AUTOMATED MULTISTEP CLASSIFIER SIZING AND TRAINING FOR DEEP LEARNER

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF UNIVERSITY OF TEXAS AT ARLINGTON
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Kanishka Tyagi
January 2018

© Copyright by Kanishka Tyagi 2018
All Rights Reserved

To my first teacher, my Mother
and my grandmother *'Amma'*

Abstract

Training algorithms for deep learning have recently been proposed with notable success, beating the start-of-the-art in certain areas like audio, speech and language processing. The key role is played by learning multiple levels of abstractions in a deep architecture. However, searching the parameters space in a deep architecture is a difficult task. By exploiting the greedy layer-wise unsupervised training strategy of deep architecture, the network parameters are initialized near a good local minima. However, many existing deep learning algorithms require tuning a number of hyperparameters including learning factors and the number of hidden units in each layer. Apart from this, a predominant methodology in training deep learning models promotes the use of gradient-based algorithms which require heavy computational resources. Poor training algorithms and excessive user chosen parameters in a learning model makes it difficult to train a deep learner. In this dissertation, we break down the training of deep learning into basic building blocks of unsupervised approximation training followed by a supervised classification learning block. We propose a multi-step training method for designing generalized linear classifiers. First, an initial multi-class linear classifier is found through regression. Then validation error is minimized by pruning of unnecessary inputs. Simultaneously, desired outputs are improved via a method similar to the Ho-Kashyap rule. Next, the output discriminants are scaled to be net functions of sigmoidal output units in a generalized linear classifier. This classifier is trained via Newton's algorithm. Performance gains are demonstrated at each step. We then develop a family of batch training algorithm for the multi layer perceptron that optimizes its hidden layer size and number of training epochs. At the end of each training epoch, median filtering removes any kind of noise in the validation error vs number of hidden units curve and the networks get temporarily pruned. Since, pruning is done at each epoch, we save the best network thereby optimizing the number of hidden units as well as the number of epochs simultaneously. Next, we combine pruning with a growing approach. Later, the input units are scaled to be the net function of the sigmoidal output units that are then feed into as input to the MLP. We then propose resulting improvements in each of the deep learning blocks thereby improving the overall performance of the deep architecture. We discuss the principles and formulation regarding learning algorithms for deep autoencoders. We investigate several problems in deep autoencoders networks including training

issues, the theoretical, mathematical and experimental justification that the networks are linear, optimizing the number of hidden units in each layer and determining the depth of the deep learning model. A direct implication of the current work is the ability to construct fast deep learning models using desktop level computational resources. This, in our opinion, promotes our design philosophy of building small but powerful algorithms. Performance gains are demonstrated at each step. Using widely available datasets, the final network's ten fold testing error is shown to be less than that of several other linear, generalized linear classifiers, multi layer perceptron and deep learners reported in the literature.

Acknowledgments

For all these blissful years of my experience as a Ph.D. student, I must thank a number of faculties, staff, friends, and family for there contributions.

First I would like to thank my supervising professor Dr. Michael Manry who, through his blackboard discussions and innumerable meetings over four years, shaped me from a restless graduate student to a capable researcher. I found his passion for perfection, sense of humor and a calm attitude extremely infectious. He listened to all my overambitious ideas patiently and then carefully exposed the fundamental theory behind it by putting it in a wider context. When I produced some horrible paper drafts, his sheer patience to correct till I get it right is still a mystery to me. My priceless moments of education includes observing him as he makes his way out of a mathematical proof or a pseudo code while sipping his coffee. I am grateful to Dr. Manry for teaching me not just a majority of neural networks but also for teaching me how to convert a raw idea into a mature concept.

I sincerely thank Dr.Victoria Chen, Dr.Kamesh Subbarao, Dr.Vassilis Athitsos, Dr.Wei-Jen Lee and Dr.Ionnis Schizas for their interest in my research and for taking time to serve on my dissertation committee.

My thinking and research philosophy has been shaped over the years by many inspiring people who I have a lot of respect and admiration. I would like to thank my first mentor Dr.Abhishek Yadav who motivated me to get in the neural networks field and Dr.Prem Kalra who shaped my approach to research and inspired me to think big. I would especially like to thank Dr.H.C Verma whose unique perspective on the philosophy of life had a deep impact on me.

I am thankful to my collaborators who I've had the pleasure of working with. This includes Rohit Rawat, Son Nguyen, Yilong Hao, Soumitro Auddy and Parastoo Kheirkhah.

I have to thank many people who I was fortunate to get to know and become friends with: Raghavendra Sriram, Babak Namazi, Akshay Malhotra, Bito Irie, James Grisham, Sarika Nagaraj, Chetan Bisht, Mohammad Reza, Jiafan Wang, Julia Martius, Nayana Parashar, Chaitanya Rani, Mythreya Lakshman, Vivek Nair, Manu Ajmani, Sunny Agrawal and Ritesh Joshi.

During my Ph.D, I was able to squeeze in a visiting research position at Seoul National University and summer internships at Mathworks and Google Research. All three were an extremely enjoyable learning experience that had an enormous impact on my research path. From this experience, I would especially like to thank Dr.Kyogu Lee, Dr.Nojun Kwak, Dr.Tom Lane, Dr.Ilya Narsky, Dr.Mark Sandler, Dr.Andrey Zhmoginov, Dr.Xiao Zhang, Roza Chojnacka, Jing Wang, Casey Chu, Ken Pfeuffer, Pritam Bhattacharya, Il Young Jeong, Hama Lok and Jung Wook Park.

I would also especially thank friends who I had been interacting from my Master's degree, especially Rajinikanth Reddy, Himanshu Jain, Sayan Bardhan, Xueyang Cheng, Eshwar Sachidananda and my high-school friends Abhinav Mitra Anurag Kumar and Ritwik Raj. Special thanks to Sandy Kurtzman for her warm nature that always made me feel at home.

Back home in India, a big thank you to my uncle Pankaj for his endless love and FaceTime calls during all these years of studies. Thanks to my sisters Anubhuti, Prachita, Aditi, my brothers Harshvardhan, Anubhav, Aman, Priyansh for always being cheerful and supportive. And to my parents who always believed in me and gave me the freedom to take my own decisions. A large part of my motivation, to be always ambitious, has been to defend the faith that my parents had on me.

Contents

Abstract **iv**

Acknowledgments **vi**

1 Introduction **1**

1.1 Introduction 1

1.1.1 Overview of problem in current methods 3

1.1.2 Proposed work 6

2 Linear Networks Review **7**

2.1 Review: Linear Networks 7

2.1.1 Structure and notation 7

2.1.2 Regression based classifier 7

2.1.3 LIBLINEAR 9

2.1.4 Sequential dual method 10

2.1.5 Iteratively reweighted least squares 10

2.1.6 Sigmoidal SVM 10

3 Review: MLP with single hidden layer **12**

3.1 Review: MLP with single hidden layer 12

3.1.1 Structure and Notation 12

3.1.2 Initialization 14

3.1.3 First order learning algorithms 16

3.1.4 Softmax classifier 19

3.1.5 Radial basis function classifier 20

3.1.6 Nonlinear SVM classifier 20

3.1.7 Second order learning algorithms 21

4 Review: Deep neural networks **25**

4.1 Review: Deep neural networks . . . 25
4.1.1 Auto-Encoders: Structure, Notations, and Training . . . 25
4.1.2 Deep autoencoder: Structure and Notation . . . 27
4.1.3 Deep autoencoder:pre-training and stacking . . . 28
4.1.4 Classifier design . . . 31
4.1.5 Fine tuning . . . 31
4.1.6 Variants of autoencoders . . . 32

5 Problems, Goal and Proposed Tasks **36**
5.0.7 Problems with linear classifiers . . . 36
5.0.8 Problems with single hidden layer MLP . . . 37
5.0.9 Problems with deep learning networks . . . 38
5.0.10 Goals and proposed tasks . . . 39

6 Proposed improvements in linear networks **41**
6.0.11 Modifying targets with output reset . . . 41
6.0.12 Pruning less useful inputs . . . 43
6.0.13 Newton based improvements . . . 48

7 Improvements in MLP algorithm **53**
7.0.14 Modification with regularization . . . 54
7.0.15 Improved pruning using median filtering . . . 56
7.0.16 Growing approach . . . 58
7.0.17 Network Pruning . . . 58
7.0.18 Growing Versus Pruning . . . 59
7.0.19 Improvements in inputs in MA training algorithms . . . 62
7.0.20 Final algorithm . . . 65

8 Investigation and improvements in deep autoencoders **67**
8.0.21 Linearity in deep learning features . . . 69
8.0.22 Analysis of sparse autoencoders . . . 71
8.0.23 Linear classifiers as probe . . . 73

9 Proposed Auto encoder theory **75**
9.0.24 Proof for linearity of AE features . . . 75
9.0.25 KLT and Linear Mapping . . . 77

10 Experimental results **78**
10.0.26 Shallow classifiers . . . 79
10.0.27 Deep learning classifiers . . . 79

11 Conclusions **83**

A Datasets **85**

A.0.28 Gongtrn dataset 85
A.0.29 Comf18 dataset 86
A.0.30 MNIST dataset 86
A.0.31 Google street view dataset 87
A.0.32 CIFAR dataset 88
A.0.33 STL-10 dataset 89
A.0.34 COVER 90
A.0.35 RCV1 90
A.0.36 NEWS-20 91

B Training weights by orthogonal least squares **92**

Bibliography **95**

List of Tables

6.1 Pruning results on linear classifiers . . . 47
6.2 10-fold testing error % results for various linear classifiers . . . 48
6.3 10-fold testing error % results for GLC-Newton . . . 52

7.1 Pruning hidden units results MA-OR algorithm . . . 55
7.2 10-fold testing error % results for MOLF based classifiers with $N_{it} = 100$ . . . 56
7.3 Pruning hidden units results on various MA improvements . . . 60
7.4 Comparision results for 10-fold testing accuracy results on single hidden layer MLP with $N_{it} = 100$ . . . 61
7.5 Comparision results for 10-fold testing accuracy results on single hidden layer MLP with $N_{it} = 100$ . . . 66

8.1 Linearity test on features from deep autoencoders for MNIST dataset . . . 70
8.2 COD for deep autoencoders for MNIST dataset . . . 71
8.3 Linearity experiments on MNIST dataset with Deep sparse autoencoder . . . 72
8.4 10 fold testing accuracy with various combinations of features and classifiers for MNIST dataset . . . 72
8.5 10 fold testing accuracy results on improvements with pruning and linear probe for proposed deep AE . . . 73
8.6 10 fold testing accuracy of proposed deep autoencoder . . . 73

10.1 10-fold testing error % results for GLC-Newton and other algorithms . . . 78
10.2 Comparision results for 10-fold testing % results for MA-OR-R-MF-GP-IS with comparing algorithms . . . 79
10.3 5-fold testing error % results for claimbuster dataset using SVM . . . 80
10.4 5-fold testing error % results for claimbuster dataset using DAE-Softmax . . . 80
10.5 5-fold testing error % results for claimbuster dataset using DAE-MA-OR-R-MF-GP-IS 81
10.6 10-fold testing error % results using DAE-prune-MOLF-Adapt-MF . . . 81

A.1 Specification of datasets 85

List of Figures

1.1 Composition of a Deep Learning algorithm . . . 3
1.2 A generalized learning paradigm . . . 5

2.1 Linear classifier . . . 8

3.1 Fully connected MLP . . . 14
3.2 Typical training algorithms for training an MLP . . . 16

4.1 Cascade connected autoencoder . . . 26
4.3 Block diagram for general deep learning algorithm . . . 28
4.4 Unsupervised pre-training . . . 29
4.5 Autoencoder stacking . . . 30
4.6 Deep neural network . . . 32
4.2 Summary of deep neural network . . . 35

6.1 Generalized linear model classifier . . . 49

7.1 Validation error and smooth validation curves for *Gongtrn* file . . . 57
7.2 Testing error and smooth testing curves for *Gongtrn* file . . . 57
7.3 Input sigmoids in linear network . . . 63

8.1 Linear probe in a deep neural network . . . 68

A.1 One hundred examples of the MNIST digits chosen at random from the training set 87
A.2 Fifteen examples of the SVHN chosen at random from the training set . . . 88
A.3 One hundred examples of the CIFAR images chosen at random from the training set 89
A.4 One hundred examples of the STL-10 images chosen at random from the training set 90

Chapter 1

Introduction

1.1 Introduction

Machine learning is a consolidation of artificial intelligence and statistics in which we develop a set of algorithms that tries to confront the holy grail of building a general purpose learning system. From an artificial intelligence point of view, one focuses on building a computer to learn and automatically identifying patterns in data and then uncover patterns to either predict (regression) or to make certain kind of decisions (classification). From a statistics standpoint, the question one addresses is to find the computational parametric models that are required, in order to infer from the data. 200 years ago, David Hume made an observation that every learner constitutes some knowledge beyond the data it's given in order to generalize beyond it. The *no free lunch* theorem [179] formalizes it by stating that no machine learning algorithm can beat random guessing over all possible functions that need to be learned.

In recent past, machine learning applications have advanced dramatically from the research lab to actual commercial products. As in [123], till 1985 there was no commercial application of machine learning. A growing number of linear and non linear algorithms are used to solve classification problem in fields such as face recognition [11], [96], automatic tagging [169], speech recognition [41], natural language processing [134], computer vision [142] [12] [182], bio surveillance [85], recommendation engines [166], robotic control [132] and bio-engineering [89]. Classification algorithms are also used in many data intensive areas such as consumer services [5] [32], flight engine fault diagnosis [159], astronomy [10] and many varying fields from social science [58] to computational biology [170], [158] and criminology [28] to wildlife protection [165]. Many unconventional application such as [55] discuss a robot waiter that recognize different drinks. Machine learning applications also

includes many regression tasks such as claiming amount for insurance premiums [20] and [48], share trading [35], [76]. Modern machine learning systems are also used in transcription for advanced speech recognition [68], [57], [40] and machine translation [7], [167]. It has also found success in many structured output tasks where the expected output are all tightly interrelated such as road location from aerial photographs [124] and image captioning [84], [175] where the words produced by the image captioning program must form a valid sentence. Anomaly detection methods [24] as well as synthesis and sampling for video game applications and image style transfer [81] have all benefited from the advancements in machine learning algorithms. Following a basic structure of pattern recognition systems, a machine learning algorithm includes three components namely: 1) feature extraction, 2) feature selection, and 3) feature classification. However, the no free lunch (NFL) theorem [179] implied that no machine learning algorithm can beat random guessing over all possible functions that need to be learned.

Deep learning is a rapidly growing area within the machine learning domain that makes use of multiple levels of representations. Deep networks have been applied with success not only in classification tasks [16], [120] [95] [21] but also in regression [70], dimensionality reduction [69] [120] textures modeling [137], information retrieval [70] and robotics [59]. Deep learning has performed remarkably well in difficult non-linear pattern recognition task such as speech recognition [161] at Microsoft, object recognition on the ImageNet dataset [91], machine translation [160]and language processing at Google Image search [176]. The word *Deep* refers to the fact that the network has more than one hidden layer. Each successive layer's output represent higher level features than in the previous layer. The idea of using unsupervised learning at each stage of a deep network was put forward in [69], as part of a training procedure for the deep belief network (DBN), a generative model with many layers of hidden stochastic variables. This training strategy has inspired a more general approach to help address the problem of training deep networks. This approach can be extended to non linear autoencoders [156], as shown by [13] and is found in stacked autoencdoer networks, and in the deep convolutional neural networks [120] derived from convolutional neural networks [98]. Deep learning is known to be much more computationally efficient than multi layer perceptrons with multiple hidden units[16], [13] as well as better when the task is complex enough along with lots of data to capture that complexity [95]. In the current work, we use deep-autoencoder, which is a discriminative deep neural network whose output targets are the data input itself rather than the class labels; hence, an unsupervised learning model.

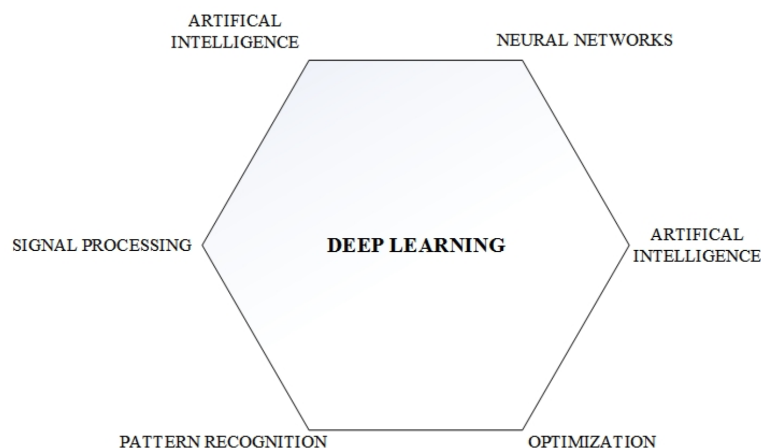

Figure 1.1: Composition of a Deep Learning algorithm

1.1.1 Overview of problem in current methods

Neural networks have seen a re-birth after their increased success in the recent past. Neural networks are particularly interesting due because of the following three reasons. Firstly, universal approximation [74], that implies that neural networks can approximate continuous discriminants arbitrarily well. Secondly, the output of the neural network classifier approximates Bayes discriminants [145]. Lastly, the training algorithms for neural networks make it an effective learning model in a machine learning framework. However, the universal approximation which throws some doubt upon the NFL theorem because the multi layer peceptrons (MLP) can approximate other discriminants. Further doubt about the NFL theorem is provided by the success of deep learners (DL) [97], [54], [157]. In addition, there is little attempts to replace DL classifier layer by any classifier other than softmax classifier. The objective in any classification problem is assign a correct class label to a given input vector. In most cases, each input pattern is assigned to only one class thereby dividing the input space into disjoint decision regions. In functional link neural networks (FLNN) and multilayer perceptrons (MLP), designed through iterative regression [42], [154], it has been shown that the outputs approximate *posterior* class probabilities $\mathrm{P}(\mathrm{i}|\mathbf{x})$, where $\mathbf{x}$ is an input vector [56], [152]. However, the approximations are frequently inaccurate because (1) regression algorithm are sensitive to outliers [150] and (2) real word data is often non Gaussian [17]. Also, in many fields, the classification problems are linear as in credit card fraud detection [46], [126], lung cancer classification [93] [164], and in other bioengineering [87], [1] applications.

In order to solve the problems in regression based linear classifier design, investigators have used L_1 methods and SVM related methods. LIBLINEAR [44] is a multi-class logistic regression and linear-SVM classifier package. It supports both L_1 and L_2 type linear SVMs. It solves an unconstrained

optimization problem using coordinate descent [75]. For multi-class problems, the one-against the rest strategy is implemented. However LIBLINEAR is difficult to modify beyond tuning the free parameters. The sequential dual method (SDM) [86], results in a multi-class classifier that combines all binary models of the one-vs the rest method. Iteratively re-weighted least squares (IRLS) [151] is an iterative method for finding the maximum likelihood estimate of a generalized linear model. Unfortunately, these methods are overly complex and usually work best for small datasets. In addition, these methods do not make use of embedded feature selection.

A serious problem in MLP training is the use of many user-chosen parameters such as initial network weight values and hyperparameters such as the number of training iterations, hidden layers and hidden units. It is a widely known fact that the initial weights have a large effect on network performance [101], [136]. Similarly, tuning the hyperparameters directly effects the model capacity which directly governs the generalization error. Methods such as early stopping [66] and pruning [144] have been used to partially solve this problem. MLP training is hindered by the large number of hidden units and large amount of heuristics that it becomes more of an art than science. MOLF based MLP [113],[112], [143] provides an intermittent solution. Unfortunately, short-term variations in the validation and testing errors are uncorrelated, leading to networks with too many hidden units.

Similarly for a DL training, various techniques exist to prevent AE from learning the identity function and to improve there ability to capture a "good" representation of the input data. [174] introduces denoising autoencoders which takes a partially corrupted input while training to recover the original undistorted input. However, tuning the noise level is cumbersome and their performance is reported on single layer autoencoders. In [21], introduces a form of sparsity regularization on autoencoders that allow sparse representations of inputs. These are useful in pre-training for classification tasks and implemented by adding additional terms in the error criterion during training. However, this increases the number of hyper-parameters to be tuned thereby increasing the user chosen parameters that need to be decided before the model can be put into use.

The goal of the present work is to simultaneously address two distinct questions. *Firstly*, in a learning model how much sample complexity is required, which means how much data is necessary to learn in order to avoid over-fitting and under-fitting ? *Secondly*, in a learning algorithm, how much computation is required for analyzing the data i.e computational complexity ? Representation (or Hypothesis) of the data plays a decisive role in answering above questions and by picking a particular error measure, we pick a final hypothesis that gives us our final learning model [2]. This has been summarized in the learning diagram as shown in Figure 1. Given a set of training examples, our aim is to find a representation $\mathbf{g}$ (or hypothesis) from a set of representations $\mathbf{H}$ that the learning algorithm can lead to minimization of an error function. Together, the representation set and the learning algorithms are called the learning model. Primarily deep learning has been seen as with

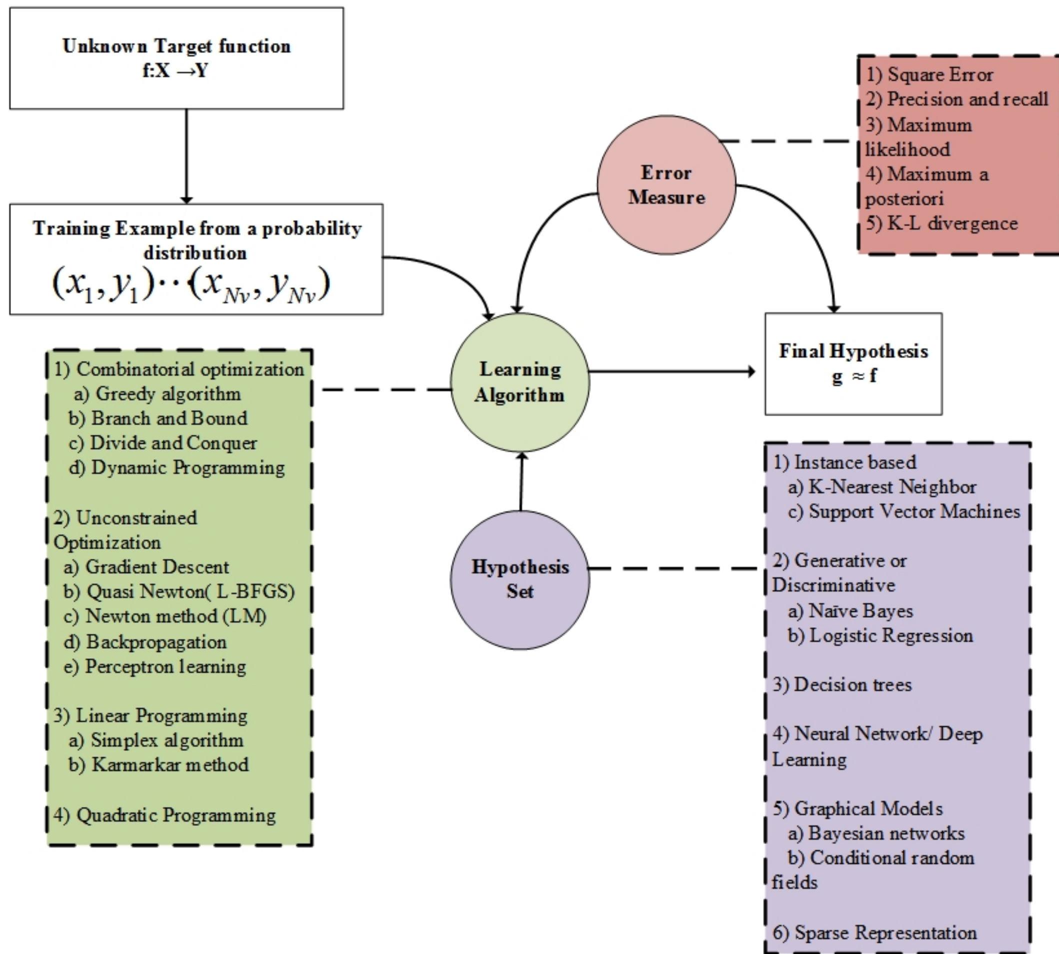

Figure 1.2: A generalized learning paradigm

both software and hardware viewpoint. From software side, mainly L-BFGS [19] or SGD [54] has been used. From the hardware side, GPU processing has been used to tackle the speed problem [23], [4]. We view the training of deep architecture purely from a software view point with a view that desktop level computers can be used to tackle problems with efficient software. Another problem from a software point of view is horizontal and vertical optimization needed in a deep neural networks. In horizontal optimization, the depth of the network is picked up heuristically thereby leaving a scope for automation. Similarly, in a vertical optimization, the number of hidden units needed in each layer is still an open research problem. Contrary to popular choice, we use sigmoidal and means square error as our activation and loss function in the classifiers. A quadratic error measure gives us the freedom to use non-gradient optimization schemes and we use orthogonal least squares at every design stage. Rather than heuristically chosen learning factors for weight updation, we use multiple learning factors that are optimal in least square sense. Optimal selection of the depth of the deep

neural network as well as developing an efficient method to determine the number of hidden units in each layer is also studied in the present proposal.

1.1.2 Proposed work

In this dissertation, firstly we develop an efficient multi-step method for designing generalized linear classifiers in applications that involve medium and high dimensional input data. The simple structure of a linear classifier is augmented by the inclusion of feature selection and methods which limit the effects of outliers. Next we develop an efficient second order multi-step method for designing adaptive MLP classifiers. The resulting classifiers have increased accuracy and are successful in replacing existing classifiers in deep learning models. We also design an efficient deep autoencoders that is computationally fast and lightweight. Chapter II reviews several types of linear classifiers. Chapter III review several types of nonlinear classifiers and problems with current training methods. A second order training algorithm utilizing pruning of hidden units is discussed in chapter III. Deep neural networks along with existing deep models are discussed in chapter IV. The problems with linear, non linear and deep learning models along with goals and proposed task is presented in chapter V. Chapter VI discuss the proposed improvements for linear classifiers. These improvements include a method for removing inconsistent errors that cause the training mean square error and misclassification probability to move in opposite directions. The proposed improvements is similar to Ho-Kashyap [72], [73]. Using a second order algorithm, the linear classifier is converted to a generalized linear classifier (GLC). To further improve performance, Newton's method is used to update the GLC weights. Chapter VII presents the proposed improvements in MLP classifiers. It includes growing and pruning algorithms and second order based input sigmoidal algorithm. A detailed investigation and improvements in deep autoencoders along with a proposed linear probe to determine the depth of deep learner is presented in chapter VIII. Chapter IX mathematically illustrates the linearity in deep learning models. Using several widely available mid-size and large datasets, we compare our proposed algorithms to various other classifiers in chapter X. Chapter XI presents our conclusions and possible enhancements to this work.

