled sets. The state-of-the-art error rate for this dataset is 23.82%, obtained by fractional max-pooling [123] on sparse convolutional networks. The following model was used to achieve this error rate: $(96kC2 - FMP\sqrt[3]{2})_{12} -$

$C2 - C1$. Dropout was also used with a ratio increasing from the first layer to the last layer. We use the same model parameters and add transformation/stability plus the mutual-exclusivity loss function. Similar to [123], we do not use data augmentation for this task ($T^j(\mathbf{x}_i)$ of Eq. 5.1 is identity function). Therefore, the proposed loss function minimizes the randomness effect due to dropout and max-pooling. We achieve **21.43%** $\pm$ **0.16** mean and standard deviation error rate, which is the state-of-the-art for this task. We perform 12 passes during the test time similar to [123].

5.4.5 ImageNet

We perform experiments on the ILSVRC 2012 challenge. The training data consists of 1281167 natural images of different sizes from 1000 categories. We create 5 labeled datasets from available training samples. Each dataset consists of 10% of training data. We form each dataset by randomly picking a subset of training samples. All available training data is used as the unlabeled set. We use cuda-convnet to train the AlexNet model [6] for this dataset. Similar to [6], all images are re-sized to 256×256. We also use data augmentation for this task following steps of [6], i.e., $T^j(\mathbf{x}_i)$ of Eq. 5.1 performs random translations, flipping and color noise. We train 2 models on each labeled dataset. One model is trained using labeled data only. The other model is trained on both the labeled and the unlabeled set using the transformation/stability plus mutual-exclusivity loss function. At each iteration, we generate 4 different transformed versions of each unlabeled sample. Therefore, each unlabeled sample is forward passed through the network 4 times. Since we use all training data as the unlabeled set, the computational cost of each iteration is roughly quadrupled. However, in practice, we found that when we use 10% of training data as the labeled set, the network converges in 20 epochs instead of the standard 90 epochs of the AlexNet model. Therefore, overall cost of our method for ImageNet is less than or equal to AlexNet. The results on the validation set are shown in Table 5.4. We also compare the results to the model trained on the mutual-exclusivity loss function only. We can see that even for a large dataset with many categories, the proposed unsupervised loss function improves the classification accuracy. The error rate of a single AlexNet model on the validation set of ILSVRC 2012 using all training data is 18.2% [6].

Table 5.4: Error rates (%) on validation set for ILSVR 2012 (Top-5).

	labeled data only:	semisupervised learning:
rep 1	45.73	39.50
rep 2	46.15	39.99
rep 3	46.06	39.94
rep 4	45.57	39.70
rep 5	46.08	40.08
mean $\pm$ std	45.91 $\pm$ 0.25	39.84 $\pm$ 0.23
mutual exclusivity [17]	45.63	42.90
[113] $\sim$1.5% of data	85.9	84.2

5.4.6 Improving Robustness with Respect to Adversarial Examples

ConvNets provide state-of-the-art accuracy for many computer vision tasks. As an example, accuracy of ConvNets on the task of object recognition has surpassed human accuracy for datasets such as ImageNet. However, it is possible to create data samples that easily fool a ConvNet [125]. For example, consider an image of a tiger that ConvNet classifies correctly with high confidence. It is possible to add small disturbances to pixel values that do not change the visual appearance of the image to human eyes. However, the new image can fool the ConvNet to switch the label with very high confidence. It is possible to improve the robustness by increasing the size of labeled data. However, as mentioned earlier, creating large labeled datasets requires a lot of manual effort.

Here we show that it is possible to use the unsupervised loss function proposed here to improve the robustness of the network with respect to adversarial examples. The general idea is to add Gaussian noise to unlabeled samples before passing through the network. In other words, $T^j(\mathbf{x}_i)$ of Eq. 5.1 adds Gaussian noise to unlabeled samples. We perform experiments on MNIST for this task. The robustness of the network is evaluated using the DeepFool [125] algorithm. For a given ConvNet model and an input image, DeepFool calculates the disturbance required for each pixel such that the resulting image changes the prediction of the ConvNet model for the original image. We use the l_2-norm of the disturbance image as a measure of network robustness. If l_2-norm of the disturbance image is higher, it means that the DeepFool needs to perturb the original image more significantly which means the network is more robust. For this experiment, we implemented our unsu-

pervised loss function in the *Caffe* deep learning framework [21]. Similar to experiments of Section 5.4.1, we create a labeled set by randomly selecting 10 samples from each class of MNIST. The unlabeled set is the entire 60000 samples of the MNIST training set. Unlike experiments of Section 5.4.1, we use random cropping of images for both labeled and unlabeled sets. In other words, every 28×28 image of MNIST dataset is cropped to 24×24 at random locations. The reason is that here we use a much simpler network architecture compared to the network of Section 5.4.1 and random cropping helps to improve the accuracy significantly. The network we use is a version of LeNet that is available in *Caffe*. In Eq. 5.1, we set n to be 8. The batch size is 128 which contains 64 labeled samples and repetitions of 8 unlabeled samples. All the input images are divided by 256 such that the input is between 0 and 1. The first convolutional layer produces 20 maps and the second one produces 50 maps. There is a 2×2 max-pooling layer after each convolution. The first fully-connected layer has 500 neurons and uses a ReLU activation unit. We added a dropout layer with ratio of 0.5 between this layer and the second fully-connected layer that produces 10 outputs. This layer is connected to a softmax layer. The labeled part of the batch is connected to a multinomial logistic loss layer and the unlabeled part of the batch is connected to both transformation/stability and mutual-exclusivity losses. The λ is set to 0.005 for both unsupervised losses.

In separate experiments, we add random Gaussian noise $\mathcal{N}(0, \sigma^2)$ with different σ values to unlabeled data. In other words, $T^j(\mathbf{x}_i)$ of unlabeled data is random cropping and Gaussian noise. It must be noted that we do not add noise to labeled data. The only transformation on labeled data is random cropping. The reason is that we want to show that it is possible to improve the robustness of a model by adding noise to unlabeled data and using unsupervised losses.

In the first experiment, we set the $\lambda = 0$ to train a model only based on labeled data. In the second experiment, we set the $\lambda = 0.005$ for both losses but set $\sigma = 0$. In other words, we train a model with unsupervised losses but without any noise. In the next set of experiments, again we set the $\lambda = 0.005$ for unsupervised losses and set the σ to 0.05, 0.1, 0.15, 0.2, 0.25 and 0.3 accordingly. The goal is to study the effect of unsupervised loss as we add more noise to unlabeled data. We repeat each of the experiments 5 times. After running all the experiments with all their repetitions, we select the subset of the

samples of the MNIST test set with all the models and all of their repetitions classified correctly. This subset contains 75% of the test samples. Then we used DeepFool to test all the models with this subset of test data. For each model, we calculate the l_2-norm of the disturbance image for every sample in this subset and calculate the mean of this l_2-norm as a representative for robustness of that model against adversarial examples. Since for each setting we trained 5 different models, we report the mean and standard deviation of this metric over 5 repetitions. The results are given in Table 5.5. The first thing to observe is that even without adding any noise to unlabeled data, we can considerably improve the robustness of the model by only using unsupervised loss functions. In addition, we can observe that as we add more noise to unlabeled data, the resulting model is more robust to adversarial examples which shows that we can improve the robustness of a model by applying our unsupervised loss function.

Figure 5.10 shows visual examples from the MNIST test set. In this figure, the left column shows 3 examples from the subset of MNIST test samples with all the models and all of their repetitions classified correctly. The middle column is the output of the DeepFool algorithm for the ConvNet trained with $\lambda = 0$ which corresponds to the first row of Table 5.5 and the last column corresponds to our model with $\lambda = 0.005$ and $\sigma = 0.25$. We can observe that the output of the DeepFool algorithm for the conventional ConvNet is visually similar to original input images which shows that the regular ConvNet can be easily fooled using DeepFool. However, the output of DeepFool for the model trained using our method is much noisier compared to the output corresponding to the

Table 5.5: Robustness of the models trained with different settings against adversarial examples.