Chapter 2

Linear Networks Review

2.1 Review: Linear Networks

2.1.1 Structure and notation

In this chapter, we discuss the structure and notation of linear classifiers. As shown in Fig. 2.1, a linear classifier is a weight matrix $\mathbf{W}$ that transforms an input vector $\mathbf{x}$ into a discriminant vector $\mathbf{y}$ [2]. The weight $w(m,n)$ connects the nth input to the mth output. The training dataset $(\mathbf{x}_p, \mathbf{t}_p)$ consists of N dimensional input vectors $\mathbf{x}_p$ and M dimensional desired output vectors $\mathbf{t}_p$. The pattern number p varies from 1 to N_v, where N_v denotes the number of training patterns. The threshold is handled by augmenting $\mathbf{x}$ with an extra element which is equal to one as $\mathbf{x}_a = [\mathbf{1} : \mathbf{x}^T]^T$. So $\mathbf{x}_a$ contains N_u basis functions, where $N_u = N + 1$. For the pth training pattern, the network output vector, $\mathbf{y}_p$ can be written as

$$\mathbf{y}_p = \mathbf{W} \cdot \mathbf{x}_{ap} \tag{2.1}$$

where $\mathbf{x}_{ap}$ denotes $\mathbf{x}_a$ for the pth pattern.

2.1.2 Regression based classifier

To train the linear classifier, we minimize the error function E that is a surrogate for a non smooth classification error. As in [17], from a Bayesian point of view, we consider maximizing likelihood function or minimizing mean square error (MSE) in a least square sense where the MSE between

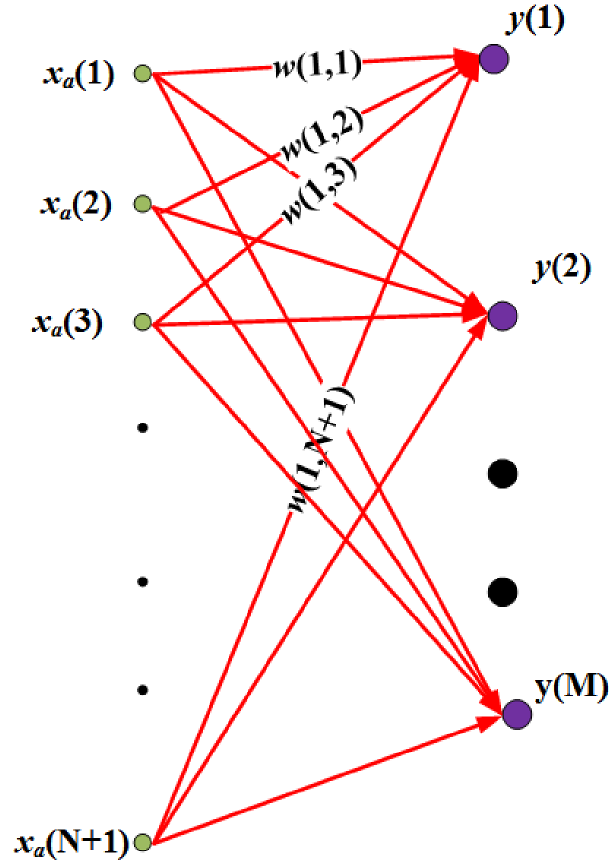

Figure 2.1: Linear classifier

the inputs and the outputs is

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - y_p(i)]^2 \tag{2.2}$$

Here the target output for the correct class i_c and the output for every incorrect class i_d satisfy $t_p(i_c) = 1$ and $t_p(i_d) = 0$. M denotes the number of classes, so our output coding is one-versus-all. We minimize the error function from equation (2.2) with respect to $\mathbf{W}$ by solving the M sets of $N+1$ linear equations given by,

$$\mathbf{C} = \mathbf{R} \cdot \mathbf{W^T} \tag{2.3}$$

where the cross-correlation matrix $\mathbf{C}$ and the auto-correlation matrix $\mathbf{R}$ are respectively

$$\mathbf{C} = \frac{1}{N_v} \sum_{p=1}^{N_v} \mathbf{x}_{ap} \cdot \mathbf{t}_p^T \tag{2.4}$$

$$\mathbf{R} = \frac{1}{N_v} \sum_{p=1}^{N_v} \mathbf{x}_{ap} \cdot \mathbf{x}_{ap}^T \tag{2.5}$$

Since $\mathbf{R}$ in equation (3.5) is often ill-conditioned, it is unsafe to use Gauss-Jordan elimination. Equation (3.5) is solved using orthogonal least squares (OLS) [115]. In [31], OLS is used to solve for radial basis function network parameters. OLS is useful for practical applications for two primary reasons. First, the training is fast, since solving linear equations is straightforward. Second, it helps us to avoid some local minima [118]. In terms of optimization theory, solving equation(3.5) for $\mathbf{W}$ is merely Newton's algorithm for the output weights [148]. Since the linear output activation is considered as a class discriminant, the classifier is said to have correctly classified the pth pattern when $y_p(i_c)$ has the largest output where the integer i_c denotes the correct class. Let $i_c^{'}$ denote the estimated class for a particular pattern where

$$i_c^{'} = \underbrace{argmax}_{i} \, y_p(i) \tag{2.6}$$

If $i_c^{'} = i_c$, we say that the network classified the current pattern correctly. Otherwise, the pattern is misclassified. Note that the errors in regression-based classifiers satisfy $-\infty < y_p(i)$ - $t_p(i) < \infty$, so we can say that these errors are unbounded.

2.1.3 LIBLINEAR

LIBLINEAR [44] is a well known binary and multi-class logistic regression and linear-SVM classifier package. The linear SVM is further split into L_1 and L_2 error functions. In this dissertation, we use the L_2 linear SVM. Given a set of training patterns, for a two class case, LIBLINEAR solves the following unconstrained optimization problem with a loss function as:

$$\underset{\mathbf{w}}{min} \frac{1}{2} \mathbf{w}^T \mathbf{w} + c \sum_{p=1}^{N_v} E_{lin}(p) \tag{2.7}$$

where $c \geq 0$ is a penalty parameter, $\mathbf{w}$ is an N by 1 column vector and E_{lin}(p) = $(max(1$ - $t_p(\mathbf{w}^{'})^T \mathbf{x}_{ap}, 0))^2$ where $\mathbf{w}^{'} = [\mathbf{w}^T : \text{b}]^T$ and b is a bias term. The coordinate descent algorithm [60] is used for training. For multi-class problems, one-vs the rest strategy is implemented.

2.1.4 Sequential dual method

SDM-CS is a multi-class classifier designed using Crammer and Singer's approach [86]. There are M weight vectors $\mathbf{w}_1, \cdots, \mathbf{w}_M$ for M classes. We consider the minimization of the following objective function

$$\min_{\mathbf{w}_1,\cdots\mathbf{w}_M} \frac{1}{2}\sum_{i=1}^{M} \|\mathbf{w}_i\|_2^2 + c\sum_{p=1}^{N_v} E_{sdm}(p) \tag{2.8}$$

where c is the regularization parameter. The loss function for the pth pattern is [86]

$$E_{sdm}(p) = \max_{i \neq t}[max(0, 1 - (\mathbf{w}_t - \mathbf{w}_M)^T \mathbf{x}_{ap})] \tag{2.9}$$

and $\mathbf{w}_t$ and $\mathbf{w}_M$ are the weight vectors for the tth and M^{th} classes respectively that correspond to the tth and M^{th} rows of $\mathbf{W}$.The basic premise of this is to combine all binary models of the one-against the rest method. The estimated class for the pth pattern is

$$i_c^{'}(p) = \underset{i}{argmax}(\mathbf{w}_i^T \mathbf{x}_{ap}) \tag{2.10}$$

2.1.5 Iteratively reweighted least squares

As a generalized linear model, iteratively reweighted least squares (IRLS) [17] is an iterative method which solves the following optimization problem

$$E_{IRLS} = \min_{\mathbf{W}} \frac{1}{N_v}\sum_{p=1}^{N_v} ||\mathbf{t}_p - \mathbf{y}_p||^2 \tag{2.11}$$

where E_{IRLS} is the error function that is minimized. IRLS is used to find the maximum likelihood estimate $\mathbf{W}$ of a generalized linear model. Two advantages of IRLS over the convex programming used in SDM-CS and LIBLINEAR are that it can be used with Levenber Marquardt (LM) [105], [121] or Newton numerical algorithms [177] which exhibit quadratic convergence [111], [3] and that it's relatively easy to write the software.

2.1.6 Sigmoidal SVM

SVM's as discussed in subsection 2.1.3, are not calibrated for posterior probabilities. For N_v training examples $\mathbf{x}_p$, $p = 1 \cdots N_v$, labeled by $t_p \in \{+1, -1\}$, the binary SVM computes a decision function $f(\mathbf{x})$ such that $sign(f(\mathbf{x}))$ is used to predict the label of any test example $\mathbf{x}$. In order to derive the posterior probabilities $Pr(y = 1|\mathbf{x})$ from the SVM output, sigmoidal functions are added to the

output layer of the SVM [138]

$$Pr(y = 1|\mathbf{x}) \approx \mathbf{s}_p \equiv \frac{1}{1 + exp(A \cdot f_p + B)} \tag{2.12}$$

for $f_p = f(\mathbf{x}_p)$. Here A and B are the scaling and shifting parameters respectively. If f_p denotes $f(\mathbf{x}_p)$ then the best parameter setting $z^* = (A^*, B^*)$ is determined by solving the following cross entropy error equation.

$$\min_{z=(A,B)} E_{sig-svm}(z) = -\sum_{p=1}^{N_v} y_p log(s_p) + (1 - y_p) log(1 - s_p) \tag{2.13}$$

where

$$y_p = \begin{cases} \frac{N_v^+ + 1}{N_v^+ + 2}, & t_p = +1 \\ \frac{1}{N_v^- + 2}, & t_p = -1 \end{cases} \tag{2.14}$$

Since equation (2.13) is a two parameter minimization, [138] uses the model-trust minimization algorithm [49].

Chapter 3

Review: MLP with single hidden layer

3.1 Review: MLP with single hidden layer

In the present chapter, we start by describing the multi layer perceptron (MLP), which is a non linear signal processor that has good approximation and classification properties. The MLP has basis functions that can adapt during the training process by utilizing example input and desired outputs. An MLP will minimize an error criterion and closely mimic an optimal processor in which the computational burden in processing an input vector is controlled by slowly varying the number of coefficients [117] [66]. We review the first and second order training algorithms for MLP followed by a classifier design of MLP through regression.

3.1.1 Structure and Notation

Figure 3.1 illustrates a single layer fully connected MLP. The input weights $w(k, n)$ connect the nth input to the kth hidden unit. Output weights $w_{oh}(m, k)$ connect the kth hidden unit's non-linear activation $O_p(k)$ to the mth output $y_p(m)$, which has a linear activation. The bypass weights $w_{oi}(m, n)$ connects the nth input to the mth output. The training data, described by the set of independent, identically distributed input-output pair $\{\mathbf{x_p}, \mathbf{t_p}\}$ consists of N dimensional input vectors $\mathbf{x_p}$ and M dimensional desired output vectors, $\mathbf{t_p}$. The pattern number p varies from 1 to N_v, where N_v denotes the number of training vectors present in the datasets. Let N_h denote the number of hidden units. In order to handle the thresholds in the input layer, the input unit is

augmented by an extra element $x_p(N+1)$, where $x_p(N+1) = 1$. For each training pattern p, the hidden layer net function vector $\mathbf{n_p}$ can be written as

$$\mathbf{n_p} = \mathbf{W} \cdot \mathbf{x_p} \tag{3.1}$$

The kth element of the hidden unit activation vector $\mathbf{O_p}$ is calculated as $O_p(k) = f(n_p(k))$ where $f(\cdot)$ denotes the sigmoid activation function. The network output vector, $\mathbf{y_p}$ can be written as

$$\mathbf{y_p} = \mathbf{W_{oi}} \cdot \mathbf{x_p} + \mathbf{W_{oh}} \cdot \mathbf{O_p} \tag{3.2}$$

The expression for the actual outputs given in equation (3.2) can be re-written as

$$\mathbf{y_p} = \mathbf{W_o} \cdot \mathbf{X_a} \tag{3.3}$$

where $\mathbf{X_a} = [\mathbf{x_p}^T : \mathbf{O_p}^T]^T$ is the augmented input column vector with N_u basis functions, where $N_u = 1 + N + N_h$. The total number of weights $N_w = N_h \cdot (1 + N) + M \cdot N_h$. Similarly, $\mathbf{W_o}$ is the M *by* N_u dimensional augmented weight matrix defined as $\mathbf{W_o} = [\mathbf{W_{oh}}, : \mathbf{W_{oi}}]$.

To train an MLP, we re-cast the MLP learning problem as an optimization problem and use an structural risk minimization framework to design the learning algorithm [17] [95]. Essentially, this framework is used to minimize the error function E as in equation 2.2 that is a surrogate for a non smooth classification error. As in [17], from a Bayesian point of view, we consider maximizing liklihood function or minimizing mean square error (MSE) in a least square sense. Therefore, the MSE between the inputs and the outputs is defined as:

$$E^{'} = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{m=1}^{M} [t_p(m) - y_p(m)]^2 + \lambda \cdot \|\mathbf{W_o}\|^2 \tag{3.4}$$

Here, λ is an L_2 regularization parameter that is used to avoid memorization and over-fitting. The nonlinearity in $\mathbf{y_p}$ causes the error E to be non-convex, and so in practice, local minima of the error function may be found. In the above discussion, we have assumed that $\mathbf{t_p}$ has a Gaussian distribution with input $\mathbf{x_p}$. In case the conditional distribution of targets, given input has a Bernoulli distribution, the error function, which is given by the negative log liklihood, is then a cross entropy error function [17].

In [17] it is concluded that using a cross-entropy error function instead of the mean square error for a classification problem leads to a faster training as well as improved generalization. Apart from cross entropy and L_2 error form we also have a L_1 error measure. [52] and [163] discuss a good comparison between the L_2 and cross entropy and suggests using cross entropy error function for classification in order to have faster training and improved generalization. Our goal is to obtain an

optimal value of the weights connected in an MLP. In order to achieve this, we use empirical risk minimization [66] framework to design the learning algorithms. An important benefit of converting the training of an MLP into an optimization problem is that we can now use a variety of optimizing algorithms to optimize the learning of an MLP.

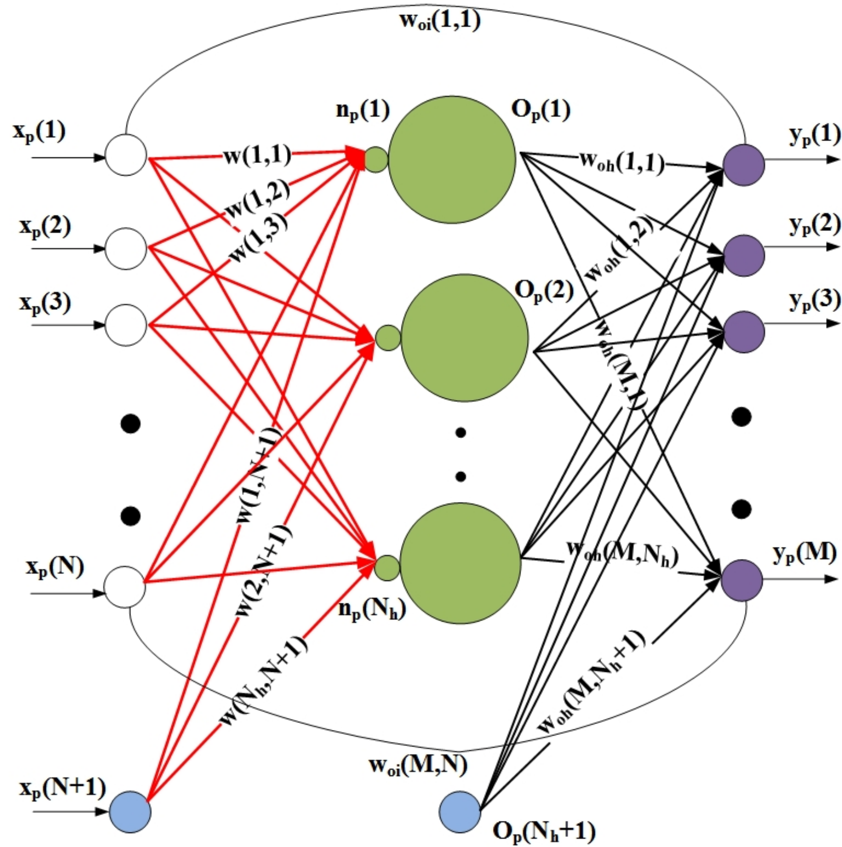

Figure 3.1: Fully connected MLP

3.1.2 Initialization

As from [117], the input weights matrix $\mathbf{W}$ is initialized randomly from a zero mean Gaussian random number generator. The training of the input weights, strongly depends on the gradient of the hidden units activation functions with respect to the inputs. Training of input weights will cease if the hidden units it feeds into has an activation function derivative of zero for all patterns. In order to remove the dominance of a large variance inputs, we divide the input weights by the input's standard deviation. Therefore we adjust the mean and standard deviation of all the hidden units net functions. This is called net control as in [135]. At this point, we have determined the initial input

weights and we are now ready to initialize the output weights. To solve for the weights connected to the output of the network, we use a technique called output weight optimization (OWO) [119], [155]. OWO minimizes the error function from equation (2.2) with respect to $\mathbf{W_o}$ by solving the M sets of N_u equations given by,

$$\mathbf{C} = \mathbf{R} \cdot \mathbf{W_o^T} \tag{3.5}$$

Here the cross-correlation matrix $\mathbf{C}$, auto-correlation matrix $\mathbf{R}$

$$\mathbf{C} = \frac{1}{N_v}\sum_{p=1}^{N_v} \mathbf{t_p} \cdot \mathbf{X_a^T} \qquad \text{and} \qquad \mathbf{R} = \frac{1}{N_v}\sum_{p=1}^{N_v} \mathbf{X_a} \cdot \mathbf{X_a^T} \tag{3.6}$$

In order to incorporate the regularization, we modify the $\mathbf{R}$ matrix elements except the threshold as

$$\mathbf{R} \leftarrow \mathbf{R} + \lambda \cdot diag(\mathbf{r}) \tag{3.7}$$

where $\mathbf{r}$ is a vector containing the diagonal elements of $\mathbf{R}$ and $diag()$ is an operator that creates a diagonal matrix from the vector.

The MLP network is now initialized and ready to be trained with first or second order algorithms. Training an MLP can be seen as an unconstrained optimization problem that usually involves first order gradient methods such as backpropagation (BP), conjugate gradient (CG) and second order Levenberg-Marquardt (LM), Newton's method as the most popular learning algorithm. Training algorithms can be classified as

1. One Stage, in which all the weights of the network are updated simultaneously and
2. Two Stage, in which input and output weights are trained alternately.

Figure 3.2 shows a flowchart that summarizes all the training algorithms that will be described in subsequent sections. The two-stage algorithms namely *output weight optimization- hidden weight optimization* (OWO-HWO) and *output weight optimization- multiple optimal learning factors* (OWO-MOLF) will serve as basic building algorithms that will be further refined in this dissertation.

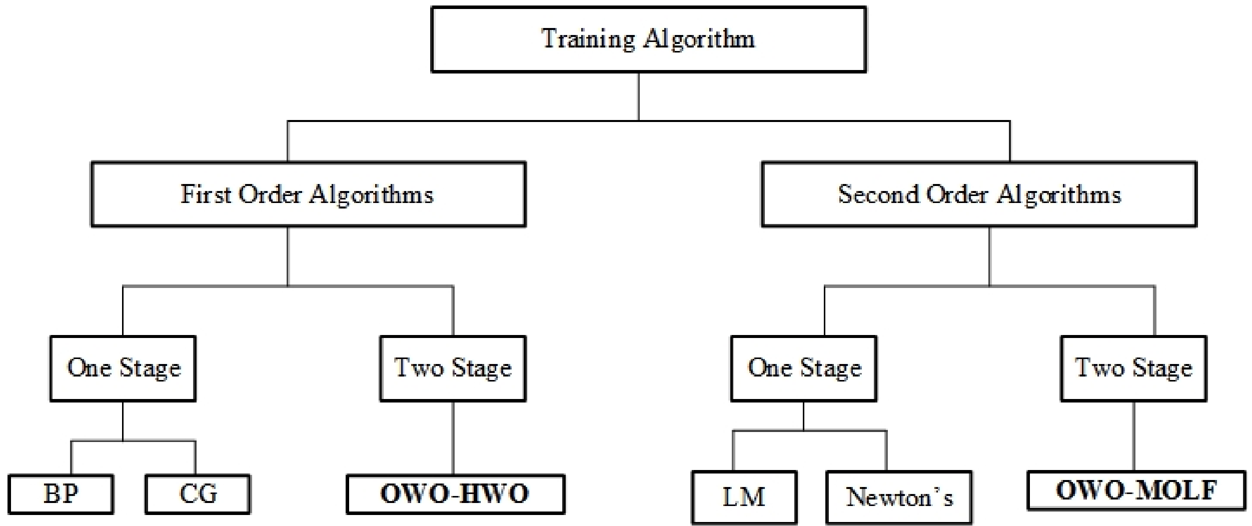

Figure 3.2: Typical training algorithms for training an MLP

3.1.3 First order learning algorithms

The first order learning algorithms update the weights of the MLP based on gradient matrices i.e. the first order information, hence the name. We start by discussing the training of an MLP with a one stage algorithm. In this, we train both the output and input weights simultaneously using either backpropagation or conjugate gradient algorithm. We then describe a two stage algorithm called output weight optimization-hidden weight optimization.

Backpropagation algorithm

The backpropagation (BP) algorithm is a greedy line search algorithms that has a step size to achieve the maximum amount of decrease of the objective function at each individual step [17]. Backpropagation is a computationally efficient method in conjunction with gradient based algorithms that is used widely to train an MLP [153]. However due to non-convexity of error function equation 2.2 in neural networks, backpropagation is not guaranteed to find global minima but rather only local minima. Although this is considered as a major drawback, recently in [34] it is discussed as to why local minima is still useful in many practical problems. In each training epoch, we update all the weights of the network in a backpropagtion algorithm as follows:

$$\mathbf{w} \leftarrow \mathbf{w} + z \cdot \mathbf{g} \tag{3.8}$$

Here $\mathbf{w}$ is a vector of network weights as $\mathbf{w} = \text{vec}(\mathbf{W}, \mathbf{W_{oh}}, \mathbf{W_{oi}})$ and $\mathbf{g}$ is a vector of network gradients $\mathbf{g} = \text{vec}(\mathbf{G}, \mathbf{G_{oh}}, \mathbf{G_{oi}})$. The gradient matrices are negative partial of E wrt the weights, $\mathbf{G} = -\frac{\partial E}{\partial W}$, $\mathbf{G_{oh}} = -\frac{\partial E}{\partial W_{oh}}$ and $\mathbf{G_{oi}} = -\frac{\partial E}{\partial W_{oi}}$. $vec()$ operator performs a lexicographic ordering of a matrix into a vector. z is the optimal learning factor that is derived using a Taylor series expansion of the mean square error E, expressed in terms of z, as [101]

$$z = -\frac{\frac{\partial E}{\partial z}}{\frac{\partial^2 E}{\partial z^2}} \tag{3.9}$$

The backpropagation algorithm can be summarized as follows:

Algorithm 1 Backpropagation algorithm

1: Initialize $\mathbf{w}$, N_{it} , it$\leftarrow 0$
2: **while** it $< N_{it}$ **do**
3: Calculate $\mathbf{g}$
4: Compute z from equation (3.9)
5: Update $\mathbf{w}$ as $\mathbf{w} \leftarrow \mathbf{w} + \text{z} \cdot \mathbf{g}$
6: it $\leftarrow$ it $+ 1$
7: **end while**

As in [17], backpropagation algorithm has two major criticism. First, it does not scale well, i.e. it takes $O(N_w^2)$ operations for sufficiently large N_w and second, being a simple gradient descent procedure, it's unduly slow in the presence of flat error surfaces and is not a very reliable learning paradigm.

Conjugate Gradient algorithms

As we see from the previous section, in a basic gradient algorithm, the weights are updated in the negative gradient direction. Although the error function reduces most rapidly along the negative direction of the gradient, it does not necessarily create fast convergence. Conjugate gradient algorithm [67] performs a line-search in the conjugate direction and has faster convergence than backpropagation algorithm. Although conjugate gradient is a general unconstrained optimization technique, its use in efficiently training an MLP is well documented in [26].

To train an MLP using conjugate gradient algorithm, we use a direction vector that is obtained from the gradient $\mathbf{g}$ as

$$\mathbf{p} \leftarrow -\mathbf{g} + B_1 \cdot \mathbf{p} \tag{3.10}$$

Here $\mathbf{p} = \text{vec}(\mathbf{P}, \mathbf{P_{oh}}, \mathbf{P_{oi}})$ and $\mathbf{P}$, $\mathbf{P_{oi}}$ and $\mathbf{P_{oh}}$ are the direction vectors. B_1 is the ratio of the

gradient energy from two consecutive iterations. This direction vector, in turn, update all the weights simultaneously as follows:

$$\mathbf{w} \leftarrow \mathbf{w} + z \cdot \mathbf{p} \tag{3.11}$$

Conjugate gradient algorithm has many attractive qualities such as the following:

1. The number of iterations it takes to converge is equal to the number of unknowns.
2. It performs better than steepest descent and it can be applied to nonquadratic error functions.
3. Since there is no Hessian involved, we don't invert any matrix and the computational cost is O($\mathbf{w}$), where $\mathbf{w}$ is the size of the weight vector.

The conjugate gradient algorithm can be summarized as follows:

Algorithm 2 Conjugate gradient algorithm

1: Initialize $\mathbf{w}$, N_{it} , it$\leftarrow$ 0
2: **while** it $< N_{it}$ **do**
3: Calculate $\mathbf{p}$ from $\mathbf{g}$
4: Compute z from equation (3.9)
5: Update $\mathbf{w}$ as $\mathbf{w} \leftarrow \mathbf{w}$ + z $\cdot$ $\mathbf{p}$
6: it $\leftarrow$ it + 1
7: **end while**

Output weight optimization-hidden weight optimization algorithm

OWO-HWO algorithm is a two-step algorithm which alternately train the output and input weights. During the first half of the training epoch, we update the input weights $\mathbf{W}$ using hidden weight optimization (HWO) [181]. HWO finds an improved gradient matrix as $\mathbf{G_{hwo}}$ by solving the following linear equation.

$$\mathbf{G_{hwo}} \cdot \mathbf{R_i} = \mathbf{G} \tag{3.12}$$

where $\mathbf{R_i}$ is the input autocorrelation matrix defined as

$$\mathbf{R_i} = \frac{1}{N_v} \sum_{p=1}^{N_v} \mathbf{x_p} \cdot \mathbf{x_p^T} \tag{3.13}$$

In a separate result in [149], it is shown that applying OWO-HWO algorithm, is equivalent to applying OWO-BP to whitening transformed input data. Re-writing equation (3.12) as

$$\mathbf{G_{hwo}} = \mathbf{G} \cdot \mathbf{R_i^{-1}} \tag{3.14}$$

It can be shown that

$$\mathbf{G_{hwo}} = \mathbf{G} \cdot \mathbf{A^T} \cdot \mathbf{A} \tag{3.15}$$

where $\mathbf{A}$ is a whitening transform matrix [149]. Therefore by using OWO-HWO, we inherently transform our data to be de-correlated.
In order to update the input weight matrix, we then use $\mathbf{G_{hwo}}$ instead of $\mathbf{G}$ as

$$\mathbf{W} \leftarrow \mathbf{W} + z \cdot \mathbf{G_{hwo}} \tag{3.16}$$

In the second half of a training epoch, we use OWO to update the output weights by solving equation ()3.5 through OLS. The OWO-HWO algorithm is faster than one stage BP or CG algorithms because training the output weights is equivalent to solving the linear equations. The OWO-HWO algorithm can be summarized as follows:

Algorithm 3 OWO-HWO algorithm

1: Initialize $\mathbf{W}, \mathbf{W_{oi}}, \mathbf{W_{oh}}$, N_{it} , it$\leftarrow$ 0
2: **while** it $< N_{it}$ **do**
3: **HWO step** : Calculate $\mathbf{G_{hwo}}$ using equation (3.12).
4: Compute z from equation (3.9)
5: Update $\mathbf{W}$ as $\leftarrow \mathbf{W}$ + z $\cdot$ $\mathbf{G_{hwo}}$
6: **OWO step** : Solve equation (3.5) to obtain $\mathbf{W_o}$
7: it $\leftarrow$ it + 1
8: **end while**

3.1.4 Softmax classifier

The softmax classifier [17] is a generalized logistic regression classifier that outputs approximate class probabilities. Structurally, it's a linear model with softmax function [42] at the output units. For the pth pattern, it maps the input vector $\mathbf{x}_p$ to the output class labels as

$$\mathbf{y}_p = \mathbf{W} \cdot \mathbf{x}_p \tag{3.17}$$

where $\mathbf{W}$ is a weight matrix. The performance measure is a cross-entropy loss function [17] as

$$E_{softmax} = -\frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} log(\frac{e^{y_p(i)}}{\sum_{j=1}^{M} e^{y_p(j)}}) \tag{3.18}$$

The softmax classifier is often trained using L-BFGS training algorithm[183].

3.1.5 Radial basis function classifier

A second order feed forward training algorithm for a radial basis function (RBF) classifier is described in [171]. Using a weighted measure $d(\mathbf{x}_p, \mathbf{m}_k) = \sum_{n=1}^{N+1} c(n)\ (x_p(n) - m_k(n))^2$ between K cluster centers and input patterns, the RBF basis functions are given as

$$\mathbf{X}_p(k) = e^{-\beta_k \cdot d(\mathbf{x}_p, \mathbf{m}_k)} \quad ; \quad k = [1, \cdots K]. \tag{3.19}$$

Modifying equation (2.2), the RBF objective function is

$$E_{rbf}(\mathbf{W}_o) = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - y_p(i)]^2 \tag{3.20}$$

where $y_p(i)$ is the ith element of the output discriminant vector

$$\mathbf{y}_p = \mathbf{W}_o \cdot \mathbf{X}_p \tag{3.21}$$

Equating the gradient of $E_{rbf}(\mathbf{W}_o)$ to zero, we solve for the output weight matrix $\mathbf{W}_o$ using OWO [171]. Let $\mathbf{m}_k$, β_k and $c(n)$ denote the kth center vector, the kth spread parameter and the weighted euclidean distance measure weight for the nth input. These parameters are optimized using an efficient Newton's algorithm [171].

3.1.6 Nonlinear SVM classifier

A nonlinear support vector machine (SVM) follows the structure similar to the RBF classifier. In a nonlinear SVM, $K(\mathbf{X}_p,\, \mathbf{X}_q) \equiv \phi(\mathbf{X}_p)^T\ \phi(\mathbf{X}_q)$ is called the kernel function and $\phi(\mathbf{X}_q)$ is a feature function. For our comparisons $K(\mathbf{X}_p,\, \mathbf{X}_q) = exp(-\gamma||\mathbf{x}_p - \mathbf{x}_q||^2)$, $\gamma > 0$. For a given two class classification problem, a nonlinear SVM solves the following convex optimization problem

$$E_{svm} = \min_{\mathbf{W}_o, b, \xi} \frac{1}{2} \mathbf{W}_o^T \mathbf{W}_o + C \sum_{p=1}^{N_{sv}} \xi_p \tag{3.22}$$

subject to $y_p(\mathbf{W}_o^T \phi(X_p) + b) \geq 1 - \xi_i$, $\xi_i \geq 0$. Here the weight vector $\mathbf{W}_o$ is N_{sv} by 1 and b is a bias. The basis vector $\mathbf{X_p}$ is N_{sv} by 1, and $1 \leq p \leq N_{sv}$. C is a user-specified positive parameter, and the ξ_p are called the slack variables. We need to find the optimum values of the weight vector $\mathbf{W}_o$, bias b and the slack variables ξ_p to minimize E_{svm}. The coordinated descent algorithm [61] is used for training. For multi-class problems, one-vs the rest strategy is implemented.

3.1.7 Second order learning algorithms

The basic idea behind using a second order method is to improve the first order algorithms by using the second derivative along with the first derivative [17]. We present two, one stage, algorithm, namely Newton's method and Levenberg-Marquardt (LM) and then a two stage algorithm called Output weight optimization-multiple optimal learning factor (OWO-MOLF).

Newton's method

For Newton's method, given a starting point, we construct a quadratic approximation to a double differentiable error function that matches the first and second order derivative value at that point. We then minimize this quadratic function instead of the original error function by expanding the Taylor series of $E^{'}$ about the point w_k as is clear from the equation below :

$$E^{'} \approx E + (\mathbf{w} - \hat{\mathbf{w}}))^T\mathbf{g} + \frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^T\mathbf{H}(\mathbf{w} - \hat{\mathbf{w}}) \tag{3.23}$$

Here $\hat{\mathbf{w}}$ is the new version of $\mathbf{w}$ that we're trying to find. We calculate the Hessian and gradient, $\mathbf{H}$ and $\mathbf{g}$ of the MSE error function where the elements of $\mathbf{H}$ are given by

$$h(m,n) = \frac{\partial^2 E}{\partial w(m) \partial w(n)} \tag{3.24}$$

and the elements of $\mathbf{g}$ are given by

$$g(n) = \frac{\partial E}{\partial w(n)} \tag{3.25}$$

We calculate the second order direction, $\mathbf{d}$, by solving the set of linear equations with OLS

$$\mathbf{H} \cdot \mathbf{d} = \mathbf{g} \tag{3.26}$$

Assuming quadratic error function in equation (2.2) and $\mathbf{H}$ to be positive definite, applying first order necessary condition (FONC) [45], on all the weights in an MLP, we update the weights as,

$$\hat{\mathbf{w}} = \mathbf{w} + \mathbf{d} \tag{3.27}$$

The Newton's algorithm can be summarized as follows:

Newton's method is quadratic convergent and affine invariant [117]. Since it converges fast, we would like to use it to train an MLP, but generally the Hessian $\mathbf{H}$ is singular [177].

Algorithm 4 Newton's algorithm

1: Initialize $\mathbf{w}$, N_{it} , it$\leftarrow 0$
2: **while** it $< N_{it}$ **do**
3: Calculate $\mathbf{g}$ and $\mathbf{H}$ from equation (3.25) and equation (3.24).
4: Compute $\mathbf{d}$ from equation (3.26).
5: Update $\mathbf{w}$ as $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{d}$
6: it $\leftarrow$ it $+ 1$
7: **end while**

If the error function is quadratic, then the approximation is exactly a one step solution; otherwise the approximation will provide only an estimate to the exact solution. In case of a non-quadratic error measure, we will require a line search and $\mathbf{w}$ is updated as

$$\hat{\mathbf{w}} = \mathbf{w} + z \cdot \mathbf{d} \tag{3.28}$$

Levenberg-Marquardt algorithm

The LM algorithm is a compromise between Newton's method, which converges rapidly near local or global minima but may diverge, and gradient descent, which has assured convergence through a proper selection of step size parameter but converge slowly. Following equation (3.23), the LM algorithm is a sub-optimal method. Since usually $\mathbf{H}$ is singular in Newton's method, an alternate is to modify the Hessian matrix as in LM [105] algorithm or use a two step method such as layer by layer training [104]. In LM, we modify the Hessian as

$$\mathbf{H_{LM}} = \mathbf{H} + \lambda \cdot \mathbf{I} \tag{3.29}$$

Here $\mathbf{I}$ is the identity matrix of the same dimensions as $\mathbf{H}$ and λ is a regularizing parameter that forces the sum matrix $(\mathbf{H} + \lambda \cdot \mathbf{I})$ to be positive definite and safely well conditioned throughout the computation. We calculate the second order direction, $\mathbf{d}$, similar to Newton's method as

$$\mathbf{H_{LM}} \cdot \mathbf{d} = \mathbf{g} \tag{3.30}$$

After obtaining $\mathbf{H_{LM}}$, weights of the model are updated using equation (3.27).

The regularizing parameter λ plays a crucial role in the way the LM algorithm functions. If we set λ equal to zero, then the equation (3.30) reduces to Newton's method equation (3.27). On the other hand, if we assign a large value to λ such that $\lambda \cdot \mathbf{I}$ overpowers the Hessian $\mathbf{H}$, the LM algorithms is effective as a gradient descent algorithm. [140] recommends an excellent *Marquardt recipie* for the selection of λ.

From a practical perspective, the computational complexity of obtaining $\mathbf{H_{LM}}$ can be demanding, particularly when the dimensionality of the weight vector $\mathbf{w}$ is high. Therefore, due to scalability constraints, LM is particularly suitable for a small network.

The LM algorithm can be summarized as follows:

Algorithm 5 LM algorithm

1: Initialize $\mathbf{w}$, N_{it} , it$\leftarrow$ 0
2: **while** it $< N_{it}$ **do**
3: Present all patterns to the network to computer error E_{old} from equation (2.2)
4: Calculate $\mathbf{g}$ and $\mathbf{H}$ from equation (3.25) and equation (3.24)
5: Obtain $\mathbf{H_{LM}}$ from equation (3.29)
6: Compute $\mathbf{d}$ from equation (3.30)
7: Update $\mathbf{w}$ as $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{d}$
8: Re-compute the error E_{new} by using the updated weights.
9: **if** $E_{new} < E_{old}$ **then**
10: Reduce the value of λ
11: **goto** step 3
12: **else**
13: Increase the value of λ
14: **end if**
15: it $\leftarrow$ it + 1
16: **end while**

Output weight optimization-multiple optimal learning factor algorithm

As mentioned in subsection 3.1.7, an alternate approach apart from LM and Newton's method is to try a "layer by layer" or two stage approach . We describe one such algorithm called OWO-MOLF algorithm. It's a two-stage algorithm that uses OWO to solve the output weights and uses a multiple optimal learning factors (MOLF) for every hidden unit [143] in order to increase the speed of learning and overall convergence. In the past, researchers have used multiple learning rates or momentum terms in order to speed up the learning process [66]. Unfortunately, these methods are mostly heuristics [78], and their performance relies on the settings of some user chosen parameters. In OWO-MOLF algorithm the key idea is that while updating the input weight matrices for each epoch, instead of using a single optimal learning factor z, we use Newton's method to estimate a vector $\mathbf{z}$ of length N_h. It is obtained by solving the following equation through OLS.

$$\mathbf{H_{molf}} \cdot \mathbf{z} = \mathbf{g_{molf}} \tag{3.31}$$

$\mathbf{H_{molf}}$ and $\mathbf{g_{molf}}$ are the Hessian and *negative* gradient, respectively, of the error with respect to $\mathbf{z}$. During the first half of the training, we obtain the gradient matrix $\mathbf{G}$ and MOLF $\mathbf{z}$ to update the

input weight matrix $\mathbf{W}$ as

$$\mathbf{W} \leftarrow \mathbf{W} + diag(\mathbf{z}) \cdot \mathbf{G} \tag{3.32}$$

In the second half of the training, OWO updates the output weights $\mathbf{W_o}$ by solving the linear equation ()3.5. Note here that $\mathbf{G}$ can be replaced with $\mathbf{G_{hwo}}$ in equation ()3.32 in order to incorporate the hidden weight optimization obtained from equation ()3.12

The OWO-MOLF algorithm can be summarized as follows:

Algorithm 6 OWO-MOLF algorithm

1: Initialize $\mathbf{W}, \mathbf{W_{oi}}, \mathbf{W_{oh}}$, N_{it} , it$\leftarrow$ 0
2: **while** it $< N_{it}$ **do**
3:     Compute $\mathbf{G}$
4:     **MOLF step** : Solve equation (7.8) for $\mathbf{z}$ using OLS
5:     Update $\mathbf{W}$ as $\mathbf{W} \leftarrow \mathbf{W}$ + diag($\mathbf{z}$) $\cdot$ $\mathbf{G}$
6:     **OWO step** : Solve equation (3.5) to obtain $\mathbf{W_o}$
7:     it $\leftarrow$ it + 1
8: **end while**

Chapter 4

Review: Deep neural networks

4.1 Review: Deep neural networks

In chapter 3.1, we discussed the single hidden layer MLP along with first and second order training algorithms. In this chapter, we introduce autoencoders, which are essentially single hidden layer MLP's, often used to learn effective encoding of the original data. The autoencoders are then used to train deep neural networks by stack several layers of it. Therefore, training a deep neural network is essentially training a multi hidden layer MLP. A typical deep neural network has two major parts, an unsupervised learning based feature extraction and a supervised classification of the generated features. We examine both the parts here and end the chapter by covering two variants of deep autoencoders that will be later used for comparisons with our proposed algorithm.

4.1.1 Auto-Encoders: Structure, Notations, and Training

We first describe an auto encoder (AE) [22], [71] as shown in figure 4.1 that has the same structure as that of an fully connected regression type MLP as shown in figure 3.1 except that the bypass weights $\mathbf{W_{oi}}$ are removed. In an autoencoder framework, an input layer represents the original data, the hidden layer represents the transformed features and the output layer matches the input layer for reconstruction. The dimension of the hidden layer can be either

1. smaller than the input layer, i.e under-complete. It's used when the goal is feature compression.
2. larger than the input layer i.e over-complete. It's used when the goal is mapping the input to a higher-dimensional space.

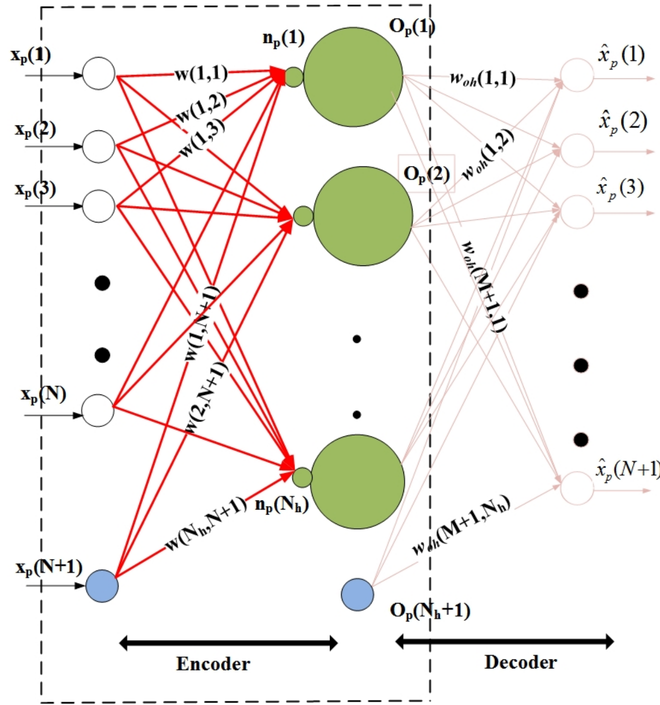

Figure 4.1: Cascade connected autoencoder

Given the input vector $\mathbf{x_p}$, the feature-extracting function called *encoder* allow a straightforward and efficient computation of a feature vector $\mathbf{O_p}$. Here $\mathbf{O_p}$ is the output of the hidden units as in subsection 3.1.1. The hidden to output layer is called *decoder* that maps from feature space back into the input space, producing a reconstruction vector $\widehat{\mathbf{x}}_\mathbf{p}$. Training an AE involves minimizing the mean squared reconstruction error between the inputs $\mathbf{x_p}$ and the reconstructed inputs $\widehat{\mathbf{x}}_\mathbf{p}$. From equation ()2.2, replacing $\mathbf{t_p}$ with $\widehat{\mathbf{x}}_\mathbf{p}$, we get

$$E = \frac{1}{N_v} \sum_{p=1}^{Nv} \sum_{i=1}^{M} [x_p(i) - \widehat{x}_p(i)]^2 + \lambda \cdot \|\mathbf{W_{oh}}\|^2 \tag{4.1}$$

The criterion that representation $\mathbf{O_p}$ should retain information about $\mathbf{x_p}$ is not self-sufficient to yield a useful representation. Indeed mutual information can be trivially maximized by setting $\mathbf{O_p} = \mathbf{x_p}$ as in [13]. Similarly, an autoencoder where $\mathbf{O_p}$ is of the same dimensionality as $\mathbf{x_p}$ (or larger) can achieve perfect reconstruction simply by learning an identity mapping. Without any other constraints, this criterion alone is unlikely to lead to the discovery of a more useful representation than the input. Thus further constraints need to be applied to attempt to separate useful information. This will naturally translate to non-zero reconstruction error. The traditional

approach to autoencoder uses a under-complete representation where $N_h <$ N. The resulting low-dimensional hidden units features can thus be seen as a lossy compresses representation of the input. Since the autoencoder were primarily developed as MLP's predicting their input, the most commonly used forms of the encoder and decoder are affine mapping, followed by a non-linearity.

$$\begin{aligned} \mathbf{O_p} &= f(\mathbf{W} \cdot \mathbf{x_p}) \\ \mathbf{y_p} &= g(\mathbf{W_{oh}} \cdot \mathbf{O_p}) \end{aligned} \tag{4.2}$$

Here $f(\cdot)$ and $g(\cdot)$ are the encoder and decoder activation function. When using affine encoder and decoder without any nonlinearity and a squared error function, the autoencoder essentially performs a PCA as shown in [9]. This is also true when using a sigmoid non-linearity in the encoder [22], since its possible to stay in the linear regime of the sigmoid, but arguably not the only one [79]. Also, when using the cross-entropy cost function rather than a squared error the optimization objective is no longer same as that of PCA and will likely learn different features.

To summarize, in order to capture the structure of the distribution that generated the input data, it is important to modify the training criterion so as to prevent the autoencoder from learning the identity function, which has zero reconstruction error or avoids it to behave like a PCA. This is done through many variations in autoencoder such as tying the weights together [69], using regularized autoencoder [21], sparse autoencoder [120], denoising autoencoder [174] and contractive autoencoder [147].

4.1.2 Deep autoencoder: Structure and Notation

This section describes the structure and notation for a deep autoencoder. Consider an MLP with L hidden layers. A deep autoencoder contains an input layer and an output layer, separated by L layers of hidden units. Given an input pattern $\mathbf{x_p}$, the value of hidden units activation in the lth layer is denoted by ${\mathbf{O_p}}^l$, with $l = 0$ referring to the input layer and $l = L{+}1$ refer to the output layer. We refer to the size of a layer as ${\mathbf{N_h}}^l$. The set of weights $\mathbf{W}^l$ between ${\mathbf{O_p}}^{l-1}$ in the layer $l-1$ and unit ${\mathbf{O_p}}^l$ in layer l determines the activation of units ${\mathbf{O_p}}^l$ as follows:

$${\mathbf{O_p}}^l = f(\mathbf{W}^l \cdot {\mathbf{O_p}}^{l-1}) \tag{4.3}$$

Here ${\mathbf{O_p}}^0 = \mathbf{x_p}$ and $f(\cdot)$ is the sigmoidal function. Given the last hidden layer, the output layer is computed similarly by

$$\mathbf{y_p} = {\mathbf{O_p}}^{L+1} = g(\mathbf{W}^{L+1} \cdot {\mathbf{O_p}}^l) \tag{4.4}$$

Typically, $g(\cdot)$ will be the identity function for a regression problem and the softmax function for a classification problem [94]. As from figure 4.2, when an input sample $\mathbf{x_p}$ is presented to the network,

the application of equation ()4.3 at each layer will generate a pattern of activity in the different layers of MLP. Intuitively, we would like the activity of the first layer neuron to correspond to low level features of the input and to the higher level of abstractions in the last hidden layer.

4.1.3 Deep autoencoder:pre-training and stacking

It has been shown in [74], that a single hidden layer MLP with only one arbitrarily large hidden layer could approximate a function to any level of precision (also add from haykin end of chapter stuff). However, single hidden layer MLP are very inefficient in terms of a number of computational units and thus in terms of training examples required [15],[16]. A deep neural network with more than one hidden layer is difficult to train if we frame the training as a purely supervised learning optimization problem. Unsupervised learning is a promising paradigm for greedy layer-wise training. Therefore, as in [94],the deep network learning algorithms combine the idea of greedily learning the network by breaking down the learning problem into easier steps and then using unsupervised learning tp provide an effective hint about what hidden units should learn, thereby bringing along the way a form of regularization that prevents overfitting even in deep networks with many degrees of freedom. The greedy layer-wise unsupervised strategy provides an initialization procedure, after which the neural network is fine tuned to the global supervised objective. A general paradigm followed by these algorithms can be decomposed into three phases. In the next three subsections, we will layout the training design paradigm in three phases that are used to train a deep neural network.

The following block diagram gives a general framework for a Deep Learning algorithm.

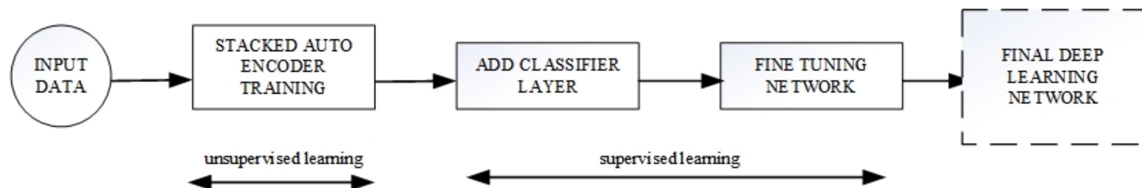

Figure 4.3: Block diagram for general deep learning algorithm

Unsupervised Pre-training

One of the difficulty in training a deep neural network is that the gradient descent can easily stuck in a poor local minimum [6]. The number and quality of these local minima clearly also influence the chance for random initialization to be in the valley of attraction. It's possible that with more hidden layers, the width of these poor valleys increases. Although using regularization parameter that will favor simpler models is suggested [43] [104], however, in practice these approaches normally overfits

and are not data dependent. Unsupervised pre-training helps to select the valley of attraction from which the learning leads to a good generalization.

Pre-training is an unsupervised initialization procedure that is used to train the L layer deep autoencoder. In order to understand the deep learning being used for feature extraction, we have to understand the *Information Storage Property* (ISP) [128] which states that if the feature vector $\mathbf{x}$ can be used to reconstruct the raw data from which it was extracted, as the dimension of input increases, then $\mathbf{x}$ contains enough information for useful classification. The aim of an unsupervised pre-training is to obtain a useful representation of the inputs. The representation learning from the input that is formed by the output of the hidden layer can reduce feature engineering thereby making supervised classifier and regression models much efficient. In pre-training, we greedily train subsets of the network parameters using a layer-wise and unsupervised learning criterion by repeating the following steps:

Algorithm 7 Unsupervised pre-training for deep autoencoders

1: **while** a stopping criterion is met, for each layer ($l \in (1 \cdots \cdots L)$) **do**
2: Map input training sample $\mathbf{x_p}$ to represent $\mathbf{O_p}^l$ using a unsupervised learning algorithm.
3: Update $\mathbf{W}^l$ of layer l and delete the output weights from hidden to output layer from the trained auto encoder.
4: **end while**

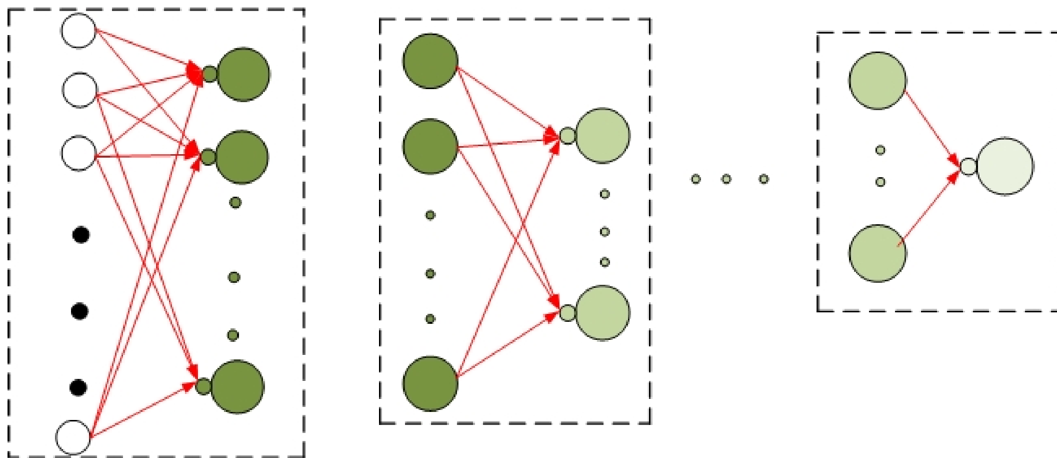

Figure 4.4: Unsupervised pre-training

Figure 4.4 shows the above greedy layer-wise pre-training procedure. The pre-training helps to initialize the networks and also encourages to encode the latent input structure in the hidden layer. In other words training an auto encoder in each layer is an unsupervised step i.e. $t_p(i) = x_p(i)$ therefore, if validation error is close to zero than according to ISP, all the information from input

layer is passed onto the corresponding forward hidden layer either as compressed features (when N_h $< N$) or mapped features in high-dimensional space (when $N_h > N$)

Stacking

Stacking is a key aspect of deep architecture. It's basically taking advantage of the depth in a representative learning and the stacked autoencoder harness the power of re-using the features by learning higher level features in each layer. Theoretically, it's well-established that deep representation are exponentially more efficient that those that are not sufficiently deep [65], [15], [14]. Therefore, once the pre-training of autoencoder is completed, we are ready to develop the deep network by stacking each of these trained autoencoder. figure 4.4 shows the stacked autoencoder in a fan-in structure. We show only 2 layers deep network for simplicity, in-fact we can stack the autoencoder with sufficient depth. It is tobe notes that when the number of hidden layers is greater than one, the autoencoder is considered to be deep.