λ	σ	robustness (mean $\pm$ std)	accuracy (mean % $\pm$ std)
0	0	1.30 ± 0.04	84.76 ± 0.92
0.005	0	1.59 ± 0.03	95.56 ± 1.44
0.005	0.05	1.67 ± 0.05	96.20 ± 0.71
0.005	0.10	1.70 ± 0.03	95.77 ± 1.68
0.005	0.15	1.79 ± 0.05	95.11 ± 1.47
0.005	0.20	1.89 ± 0.06	93.58 ± 0.98
0.005	0.25	1.98 ± 0.06	94.23 ± 0.75
0.005	0.30	1.96 ± 0.05	91.31 ± 0.79

Figure 5.10: Visual examples from MNIST test set. *Left column*: 3 images from MNIST test set. *Middle column*: Output of DeepFool algorithm for a conventional ConvNet. *Right column*: Output of DeepFool algorithm for our method trained using Gaussian noise with $\sigma = 0.25$.

conventional ConvNet which means that DeepFool had to generate disturbances with higher magnitude to fool our model. However, we as humans can still easily recognize the images of the DeepFool algorithm that are corresponding to our method. Nonetheless, we believe that our approach is a good first step.

5.5 Discussion

We can see that the proposed loss function can improve the accuracy of a ConvNet regardless of the architecture and implementation. We improve the accuracy of 2 relatively different implementations of ConvNets, i.e., cuda-convnet and sparse convolutional networks. For SVHN and NORB, we do not use dropout or randomized pooling for the experiments performed using cuda-convnet. Therefore, the only source of variation in different passes of a sample through the network is random transformations (translation and rotation). For the experiments performed using sparse convolutional networks on these 2 datasets, we do not use data transformation. Instead, we use dropout and randomized pooling. Based on the results, we can see that in both cases, we can significantly improve the accuracy when we have a small number of labeled samples. For CIFAR100, we achieve state-of-the-art error rate of 21.43% by taking advantage of the variations caused by dropout and randomized pooling. In ImageNet and CIFAR10 experiments, we use both data transformation and dropout. For CIFAR10, we also have randomized pooling

and achieve the state-of-the-art error rate of 3.00%. In MNIST experiments with 100 labeled samples and NORB experiments with 1% of labeled data, we achieve accuracy reasonably close to the case when we use all available training data by applying mutual-exclusivity loss and minimizing the difference in predictions of multiple passes caused by dropout and randomized pooling.

5.6 Conclusion

In this chapter, we proposed an unsupervised loss function that minimizes the variations in different passes of a sample through the network caused by nondeterministic transformations and randomized dropout and max-pooling schemes. We evaluated the proposed method using 2 ConvNet implementations on multiple benchmark datasets. We showed that it is possible to achieve significant improvements in accuracy by using the transformation/stability loss function along with mutual-exclusivity of [17] when we have a small number of labeled data available. We also showed that it is possible to use the proposed loss function to improve the robustness of a model by adding noise to repetitions of unlabeled data.

CHAPTER 6

CONCLUSION

In this dissertation, learning methods were proposed contributing to both supervised learning and semisupervised learning. We introduced a disjunctive normal network model which is based on artificial neural networks and provides efficient coverage of decision space that can be local or nonlocal. An intuitive deterministic initialization scheme was also introduced for the network parameters. This initialization method minimizes the risk of stopping in a poor local minima. We proposed a classification method called LDNN and a regression method named LPBN based on this idea. Through extensive set of experiments on synthetic and real-world benchmarks, it was shown that our models are efficient in terms of number of parameters needed and also provide competitive accuracy and speed for the tasks of general classification and regression.

We contributed to semisupervised learning by introducing 2 unsupervised loss functions called mutual-exclusivity and transformation/stability. These loss functions make use of unlabeled data to improve the accuracy of supervised classifiers. The proposed unsupervised losses are general and can be used with any classifier optimized by gradient descent. Mutual-exclusivity loss encourages the prediction vector associated to an unlabeled sample to have only one nonzero element. In other words, this loss function encourages the dimensions of the prediction vector to be mutually-exclusive. It was shown that this loss function pushes the decision boundary towards less dense areas of decision space and provides better generalization. It was also experimentally proved that when the number of labeled data is small, this unsupervised loss can help to improve the accuracy of a supervised classifier significantly. It was observed that given a fixed labeled set, as we add more unlabeled data, the accuracy of the trained model increases.