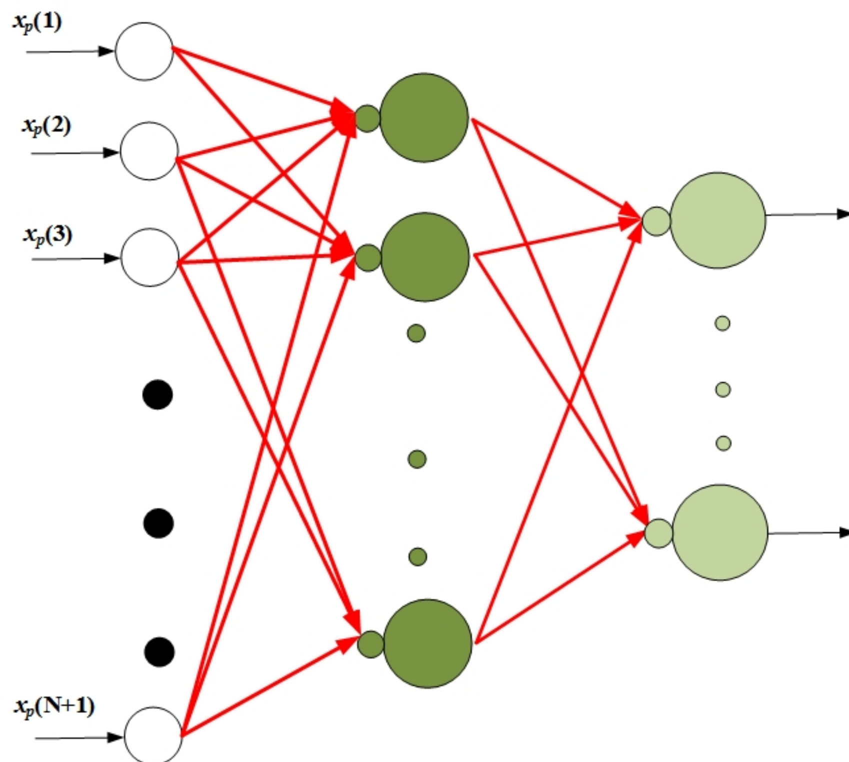

Figure 4.5: Autoencoder stacking

4.1.4 Classifier design

Once the unsupervised pre-training and stacking is completed, deep autoencoder features can be used for classification problem by feeding them into a linear or non-linear classifier. It's quite common to use either a softmax classifier [17] with cross entropy loss. The mth output of a softmax classifier [102] will be

$$O_p(m) = \frac{exp(y_p(m))}{\sum_{m=1}^{M}(exp(y_p(m))} \tag{4.5}$$

The predicted class $i^{'}$ will be $i^{'} = \underbrace{argmax}_{i} O_p(i)$. The MSE in equation (2.2) is modified by replacing $\mathbf{y_p}$ with $\mathbf{O_p}$ from equation (4.5). In chapter 3.1, we described the training of an MLP network as a regression model. [52] concludes using cross entropy error function for classifiers. Contrary to this, we design our classifier via regression models by modifying the desired output with output reset (OR) [108] algorithm. From [152], an MLP network is a Bayes discriminant where the output is posterior probability p(y = i_c | $\mathbf{x}$) and the performance comes partially from the error criterion used in training. Typically the output of the MLP is a discriminant function $d_i(x)$ and approximates posterior probabilities. The largest discriminant corresponds to the estimated class, while the other values correspond to estimated incorrect classes. Universal approximation theorem [74] implies that the MLP can approximate any continuous discriminants arbitrary well. However, given an MLP with MSE as error criterion and fixed binary desired outputs, the training is damaged when actual outputs are larger than 1 for the correct class and less than zero for the incorrect class. Therefore we use output reset (OR) algorithm [108], [56] to modify the desired outputs so that the discriminant types can be approximated. Following [117], we modify the target output $\mathbf{t}_p$ to $\mathbf{t}^{'}_p$. Here $t^{'}_p(i_c)$ = $y_p(i_c)$ if $y_p(i_c) > t_p(i_c)$ and $t^{'}_p(i_d) = y_p(i_d)$ if $y_p(i_d) < t_p(i_d)$. Following [108], by designing classifier using this method, we greatly decrease the training and validation error. However, the OR algorithm is fairly immune to the outliers but a strong disadvantage is that the decision boundary may not necessarily be optimally placed. Clearly, it's a fundamental research problem that will be investigated in Chapter VII.

4.1.5 Fine tuning

Fine tuning is a well-known technique [69] that is used to update all the weights in the stacked deep autoencoder including the classifier layer, in order to improve the overall performance. In each iteration, fine tuning updates all the weights in all the layer of a stacked autoencoder along with the classifier. Overall the entire deep learning network for 2 layers along with the classifier is shown in figure 4.6.

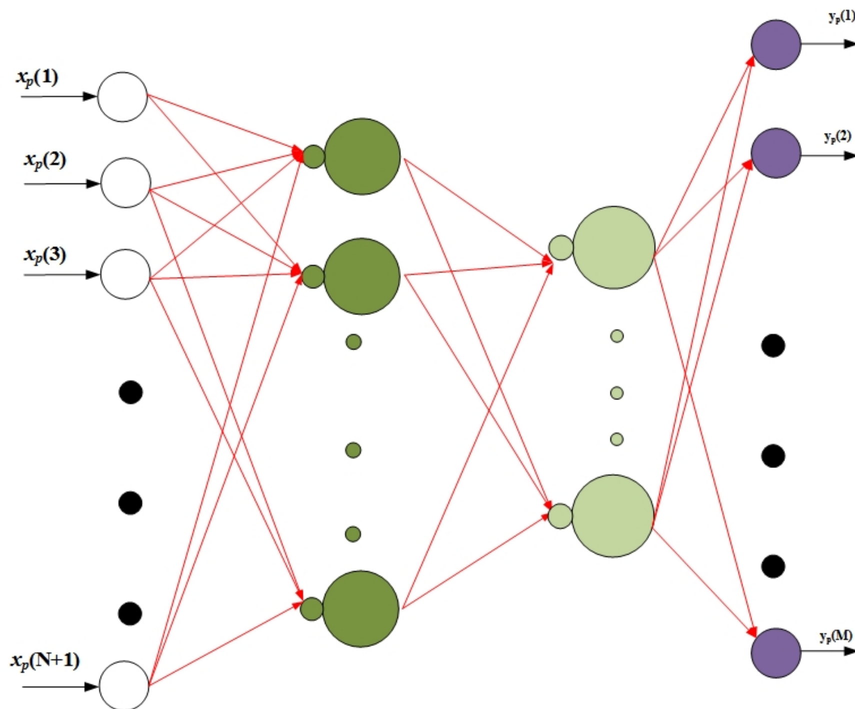

Figure 4.6: Deep neural network

The deep neural network algorithm can be summarized as follows:

Algorithm 8 Greedy layerwise training of deep autoencoder

1: **Unsupervised pre-training**: Train each autoencoder as illustrated in algorithm 7.
2: **Stacking**: Stack the trained autoencoders to form the deep autoencoder network.
3: **Classifier**: Add a classifier at the end of the last hidden layer of the stacked autoencoder and train it using a supervised cost function.
4: **Fine tuning**: The entire network using BP and gradient descent on a global supervised cost function.

4.1.6 Variants of autoencoders

As mentioned in the previous section, there are many variants of autoencoder that are useful in generating the effective representation of the data instead of learning the identity function. Here, we discuss two such variants. Sparse autoencoder is an under complete representation where the goal is feature compression whereas denoising autoencoders generate overcomplete features that map in the higher dimension.

Sparse autoencoder

In autoencoder, sparsity is achieved by penalizing the hidden unit activations thereby making them closer to there saturating value of 0 [21] [103], [184]. [130] [102] [141] describes a sparse autoencoders (SAE) as in where a KL divergence penalty term is introduced into the error function. KL divergence is a standard function for measuring how two distributions are different.

$$KL(\overline{O}(k)) = \rho \cdot log(\frac{\rho}{\overline{O}(k)}) + (1-\rho) \cdot log(\frac{1-\rho}{1-\overline{O}(k)}) \quad (4.6)$$

Here $\overline{O}(k)$ is the average activation of the kth hidden unit,

$$\overline{O}(k) = \frac{1}{N_v} \sum_{p=1}^{N_v} O_p(k) \quad (4.7)$$

ρ is the sparsity parameter, typically a small value close to zero. In equation ()4.6, for $\rho = \overline{O}(k)$, $KL(\overline{O}(k)) = 0$. In order to avoid overfitting, we add an L_2 regularization term or weight decay term. Adding these two terms in equation ()2.2, the final error function is now modified as follows

$$E_{SAE} = E + \beta \cdot KL(\overline{O}(k)) + \frac{\lambda}{2} \sum_{l=1}^{N_l} \sum_{i=1}^{N_h^l} \sum_{j=1}^{M} [w^l(j,i)]^2 \quad (4.8)$$

Here β is the sparsity penalty term and λ is the weight decay parameter. The regularization term tends to decrease the magnitude of the weighs and is not applied to the bias terms in weight matrices. The regularization term is a variant of the Bayesian regularization method by replacing the Gaussian prior on the parameters and doing Maximum apriori probability estimation.

Denoising autoencoder

In an over-complete representation where N $> N_h$, there is no compression and each hidden layer can just copy a different input unit. To avoid this problem, Denoising autoencoders (DAE) [174] is a modification to the basic autoencoders. The key idea in a DAE is that the input representation should be robust to noise. Therefore the noise is added by

1. Switching off the random subset of input units to zero.
2. Adding Gaussian noise to the turned on input units.

Therefore the input $\mathbf{x}$ is first corrupted by the above two steps to get a partially destroyed input $\tilde{\mathbf{x}}$ which is then fed to a basic autoencoder to obtain the reconstructed input $\hat{\mathbf{x}}$. The error measure in

a cross entropy reconstruction function as

$$E_{DAE} = -\sum_{i=1}^{N} \mathbf{x(i)log(\hat{x}(i)) + (1 - x(i))log(1 - \hat{x}(i))} \tag{4.9}$$

Note that the error measure compares $\hat{\mathbf{x}}$ with the noiseless input $\mathbf{x}$. By adding noise in the input, we push it away from the manifold and then the basic autoencoder training try to project $\hat{\mathbf{x}}$ onto the manifold. Overall this helps in learning more meaningful features than a linear autoencoder and overcome the over-complete representation problem. Unlike SAE, DAE is a fan-out structure. It is reported in [173] that adding Gaussian noise in input layer is not equivalent to weight decay or regularization.

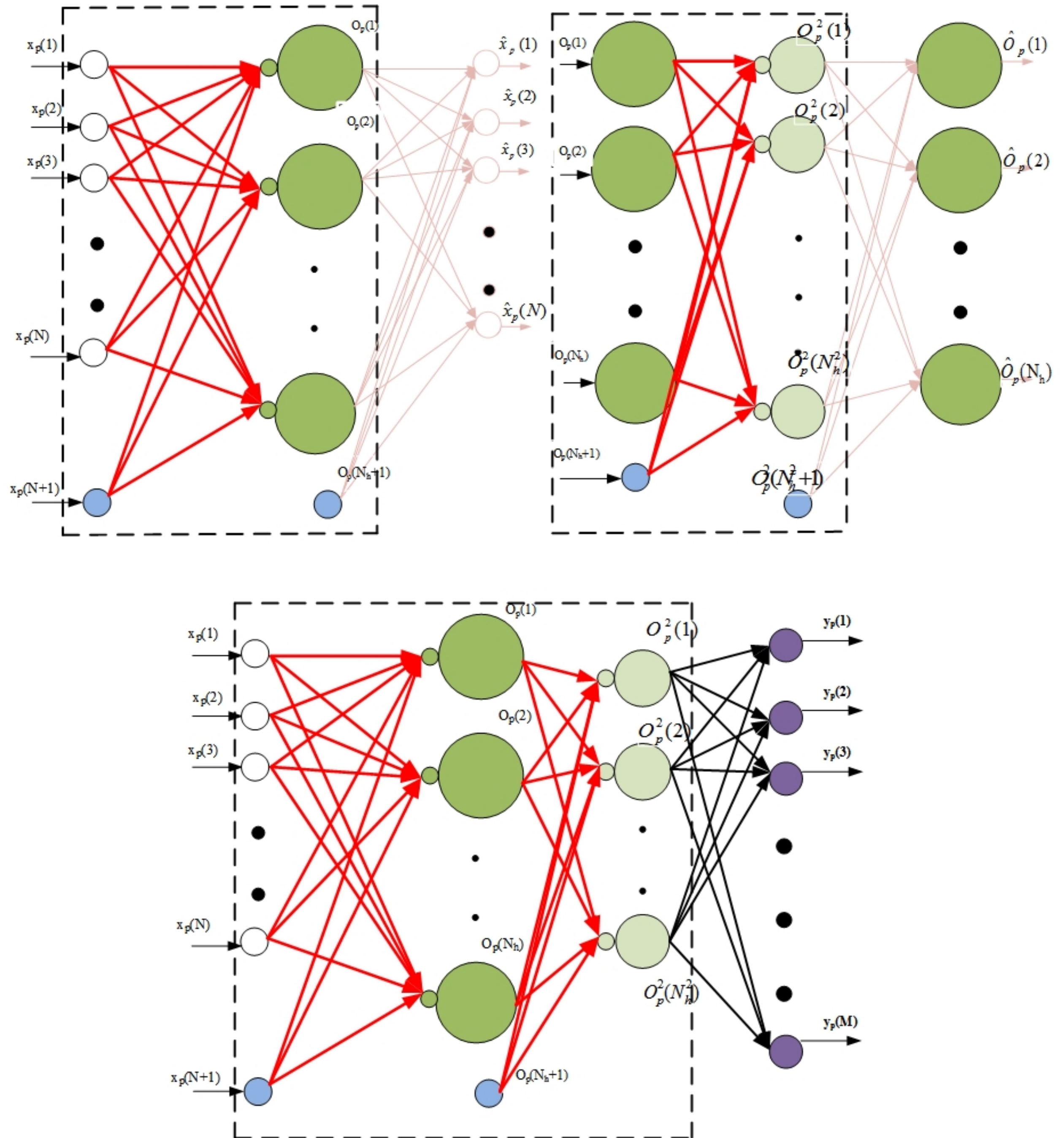

Figure 4.2: Summary of deep neural network

Chapter 5

Problems, Goal and Proposed Tasks

In this chapter we start by enumerating major problems and bottlenecks associated with training a linear classifier, single hidden layer MLP and a deep learners. We also detail out the goals and proposed tasks that we will be focusing on this dissertation.

5.0.7 Problems with linear classifiers

Existing regression based classifiers, and some others, have the following problems which damage performance.

(L1) There are *inconsistent* errors $y_p(i)$ - $t_p(i)$ whose absolute value tends to move in a direction opposite to that of the probability of classification error P_e while increasing the mean square error. Specifically, this occurs when $y_p(i_c) \geq t_p(i_c)$ for the correct class i_c or $y_p(i_d) \leq t_p(i_d)$ for any incorrect class i_d.

(L2) There are *consistent* errors $y_p(i)$ - $t_p(i)$ whose absolute value tends to move in the same direction as P_e. Specifically, this occurs when $y_p(i_c) \leq t_p(i_c)$ or $y_p(i_d) \geq t_p(i_d)$ for any incorrect class i_d.

(L3) When small *consistent* or *inconsistent* errors grow to become outliers, performance is severely impacted [109].

(L4) When we have too many useless inputs, testing errors increase due to *Hughes phenomenon*

[77]. This is often called over-training in the machine learning literature. Useless inputs fall into at least two categories:

(C1) Some inputs can be linearly dependent upon others.

(C2) Some inputs are not dependent upon others but have no information that improves the calculation of the output discriminant vector $\mathbf{y}$.

(L5) Performance suffers when there is a mismatch between the assumed and actual statistics of the data.

(L6) Current training algorithms for the linear or generalized linear classifier do not include pruning of noisy or dependent inputs.

Margin based classifiers are little effected by problems (L1) through (L5) because they are principally effected by support vectors. In addition, some of these algorithms make use of slack variables which limit the damage of support vectors due to imperfect outputs. However, even margin based classifiers can be strongly effected by problem (L6) since they often fail to eliminate useless inputs.

It is to be noted that problem (L5) is obviously relevant for Bayes-Gaussian linear classifiers and regression based classifiers, for which it is assumed that there are no outliers. The proposed algorithm in this dissertation makes no assumptions about the data statistics and is tolerant of outliers. Similar to this, [172] discusses the V matrix method that captures the geometric properties of the observation data that are usually ignored by the classical statistical methods. This makes for a smoother approximation.

5.0.8 Problems with single hidden layer MLP

(M1) Ideally, one would like to have a good generalization performance with the smallest size network that does not have any dependent hidden units. In order to find the optimal smallest independent number of hidden units, two methods are employed called growing and pruning methods. In the current work, we look into pruning methods in detail. Pruning is essentially training an MLP with larger than necessarily hidden units so that it learns reasonably quickly with less sensitivity to the initial condition and then removing each unit to reduce to size so that the generalization error is less. Pruning methods is further divided into dynamic pruning methods such as weight decaying [27], [139] and static pruning methods such as skeletonization [125], optimal brain damage [100], hidden unit reduction methods [60] and Frobenius norm approximation method [92]. [144] provides an excellent survey of the pruning algorithms. Therefore pruning of the hidden units becomes extremely important.

(M2) A typical MLP training procedure is a supervised training that involves training using only the labeled data. For any machine learning problem, Labeled data is usually very expensive and difficult to obtain. Inability to use the unlabeled data holds it significantly in terms of its applicability in various fields. Broadly, there are two types of learning algorithms that use unsupervised data. Firstly, semi-supervised learning [133] and secondly, self-taught learning [141]. Unlike semi-supervised learning, self-taught learning does not require the unlabeled data to be from the same generative distribution as the labeled data.

(M3) The structural optimization of current MLP models is inadequate, resulting in networks with more hidden units than necessary to minimize the generalization error.

5.0.9 Problems with deep learning networks

From a neural networks perspective, deep learning is a way to train a multi hidden layer MLP. Until the seminal paper by Hinton [69], it appeared that deep neural network is hard to train due to computational constraints. However, there are several problems associated with training a deep learning algorithm that is summarized below:

(D1) Majority of literature on deep learning uses first order gradient based training [15], [94], [141], [102] that is often computationally expensive.

(D2) In our opinion, the useful properties of sigmoid activation function are not completely exploited. There is a large scope of research in investigating the mean square criterion along with sigmoid activation functions.

(D3) For any highly flexible deep network, there actually exists any valleys of attraction in the parameter space that can give a poor generalization. So when a gradient descent is able to find a good low training error, it's not guaranteed that the associated parameter configuration will provide good generalization. Therefore initialization of a deep neural network continues to be an open problem.

(D4) The current training algorithms involves many phases i.e unsupervised pre-training, supervised training and fine-tuning thereby making the entire training paradigm complicated.

(D5) Similar to MLP, there are many parameters in a deep learner that are user chosen. This caused a problem while comverging to an optimal deep learner structure.

(D6) The deep autoencoders are unable to adapt to the unknown amount of nonlinearity in the data. Even though the unsupervised pre-training helps in initializing the model, the abstract features that are formed in a hierarchical manner are not entirely transferring the information

to further deeper into the network.

(D7) Since deep learning is fairly a new area, there is a clear lack of theory that explains the learning paradigm for a deep neural network. A strong theoretical analysis is still lacking in order to explain the success of deep learning models for various applications. There is a clear need of developing investigative tools for understanding the training of deep neural networks.

The high dependency on first order gradient based training as in (D1) is an appealing area of research. Certainly, there is a need to investigate non-gradient methods like Grahm-Schmidt procedure along with our successful implementation of second order MLP training algorithms [113], [143], [171], [88], [62], [131]. As mentioned in (D2), the choice of activation function is extremely important while training the deep learner. Sigmoid activation function usually leads to saturation of hidden units [51] and many other alternative activation function have been suggested [69], [13]. It's been reported that the cross entropy cost function coupled with softmax activation works better than the MSE and sigmoidal activation in the classification task. [106] [153], [52]. The difficulty in training in (D4) is usually compensated by using cross-validation but usually the number of good generalization configuration for the MLP model is small in comparison to a good training configuration. To elaborate (D5), the number of parameters that needed to be adjusted before we start training includes number of hidden layers, a number of hidden units in each layer and learning factor for the hidden units.

5.0.10 Goals and proposed tasks

We narrow down to the following *goals* in the present work

1. Improve and automate the design of linear and MLP networks with small testing error and minimum user intervention.

2. Adapt the automated feed forward technology to deep learning network design and minimize the number of weights in a deep learner.

In order to expand on the above goals, we have following *proposed tasks* marked out, that we focus on in this dissertation.

1. Develop and automate the linear network design by identifying and removing dependent inputs.

2. Improve the design of linear classifiers by adding a nonlinear activation function at the output.

3. Improve the pruning algorithm for hidden units in a single hidden layer MLP and an autoencoder.

4. Develop robust MLP training algorithm that are prune to outliers effects.

5. Apply all the above improvements to the deep learning paradigm.

6. Automate user chosen parameter that are specific for deep learning models.

7. Improve the autoencoder theory by detailing out the mathematical framework of it's working.

Chapter 6

Proposed improvements in linear networks

In this chapter, we start with the linear classifier $\mathbf{W}$ of subsection 2.1.2 and describe several incremental improvements to it, that partially solve problems (L1) through (L6). These improvements consist of changes to the elements of matrices $\mathbf{R}$ and $\mathbf{C}$ in equation (3.5).

6.0.11 Modifying targets with output reset

In this subsection, our goal is to eliminate inconsistent errors of (L1) and unwanted variations in output vectors $\mathbf{y}_p$ due to (L5) by developing new target outputs $t_p^{'}(i)$, while keeping the constraint that the target margin satisfies $t_p^{'}(i_c)$ - $t_p^{'}(i_d) \geq 1$. There are two kinds of inconsistent errors which simultaneously increase E and decrease P_e or leave P_e the same. First each pattern's output can have a bias so that the average of $t_p(i)$ over i, differs from the average of $y_p(i)$. Second, as stated in problem (L1), we can have $y_p(i_c) > t_p(i_c)$ or $y_p(i_d) < t_p(i_d)$. In order to remove these inconsistent errors we design a new error function $E^{'}$ [108] [56] in which targets, but not labels are changed [53]. Specifically as in [47]

$$E^{'} = \frac{1}{N_v} \sum_{p=1}^{Nv} \sum_{i=1}^{M} [t_p^{'}(i) - y_p(i)]^2 \tag{6.1}$$

where $t_p^{'}(i)$ is modeled as

$$t_p^{'}(i) = t_p(i) + a_p + d_p(i) \tag{6.2}$$

and where a_p and $d_p(i)$ are initially equal to zero. Since a_p is the same for each class, it has no

effect on P_e. Following [56], [108], we calculate the closed form expression for a_p by setting $\frac{\partial E'}{\partial a_p} = 0$, therefore obtaining

$$a_p = \frac{1}{M}\sum_{i=1}^{M}[y_p(i) - t_p'(i) - d_p(i)] \tag{6.3}$$

Similarly, $d_p(i)$ is defined as

$$d_p(i) = y_p(i) - t_p'(i) - a_p \tag{6.4}$$

such that $d_p(i_c) \geq 0$ and $d_p(i_d) \leq 0$. It is true that a_p and $d_p(i)$ can be included in $t_p'(i)$ or $y_p(i)$ during training. However, these parameters are not available during testing because they make use of the correct class $i_c(p)$, which is unknown. Therefore we include them in $t_p'(i)$.

To avoid inconsistent errors we need $t_p'(i_c) \geq y_p(i_c)$ and $t_p'(i_d) \leq y_p(i_d)$ so

$$d_p(i_c) = [y_p(i_c) - a_p - t_p(i_c)]u(y_p(i_c) - a_p - t_p(i_c)) \tag{6.5}$$

and

$$d_p(i_d) = [y_p(i_d) - a_p - t_p(i_d)]u(t_p(i_d) - a_p - y_p(i_d)) \tag{6.6}$$

where u($\cdot$) denotes the unit step function. Note here that for a classifier, the training halts when E' becomes zero, even if E is nonzero. The effect of (L5) is reduces because $\lim_{y_p(i_c)\to\infty}(y_p'(i_c) - t_p(i_c)) = 0$ and similarly $\lim_{y_p(i_d)\to\infty}(y_p'(i_d) - t_p(i_d)) = 0$. We denote the process of obtaining t_p' as an output reset (OR) algorithm and describe it as follows

Algorithm 9 OR algorithm

Given p, $t_p(i)$ and $y_p(i)$ from the current regression based classifier, initialize a_p to zero and $d_p(i)$ to zero for 1 ≤ i ≤ M.

for $i_t = 1$ to 3 **do**
  Calculate a_p using (6.3).
  for $i = 1$ to M **do**
    Calculate $d_p(i)$ using (6.5) and (6.6)
    Calculate $t_p'(i)$ using (6.2)
  end for
end for

Note that in algorithm 1, letting the maximum number of iterations equal 3 allows considerable improvement in performance without a significant change in training time [110]. In [56], the heuristic iterative OR algorithm of [110] is replaced by an efficient closed from expression for the target outputs. Using OR, we modify regression-based LC training as follows:

Algorithm 10 LC-OR algorithm

Read the training data. Set the value of $N_{it}^1 = 10$.
Calculate **R** and **C** using equation (3.6).
Solve equation (3.5) for the initial weight matrix **W**.
Initialize $i_t \leftarrow 0$.
while $i_t < N_{it}^1$ **do**
  Initialize $\mathbf{C} \leftarrow 0$.
  for $p = 1$ to N_v **do**
    Use OR algorithm to process $\mathbf{t}_p$ into $\mathbf{t}_p^{'}$.
    Update **C** by replacing $\mathbf{t}_p$ by $\mathbf{t}_p^{'}$ in equation (3.6) as $\mathbf{C} \leftarrow \mathbf{C} + \mathbf{x}_{ap}\ (\mathbf{t}_p^{'})^T$
  end for
  Solve equation (3.5) for weight matrix **W**.
  $i_t \leftarrow i_t + 1$
end while

In this subsection we have removed inconsistent errors of (L1) from $E^{'}$ by changing elements of the cross correlation matrix **C**. We have reduced the effects of output vector bias and outliers due to (L5).

Another method for changing target outputs is the Ho-Kashyap procedure [72], [73]. Unlike OR, Ho-Kashyap (1) does not handle pattern biases, (2) is used with binary or one-versus-one output coding, and (3) solves linear equations simultaneously for both target vectors and classifier weights.

6.0.12 Pruning less useful inputs

As we discussed in problem (L4), many classifiers fail to eliminate less useful inputs, which can lead to Hughes phenomenon [77] and other forms of overtraining. Our first solution for both categories of (L4) is to utilize pruning of inputs when regression is used to solve for classifier weights. In pruning which can be viewed as a type of L_0 regularization, we:

- Order the inputs as x(o(1)), x(o(2)),x(o(N+1)), where o(n) is an order function denoting the index of the nth most useful input.
- Find the number of ordered inputs N_i such that validation error is minimized.

Basic algorithm

Here we describe a direct but inefficient version of our algorithm. The efficient version is discussed in the next subsection. Let S_m denote an m-element subset of the indices of the m most useful inputs.

The first step in pruning is to order the inputs using sequential forward selection (SFS) [53] so that the input set $S_1 = \{o(1)\}$ yields the minimum training error for a classifier having one input. Each successive set S_m is found as

$$S(m) = S(m-1) \bigcup \{o(m)\} \tag{6.7}$$

such that the input $x(o(m))$ is that which causes the maximal decrease in training error out of all the unchosen inputs. Let $E_m(k)$ denote the training error when chosen inputs $x(o(1))$ through $x(o(m1))$ are used, along with $x(k)$. The integer k denotes a candidate value of $o(m)$. Then

$$E_m(k) = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - y_{pm}(i)]^2 \tag{6.8}$$

where the output $y_{pm}(i)$ is

$$y_{pm}(i) = w_o(i,k)x_p(k) + \sum_{n=1}^{m-1} w(i,n)x_{po}(n) \tag{6.9}$$

and where

$$x_{po}(n) \equiv x_p(o(n)) \tag{6.10}$$

Here $\mathbf{W}_o$ denotes the classifiers weight matrix when the inputs are ordered. Therefore,

$$\mathbf{y}_p = \mathbf{W} \cdot \mathbf{x}_p = \mathbf{W}_o \cdot \mathbf{x}_{po} \tag{6.11}$$

The set S_U of unchosen input indices is initially $\{1, 2, 3, \cdots, N+1\}$. Sequential forward selection [53] is implemented to order the inputs as in algorithm (3)

Algorithm 11 SFS algorithm

```
for i = 1 to N+1 do
    o(m)= argmin_{k∈S_U} { min_{W_o} E_m(k) }
    S_U ← S_U - {o(m)}.
end for
```

The second step in our algorithm is to find the number of ordered inputs that minimizes the classifiers validation error. Let $E_v(m)$ denote the validation error when m ordered inputs are used, as

$$E_v(m) = \frac{1}{N_{vv}} \sum_{p=1}^{N_{vv}} \sum_{i=1}^{M} [t_{pv}(i) - v_{pm}(i)]^2 \tag{6.12}$$

where

$$v_{pm}(i) = \sum_{n=1}^{M} w_o(i,n) x_{pvo}(n) \tag{6.13}$$

where N_{vv} is the number of validation patterns, subscript m denotes the number of inputs being used, $t_{pv}(i)$ denotes the ith target output for the pth validation pattern, $v_{pm}(i)$ is the ith output for the pth validation pattern, and $x_{pvo}(n)$ denotes the nth ordered input in the pth validation pattern. The number N_i of chosen ordered inputs is now found as

$$N_i = \underset{m}{argmin} \{E_v(m)\} \tag{6.14}$$

Efficient implementation

The algorithm of the previous subsection is efficiently implemented using a form of orthogonal least squares, related to that in [31] for RBF networks. In our pruning algorithm [127], however, the training data is represented using correlation matrices, so that N_v patterns are not stored in main memory. Also, our method efficiently determines N_i. Using the Gram-Schmidt process [53], our algorithm develops an $(N + 1)$ by $(N + 1)$ lower triangular matrix $\mathbf{A}$, so that input vectors with orthonormal elements can be found as

$$\mathbf{x}_p^{'} = \mathbf{A} \cdot \mathbf{x}_{po} \tag{6.15}$$

The output vector $\mathbf{y}_p$ can be written as

$$\mathbf{y}_p = \mathbf{W}^{'} \cdot \mathbf{x}_{ap} \tag{6.16}$$

Now, $E_m(k)$ can be rewritten as

$$E_m(k) = E_t - P(m,k) \tag{6.17}$$

$$E_t = \sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p(i)]^2 \tag{6.18}$$

$$P(m,k) = \sum_{i=1}^{M}[w^{'}(i,m,k)]^2 + \sum_{n=1}^{M-1}[w^{'}(i,n)]^2 \tag{6.19}$$

In the orthonormal network of equations (6.15) and (6.16), $w^{'}(i,m,k)$ is the weight from $x^{'}_p(m)$ to $y_{pm}(i)$ when $o(m)$ equals the candidate value k. The *argmin* operation in algorithm 3 can now be replaced by

$$o(m) = \underset{k \in S_U}{argmax}\{P(m,k)\} \tag{6.20}$$

Minimizing validation error

In algorithm 3, all elements of the weight matrix $\mathbf{W_o}$ change when m increases. Therefore, in equations (6.12) and (6.13), we have to store a different $\mathbf{W}_o$ for each value of m, use m multiplies for each value of $v_{pm}(i)$, and potentially access the validation patterns every time $E_v(m)$ is calculated for a different value of m. These problems are eliminated when using the orthonormal network [127]. Specifically, $v_{pm}(i)$ in equation (6.13) is now written as

$$v_{pm}(i) = \sum_{n=1}^{m} w^{'}(i,n)x^{'}_{pv}(n) \tag{6.21}$$

where $x^{'}_{pv}(i)$ is the ith element of the pth validation pattern's transformed input $\mathbf{x}^{'}_{pv}$, found as

$$\mathbf{x}^{'}_{pv} = \mathbf{A}\cdot\mathbf{x}_{pvo} \tag{6.22}$$

Note that $v_{pm}(i)$ in equation (6.21) can be written recursively as

$$v_{pm}(i) = v_{p,m-1}(i) + w^{'}(i,m)x^{'}_{pv}(m) \tag{6.23}$$

Therefore to summarize,

1. For a given pattern, the calculation of $v_{pm}(i)$ in equation (6.23) requires one multiply.
2. One matrix $\mathbf{W}^{'}$ needs to be stored rather than $(N+1)$ of them.
3. One pass through the data is necessary to calculate the $E_v(m)$ sequence and N_i [127].

4. After N_i and $\mathbf{W}^{'}$ are found, $\mathbf{W}^{'}$ is mapped back to $\mathbf{W}$. $\mathbf{W}_o$ is generated from $\mathbf{W}^{'}$ as $\mathbf{W}_o = \mathbf{W}^{'} \cdot \mathbf{A}$. The matrix $\mathbf{W}$ is found from $\mathbf{W}_o$ by rearranging its columns.

Further details of subsections 3.2.2 and 3.2.3 are given in the pruning description of [127]. Linear classifier training using regression OR and pruning is denoted as LC-OR-P. The algorithm follows.

Algorithm 12 LC-OR-P algorithm

Read the training data.
Using training data, find weight matrix $\mathbf{W}$ using the LC-OR algorithm.
Randomly split off 30 % of the training data to be validation data.
Initialize N_{it}^2 , $i_t \leftarrow 0$.
while $i_t < N_{it}^2$ **do**
  Using equation (2.5), find the $\mathbf{R}$ matrix. Find the $\mathbf{C}$ matrix using OR.
  Find the order function $o(n)$ and the triangular matrix $\mathbf{A}$.
  Using pruning, calculate the weight matrix $\mathbf{W}$ whose non-zero elements are $w(i, o(n))$ for $1 \leq n \leq N_i$
  $i_t \leftarrow i_t + 1$
end while
Save $\mathbf{W}$, $o(n)$ and N_i.

Table 6.1 compares the numbers of inputs used when LC-OR and LC-OR-P are applied. As we see from Table 6.1, pruning has the largest effect on data files with large input dimension, where it helps to prevent over-fitting.

Table 6.1: Pruning results on linear classifiers

Datasets	LC-OR number of input units (N)	LC-OR-P number of input units (N_i)
Gongtrn	16	16
Comf18	18	18
MNIST	784	563
SVHN	1024	962
CIFAR-10	1024	985
COVER	54	52
SECTOR	55197	54871
RCVI	47236	46923
NEWS-20	9216	8521
Breast-Cancer	40	40

We use 10-fold testing errors to evaluate the various versions of our classifiers. For each fold, 1/10 of the data is used for testing and 9/10 for training and validation. Out of the 9/10 data, 70 % is used for training and the rest is for validation. Table 6.2 lists the 10-fold testing errors along with the standard deviation that results from each incremental improvement to the regression based linear classifier (LC). For each datafile, the 10-fold testing errors either decrease or stay the same in each successive step of LC-OR-P. Therefore LC-OR-P is the best algorithm we've developed up to this point.

Table 6.2: 10-fold testing error % results for various linear classifiers

Datasets	LC	LC-OR	LC-OR-P
Gongtrn	16.2577 ± 0.02	11.3668 ± 0.02	11.3668 ± 0.02
Comf18	28.6579 ± 0.37	20.3308 ± 0.32	20.3308 ± 0.27
MNIST	15.2478 ± 0.13	7.3575 ± 0.13	5.1816 ± 0.14
SVHN	18.2487 ± 0.31	5.8161 ± 0.33	4.7459± 0.26
CIFAR-10	20.2478± 1.08	11.3426 ± 0.99	10.8348 ± 0.85
COVER	28.1452 ± 0.44	22.3202 ± 0.43	20.0021 ± 0.44
SECTOR	11.3424 ± 1.18	9.34532 ± 1.15	8.34521 ± 1.04
RCV1	15.3245 ± 1.52	11.2345 ± 1.24	10.2343 ± 0.84
NEWS-20	29.6746 ± 0.38	17.6786 ± 0.34	15.4161 ± 0.28
Breast Cancer	19.3571 ± 0.94	17.987 ± 0.12	17.987 ± 0.12

6.0.13 Newton based improvements

The output reset (OR) algorithm discussed in section III eliminates inconsistent errors of problem (L1) but is not designed to eliminate the consistent errors of problem (L2) that can become outliers. In this section we develop a method for bounding consistent errors. We then formulate second order algorithms for updating weights.

Initialization

At this point, the LC-OR-P algorithm uses regression and OR to design linear classifiers with errors satisfying

$$0 \leq t_p^{'}(i_c) - y_p(i_c) < \infty \tag{6.24}$$

and

$$-\infty \leq t_p^{'}(i_d) - y_p(i_d) < 0 \tag{6.25}$$

In other words, inconsistent errors are bounded but consistent errors are unbounded and can still cause problems. Platt [138] has shown the utility of using non linear output activations such as sigmoids. We therefore choose to limit both consistent and inconsistent errors in a new training stage by using sigmoidal output activations to bound network outputs as shown in figure 6.1. The error of the resulting generalized linear classifier (GLC) is

$$E_G = \frac{1}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p(i) - O_p(i)]^2 \tag{6.26}$$

where $O_p(i)$ is defined as $f(y_p(i))$ and $f(\cdot)$ denotes the sigmoid activation function. E_G can be larger than $E^{'}$ in (6.1) because $y_p(i)$ equal to 0 or 1 results in $O_p(i)$ equal to $\frac{1}{2}$ or $\frac{1}{1+e}$ respectively.

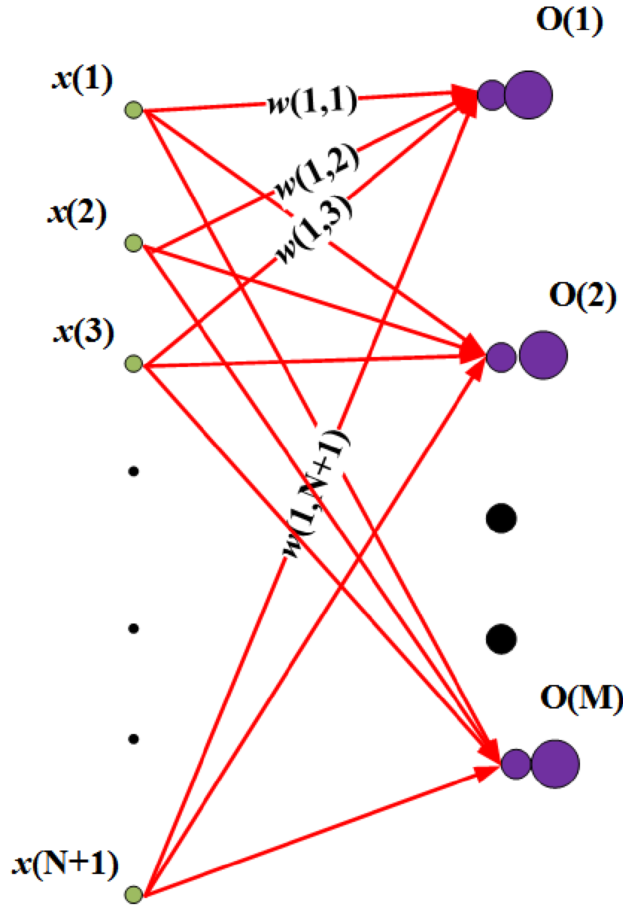

Figure 6.1: Generalized linear model classifier

In order to mitigate this increase in E_G, we need to map the linear output activations $\mathbf{y}_p$ to sigmoidal

outputs $\mathbf{O}_p$ as

$$O_p(i) = f(a \cdot y_p(i) + b) \tag{6.27}$$

where a and b are scalars that are found such that E_G is minimized. We choose to find the values of a and b using Newton's method. Given the error function in equation (2.2), we calculate the negative gradient vector $\mathbf{g}_{ab} = [g_a : g_b]^T$. Here the negative partial derivative g_a is calculated as

$$g_a = -\frac{\partial E_G}{\partial a} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}(t_p(i) - O_p(i))\frac{\partial O_p(i)}{\partial a} \tag{6.28}$$

and g_b is calculated as

$$g_b = -\frac{\partial E_G}{\partial b} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}(t_p(i) - O_p(i))\frac{\partial O_p(i)}{\partial b} \tag{6.29}$$

Similarly the Hessian matrix elements will be

$$h_{aa} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}(\frac{\partial O_p}{\partial a})^2 \tag{6.30}$$

$$h_{ab} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\frac{\partial O_p}{\partial a} \cdot \frac{\partial O_p}{\partial b} \tag{6.31}$$

$$h_{bb} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}(\frac{\partial O_p}{\partial b})^2 \tag{6.32}$$

The Hessian matrix is now assembled as:

$$\mathbf{H} = \begin{bmatrix} h_{aa} & h_{ab} \\ h_{ab} & h_{bb} \end{bmatrix}$$

The weight gain vector $\mathbf{e}$ is obtained by solving the following linear equations

$$\mathbf{H} \cdot \mathbf{e} = \mathbf{g}_{ab} \tag{6.33}$$

The scaling factor a and b are then updated as

$$a \leftarrow a + e(1) \tag{6.34}$$

$$b \leftarrow b + e(2) \tag{6.35}$$

After the weight matrix is updated as

$$\mathbf{W} \leftarrow a \cdot \mathbf{W}, \tag{6.36}$$

each output threshold in $\mathbf{W}$ is updated as

$$w(i, N+1) \leftarrow w(i, N+1) + b \tag{6.37}$$

In case the error starts to increase, we backtrack, as in Rprop [146], from the previously saved network.

Proposed algorithm

After, we have initialized the scaling factors, our GLC model is ready to be trained. In each training epoch, the weight connecting the ith output to the nth input in the network is updated as follows:

$$w(i, n) \leftarrow w(i, n) + e_i(n) \tag{6.38}$$

Here $\mathbf{e}_i$ is the $N+1$ length Newton's direction vector for the ith output. To obtain the value of $e_i(n)$, we calculate the gradient vector and Hessian matrix for each ith output with respect to error as:

$$\begin{aligned} g_i(m) &= \frac{2}{N_v} \sum_{p=1}^{N_v} [t_p(i) - O_p(i)] O_p^{'}(i) x_{ap}(m) \\ h_i(m, u) &= \frac{2}{N_v} \sum_{p=1}^{N_v} \frac{\partial O_p(i)}{\partial w(i, m)} \frac{\partial O_p(i)}{\partial w(i, u)} \end{aligned} \tag{6.39}$$

for each ith output, we solve the following sets of linear equation through OLS.

$$\mathbf{H_i} \cdot \mathbf{e_i} = \mathbf{g_i} \tag{6.40}$$

As seen in equation (6.40), we have a separate Hessian matrix for each output, which is a clear disadvantage of this approach. We denote this Newton-based weight modification algorithm as GLC-Newton, which is described below

Algorithm 13 GLC-Newton algorithm

1: Initialize $\mathbf{W}$ using LC-OR-P.
2: Combine validation and training data.
3: Add sigmoid activation functions at each output to get $\mathbf{O}_p$ as in equation (6.27).
4: Find a and b and re-scale $\mathbf{W}$ as in equations (6.36) and (6.37) .
5: Initialize N_{it}^3 , $i_t \leftarrow 0$
6: **while** $i_t < N_{it}^3$ **do**
7: for i = 1 , $\cdots$ M, solve equation (6.40) for $\mathbf{e}_i$.
8: Update $\mathbf{W}$ using equation (6.38)
9: $i_t \leftarrow i_t + 1$
10: **end while**

In Table 6.3, we now tabulate the testing error along with the standard deviation of the resulting improvements in this section from GLC-Newton.

Table 6.3: 10-fold testing error % results for GLC-Newton

Datasets	LC-OR-P	GLC-Newton
Gongtrn	11.3668 ± 0.02	9.4567 ± 0.004
Comf18	20.3308 ± 0.27	17.1813 ± 0.27
MNIST	5.1816 ± 0.14	4.0775 ± 0.15
SVHN	4.7459± 0.26	4.4158 ± 0.22
CIFAR-10	10.8348 ± 0.85	8.4176 ± 0.85
COVER	20.0021 ± 0.44	15.0121 ± 0.47
SECTOR	8.34521 ± 1.04	6.1120 ± 0.99
RCV1	10.2343 ± 0.84	8.1203 ± 0.72
NEWS-20	15.4161 ± 0.28	13.6705 ±0.19
Breast Cancer	17.987 ± 0.12	10.9382 ±0.25

As seen in Table 6.3 GLC-Newtons handling of consistent errors is significantly better than that of LC-OR-P. It is to be noted here that the incremental improvement comes at a cost of minor increment in time complexity of the algorithm. So when we compare GLC-Newton with LC-OR-P, there is a increment in time complexity but the model performance is far better.

Chapter 7

Improvements in MLP algorithm

In this section we start with the MA algorithm of chapter III and describe several incremental improvements to it, that particularly solve problems (M1) through (M4). These improvements consist of changes to the elements of matrices **R** and **C** in equation (3.5). We combine the MOLF from Chapter III and pruning from chapter VI to form MA algorithm. The MA algorithm can be summarized as follows:

Algorithm 14 MA algorithm

1: Read the training data.
2: Calculate **R** and **C** using equation (3.6).
3: Solve equation (3.5) for the initial weight matrix $\mathbf{W}_o$.
4: Initialize N_{it}^3 , $i_t \leftarrow 0$, $\mathbf{C} \leftarrow 0$ and input weight matrix **W** with Gaussian random numbers.
5: **while** it < N_{it}^3 **do**
6: **BP step** : Compute **G** using equation (3.12).
7: **MOLF step** : Solve equation (7.8) for **z** using OLS
8: Update **W** as $\mathbf{W} \leftarrow \mathbf{W} + \text{diag}(\mathbf{z}) \cdot \mathbf{G}_{hwo}$
9: **OWO step** : Solve equation (3.5) to obtain $\mathbf{W_o}$
10: Find the order function $o(n)$ and the triangular matrix **A**.
11: Using pruning, calculate the weight matrix **W** whose non-zero elements are $w(i, o(n))$ for 1 ≤ n ≤ N_i
12: it ← it + 1
13: **end while**
14: Save **W**, $o(n)$ and N_i.

During training the MA algorithm, we prune the hidden units after every third iteration since its quite certain that the hidden units that are pruned are not yet trained and they might become useful after few more iterations. All the hidden unit outputs are linearly dependent upon the inputs so we have regular inputs and the linear combination of the inputs. While pruning, we prune all the hidden unit weights and set the input weights to zero. The MA algorithm is an interesting algorithm and potentially good in the sense that it paves way for reducing model complexity but it suffers from serious flaws that we address in the subsequent sections.

7.0.14 Modification with regularization

The no free lunch theorem [179] implies that our algorithms should be desined to perform well on specific tasks. By using regularization, we give preference to one solution of the learning algorithm over another in its hypothesis space. In order to prevent over-training [50] and solve problem P4, we modify equation (2.2) by adding an L_2 regularization term. The resulting error function is

$$E^{'} = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - y_p(i)]^2 + \lambda \cdot \|\mathbf{W_{ih}}\|^2 \tag{7.1}$$

where, λ is the L_2 regularization parameter. Regularization changes elements in the auto-correlation matrix $\mathbf{R}$ as

$$\mathbf{R} \leftarrow \mathbf{R} + \lambda \cdot diag(\mathbf{r}) \tag{7.2}$$

where $\mathbf{r}$ is a vector containing the diagonal elements of $\mathbf{R}$ such that $r(1) = 0$ so that the bias is not regularized. $diag()$ is an operator that creates a diagonal matrix from a vector. Using a grid search method, for different values of λ, we calculate validation error $E_v(\lambda)$. The value of λ is selected as the value that leads to minimum validation error E_v. Note that like any other regularizer, the L_2 regularizer reduces the generalization error but not its training error. By adding OR and regularization in MA algorithm, we improve it to build an adaptive algorithm denoted by MA-OR-R. The MA-OR-R algorithm can be summarized as follows:

Algorithm 15 MA-OR-R algorithm

1: Read the training data.
2: Randomly split off 30 % data of the training data to be validation data.
3: Calculate **R** and **C** using equation (3.6).
4: Solve equation (3.5) for the initial weight matrix $\mathbf{W}_o$.
5: Initialize N_{it}^3 , $i_t \leftarrow 0$, $\mathbf{C} \leftarrow 0$ and input weight matrix **W** with Gaussian random numbers.
6: **while** it $< N_{it}^3$ **do**
7: Using equation (3.6), find the **R** matrix. Find the **C** matrix using OR.
8: Save the best sized $\mathbf{W}_o$ and $\mathbf{W}_i$ using MA algorithm.
9: it $\leftarrow$ it + 1
10: **end while**
11: For each candidate λ value, update **R** using equation (7.2).
12: Solve equation (3.5) for **W**.
13: Calculate $E_v(\lambda)$ and save the corresponding weight matrix $\mathbf{W}_\lambda$
14: Find $\lambda^{'} = \underset{\lambda}{argmin} E_v(\lambda)$
15: Update output weight matrices $\mathbf{W}_o$ as $\mathbf{W}_o \leftarrow \mathbf{W}_o^{\lambda^{'}}$.
16: Recombine validation and training data.
17: Recalculate **R** using equation (3.6) and **C** using $\mathbf{t}_p^{'}$.
18: Solve equation (3.5) for **W**.

Table 7.1 compared the pruning results for MA-OR algorithm. MA is a two-stage algorithm described in algorithm 14 that has pruning based on validation error. As we see from Table 7.1, pruning has the largest effect on data files with large input dimension, where it helps to prevent over-fitting.

Table 7.1: Pruning hidden units results MA-OR algorithm

Datasets	Initial Nh	Nh in MA algorithm
Gongtrn	30	26
Comf18	30	28
MNIST	200	164
SVHN	200	172
CIFAR-10	200	189
COVER-10	100	63
SECTOR	400	326
NEWS-20	300	242
Breast Cancer	100	82

We use 10-fold testing errors to evaluate the various version of the classifiers. For each fold, 1/10 of the data is used for testing and 9/10 for training and validation. Out of the 9/10 data, 70 % is used for training and the rest is for validation. Table 7.2 lists the 10-fold testing errors along with the

standard deviation that results from each incremental improvement to the MOLF based classifiers. For each datafile, the 10-fold testing errors either decrease or stay the same in each successive step of MA-OR-R. Therefore MA-OR-R is the best algorithm weve developed up to this point.

Table 7.2: 10-fold testing error % results for MOLF based classifiers with $N_{it} = 100$

Datasets	MA	MA-OR-R
Gongtrn	8.2667 $\pm$0.012	7.2667 $\pm$0.196
Comf18	15.6881 $\pm$0.001	14.4901 $\pm$0.001
MNIST	4.5735$\pm$0.254	4.3461 $\pm$ 0.324
SVHN	4.5163 $\pm$ 0.269	4.0846 $\pm$ 0.210
CIFAR-10	7.8613 $\pm$ 0.214	6.2388 $\pm$ 0.229
COVER	14.8578 $\pm$ 0.251	12.1333 $\pm$ 0.281
SECTOR	5.87451 $\pm$ 0.205	4.7896 $\pm$ 0.264
NEWS-20	11.2472 $\pm$ 0.014	10.2478 $\pm$ 0.011
Breast Cancer	10.1470 $\pm$ 0.250	9.8771 $\pm$ 0.210

7.0.15 Improved pruning using median filtering

In this subsection, we will discuss the modifications for MA-OR-R algorithm. We first develop a median filtering algorithm to improve pruning and then formulate the growing approach to combine it with pruning. Following [143], we use OLS to perform one-pass pruning that removes the dependent hidden units. Using OLS, we get orthonormal basis functions for the hidden units that will be ordered based on their usefulness and eliminate the remaining hidden units. While training each MLP, we prune the hidden units based on validation data. During pruning, we plot a validation error E_v *vs* the hidden units Nh curve. The optimal number of hidden units is the one that gives the minimum validation error. The pruning based on the above technique gives us a smaller network that does not have dependent features. Pruning hidden units with validation helps us to get smaller models, however, in certain scenarios, the noise in the curve induces small impulse in the validation error that can lead us to pick far more hidden units than required. The noise in the data can deviate us to get an optimal hidden units. In order to investigate it in detail, we examine the correlation between the training and testing curve with respect to the number of hidden units.

Figure 7.1 shows the validation E_v and smooth validation $\overline{E_v}$ curves for the *Gongtrn* dataset as a function of N_h. Similarly, figure 7.2 shows the testing E_{test} and smooth testing $\overline{E_{test}}$ curves for the *Gongtrn* dataset as a function of N_h. $\overline{E_v}$ and $\overline{E_{test}}$ denotes the 3 point moving average. In both figures, the differences between normal and smooth curves represent noise.