Transformation/stability was introduced as an unsupervised loss function that exploits the randomness of the training process to improve the accuracy of a supervised classifier.

This loss function can use randomized data augmentation and internal random variations of a network such as dropout and stochastic pooling schemes to improve the representation power of the learning system.

The combination of mutual-exclusivity and transformation/stability losses with a general supervised loss function can create a novel and state-of-the-art semisupervised learning method. For example, using only 100 labeled samples of MNIST dataset, we trained a model that in terms of accuracy is reasonably close to a model trained using all 60000 labeled samples. For this purpose, we used all 60000 samples of MNIST as unlabeled data. Similar results were also shown with the NORB dataset with using only 1% of labeled data. It was also experimentally demonstrated that we can improve the accuracy of the models trained on the ImageNet dataset that has 1000 classes and contains more than 1.2 million images.

We believe that unlabeled data can help to describe the underlying distribution of the data samples even further. Additionally, the process of learning in human beings is semisupervised and proper processing of unlabeled data can provide significant information for a learning system.

One possible direction to exploit unlabeled data is to design cost functions for stabilizing the gradient of a learning function with respect to input data. For example, we know that adding small amounts of noise and disturbances to a data sample should not change the prediction of a learning function. When we change the input data by moving its corresponding high dimensional data point in the direction of these disturbances, the prediction of the learning function should not change. In other words, the direction of the gradient of the learning function should be orthogonal to these disturbances. This simple observation puts restrictions on the gradient of the decision function that can be used for regularization of the network. Note that we don't need the label of a data sample to enforce this property. So, we can design cost functions to define the behavior of the gradient function using only unlabeled data.

REFERENCES

[1] T. Hastie, R. Tibshirani, and J. Friedman, *Unsupervised Learning*, pp. 485–585. New York, NY: Springer New York, 2009.

[2] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-supervised Learning*. Adaptive computation and machine learning, MIT Press, 2006.

[3] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learning*, vol. 20, pp. 273–297, Sep, 1995.

[4] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, Mar, 1986.

[5] L. Breiman, "Random forests," *Mach. Learning*, vol. 45, pp. 5–32, Oct, 2001.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances Neural Inform. Process. Syst. 25*, pp. 1097–1105, 2012.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pp. 770–778, Jun. 2016.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[9] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pp. 3431–3440, Jun. 2015.

[10] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, pp. 82–97, Nov, 2012.

[11] D. S. Broomhead and D. Lowe, "Radial basis functions, multi-variable functional interpolation and adaptive networks," Tech. Rep., DTIC Document, 1988.

[12] M. Hazewinkel, ed., *Encylopedia of Mathematics*. Springer, 2001.

[13] M. Sajjadi, M. Seyedhosseini, and T. Tasdizen, "Disjunctive normal networks," *Neurocomputing*, vol. 218, pp. 276–285, 2016.

[14] M. Sajjadi, M. Seyedhosseini, and T. Tasdizen, "Nonlinear regression with logistic product basis networks," *IEEE Signal Process. Lett.*, vol. 22, pp. 1011–1015, Aug, 2015.

[15] X. Zhu, "Semi-supervised learning literature survey," Tech. Rep. 1530, Computer Sciences, University of Wisconsin-Madison, 2005.

[16] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," Tech. Rep. CMU-CALD-02-107, Carnegie Mellon University, 2002.

[17] M. Sajjadi, M. Javanmardi, and T. Tasdizen, "Mutual exclusivity loss for semi-supervised deep learning," in *2016 IEEE Int. Conf. Image Process. (ICIP)*, pp. 1908–1912, Sep. 2016.

[18] M. Sajjadi, M. Javanmardi, and T. Tasdizen, "Regularization with stochastic transformations and perturbations for deep semi-supervised learning," in *Advances Neural Inform. Process. Syst. 29*, pp. 1163–1171, 2016.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jan, 2014.

[20] B. Graham, "Spatially-sparse convolutional neural networks," *arXiv preprint arXiv:1409.6070*, 2014.

[21] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[22] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359 – 366, 1989.

[23] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics Control, Signals Syst.*, vol. 2, pp. 303–314, Dec, 1989.

[24] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 9, pp. 533–536, 1986.

[25] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances Neural Inform. Process. Syst. 2*, pp. 396–404, 1990.

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, pp. 2278–2324, Nov. 1998.

[27] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, pp. 1915–1929, Aug, 2013.

[28] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[29] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE Conf. Comput. Vision Pattern Recognition*, pp. 3642–3649, Jun. 2012.

[30] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proc. 30th Int. Conf. Mach. Learning - Vol. 28*, ICML'13, pp. III–1058–III–1066, JMLR.org, 2013.

[31] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *Proc. 30th Int. Conf. Mach. Learning - Vol. 28*, ICML'13, pp. III–1319–III–1327, JMLR.org, 2013.

[32] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances Neural Inform. Process. Syst. 2*, pp. 524–532, 1990.

[33] R. Kohn, M. Smith, and D. Chan, "Nonparametric regression using linear combinations of basis functions," *Statist. Computing*, vol. 11, pp. 313–322, Oct, 2001.

[34] T. Heskes and B. Kappen, *Mathematical Approaches to Neural Networks*, vol. 51, ch. On-line learning processes in artifical neural networks, pp. 199–233. Elsevier, 1993.

[35] G. Orr, *Dynamics and Algorithms for Stochastic Learning*. PhD thesis, Oregon Graduate Inst., 1995.

[36] M. Joost and W. Schiffmann, "Speeding up backpropagation algorithms by using cross-entropy combined with pattern normalization," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 6, pp. 117–126, Apr, 1998.

[37] D. Saad and S. A. Solla, "Exact solution for on-line learning in multilayer neural networks," *Physical Rev. Lett.*, vol. 74, no. 21, pp. 4337–4340, 1995.

[38] N. Murata, K.-R. Müller, A. Ziehe, and S. ichi Amari, "Adaptive on-line learning in changing environments," in *Advances Neural Inform. Process. Syst. 9*, pp. 599–605, 1997.

[39] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient BackProp*, pp. 9–48. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[40] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, pp. 1527–1554, Jul, 2006.

[41] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[42] P. K. Simpson, "Fuzzy min-max neural networks. i. classification," *IEEE Trans. Neural Networks*, vol. 3, pp. 776–786, Sep, 1992.

[43] B. G. Song, R. J. Marks, S. Oh, P. Arabshahi, T. P. Caudell, and J. J. Choi, "Adaptive membership function fusion and annihilation in fuzzy if-then rules," in *[2nd IEEE Int. Conf. Fuzzy Syst.*, vol. 2, pp. 961–967, 1993.

[44] A. V. Nandedkar and P. K. Biswas, "A fuzzy min-max neural network classifier with compensatory neuron architecture," in *Proc. 17th Int. Conf. Pattern Recognition, 2004. ICPR 2004.*, vol. 4, pp. 553–556, Aug. 2004.

[45] B. L. Lu, H. Kita, and Y. Nishikawa, "A multi-sieving neural network architecture that decomposes learning tasks automatically," in *Neural Networks, IEEE World Congr. Computational Intell.*, vol. 3, pp. 1319–1324, Jun. 1994.

[46] H.-M. Lee, K.-H. Chen, and I.-F. Jiang, "A neural network classifier with disjunctive fuzzy information," *Neural Networks*, vol. 11, no. 6, pp. 1113–1125, 1998.

[47] B.-L. Lu and M. Ito, "Task decomposition and module combination based on class relations: a modular neural network for pattern classification," *IEEE Trans. Neural Networks*, vol. 10, pp. 1244–1256, Sep, 1999.

[48] R. M. II, S. Oh, P. Arabshahi, T. Caudell, J. Choi, and B. Song, "Steepest descent adaptations of min-max fuzzy if-then rules," in *Proc Int. Joint Conf. Neural Networks*, vol. III, pp. 471–477, 1992.

[49] H. Nomura, I. Hayashi, and N. Wakami, "A learning method of fuzzy inference rules by descent method," in *[1992 Proc.] IEEE Int. Conf. Fuzzy Syst.*, pp. 203–210, Mar. 1992.