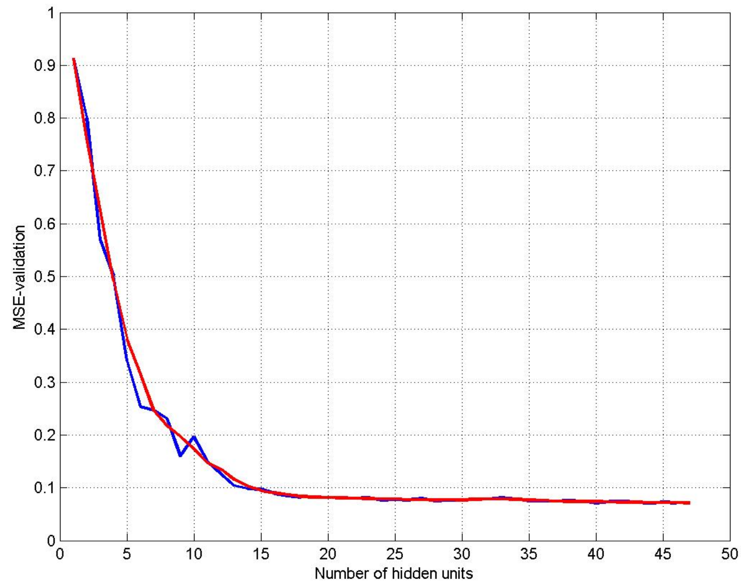

Figure 7.1: Validation error and smooth validation curves for *Gongtrn* file

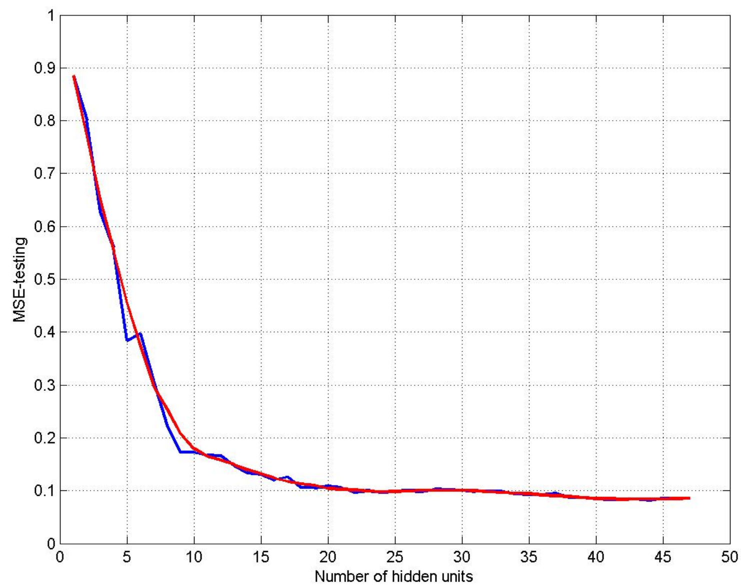

Figure 7.2: Testing error and smooth testing curves for *Gongtrn* file

In order to demonstrate that these noises are unimportant, we calculate the Pearson's correlation

coefficient ρ between E_v and $\overline{E_v}$ and similarly for E_{test} and $\overline{E_{test}}$ curves from figure 7.1 and figure 7.2 respectively. The correlation coefficient $\rho_{E_v,E_{test}}$ is given as [116]

$$\rho_{E_v,E_{test}} = \frac{\frac{1}{N_v}\sum_{k=1}^{N_h}[E_v(k) - \overline{E_v}(k)] \cdot [E_{test}(k) - \overline{E_{test}}(k)]}{\sqrt{[E_v(k) - \overline{E_v}(k)]^2} \cdot \sqrt{[E_{test}(k) - \overline{E_{test}}(k)]^2}} \tag{7.3}$$

From our experiments with the *Gongtrn* datafile on the MA algorithm with 100 iterations and 100 hidden units, we find that $\rho_{E_v,E_{test}} = 0.2$. Therefore, we observe that the testing and validation mean square errors are not correlated. From a signal processing point of view, the curves are non-stationary. The nonlinear median filter has the ability to preserve edges while filtering out any impulses. Therefore, we use it to smooth out the E_v vs N_h curve before pruning. Another property of median filter that we put to use is that the low window size preserves the overall signal shape. Therefore in current investigation, we take the window size to be of length 3.

7.0.16 Growing approach

While pruning of MLP reduces the network size, with large datasets, the pruning algorithms gets slower owing to more processing to prune large number of hidden units in the starting. One way to circumvent this progemblem is to take a growing approach and add hidden units on a certain validation error criterion. Growing offers an advantage over pruning in the sense that the training algorithm is relatively faster in the starting as we have less number of hidden units and the network usually settles down to a much lower stable structure. In our experiments, for each growing steps, we prune the number of hidden units and then calculate the validation error. We stop as soon as the validation error increases and save the network with best size.

7.0.17 Network Pruning

In an MLP, pruning is the process of removing less useful weights [100], [64] or hidden units [162], [125], [119], [83], [30], [29], [80], [25], in order to promote sparsity, prevent overtraining, and reduce testing errors. In this section, we use a hidden unit pruning method [114], [127], which is defined as follows.

Lemma 1: Given an MLP with no linearly dependent or unused hidden units, pruning alone cannot convert it to a functionally equivalent MLP [178], [82] of smaller size. If a useful hidden unit is removed from an MLP, there are no dependent hidden units to replace it, so that the pruned network is equivalent

Lemma 2: Given an MLP with no linearly dependent hidden units, that performs a nonlinear mapping, a functionally equivalent MLP can be constructed with a larger number of independent hidden units.

Proof: Given the target function $\mathbf{t}(\mathbf{x})$ and the hidden unit basis functions $O(k, \mathbf{x})$, for $1 \leq k \leq N_h$,

1. Generate a new hidden unit $O(N_h + 1, \mathbf{x})$ that is linearly independent of the other hidden units. This is possible, since otherwise, Covers theorem [38] is violated.
2. Train a new MLP that approximates $\frac{1}{2}[O(N_h, \mathbf{x}) + O(N_h + 1, \mathbf{x})]$ arbitrarily well [74] using $N_h^1 > 2$ linearly dependent hidden units. Use this new MLP's hidden units to replace unit $O(N_h, \mathbf{x})$ in the first MLP.
3. The resulting modified network approximates the first one arbitrarily well, using $N_h + N_h^1$ linearly independent hidden units.

Lemma 3: MLPs of many different sizes have virtually the same performance.

Proof: Given an acceptable MLP with Nh hidden units, many larger ones exist, and can be generated via ***lemma 2***.

An MLP training algorithm [143] denoted as MA-OR-R has been developed, which starts out with Nh_i hidden units. At every 3rd iteration, pruning with validation [127] is used is remove dependent or useless hidden units. The final MLP is more sparse than the initial network. From ***lemma 2*** and ***lemma 3***, we cannot guarantee that this pruning based approach yields the smallest network out of all those which are functionally equivalent. Therefore we can consider using MA-OR-R in an algorithm based upon both growing and pruning [39], [127].

7.0.18 Growing Versus Pruning

Basically, our goal is to run MA-OR-R using many initial Nh values $N_h^i(k)$ for $k = 1$ to k_f.

Two Approaches

In the pruning approach for applying MA-OR-R, $N_h^i(1)$, the initial number of hidden units is large. MA-OR-R is used to generate successively smaller values $N_h^i(k)$ for $k = 1$ to k_f . In the growing approach, $N_h^i(1)$ is small and the $N_h^i(k)$ values for $k > 1$ are successively larger values.

Characteristics of the Pruning Approach

The pruning approach has these characteristics.

1. The initial N_h, $N_h^i(1)$ is unknown, and it is not clear how to pick its value.

2. If $N_h^i(1)$ is too small, we cannot find $min(N_h)$ since only network with $N_h \leq N_h^i(1)$ can be generated.

3. If $N_h^i(1)$ is too large, the algorithm may generate smaller networks, but it may use many training iterations on large networks, resulting in computational complexity.

4. We can't guarantee that $min(N_h)$ is found because of ***lemma 3***.

Characteristics of the Growing Approach

The growing approach has these characteristics.

1. $N_h^i(1)$ can be 0 or some other small number. Because $N_h^i(k)$ can only increase, we can find a good value of $min(N_h)$, since there are no smaller values of N_h^i to try.

2. An early stopping related approach can be used. We can stop when the network for $N_h^i(k+1)$ is no better than that for $N_h^i(k)$.

3. We can add to the existing hidden units when $N_h^i(k)$ changes.

Table 7.3 presents the effect of median filter and combined growing and pruning approach on various datasets. The median filtering based pruning algorithm is denoted by MA-OR-R-MF and the combined growing and pruning approach by MA-OR-R-MF-GP.

Table 7.3: Pruning hidden units results on various MA improvements

Datasets	Initial N_h	MA-OR-R-MF	MA-OR-R-MF-GP
Gongtrn	30	23	10
Comf18	30	26	11
MNIST	200	158	146
SVHN	200	172	162
CIFAR-10	200	182	166
COVER-10	100	58	42
SECTOR	400	312	236
NEWS-20	300	228	148
Breast Cancer	100	68	32

We observe that the median filter approach results in smaller networks. The primary goal of using a median filter is to remove noise that might cause any impulse, thereby leading us to pick a larger network. The growing pruning algorithm clearly makes a very small size model. The primary reason for that being the growing starts with a smaller hidden units and eventually grows until it starts overfitting and stop. At each growing step we do median filtering based pruning so as to prune down the growing model. By adding median filtering and combined growing and pruning in MA-OR-R algorithm, we improve it to build an adaptive algorithm denoted by MA-OR-R-MF-GP. The

MA-OR-R-MF-GP algorithm can be summarized as follows:

Algorithm 16 MA-OR-R-MF-GP algorithm

1: Read the training data.
2: Randomly split off 30 % data of the training data to be validation data.
3: Calculate **R** and **C** using equation (3.6).
4: Solve equation (3.5) for the initial weight matrix $\mathbf{W}_o$.
5: Initialize N_{it}^3 , $i_t \leftarrow 0$, $\mathbf{C} \leftarrow 0$, input weight matrix **W** with Gaussian random numbers and Nh.
6: **while** $i_t < N_{it}^3$ **do**
7: Using equation (3.6), find the **R** matrix. Find the **C** matrix using OR.
8: Save the best sized $\mathbf{W}_o$ and $\mathbf{W}_i$ using MA-OR-R-MF algorithm.
9: **if** $\frac{E_v(i_t)-E_v(i_t-1)}{Nh(i_t))} < \epsilon$ **then**
10: Increase the number of hidden units at exponential rate.
11: **else**
12: Save the best $\mathbf{W}_o$, $\mathbf{W}_i$ and N_h.
13: **end if**
14: $i_t \leftarrow i_t + 1$
15: **end while**
16: Recombine validation and training data.
17: Recalculate **R** using equation (3.6) and **C** using $\mathbf{t}_p^{'}$.
18: Solve equation (3.5) for **W**.

Table 10.2 presents the 10-fold testing error results for the proposed modification of MA on various datasets.

Table 7.4: Comparision results for 10-fold testing accuracy results on single hidden layer MLP with $N_{it} = 100$

Datasets	MA-OR-R	MA-OR-R-MF	MA-OR-R-MF-GP
Gongtrn	7.2667 ±0.196	7.0333 ±0.157	7.001 ±0.152
Comf18	14.4901 ±0.001	14.3691 ±0.002	14.1487 ±0.001
MNIST	4.3461 ± 0.324	3.3116 ±0.221	3.2125 ±0.214
SVHN	4.0846 ± 0.210	4.0846 ± 0.247	3.8745 ± 0.124
CIFAR-10	6.2388 ± 0.229	6.0015 ± 0.210	6.998 ± 0.229
COVER	12.1333 ± 0.281	10.2147 ± 0.214	10.0258 ± 0.201
SECTOR	4.7896 ± 0.264	3.3475 ±0.214	3.2478 ± 0.211
NEWS-20	10.2478 ± 0.011	8.5475 ± 0.018	8.4127 ± 0.017
Breast Cancer	9.8771 ± 0.210	9.2458 ±0.278	9.2222 ± 0.219

Ideally, the use of median filter should either lead to a lower testing error or it should not change the testing error at all. If we compare the testing accuracy for MA-OR-R-MF-GP with rest of the algorithms, we observe that the proposed method has superior performance for all the datasets. This is an encouraging result since MA-OR-R-MF-GP comes with an added advantage of generating models with optimal number of hidden units. The 10-fold testing accuracy clearly suggest that

MA-OR-R-MF-GP is the best algorithm we've developed up to this point.

7.0.19 Improvements in inputs in MA training algorithms

The output reset (OR) algorithm discussed in chapter V eliminates inconsistent errors of problem (L1) but is not designed to eliminate the consistent errors of problem (L2) that can become outliers. In this section we develop a method for bounding consistent errors. We then formulate second order algorithms for modifying input units.

Initialization

At this point, the MA-OR-R algorithm uses regression and OR to design MLP classifiers with errors satisfying

$$0 \le t_p^{'}(i_c) - y_p(i_c) < \infty \tag{7.4}$$

and

$$-\infty \le t_p^{'}(i_d) - y_p(i_d) < 0 \tag{7.5}$$

In other words, inconsistent errors are bounded but consistent errors are unbounded and can still cause problems. Platt [138] has shown the utility of using non linear output activations such as sigmoids. We therefore choose to limit both consistent and inconsistent errors in a new training stage by using sigmoidal input activations to bound network inputs as shown in figure 7.3. The error for the linear classifier is

$$E_{sig} = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - O_p(i)]^2 \tag{7.6}$$

where the classifier output $\mathbf{y}_p$ is given as

$$y(i) = \sum_{n=1}^{N+1} w(i,n) \cdot z(n) \tag{7.7}$$

where

$$z(n) = f(a(n) \cdot x(n) + b(n)) \tag{7.8}$$

where $f(\cdot)$ denotes the sigmoid activation function. E_{sig} can be larger than $E^{'}$ in (2.2) because $y_p(i)$ equal to 0 or 1 results in $z(i)$ equal to $\frac{1}{2}$ or $\frac{1}{1+e}$ respectively. In order to mitigate this increase in E_{sig}, we map the linear input activations $\mathbf{x}$ to sigmoidal outputs $\mathbf{z}$ in equation (7.8)

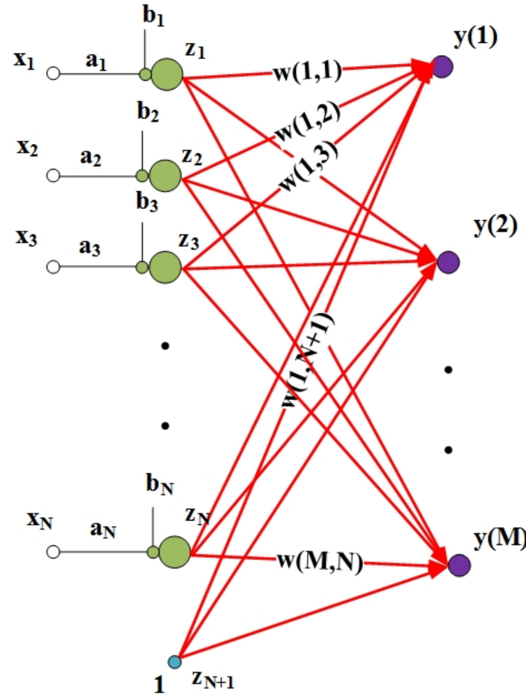

Figure 7.3: Input sigmoids in linear network

where **a** and **b** are N dimensional vectors that are found such that E_{sig} is minimized. We choose to find the values of a and b using Newton's method. To initialize, we use

$$a(n) = \frac{1}{\epsilon + \sigma(n)}, \quad b(n) = a(n) \cdot m(n) \quad where \quad \epsilon = 0.001 \tag{7.9}$$

we use,

$$t_p(i) = \delta(i - i_c(p)) \tag{7.10}$$

and use OWO along with it.

Gradient Calculations

Given the error function in equation (7.6), we calculate the negative gradient vector $\mathbf{g}_{ab} = [g_a : g_b]^T$. Here the negative partial derivative g_a is calculated as

$$g_a = -\frac{\partial E}{\partial a} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p^{'}(i) - y_p(i)]\frac{\partial y_p(i)}{\partial a} \tag{7.11}$$

and g_b is calculated as

$$g_b = -\frac{\partial E}{\partial b} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p^{'}(i) - y_p(i)]\frac{\partial y_p(i)}{\partial b} \tag{7.12}$$

We know that

$$z(n) = f([a(n) + z_a \cdot g_a(n)]x_n + [b(n) + z_b \cdot g_b(n)]) \tag{7.13}$$

Applying MOLF to calculate the learning factor $\mathbf{z} = [z_a : z_b]^T$, the Hessian matrix elements will be

$$h_{aa} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}(\frac{\partial y_p}{\partial z_a})^2 \tag{7.14}$$

$$h_{ab} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\frac{\partial y_p}{\partial z_a}\cdot\frac{\partial y_p}{\partial z_b} \tag{7.15}$$

$$h_{bb} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}(\frac{\partial y_p}{\partial z_b})^2 \tag{7.16}$$

The Hessian matrix is now assembled as:

$$\mathbf{H} = \begin{bmatrix} h_{aa} & h_{ab} \\ h_{ab} & h_{bb} \end{bmatrix}$$

The learning factor vector $\mathbf{z}$ is obtained by solving the following linear equations

$$\mathbf{H}\cdot\mathbf{z} = \mathbf{g}_{ab} \tag{7.17}$$

The scaling factor $\mathbf{a}$ and $\mathbf{b}$ are then updated as

$$\mathbf{a} \leftarrow \mathbf{a} + z_a \cdot \mathbf{g}_a \tag{7.18}$$

$$\mathbf{b} \leftarrow \mathbf{b} + z_b \cdot \mathbf{g}_b \tag{7.19}$$

We denote this Newton-based input sigmoid algorithm as IS algorithm, which is described below

Algorithm 17 IS algorithm

Read the training data.
Calculate $\mathbf{R}$ and $\mathbf{C}$ using equation (3.6).
Solve equation (3.5) for the initial weight matrix $\mathbf{W}$.
Initialize N_{it}^{1} , $i_t \leftarrow 0$, $\mathbf{C} \leftarrow 0$
while $i_t < N_{it}^{1}$ **do**
  for $p = 1$ to N_v **do**
    Calculate $\mathbf{g}_a$ and $\mathbf{g}_b$.
  end for
  for $p = 1$ to N_v **do**
    Calculate h_{aa}, h_{ab}, h_{bb}.
  end for
  Solve for $\mathbf{z}$ using equation (7.17) and update $\mathbf{a}$ and $\mathbf{b}$.
  for $p = 1$ to N_v **do**
    Use OR algorithm to process $\mathbf{t}_p$ into $\mathbf{t}_p^{'}$.
    Update $\mathbf{C}$ by replacing $\mathbf{t}_p$ by $\mathbf{t}_p^{'}$ in equation (3.6) as $\mathbf{C} \leftarrow \mathbf{C} + \mathbf{x}_{ap}\ (\mathbf{t}_p^{'})^T$
    Perform OWO.
  end for
  $i_t \leftarrow i_t + 1$
end while

Table 10.2 presents the results for the proposed modification of MLP on various datasets. We also present the performance of median filter based pruning and compare it with the best reported results. We can view input sigmoids is a type of normalization that converges the MLP quickly, so overall training is fast. Since the inputs are now bounded well within the limits, the network weights are easier to initialize and therefore it helps to reduce the sensitivity towards initial starting weights. By shifting and scaling the inputs, we actually introduce a little noise into the model, and hence the input sigmoids provides some regularization.

7.0.20 Final algorithm

We combine the IS algorithm with MA-OR-R-MF-GP algorithm to give our final algorithm denoted by MA-OR-R-MF-GP-IS algorithm, which is described below.

Algorithm 18 MA-OR-R-MF-GP-IS algorithm

1: Read the training data.
2: Find a and b using IS and re-scale the training data.
3: Use the rescaled data to save the best sized $\mathbf{W}_o$ and $\mathbf{W}_i$ using MA-OR-R-MF-GP algorithm.

If we compare the testing accuracy, we observe that the proposed method has superior performance for all the datasets. The 10-fold testing accuracy clearly suggests that MA-OR-R-MF-GP-IS is the

Table 7.5: Comparision results for 10-fold testing accuracy results on single hidden layer MLP with $N_{it} = 100$

Datasets	MA-OR-R-MF-GP	MA-OR-R-MF-GP-IS
Gongtrn	7.001 ±0.152	6.5789 ± 0.179
Comf18	14.1487 ±0.001	14.015 ±0.001
MNIST	3.2125 ±0.214	2.8965 ± 0.112
SVHN	3.8745 ± 0.124	3.2478 ± 0.127
CIFAR-10	6.998 ± 0.229	6.2147 ± 0.227
COVER	10.0258 ± 0.201	9.1245 ± 0.201
SECTOR	3.2478 ± 0.211	2.8789 ± 0.214
NEWS-20	8.4127 ± 0.017	8.2478 ± 0.019
Breast Cancer	9.2222 ± 0.219	9.1212 ± 0.217

best algorithm we've developed.

Chapter 8

Investigation and improvements in deep autoencoders

We now incorporate all the improvement mentioned above into the deep neural network. During the autoencoder training, we prune each layer thereby doing feature selection. Pruning with the median filter ensures that the learning do not go into the saturation area of sigmoid thereby giving us an overall improved feature generation and selection method. The improved initialization scheme ensures that the reconstruction error is small and the compressed features are able to re-construct back. The multi-class OR algorithm, helps us to classify the misclassified patterns and making sure that none of the desired output remain in the linear region of sigmoid. this helps us to give a good performance even if we use a mean square error criterion and a sigmoid activation function. Usually for classification, cross entropy and softmax are preferred. We are working more towards finding better algorithm that can surpass the performance of cross entropy based models.

While utilizing the improvements from the linear classifier we try to address an open problems in MLP based neural networks which is to determine the depth of the deep learner. In a typical greedy layer-wise training of deep learner [15], we train the first layer for a particular number of iterations or if it's within acceptable reconstruction error limits and then add a second layer to start the training again and so on. On insight, the features obtained from the trained autoencoder are good for reconstruction but since we want to classify these high level features, we go the required depth to which this is feasible. In our work, we try to have a methodology where we automatically choose the number of layers by using a linear probe. The linear classifier as described in chapter II are used as linear probe to determine the depth of the deep learning network as shown in figure 8.1. At each stage, the hidden units features are feed as input to a linear probe and the desired output are

the labels. We calculate the feature classification error and stop when it increases. This gives us a good indication as to whether we need to add more layer of autoencoder. The layer just before the classification error increases is the required depth. Adding to this, we also have a maximum depth limit set in case the linear probe. Although it works well for a good number of datasets, it suffers from two major problems, firstly, it requires the labeled data for classification error and secondly, the stopping criterion is not meet sometime causing it to create an extremely deep network that brings its own computational burden with it.

Alternatively we also experimented with the nonlinear probe where we feed the hidden units features into a non linear MLP to observe the MSE. We prefer using a linear probe over it because of speed considerations and the intention here is to compare the MSE with the previous layer so as to decide to add more trained layer of auto encoder.

In summary, we develop compact stacked autoencoder that are pruned to generate independent features. We feed these features to better classification scheme in the supervised training step to give a lower generalization error.

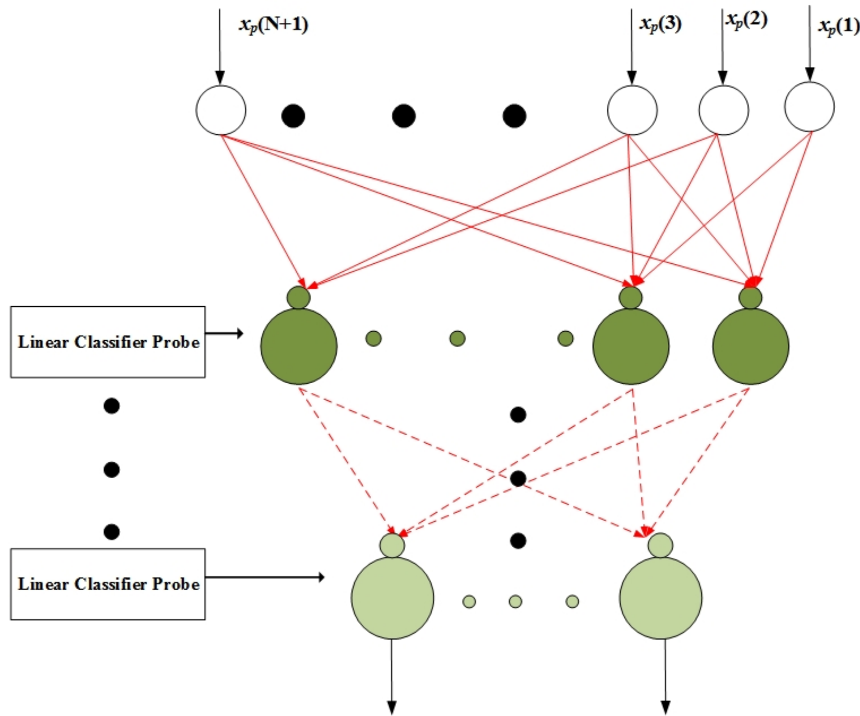

Figure 8.1: Linear probe in a deep neural network

8.0.21 Linearity in deep learning features

In this subsection, our goal is to develop tools to evaluate the quality of features obtained through various deep autoencoders. Our task is to investigate the linearity of the features that are obtained from training stacked autoencoder. We are motivated by two results regarding autoencoder training. In the first reported result [9] we see that if we use an autoencoder, without any non linearity and a squared error loss, the autoencoder essentially perform KLT. Secondly as reported in [22], even when a nonlinearity such as sigmoid us used in the encoder, obtaining KLT subspace is a likely possibility since its possible to stay in the linear region of sigmoid. We therefore inspect the features obtained from SAE and DAE where they are stacked to form the deep neural network. In this study, we inspect the features with a series of experiments to verify our claim about linearity. For all our experiments, we use MNIST dataset therefore, the linearity of the features that we report is with respect to the MNIST.

Linear mapping of inputs to autoencoder features

In our first experiment, our goal is to see if the entire stacked autoencoder pipeline can be replaced by a linear network. In order for it to work well, there should be a linear mapping between the raw inputs and the features from the last layer of the trained stacked autoencoder. We then use the testing data to produce features that are used as input to the fine tuned classifier to measure the testing accuracy.

We formalize our proposition that the features obtained from SAE and DAE are linear as follows

Algorithm 19 Linearity check for deep autoencoder features

1: Train each autoencoder in a greedy unsupervised method. Stack them and complete the training with fine-tuning.
2: Pass the training data through the trained autoencoder layers and get the output feature vector from the last layer of the auto encoder (excluding the classifier). If l is the last layer of autoencoder then the feature vector will be $\mathbf{O}_p^l$.
3: Train a linear network with training input data as $\mathbf{x_p}$ and target output as $\mathbf{O}_p^l$.
4: Put the testing data through the trained linear network to obtain feature vector $\mathbf{O}_p^{linear}$.
5: Feed the output of the linear network $\mathbf{O}_p^{linear}$ to the classifier that is trained and fine tuned in Step 1.
6: Measure the testing accuracy.

Based on the algorithm 19, we present the experimental results in Table 8.1. The sparse deep autoencoder and the denoising deep autoencoder have softmax classifier and cross entropy error function as described in subsection 4.1.6 and our proposed deep autoencoder has MSE error function and MLP classifier with initialization through Newton's method as described in section Chapter V.

Table 8.1: Linearity test on features from deep autoencoders for MNIST dataset

Deep autoencoder model	Deep autoencoder features	Linear network features
Sparse deep autoencoder	1.6213	1.9852
Denoising deep autoencoder	1.2858	2.0719
Proposed deep autoencoder	3.6124	3.8247

We observe from Table 8.1 that the accuracy of the linear network features is very similar to the deep autoencoder features. This observation implies that the features obtained from the deep autoencoder can be mapped by a linear network. This is an interesting observation since even though the deep autoencoder features are obtained through a hierarchical structure, there replaceability by a linear network strongly suggests that the features are in a similar subspace as KLT. Since the accuracy is not exactly same as by the deep autoencoders, one plausible explanation can be that the specific projection direction of these features is in general not correspond to the actual principal direction and it may not need be orthonormal.