[50] X. Zhang, C.-C. Hang, S. Tan, and P.-Z. Wang, "The delta rule and learning for min-max neural networks," in *Neural Networks, 1994. IEEE World Congr. Computational Intell., 1994 IEEE Int. Conf.*, vol. 1, pp. 38–43, Jun. 1994.

[51] X. Zhang, C.-C. Hang, S. Tan, and P.-Z. Wang, "The min-max function differentiation and training of fuzzy neural networks," *IEEE Trans. Neural Networks*, vol. 7, pp. 1139–1150, Sep, 1996.

[52] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[53] S. E. Yuksel, J. N. Wilson, and P. D. Gader, "Twenty years of mixture of experts," *IEEE Trans. Neural Networks Learning Syst.*, vol. 23, pp. 1177–1193, Aug, 2012.

[54] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, pp. 1771–1800, Aug, 2002.

[55] M. J. Orr, "Recent advances in radial basis function networks," Tech. Rep., Institute for Adaptative and Neural Computation, Edinburgh University, 1999.

[56] F. Schwenker, H. A. Kestler, and G. Palm, "Three learning phases for radial-basis-function networks," *Neural Networks*, vol. 14, no. 4, pp. 439 – 458, 2001.

[57] T. Hastie and R. Tibshirani, "Discriminant analysis by gaussian mixtures," *J. Roy. Statistical Soc.. Series B (Methodological)*, vol. 58, no. 1, pp. 155–176, 1996.

[58] M. Zhu and A. M. Martinez, "Subclass discriminant analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, pp. 1274–1286, Aug, 2006.

[59] T.-K. Kim and J. Kittler, "Locally linear discriminant analysis for multimodally distributed classes for face recognition with a single model image," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, pp. 318–327, Mar, 2005.

[60] J. Wang and V. Saligrama, "Local supervised learning through space partitioning," in *Advances Neural Inform. Process. Syst. 25*, pp. 91–99, 2012.

[61] J. Dai, S. Yan, X. Tang, and J. T. Kwok, "Locally adaptive classification piloted by uncertainty," in *Proc. Int. Conf. on Mach. Learning*, ICML '06, (New York, NY, USA), pp. 225–232, ACM, 2006.

[62] M. Toussaint and S. Vijayakumar, "Learning discontinuities with products-of-sigmoids for switching between local models," in *Proc. Int. Conf. on Mach. Learning*, ICML '05, (New York, NY, USA), pp. 904–911, ACM, 2005.

[63] O. Dekel and O. Shamir, "Theres a hole in my data space: Piecewise predictors for heterogeneous learning problems," in *Proc. Int. Conf. on Artificial Intell. Mach. Learning*, pp. 291–298, 2012.

[64] H. Cheng, P.-N. Tan, and R. Jin, "Localized support vector machine and its efficient algorithm," in *Proc. SIAM Int. Conf. on Data Mining*, pp. 461–466, 2007.

[65] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics, Springer New York, 2013.

[66] B. Schulmeister and F. Wysotzki, *Dipol - a hybrid piecewise linear classifier*, ch. Machine Learning and Statistics: the Interface, pp. 133–151. John Wiley and Sons, 1997.

[67] T. Hastie and R. Tibshirani, "Classification by pairwise coupling," in *Advances Neural Inform. Process. Syst. 10*, pp. 507–513, 1998.

[68] B.-L. Lu, K.-A. Wang, M. Utiyama, and H. Isahara, "A part-versus-part method for massively parallel training of support vector machines," in *Proc. Int. Joint. Conf. Neural Networks*, vol. 1, pp. 735–740, Jul. 2004.

[69] T.-F. Wu, C.-J. Lin, and R. C. Weng, "Probability estimates for multi-class classification by pairwise coupling," *J. Mach. Learn. Res.*, vol. 5, pp. 975–1005, Dec, 2004.

[70] J. Wu, H. Xiong, P. Wu, and J. Chen, "Local decomposition for rare class analysis," in *Proc. 13th ACM SIGKDD Int. Conf. Knowledge Discovery Data Mining*, KDD '07, (New York, NY, USA), pp. 814–823, ACM, 2007.

[71] F. Chen, C. T. Lu, and A. P. Boedihardjo, "On locally linear classification by pairwise coupling," in *2008 8th IEEE Int. Conf. Data Mining*, pp. 749–754, Dec. 2008.

[72] H. Abbassi, R. Monsefi, and H. Sadoghi Yazdi, "Constrained classifier: a novel approach to nonlinear classification," *Neural Computing and Appl.*, vol. 23, pp. 2367–2377, Dec, 2013.

[73] H. Poon and P. Domingos, "Sum-product networks: A new deep architecture," in *2011 IEEE Int. Conf. Comput. Vision Workshops (ICCV Workshops)*, pp. 689–690, Nov. 2011.

[74] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. Wiley-Interscience, 2001.

[75] A. Asuncion and D. Newman, "Uci machine learning repository," 2007.

[76] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Trans. intelligent Syst.Technol. (TIST)*, vol. 2, no. 3, p. 27, 2011.

[77] C.-C. Chang and C.-J. Lin, "Ijcnn 2001 challenge: generalization ability and text decoding," in *Int. Joint Conf. Neural Networks*, vol. 2, pp. 1031–6, 2001.

[78] I. Nabney and C. Bishop, "Netlab neural network software," *Matlab Toolbox*, vol. 71, p. 7, 2003.

[79] T. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[80] K. R. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf, "An introduction to kernel-based learning algorithms," *IEEE Trans. Neural Networks*, vol. 12, pp. 181–201, Mar, 2001.

[81] N. Meinshausen, "Quantile regression forests," *J. Mach. Learn. Res.*, vol. 7, pp. 983–999, Dec, 2006.

[82] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*, vol. 1. MIT press Cambridge, 2006.

[83] D. Huang, R. Cabral, and F. D. l. Torre, "Robust regression," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, pp. 363–375, Feb, 2016.

[84] V. Vapnik, S. E. Golowich, and A. J. Smola, "Support vector method for function approximation, regression estimation and signal processing," in *Advances Neural Inform. Process. Syst. 9*, pp. 281–287, 1997.

[85] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statist. Computing*, vol. 14, pp. 199–222, Aug, 2004.

[86] G. B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Trans. Syst., Man, Cybernetics, Part B (Cybernetics)*, vol. 42, pp. 513–529, Apr, 2012.

[87] C. Bishop, "Improving the generalization properties of radial basis function neural networks," *Neural Computation*, vol. 3, pp. 579–588, Dec, 1991.

[88] J.-N. Hwang, S.-R. Lay, and A. Lippman, "Nonparametric multivariate density estimation: a comparative study," *IEEE Trans. Signal Process.*, vol. 42, pp. 2795–2810, Oct, 1994.

[89] A. Jaiantilal, "Classification and regression by randomforest-matlab." Available at *http://code.google.com/p/randomforest-matlab*, 2009.

[90] C.-W. Hsu, C.-C. Chang, C.-J. Lin, *et al.*, "A practical guide to support vector classification," Tech. Rep., National Taiwan University, 2003.

[91] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[92] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pp. 1–9, Jun. 2015.

[93] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proc. 26th Annu. Int. Conf. Mach. Learning*, ICML '09, (New York, NY, USA), pp. 609–616, ACM, 2009.

[94] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proc. of 2010 IEEE Int. Symp. Circuits Syst.*, pp. 253–256, May. 2010.

[95] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in *2009 IEEE 12th Int. Conf. Comput. Vision*, pp. 2146–2153, Sep. 2009.

[96] K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "Fast inference in sparse coding algorithms with applications to object recognition," *arXiv preprint arXiv:1010.3467*, 2010.

[97] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko, "Semi-supervised learning with ladder networks," in *Advances Neural Inform. Process. Syst. 28*, pp. 3546–3554, 2015.

[98] R. Johnson and T. Zhang, "Semi-supervised convolutional neural networks for text categorization via region embedding," in *Advances Neural Inform. Process. Syst. 28*, pp. 919–927, 2015.

[99] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proc. Eleventh Annu. Conf. Computational Learning Theory*, COLT' 98, (New York, NY, USA), pp. 92–100, ACM, 1998.

[100] V. R. de Sa, "Learning classification with unlabeled data," in *Advances Neural Inform. Process. Syst. 6*, pp. 112–119, 1994.