Identifying linearity using coefficient of determination

In order to understand more about the features obtained by training a linear network as in previous subsection, we apply the idea of coefficient of determination (COD) or R^2. COD is a measure that allows us to determine how certain one can be in making predictions from a certain model. The

coefficient of determination is such that $0 \leq R^2 \leq 1$. It denotes the strength of the linear association between two features obtained from trained autoencoders and the input vectors. COD represent the percentage of the data that is closest to the line of best fit. For example if COD = 0.9580, that means 95 % of the total variation in one feature can be explained by the linear relationship between the two features. For i^{th} output we calculate COD as follows,

$$COD(i) = 1 - \frac{MSE(i)}{\sigma^2(\mathbf{O_p}^l)} \tag{8.1}$$

Table 8.2 illustrates the mean COD value for all the outputs.

Table 8.2: COD for deep autoencoders for MNIST dataset

Deep autoencoder model	first layer COD	second layer COD	third layer COD
Sparse deep autoencoder	0.9992	0.9996	0.9998
Denoising deep autoencoder	0.9918	0.9921	0.9935
Proposed deep autoencoder	0.9924	3.8247	0.9975

We see from Table 8.2 that almost all the features in each stage of the deep autoencoder are linear and can be mapped by a linear network. This is an extremely important result in terms of the feature quality. It also indicates that level of abstraction in deep learning features from deep autoencoders need to be investigated in more detail.

8.0.22 Analysis of sparse autoencoders

In order to analyze more about the effect of regularization and sparsity parameter in deep sparse autoencoders [130], we experimented with various combinations of the parameters that were used in deep sparse autoencoder. Table 8.3 tabulates the results for the experiment.

Table 8.3 gives us surprisingly intriguing results. We see that the model has very low effect of regularization and sparsity on the final testing accuracy and it is the fine tuning that gives the model a boost in the performance.

Table 8.4 describes the results of combining various deep autoencoders with different classifiers.

Table 8.3: Linearity experiments on MNIST dataset with Deep sparse autoencoder

Model parameters	Pe_{test} (w/o fine tuning)	Pe_{test} (with fine tuning)
$\lambda = 0$, $\lambda_{softmax}= 0$, $\beta= 0$ $\rho =0$	87.00	97.30
$\lambda \neq 0$, $\lambda_{softmax}= 0$, $\beta= 0$ $\rho =0$	89.27	97.41
$\lambda \neq 0$, $\lambda_{softmax} \neq 0$, $\beta= 0$ $\rho =0$	73.76	98.39
$\lambda \neq 0$, $\lambda_{softmax}= 0$, $\beta \neq 0$ $\rho \neq 0$	**94.02**	98.24
$\lambda = 0$, $\lambda_{softmax}=0$, $\beta \neq 0$ $\rho \neq 0$	93.32	97.97
$\lambda = 0$, $\lambda_{softmax} \neq 0$, $\beta \neq 0$ $\rho \neq 0$	92.03	**98.22**
$\lambda = 0$, $\lambda_{softmax} \neq 0$, $\beta= 0$ $\rho= 0$	74.76	98.26
$\lambda \neq 0$, $\lambda_{softmax} \neq 0$, $\beta \neq 0$ $\rho \neq 0$	92.03	98.38

Table 8.4: 10 fold testing accuracy with various combinations of features and classifiers for MNIST dataset

Feature + Classifier	Initial configuration	Pe_{test} (w/o fine tuning)	Pe_{test} (with fine tuning)
Sparse AE + Linear Classifier	784-200-200	10.3213	5.9712
Sparse AE + MLP Classifier	784-200-200	8.1611	2.7222
Sparse AE + Softmax Classifier	784-200-200	7.9701	1.6213
DAE + Linear Classifier	784-1000-1000-1000	8.9613	5.6387
DAE + MLP Classifier	784-1000-1000-1000	7.3127	3.2162
DAE + Softmax Classifier	784-1000-1000-1000	4.8420	1.2858
proposed deep AE + Linear Classifier	784-200-200	4.9981	——
proposed deep AE + MLP Classifier	784-200-200	3.6124	——
proposed deep AE + Softmax Classifier	784-200-200	2.8128	0.9713

We see from Table 8.4 that the MLP and linear classifier has a comparable performance and we are currently working to improve it's performance.

8.0.23 Linear classifiers as probe

In order to determine the depth of the deep neural network, we use linear classifiers as probe that measure the validation error P_{ev}. We keep adding the layers until probability of validation error P_{ev} fails to decrease. Table 8.5 shows the results with the linear probe as well as with pruning in each hidden layer.

Table 8.5: 10 fold testing accuracy results on improvements with pruning and linear probe for proposed deep AE

Datasets	Initial configuration of AE	Configuration after pruning	Pe_{test} (initial configuration)	Pe_{test} (after pruning)
Gongtrn	16-200-200-200	16-151-172	6.1110	4.8521
Comf18	18-300-300-200	18-168-190	11.7844	13.6714
MNIST	784-400-400-300	784-187-201-243	4.8981	3.6124
Google-SV	1024-500-500-500	1024-387-424-428	2.8413	2.1143
CIFAR-10	1024-500-500-500	1024-358-462-442	6.8524	4.8674
STL-10	9216-4000-1000-500-300	9216-3416-880-488	26.1424	25.8674

We finally present our results with better initialization scheme and improved pruning in Table 8.6.

Table 8.6: 10 fold testing accuracy of proposed deep autoencoder

Datasets	LC	Single layer MLP	Proposed deep AE	Sparse AE	Denoising AE	Best-result reported
Gongtrn	9.4667	7.9406	4.8521	4.9991	4.9001	——
Comf18	17.1843	15.1117	3.6714	14.7113	14.8527	——
MNIST	5.0765	5.0765	3.6124	1.6213	1.2858	0.2100
Google-SV	4.4058	3.9671	2.1143	1.8824	1.8876	1.6900
CIFAR-10	8.4106	5.8187	4.8674	4.1621	4.6140	3.4609
STL-10	30.6715	27.8619	25.8644	26.8924	25.9940	25.6712

As we see that our proposed deep autoencoder has comparatively lower accuracy, however, it is not as different from the other two deep autoencoders. The best result reported is consisting of algorithms that are based on various other deep learning variants. We enumerate the reasons for our lagging performance below:

1. The validation data is not combined to get a final trained network. It means that we are

training with less number of patters as compared to other algorithms.

2. We have not included fine-tuning in our model. This implies that our model performance is compared with other algorithms that already have been fine-tuned. If we remove the fine-tuning and compare the performance, then our proposed deep autoencoder is far better than other algorithms.

Chapter 9

Proposed Auto encoder theory

Based on the theory of [74], [22], we now show a formal proof as to how the features of AE are linear and equivalent of a KLT mapping.

9.0.24 Proof for linearity of AE features

Theorem 1: *An Auto encoder with zero valued training error (E_{tr}) is linear.*
Proof: Let each auto encoder layer include the constant basis function as O$(N_h$+1) = 1 and $y(N$+1) = 1. Here $\mathbf{y} \in \Re^{(N+1)\times 1}$, $\mathbf{W_o} \in \Re^{(N+1)\times(N_h+1)}$ and $\mathbf{O} \in \Re^{(N_h+1)\times 1}$ and $N_h < N$. Now $\mathbf{O}$ can be found from $\mathbf{y}$ as

$$\mathbf{O} = \mathbf{B} \cdot \mathbf{y} \qquad where \qquad \mathbf{B} = (\mathbf{W_o^T W_o})^{-1}\mathbf{W_o^T} \tag{9.1}$$

Now $rank(\mathbf{B}) = (N_h + 1)$. Since $E_{tr} = 0$ means $\mathbf{y} = \mathbf{x}$ and $\mathbf{O} = \mathbf{B} \cdot \mathbf{x}$ so $\mathbf{O}$ is a linear function of $\mathbf{x}$

Theorem 2: An autoencoders has an infinite number of solutions
Proof: The net function vector for the hidden units is as follows:

$$\mathbf{O} = \mathbf{A} \cdot \mathbf{W} \cdot \mathbf{X} \tag{9.2}$$

where $\mathbf{A}$ is any $N_h \times N_h$ non singular matrix. The input weight matrix is $\mathbf{A} \cdot \mathbf{W}$ The actual output $\mathbf{y}$ can be written as

$$\mathbf{y} = \mathbf{W_o} \cdot \mathbf{A} \cdot \mathbf{W} \cdot \mathbf{X} \tag{9.3}$$

Since the number of inputs is equal to the number of outputs and from Theorem 1, we know that :

$$\mathbf{W_o} \cdot \mathbf{W} \cdot \mathbf{X} = \mathbf{B} \cdot \mathbf{A} \cdot \mathbf{W} \cdot \mathbf{X} \tag{9.4}$$

This leads to

$$\mathbf{W_o} = \mathbf{B} \cdot \mathbf{A} \tag{9.5}$$

therefore the output weight matrix is $\mathbf{B}$ is

$$\mathbf{B} = \mathbf{W_o} \cdot \mathbf{A^{-1}} \tag{9.6}$$

Since there are infinite number of $\mathbf{A}$ matrices, therefore it can have infinite number of solutions.

Theorem 2: As from [47], optimal features for compressing $\mathbf{x}$ that satisfy information storage property are not optimal for classifying $\mathbf{x}$ that do not minimize the classification accuracy.

Training a deep learner is a two stage process. When we train each autoencoders, that is approximately linear and has infinite number of solutions which are all equally good for reconstruction of the inputs. This is compression for reconstruction and not compression for classification. As we connect and train the classifier keeping the input weights fixed for the autoencoders, the classifier learns how to convert the linear feature of compression for approximation into linear feature of compression for classification for a minimum classification error. Later fine tuning enables it to improve further. We can break down the training of autoencoder in two stages.

Stage 1 : First compress the inputs. without destroying useful information. As from information storage property, this is a linear operation. After training an autoencoder, we get a KLT matrix times a non singular matrix of size $N_h \times N_h$. It has infinite number of equally good solutions.

Stage 2: In second stage we want a feature extraction that is good for classification. It is initially achieved by using softmax classifier and then fine tuning. This stage helps the stacked autoencoder to change it's linear transformation and weight change in softmax classifier. In essence, this stage converts the features for compression to the features for classification.

In summary, our goal is to generate feature that are good for classification in two stages. First it's compression and then linearly convert those features for classification. We do compression first since its easy and you squeeze a lot of information, although your real goal is to classification.

9.0.25 KLT and Linear Mapping

An important question that arises here is that if an autoencoders is linear then why does it have stacked layers ?

We now explain the reason as follows. As in equation (()9.2), we don't have $\mathbf{A}$ and $\mathbf{W}$ separately, so if we try to delete one row of $\mathbf{A} \cdot \mathbf{W}$, we still have the same matrix. On the side note, the entire $\mathbf{A} \cdot \mathbf{W}$ matrix is damaged if $\mathbf{W}$ was a real KLT matrix. we don't have the KLT matrix separately so when we delete row from $\mathbf{A} \cdot \mathbf{W}$, that's not equal to deleting the row from $\mathbf{W}$. $\mathbf{A} \cdot \mathbf{W}$ has same reconstruction capability as $\mathbf{W}$. Since we have $\mathbf{B} \cdot \mathbf{W} \cdot \mathbf{A^{-1}}$, if we delete any row for $\mathbf{A} \cdot \mathbf{W}$ to make it $\mathbf{A}^{'} \cdot \mathbf{W}^{'}$ we change all row for $\mathbf{W}$.ie

$$\mathbf{A}^{'} \cdot \mathbf{W}^{'} \neq \mathbf{A} \cdot \mathbf{W} \tag{9.7}$$

Now a logical question can be that if an autoencoders is simply a KLT transform then why not delete features from each layer.?

This is because throwing away hidden units from an auto encoder is not going to give a KLT. That is why an autoencoders change the number of features to get the minimum training error. Each layer of autoencoders is linearly transforming the KLT matrix, i.e. each auto encoder is designing its own $\mathbf{A} \cdot \mathbf{W}$. If we have $\mathbf{W}$ then we know how to remove the features by deleting the last row ot if but since we don't have $\mathbf{W}$ we are unable to do it. An autoencoders designs the $\mathbf{A} \cdot \mathbf{W}$ matrix and not $\mathbf{W}$ matrix. In the second layer of the autoencoders, we transfer the information of $\mathbf{A} \cdot \mathbf{W}$ by multiplying it by a new matrix that reduce the feature size vector as

$$\mathbf{A}^{'} \cdot \mathbf{W_{reduced}} \tag{9.8}$$

where $\mathbf{W_{reduced}}$ is a reduced size KLT matrix.

Chapter 10

Experimental results

In this section, we first compare 10 fold testing performances of GLC-Newton with those of linear classifiers such as linear SVM [44] [86], SDM-CS [75], sigmoidal SVM (Sig-SVM) [138], and with the IRLS generalized linear classifier [151].

Table 10.1: 10-fold testing error % results for GLC-Newton and other algorithms

Datasets	GLC-Newton	LIBLINEAR	IRLS	SDM-CS	Sig-SVM
Gongtrn	9.4567 ± 0.004	13.8674 ± 0.001	12.5789 ± 0.004	13.7895 ± 0.004	11.0254 ± 0.007
Comf18	17.1813 ± 0.27	25.6671 ± 0.34	21.8746 ± 0.29	20.0145 ± 0.32	21.0478 ± 0.31
MNIST	4.0775 ± 0.15	9.4306 ±0.18	5.2874 ± 0.29	8.2472 ± 0.42	7.0874 ± 0.17
SVHN	4.4158 ± 0.22	9.8753 ± 0.35	5.2864 ±0.26	8.2578 ± 0.31	7.3278 ± 0.13
CIFAR-10	8.4176 ± 0.85	11.0483 ± 0.79	10.2578 ± 0.94	10.8745 ± 1.01	10.7954 ± 0.92
COVER	15.0121 ± 0.47	19.2342 ± 0.59	18.2302 ± 0.54	18.3109 ± 0.78	16.1487 ± 0.25
SECTOR	6.1120 ± 0.99	6.1998 ± 1.02	7.2138 ± 1.09	7.9873 ± 0.97	6.1333 ± 1.11
RCV1	8.1203 ± 0.72	10.1245 ± 0.82	11.3958 ± 0.94	11.2344 ± 0.81	9.2547 ± 0.65
NEWS-20	13.6705 ± 0.19	15.2875 ± 0.24	14.7894 ± 0.21	15.1785 ± 0.22	14.8794 ± 0.17
Breast Cancer	10.9382 ±0.25	15.2472 ± 0.64	14.5873 ± 0.32	13.4785 ± 0.01	14.9871 ± 0.58

From Table 10.1, we see that GLC-Newton consistently performs better than the other techniques. For smaller dimensional, uncorrelated datasets (*Gongtrn* and *Comf18*) GLC-Newton gives the best performance. One reason of the better performance for GLC-Newton is that the pruning algorithm removes useless inputs that damage the performance of the other algorithms. For object recognition datasets (*MNIST*, *SVHN*, *CIFAR-10*), linear dataset (*Breast Cancer*) and text recognition

datasets (*RCV1* and *NEWS-20*), GLC-Newton performs better due to the second order algorithm that suppresses *consistent* and *inconsistent* errors.

Next, we compare the 10 fold testing performance of MA-OR-R-MF-GP-IS with non-linear classifier such as softmax classifier [130], standard MLP [66], non linear SVM [37] and RBF classifier [171]. We later use MOLF-Adapt-MF as a classifier in a deep learning architecture for various datasets.

10.0.26 Shallow classifiers

While comparing MA-OR-R-MF-GP-IS with other nonlinear classifiers, we set the initial hidden units as 2 for all the datasets. For MA-OR-MF-GP-IS, we used 30% of training data as validation data.

Table 10.2: Comparision results for 10-fold testing % results for MA-OR-R-MF-GP-IS with comparing algorithms

Datasets	MA-OR-R- -MF-GP-IS	Softmax	Nonlinear SVM	RBF	CG
Gongtrn	6.5789 ± 0.179	9.1578 ± 0.145	14.2231 ±0.104	8.8652 ±0.121	8.4785 ± 0.147
Comf18	14.015 ± 0.001	18.6321 ±0.001	21.3429 ±0.002	19.2283 ±0.010	18.9871 ± 0 006
MNIST	2.8965 ± 0.112	7.4503 ±0.221	11.2346 ±0.227	9.9987 ± 0.229	8.9654 ±0.247
SVHN	3.2478 ± 0.127	6.9541 ±0.197	10.2312 ± 0.139	9.3428 ± 0.167	8.7891 ± 0.167
CIFAR-10	6.2147 ± 0.227	8.3328 ±0.297	13.5453 ± 0.671	11.2396 ± 0.347	10.9987 ± 0.336
COVER	9.1245 ± 0.201	11.1478±0.214	14.2487 ± 0.210	12.4785 ± 0.241	12.1478 ± 0.200
SECTOR	2.8789 ± 0.214	3.8978 ± 0.252	4.2141 ± 0.201	4.2017 ± 0.201	4.1245 ± 0.222
NEWS-20	8.2478 ± 0.019	10.2547 ± 0.014	12.0214 ± 0.014	10.3257 ± 0.012	10.9874 ± 0.017
Breast Cancer	9.1212 ± 0.217	10.1114 ± 0.214	13.2547 ± 0.211	13.0101 ± 0.210	12.8795 ± 0.214
Claimbuster	14.2158± 0.024	14.9874 ± 0.021	17.6589 ± 0.021	16.1248 ± 0.022	15.6987 ± 0.021

Simulation results show that MA-OR-R-MF-GP-IS generalizes much better than other comparable algorithms. This is because the proposed algorithm offers a model flexibility that adapts itself to capture the inherent data structure.

10.0.27 Deep learning classifiers

We now demonstrate the MA-OR-MF-GP-IS algorithm performance as replacement classifier in deep learning framework and compare it with pre-existing classifiers in stacked autoencoder [102],

convolutional network [98]. For *claimbuster* dataset [63], features are extracted using Alchemy API [**?**]. In our experiment, we use the claimbuster data along with a deep stacked autoencoder that we designed for this problem. We pre-process the data by removing the zero column feature vector and then splitting the data into 80%, 10% and 10% as training, validation and testing respectively. We use principal component analysis to reduce the input dimension from 5413 to 1000. We compare the above results with deep autoencoder (DAE) along with softmax classifier as in [102]. For our final experiment, we use MA-OR-R-MF-GP algorithm to built and prune the deep autoencoder (DAE-prune). A 2 hidden layer deep stacked autoencoder with 800 and 500 hidden units in first and second layer respectively is trained for 500 iterations. The deep learner features are then feed into the proposed MA-OR-R-MF-GP-IS classifier with 300 hidden units. Pruning the hidden units makes a 232 hidden units. The results are presented in Table 10.5. Table 10.4 presents the results.

The precision, recall and F-measure value are given in Table 10.3

Table 10.3: 5-fold testing error % results for claimbuster dataset using SVM

	Precision	Recall	F-measure
NFS	0.90	0.96	0.93
UFS	0.65	0.26	0.37
CFS	0.79	0.74	0.77

For claimbuster dataset [63], features are extracted using Alchemy API and SVM is used as classifier. In our experiment, we use the claimbuster data along with a deep stacked autoencoder that we designed for this problem. We pre-process the data by removing the zero column feature vector and then splitting the data into 80%, 10% and 10% as training, validation and testing respectively. We use principal component analysis to reduce the input dimension from 5413 to 1000. We compare the above results with deep autoencoder (DAE) along with softmax classifier as in [102]. Table 10.4 presents the results.

Table 10.4: 5-fold testing error % results for claimbuster dataset using DAE-Softmax

	Precision	Recall	F-measure
NFS	0.92	0.96	0.90
UFS	0.45	0.42	0.44
CFS	0.80	0.81	0.78

For our final experiment, we use MA-OR-R-MF-GP-IS algorithm to built and prune the deep autoencoder (DAE-prune). A 2 hidden layer deep stacked autoencoder with 800 and 500 hidden units in first and second layer respectively is trained for 500 iterations. The deep learner features are then feed into the proposed MOLF-Adapt-MF classifier with 300 hidden units. Pruning the hidden units makes a 232 hidden units. The results are presented in Table 10.5.

Table 10.5: 5-fold testing error % results for claimbuster dataset using DAE-MA-OR-R-MF-GP-IS

	Precision	Recall	F-measure
NFS	0.95	0.94	0.95
UFS	0.57	0.66	0.61
CFS	0.85	0.83	0.84

Unlike claimbuster, rest of the datasets have balanced classes. For *MNIST* [99], we divided the dataset into 60,000 training samples out of which 12,000 samples are used in validation. A test set of 10,000 samples ar used as testing data. A 2 layer deep stacked autoencoder (DAE) with 200 hidden units in each layer, is trained with 300 iterations and the MA-OR-R-MF-GP-IS algorithm with 200 hidden units is used to classify. After training, the features in autoencoder are pruned to 182 and 162 in first and second layer respectively. Similarly hidden units are pruned to 158 in MA-OR-R-MF-GP-IS classifier. A 2 layer deep stacked autoencoder described in [102] with 200 hidden units in each layer is used with softmax classifier for comparison. In Table 10.6, we see that DL based on MA-OR-R-MF-GP-IS classifier performs slightly better than the comparing softmax classifiers. One reason only a slight improvement is due to the linearity in the CNN and DAE features that are feed into the classifier. Similar results are obtained for Google street view housing numbers (*SVHN*) [129], *CIFAR-10* dataset [90] and *Scrap* dataset.

For the object detection datasets *MNIST, SVHN, CIFAR, Scrap*, a pre-trained Inception v3 [168] is used in Keras [33]. Inception v3 is a deep CNN model that is trained from the 2012 ImageNet large visual recognition challenge. This network is trained with mini-batch gradient descent with the AdaDelta optimizer.

Table 10.6: 10-fold testing error % results using DAE-prune-MOLF-Adapt-MF

Dataset	Inception v3-Softmax	Inception v3-MA-OR-R-MF-GP-IS	DAE-Softmax	DAE-MA-OR-R-MF-GP-IS
MNIST	0.9814±0.122	0.9718± 0.134	1.9857 ± 0.041	1.0434 ± 0.041
SVHN	2.5478 ± 0.113	1.2478 ± 0.132	2.9871 ± 0.267	2.4478 ± 0.362
CIFAR-10	5.2574±0.247	5.12478 ±0.127	5.7478 ± 0.217	5.2147 ± 0.281
Scrap	12.3587 ±0.141	9.9775 ± 0.154	12.9217 ± 0.322	12.8049 ± 0.128
COVER	NA	NA	8.6274 ± 0.200	8.4210 ± 0.200
SECTOR	NA	NA	1.4458 ± 0.219	1.2147 ± 0.222
NEWS-20	NA	NA	7.6581 ± 0.104	7.3478 ± 0.121
Breast Cancer	NA	NA	8.2222 ± 0.214	8.1478 ± 0.214
Claimbuster	NA	NA	11.0817 ± 0.021	10.24784 ± 0.021

From Table 10.6, we see that the Inception v3 based deep learners are good for object recognition and DAE are good in capturing the language model as in *Claimbuster* and *NEWS-20*. For *SECTOR* and *Breast Cancer* datasets, which are nonlinear, the DAE-MA-OR-R-MF-GP-IS is slightly better than the DAE-Softmax classifier. Overall replacement classifier is better than traditional softmax classifier.

Chapter 11

Conclusions

In the present dissertation, we analyzed the deep autoencoders and made following contributions.

1. We have improved the performance of a linear classifier and propose multiple modifications that performs better than commercial package.
2. We have developed smaller MLP's with better generalization error.
3. We developed an alternate pre-training algorithm for deep learning models
4. We demonstrated an efficient automatic scheme to estimate the number and size of hidden layers in MLP.
5. We discovered (or re-discovered) the linearity of many datasets with the AE based deep learning models.
6. We discover the necessity for non linear classifiers in deep learning.

In this dissertation we have pointed out the pervasiveness of linear applications and the shortcomings in existing linear classifiers. We have then developed a new multi-step approach for linear classifier design which requires no user chosen parameters. The OR algorithm in the first step solves the problem of inconsistent errors, where outputs are better than expected. The pruning algorithm in the second step knocks out useless and dependent inputs thereby reducing the size of the linear classifier and decreasing the effects of over-training. In the last step, the linear network is mapped to a GLC and trained further using Newton's algorithm. This step limits the effects of large consistent errors. Using several widely available datasets, we have shown that our proposed algorithm results in a classifier with significantly smaller 10 fold testing errors than those seen in existing linear and generalized linear classifiers. In the future, we plan to extend GLC-Newton to the case of softmax

activations and the cross entropy error measure.We will also try GLC-Newton as a method for improving output weights in nonlinear classifiers.

We then turn our attention to make several incremental improvements to MOLF based classifiers. The pruning algorithm deletes the useless and dependent inputs thereby reducing the size of MOLF classifier and decreasing the effects of over-training. Regularization reduces over-training when the input dimension is large compared to the number of patterns. Weights for less useful inputs are reduced in size. The median filtering pruning in the third step further improves the pruning algorithm to delete more useless hidden units. Further sizing improvements of growing and pruning algorithm decrease the model size significantly without perturbing the testing performance. In the last step, input sigmoids are added to the algorithm to prevent the effect of outliers. Using several widely available datasets, we have shown that our proposed algorithm results in a classifier with significantly smaller 10 fold testing errors than those seen in existing nonlinear and deep learning classifiers. In the future, we plan to extend MA-OR-R-MF-GP-IS algorithm to include a K-fold validation and testing technique that fuses the K optimal networks into one.

All the above improvements in linear and non linear design are then directly incorporated into deep learning models. We first propose a new deep autoencoder with automated depth that is gives a solution to horizontal optimization problem in deep learning models. We use our classifiers as replacement classifiers and demonstrate a better performance than existing classifiers in deep learning.

Appendix A

Datasets

Below is the dataset we use to evaluate our work till now Description of datasets goes here

Table A.1: Specification of datasets

Datasets	N_v	N	M	N_v (train)	N_v (validation)	N_v (test)
Gongtrn	3000	16	10	2000	1000	1000
Comf18	12392	18	4	10000	2392	1000
MNIST	60000	784	10	50000	10000	10000
SVHN	73257	1024	10	73257	26032	26032
CIFAR-10	60000	1024	10	50000	10000	10000
COVER	581012	54	2	406700	110000	64312
SECTOR	6412	55197	105	4000	1000	1412
RCV1	531742	47236	103	400000	100000	31742
NEWS-20	15935	62061	20	8000	4000	3935
Breast Cancer	990	40	2	690	200	100

A.0.28 Gongtrn dataset

The geometric shape recognition data file [180] consists of four geometric shapes, ellipse, triangle, quadrilateral, and pentagon. Each shape consists of a matrix of size 64*64. For each shape, 200 training patterns were generated using different degrees of deformation. The deformations included rotation, scaling, translation, and oblique distortions. The feature set is ring-wedge energy (RNG), and has 16 features.

A.0.29 Comf18 dataset

The training data file is generated from segmented images [8]. Each segmented region is separately histogram equalized to 20 levels. Then the joint probability density of pairs of pixels separated by a given distance and a given direction is estimated. We use 0, 90, 180, 270 degrees for the directions and 1, 3, and 5 pixels for the separations. The density estimates are computed for each classification window. For each separation, the co-occurrences for for the four directions are folded together to form a triangular matrix. From each of the resulting three matrices, six features are computed: angular second moment, contrast, entropy, correlation, and the sums of the main diagonal and the first off diagonal. This results in 18 features for each classification window.