[101] D. J. Miller and H. S. Uyar, "A mixture of experts classifier with learning based on both labelled and unlabelled data," in *Advances Neural Inform. Process. Syst. 9*, pp. 571–577, 1997.

[102] T. Joachims, "Transductive inference for text classification using support vector machines," in *Proc. 16th Int. Conf. Mach. Learning*, ICML '99, (San Francisco, CA, USA), pp. 200–209, Morgan Kaufmann Publishers Inc., 1999.

[103] K. P. Bennett and A. Demiriz, "Semi-supervised support vector machines," in *Advances Neural Inform. Process. Syst. 11*, pp. 368–374, 1999.

[104] A. Blum and S. Chawla, "Learning from labeled and unlabeled data using graph mincuts," in *Proc. 18th Int. Conf. Mach. Learning*, ICML '01, (San Francisco, CA, USA), pp. 19–26, Morgan Kaufmann Publishers Inc., 2001.

[105] X. Zhu, Z. Ghahramani, and J. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proc. 20th Int. Conf. Int. Conf. Mach. Learning*, ICML'03, pp. 912–919, AAAI Press, 2003.

[106] V. Castelli and T. M. Cover, "The relative value of labeled and unlabeled samples in pattern recognition with an unknown mixing parameter," *IEEE Trans. Inform. Theory*, vol. 42, pp. 2102–2117, Nov, 1996.

[107] T. J. O'neill, "Normal discrimination with unclassified observations," *J. Amer. Statistical Assoc.*, vol. 73, no. 364, pp. 821–826, 1978.

[108] Y. Grandvalet and Y. Bengio, "Semi-supervised learning by entropy minimization," in *Advances Neural Inform. Process. Syst. 17*, pp. 529–536, 2005.

[109] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., Computer Science Department, University of Toronto, 2009.

[110] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proc. IEEE Conf. Comput. Vision Pattern Recognition, 2004. CVPR 2004.*, vol. 2, pp. II–97–104 Vol.2, Jun. 2004.

[111] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, p. 4, Granada, Spain, 2011.

[112] A. Krizhevsky, "cuda-convnet: High-performance c++/cuda implementation of convolutional neural networks," 2012.

[113] P. Agrawal, J. Carreira, and J. Malik, "Learning to see by moving," in *2015 IEEE Int. Conf. Comput. Vision (ICCV)*, pp. 37–45, Dec. 2015.

[114] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *2015 IEEE Int. Conf. Comput. Vision (ICCV)*, pp. 1422–1430, Dec. 2015.

[115] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Int. Conf. artificial Intell. Statist.*, pp. 249–256, 2010.

[116] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, *Deep Learning via Semi-supervised Embedding*, pp. 639–655. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[117] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," in *2015 IEEE Int. Conf. Comput. Vision (ICCV)*, pp. 2794–2802, Dec. 2015.

[118] D. Jayaraman and K. Grauman, "Learning image representations tied to egomotion," in *2015 IEEE Int. Conf. Comput. Vision (ICCV)*, pp. 1413–1421, Dec. 2015.

[119] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox, "Discriminative unsupervised feature learning with convolutional neural networks," in *Advances Neural Inform. Process. Syst. 27*, pp. 766–774, 2014.

[120] P. Y. Simard, Y. A. LeCun, J. S. Denker, and B. Victorri, *Transformation Invariance in Pattern Recognition — Tangent Distance and Tangent Propagation*, pp. 239–274. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998.

[121] D. Jayaraman and K. Grauman, "Slow and steady feature analysis: Higher order temporal coherence in video," in *2016 IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pp. 3852–3861, Jun. 2016.

[122] L. Sun, K. Jia, T. H. Chan, Y. Fang, G. Wang, and S. Yan, "Dl-sfa: Deeply-learned slow feature analysis for action recognition," in *2014 IEEE Conf. Comput. Vision Pattern Recognition*, pp. 2625–2632, Jun. 2014.

[123] B. Graham, "Fractional max-pooling," *arXiv preprint arXiv:1412.6071*, 2014.

[124] J.-R. Chang and Y.-S. Chen, "Batch-normalized maxout network in network," *arXiv preprint arXiv:1511.02583*, 2015.

[125] S. M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *2016 IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pp. 2574–2582, Jun. 2016.