A.0.30 MNIST dataset

The digits data used in this book is taken from the MNIST data set [99], which itself was constructed by modifying a subset of the much larger data set produced by NIST (the National Institute of Standards and Technology). It comprises a training set of 60,000 examples and a test set of 10,000 examples. Some of the data was collected from Census Bureau employees and the rest was collected from high-school children, and care was taken to ensure that the test examples were written by different individuals to the training examples. The original NIST data had binary (black or white) pixels. To create MNIST,these images were size normalized to fit in a 20 20 pixel box while preserving their aspect ratio. As a consequence of the anti-aliasing used to change the resolution of the images, the resulting MNIST digits are grey scale. These images were then centered in a 28 28 box. Examples of the MNIST digits are shown in Figure A.1. This dataset is a classic within the machine learning community and has been extensively studied.

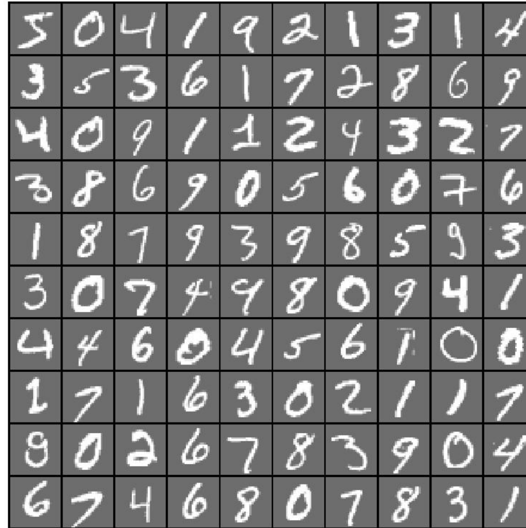

Figure A.1: One hundred examples of the MNIST digits chosen at random from the training set

A.0.31 Google street view dataset

The Google street view housing numbers (SVHN) [129] is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

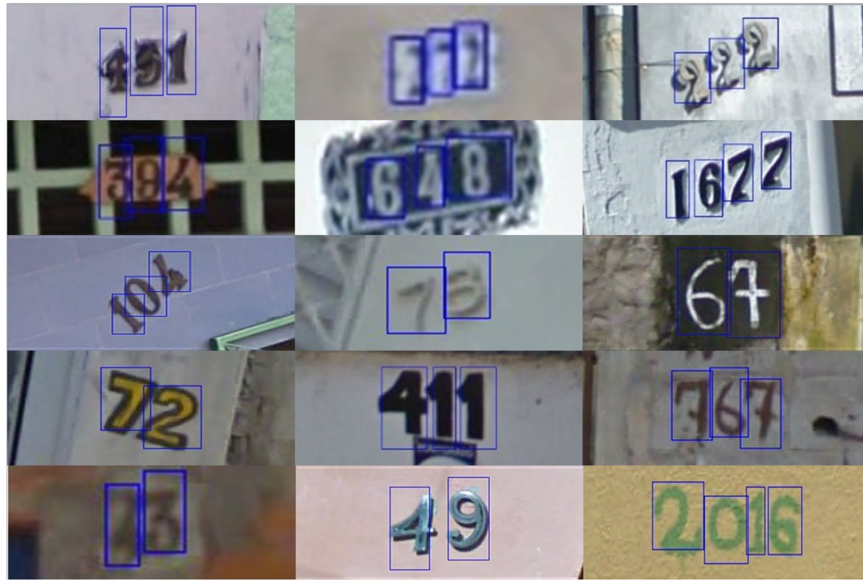

Figure A.2: Fifteen examples of the SVHN chosen at random from the training set

A.0.32 CIFAR dataset

The CIFAR-10 dataset [90] consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

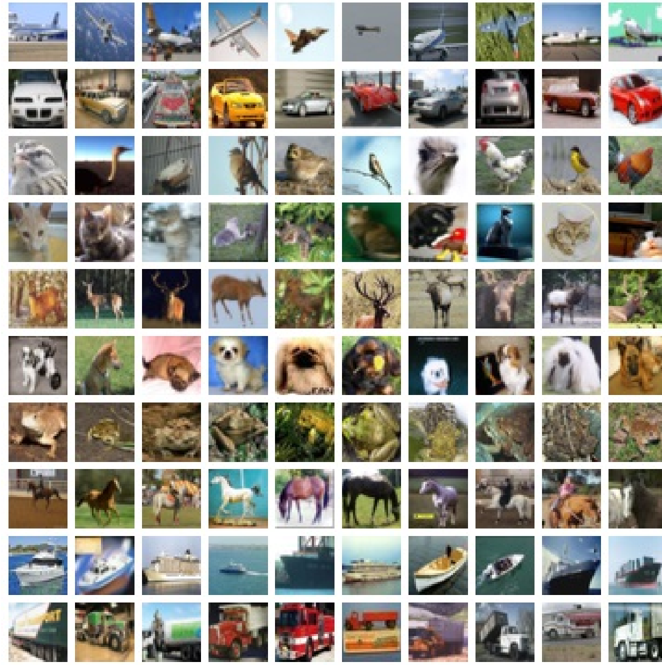

Figure A.3: One hundred examples of the CIFAR images chosen at random from the training set

A.0.33 STL-10 dataset

The STL-10 dataset [36]is an image recognition dataset for developing unsupervised feature learning, deep learning, self-taught learning algorithms. It is inspired by the CIFAR-10 dataset but with some modifications. In particular, each class has fewer labeled training examples than in CIFAR-10, but a very large set of unlabeled examples is provided to learn image models prior to supervised training. The primary challenge is to make use of the unlabeled data (which comes from a similar but different distribution from the labeled data) to build a useful prior. The 10 classes are airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck. All the images are colored with 96x96 pixels. This includes 500 training images (10 pre-defined folds), 800 test images per class and 100000 unlabeled images for unsupervised learning. These examples are extracted from a similar but broader distribution of images. For instance, it contains other types of animals (bears, rabbits, etc.) and vehicles (trains, buses, etc.) in addition to the ones in the labeled set.

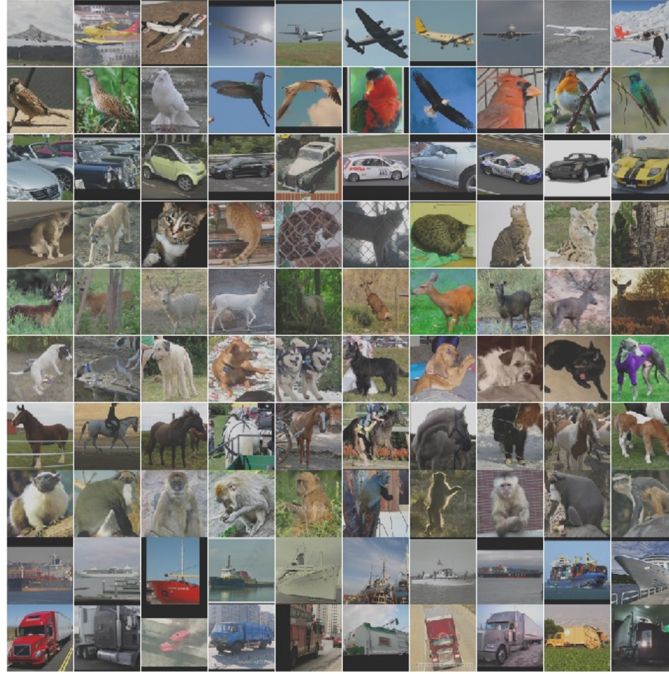

Figure A.4: One hundred examples of the STL-10 images chosen at random from the training set

A.0.34 COVER

This dataset [18] is contains forest cover type for a given observation (30 x 30 meter cell) that was determined from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. Independent variables were derived from data originally obtained from US Geological Survey (USGS) and USFS data. Data is in raw form (not scaled) and contains binary (0 or 1) columns of data for qualitative independent variables (wilderness areas and soil types).

A.0.35 RCV1

Reuters Corpus Volume I (RCV1) [107] is an archive of over 800,000 manually categorized newswire stories made available by Reuters, Ltd. for research purposes. The dataset is extensively described in [1].

A.0.36 NEWS-20

The 20 Newsgroups dataset [122] is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.

Appendix B

Training weights by orthogonal least squares

In the current work, we use orthogonal least square method to solve for the output weights, pruning of hidden units and deciding on the number of hidden units in a deep learner. an orthogonal least square method is a transformation of the set of ¡ ¡ ¿¿ ¿ into a set of orthogonal basis vectors thereby measuring the individual contribution to the desired output energy from each basis vector.

In an autoencoder, we are mapping from an (N+1) dimensional augmented input vector to it's reconstruction in the output layer. The output weight matrix $\mathbf{W_{oh}} \in \Re^{N \times N_h}$ and y_p in elements wise will be given as

$$y_p(i) = \sum_{n=1}^{N+1} w_{oh}(i,n) \cdot x_p(n) \tag{B.1}$$

To solve for the output weights by regression , we minimize the reconstruction error as in (2.2). In order to achieve a superior numerical computation, we define the elements of auto correlation $\mathbf{R} \in \Re^{N_h \times N_h}$ and cross correlation matrix $\mathbf{C} \in \Re^{N_h \times M}$ as follows :

$$r(n,l) = \frac{1}{N_v} \sum_{p=1}^{N_v} O_p(n) \cdot O_p(l) \qquad c(n,i) = \frac{1}{N_v} \sum_{p=1}^{N_v} O_p(n) \cdot t_p(i) \tag{B.2}$$

Substituting the value of $y_p(i)$ in (2.2) we get,

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{m=1}^{M} [t_p(m) - \sum_{k=1}^{N_h} w_o h(i,k) \cdot O_p(k)]^2 \quad \text{(B.3)}$$

Differentiating with respect to $\mathbf{W_{oh}}$ and using (B.2) we get

$$\frac{\partial E}{w_{oh}(m,l)} = -2[c(l,m) - \sum_{k=1}^{N_h+1} w_{oh}(m,k) r(k,l)] \quad \text{(B.4)}$$

Equating (B.4) to zero we obtain a M set of $N_h + 1$ linear equations in $N_h + 1$ variables. In a compact form it can be written as

$$\mathbf{R} \cdot \mathbf{W}^T = \mathbf{C} \quad \text{(B.5)}$$

By using orthogonal least square, the solution for computation of weights in (B.5) will speed up. For convineance, let $N_u = N_h + 1$ and the basis functions be the hidden units output $\mathbf{O} \in \Re^{(N_h+1)\times 1}$ augmented with a bias of **1**. For an unordered basis function $\mathbf{O}$ of dimension N_u , the mth orthonormal basis function $\mathbf{O}^{'}$ is defines as ¡¡ add reference ¿¿

$$O_m^{'} = \sum_{k=1}^{m} a_{mk} \cdot O_k \quad \text{(B.6)}$$

Here a_{mk} are the elements of triangular matrix $\mathbf{A} \in \Re^{N_u \times N_u}$

For $m = 1$

$$O_1^{'} = a_{11} \cdot O_1 \qquad a_{11} = \frac{1}{\|O\|} = \frac{1}{r(1,1)} \quad \text{(B.7)}$$

for $2 \le m \le N_u$, we first obtain

$$c_i = \sum_{q=1}^{i} a_{iq} \cdot r(q,m) \quad \text{(B.8)}$$

for $1 \le i \le m - 1$. Second, we set $b_m = 1$ and get

$$b_{jk} = -\sum_{i=k}^{m=1} c_i \cdot a_{ik} \quad \text{(B.9)}$$

for $1 \leq k \leq m-1$. Lastly we get the coeffeicent A_{mk} for the triangular matrix $\mathbf{A}$ as

$$a_{mk} = \frac{b_k}{[r(m,m) - \sum_{i=1}^{m-1} c_i^2]^2} \tag{B.10}$$

Once we have the orthonormal basis functions, the linear mapping weights in the orthonormal system can be found as

$$w^{'}(i,m) = \sum_{k=1}^{m} a_{mk} c(i,k) \tag{B.11}$$

The orthonormal system's weights $\mathbf{W}^{'}$ can be mapped back to the original system's weights $\mathbf{W}$ as

$$w(i,k) = \sum_{m=k}^{N_u} a_{mk} \cdot w_o^{'}(i,m) \tag{B.12}$$

In an orthonormal system, the total training error can be written from (2.2) as

$$E = \sum_{i=1}^{M} \sum_{p=1}^{N_v} [\langle t_p(i), t_p(i) \rangle - \sum_{k=1}^{N_u} (w^{'}(i,k))^2] \tag{B.13}$$

Orthogonal least square is equivalent of using the $\mathbf{QR}$ decomposition [53] and is useful when equation (B.5) is ill-conditioned meaning that the determinant of $\mathbf{R}$ is $\mathbf{0}$.

Bibliography

[1] Hussein A Abbass. An evolutionary artificial neural networks approach for breast cancer diagnosis. *Artificial intelligence in Medicine*, 25(3):265–281, 2002.

[2] Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from data*, volume 4. AMLBook Singapore, 2012.

[3] Masoud Ahookhosh, Francisco J Aragón, Ronan MT Fleming, and Phan T Vuong. Local convergence of levenberg-marquardt methods under h\"{o} lder metric subregularity. *arXiv preprint arXiv:1703.07461*, 2017.

[4] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*, 2015.

[5] Spencer E Ante. Amazon: turning consumer opinions into gold. *Business Week*, 15, 2009.

[6] Peter Auer, Mark Herbster, Manfred K Warmuth, et al. Exponentially many local minima for single neurons. *Advances in neural information processing systems*, pages 316–322, 1996.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[8] Robert R Bailey, Elaine J Pettit, Ronald T Borochoff, Michael T Manry, and Xianping Jiang. Automatic recognition of usgs land use/cover categories using statistical and neural network classifiers. In *Optical Engineering and Photonics in Aerospace Sensing*, pages 185–195. International Society for Optics and Photonics, 1993.

[9] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

[10] Nicholas M Ball and Robert J Brunner. Data mining and machine learning in astronomy. *International Journal of Modern Physics D*, 19(07):1049–1106, 2010.

[11] Marian Stewart Bartlett, Gwen Littlewort, Mark Frank, Claudia Lainscsek, Ian Fasel, and Javier Movellan. Recognizing facial expression: machine learning and application to spontaneous behavior. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 568–573. IEEE, 2005.

[12] Rodrigo Benenson, Mohamed Omran, Jan Hosang, and Bernt Schiele. Ten years of pedestrian detection, what have we learned? In *European Conference on Computer Vision*, pages 613–627. Springer, 2014.

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

[14] Yoshua Bengio and Olivier Delalleau. On the expressive power of deep architectures. In *International Conference on Algorithmic Learning Theory*, pages 18–36. Springer, 2011.

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

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

[17] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128, 2006.

[18] Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999.

[19] Joseph-Frédéric Bonnans, Jean Charles Gilbert, Claude Lemaréchal, and Claudia A Sagastizábal. *Numerical optimization: theoretical and practical aspects*. Springer Science & Business Media, 2006.

[20] Indranil Bose and Radha K Mahapatra. Business data mininga machine learning perspective. *Information & management*, 39(3):211–225, 2001.

[21] Y-lan Boureau, Yann L Cun, et al. Sparse feature learning for deep belief networks. In *Advances in neural information processing systems*, pages 1185–1192, 2008.

[22] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294, 1988.

[23] Bryan Catanzaro. Deep learning with cots hpc systems. 2013.

[24] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[25] Hema Chandrasekaran, Hung-Han Chen, and Michael T Manry. Pruning of basis functions in nonlinear approximators. *Neurocomputing*, 34(1):29–53, 2000.

[26] Christakis Charalambous. Conjugate gradient algorithm for efficient training of artificial neural networks. *IEE Proceedings G-Circuits, Devices and Systems*, 139(3):301–310, 1992.

[27] Yves Chauvin. Dynamic behavior of constained back-propagation networks. In *NIPS*, volume 89, pages 643–649, 1989.

[28] Hsinchun Chen, Wingyan Chung, Jennifer Jie Xu, Gang Wang, Yi Qin, and Michael Chau. Crime data mining: a general framework and some examples. *Computer*, 37(4):50–56, 2004.

[29] M-S Chen and Michael T Manry. Basis vector analyses of back-propagation neural networks. In *Circuits and Systems, 1991., Proceedings of the 34th Midwest Symposium on*, pages 23–26. IEEE, 1991.

[30] M-S Chen and Michael T Manry. Power series analyses of back-propagation neural networks. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume 1, pages 295–300. IEEE, 1991.

[31] Sheng Chen, Colin FN Cowan, and Peter M Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):302–309, 1991.

[32] Yun Chen, Flora S Tsai, and Kap Luk Chan. Machine learning techniques for business blog search and mining. *Expert Systems with Applications*, 35(3):581–590, 2008.

[33] François Chollet et al. Keras. https://github.com/keras-team/keras, 2015.

[34] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *AISTATS*, 2015.

[35] Rohit Choudhry and Kumkum Garg. A hybrid machine learning system for stock market forecasting. *World Academy of Science, Engineering and Technology*, 39(3):315–318, 2008.

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

[37] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[38] Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, (3):326–334, 1965.

[39] W.H. Delashmit and M.T. Manry. A neural network growing algorithm that ensures monotonically non increasing error. *Advances in Neural Networks*, 14:280–284, 2007.

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

[41] Li Deng and Xiao Li. Machine learning paradigms for speech recognition: An overview. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(5):1060–1089, 2013.

[42] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[43] Scott E Fahlman and Christian Lebiere. The cascade-correlation learning architecture. 1989.

[44] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

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

[46] Kang Fu, Dawei Cheng, Yi Tu, and Liqing Zhang. Credit card fraud detection using convolutional neural networks. In *International Conference on Neural Information Processing*, pages 483–490. Springer, 2016.

[47] Keinosuke Fukunaga. *Introduction to statistical pattern recognition*. Academic press, 2013.

[48] Guojun Gan. Application of data clustering and machine learning in variable annuity valuation. *Insurance: Mathematics and Economics*, 53(3):795–801, 2013.

[49] Philip E Gill, Walter Murray, and Margaret H Wright. Practical optimization. 1981.

[50] Federico Girosi, Michael Jones, and Tomaso Poggio. Regularization theory and neural networks architectures. *Neural computation*, 7(2):219–269, 1995.

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

[52] Pavel Golik, Patrick Doetsch, and Hermann Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *INTERSPEECH*, pages 1756–1760, 2013.

[53] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[54] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[55] Ian J Goodfellow, Nate Koenig, Marius Muja, Caroline Pantofaru, Alexander Sorokin, and Leila Takayama. Help me help you: Interfaces for personal robots. In *Proceedings of the 5th ACM/IEEE international conference on Human-robot interaction*, pages 187–188. IEEE Press, 2010.

[56] RG Gore, Jiang Li, Michael T Manry, Li-Min Liu, Changhua Yu, and John Wei. Iterative design of neural network classifiers through regression. *International Journal on Artificial Intelligence Tools*, 14(01n02):281–301, 2005.

[57] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[58] Justin Grimmer. We are all social scientists now: how big data, machine learning, and causal inference work together. *PS: Political Science & Politics*, 48(01):80–83, 2015.

[59] Raia Hadsell, Ayse Erkan, Pierre Sermanet, Marco Scoffier, Urs Muller, and Yann LeCun. Deep belief net learning in a long-range vision system for autonomous off-road driving. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 628–633. IEEE, 2008.

[60] Masafumi Hagiwara. Novel backpropagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection. In *IJCNN International Joint Conference on Neural Networks*, pages 625–630. IEEE, 1990.

[61] Masafumi Hagiwara. Novel backpropagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 625–630. IEEE, 1990.

[62] Yilong Hao, Kanishka Tyagi, Rohit Rawat, and Michael Manry. Second order design of multi-class kernel machines. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 3233–3240. IEEE, 2016.

[63] Naeemul Hassan, Chengkai Li, and Mark Tremayne. Detecting check-worthy factual claims in presidential debates. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1835–1838. ACM, 2015.

[64] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *Neural Networks, 1993., IEEE International Conference on*, pages 293–299. IEEE, 1993.

[65] Johan Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 6–20. ACM, 1986.

[66] Simon Haykin. *Neural networks and learning machines*, volume 3. Pearson Education, 2009.

[67] Magnus Rudolph Hestenes and Eduard Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS, 1952.

[68] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

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

[70] Geoffrey E Hinton and Ruslan R Salakhutdinov. Using deep belief nets to learn covariance kernels for gaussian processes. In *Advances in neural information processing systems*, pages 1249–1256, 2008.

[71] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length, and helmholtz free energy. *Advances in neural information processing systems*, pages 3–3, 1994.

[72] Yu-Chi Ho and RL Kashyap. An algorithm for linear inequalities and its applications. *IEEE Transactions on Electronic Computers*, (5):683–688, 1965.

[73] Yu-chi Ho and RL Kashyap. A class of iterative procedures for linear inequalities. *SIAM Journal on Control*, 4(1):112–115, 1966.

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

[75] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning*, pages 408–415. ACM, 2008.

[76] Wei Huang, Yoshiteru Nakamori, and Shou-Yang Wang. Forecasting stock market movement direction with support vector machine. *Computers & Operations Research*, 32(10):2513–2522, 2005.

[77] Gordon Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE transactions on information theory*, 14(1):55–63, 1968.

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

[79] Nathalie Japkowicz, Stephen Jose Hanson, and Mark A Gluck. Nonlinear autoassociation is not equivalent to pca. *Neural computation*, 12(3):531–545, 2000.

[80] Xianping Jiang, Mu-Song Chen, Michael T Manry, Michael S Dawson, and Adrain K Fung. Analysis and optimization of neural networks for remote sensing. *Remote Sensing Reviews*, 9(1-2):97–114, 1994.

[81] Bhautik Joshi, Kristen Stewart, and David Shapiro. Bringing impressionism to life with neural style transfer in come swim. *arXiv preprint arXiv:1701.04928*, 2017.

[82] Paul C Kainen, Vera Kurková, Vladik Kreinovich, and Ongard Sirisaengtaksin. Uniqueness of network parametrization and faster learning. *Neural Parallel & Scientific Comp.*, 2(4):459–466, 1994.

[83] Ehud D Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks*, 1(2):239–242, 1990.

[84] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

[85] Mujitha B KB, Vishnuprasad V Ajil Jalal, and KA Nishad. Analytics, machine learning & nlp–use in biosurveillance and public health practice. *Online journal of public health informatics*, 7(1), 2015.

[86] S Sathiya Keerthi, Sellamanickam Sundararajan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. A sequential dual method for large scale multi-class linear svms. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 408–416. ACM, 2008.

[87] Javed Khan, Jun S Wei, Markus Ringner, Lao H Saal, Marc Ladanyi, Frank Westermann, Frank Berthold, Manfred Schwab, Cristina R Antonescu, Carsten Peterson, et al. Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nature medicine*, 7(6):673, 2001.

[88] Parastoo Kheirkhah, Kanishka Tyagi, Son Nguyen, and Michael T Manry. Structural adaptation for sparsely connected mlp using newton's method. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 4467–4473. IEEE, 2017.

[89] Igor Kononenko. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in medicine*, 23(1):89–109, 2001.

[90] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[91] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[92] SY Kung and Yu Hen Hu. A frobenius approximation reduction method (farm) for determining optimal number of hidden units. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume 2, pages 163–168. IEEE, 1991.

[93] Jinsa Kuruvilla and K Gunavathi. Lung cancer classification using neural networks for ct images. *Computer methods and programs in biomedicine*, 113(1):202–209, 2014.

[94] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10(Jan):1–40, 2009.

[95] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480. ACM, 2007.

[96] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

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

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

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

[100] Yann LeCun, John S Denker, Sara A Solla, Richard E Howard, and Lawrence D Jackel. Optimal brain damage. In *NIPS*, volume 2, pages 598–605, 1989.

[101] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[102] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. Efficient sparse coding algorithms. In *Advances in neural information processing systems*, pages 801–808, 2006.

[103] Honglak Lee, Chaitanya Ekanadham, and Andrew Y Ng. Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems*, pages 873–880, 2008.

[104] Régis Lengellé and Thierry Denoeux. Training mlps layer by layer using an objective function for internal representations. *Neural Networks*, 9(1):83–97, 1996.

[105] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[106] Esther Levin and Michael Fleisher. Accelerated learning in layered neural networks. *Complex systems*, 2:625–640, 1988.

[107] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397, 2004.

[108] Jiang Li, Michael T Manry, Li-Min Liu, Changhua Yu, and John Wei. Iterative improvement of neural classifiers. In *FLAIRS Conference*, pages 700–705, 2004.

[109] Kadir Liano. Robust error measure for supervised neural network learning with outliers. *IEEE Transactions on Neural Networks*, 7(1):246–250, 1996.

[110] LM Liu, MT Manry, F Amar, MS Dawson, and AK Fung. Image classification in remote sensing using functional link neural networks. In *Proceedings of the IEEE Southwest Symposium on Image Analysis and Interpretation*, pages 54–58. IEEE, 1994.

[111] Changfeng Ma and Jia Tang. The quadratic convergence of a smoothing levenberg–marquardt method for nonlinear complementarity problem. *Applied Mathematics and Computation*, 197(2):566–581, 2008.

[112] Sanjeev S Malalur and Michael T Manry. Multiple optimal learning factors for feed-forward networks. In *SPIE Defense, Security and Sensing (DSS) Conference, Orlando, FL*, 2010.

[113] Sanjeev S Malalur, Michael T Manry, and Praveen Jesudhas. Multiple optimal learning factors for the multi-layer perceptron. *Neurocomputing*, 149:1490–1501, 2015.

[114] FJ Maldonado and MT Manry. Optimal pruning of feedforward neural networks based upon the schmidt procedure. In *Signals, Systems and Computers, 2002. Conference Record of the Thirty-Sixth Asilomar Conference on*, volume 2, pages 1024–1028. IEEE, 2002.

[115] FJ Maldonado, MT Manry, and Tae-Hoon Kim. Finding optimal neural network basis function subsets using the schmidt procedure. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 444–449. IEEE, 2003.

[116] Michael Manry. Ee 5352 statistical signal processing lecture notes. 2016.

[117] Michael Manry. Ee 5353 neural networks lecture notes. 2016.

[118] MT Manry, H Chandrasekaran, and CH Hsieh. Signal processing applications of the multilayer perceptron, 2001.

[119] MT Manry, MS Dawson, AK Fung, SJ Apollo, LS Allen, WD Lyle, and W Gong. Fast training of neural networks for remote sensing. *Remote sensing reviews*, 9(1-2):77–96, 1994.

[120] Christopher Poultney MarcAurelio Ranzato, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In *Proceedings of NIPS*, 2007.

[121] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[122] Tom M Mitchell. Machine learning book, 1997.

[123] Tom Michael Mitchell. *The discipline of machine learning*, volume 9. Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006.

[124] Volodymyr Mnih and Geoffrey E Hinton. Learning to detect roads in high-resolution aerial images. In *European Conference on Computer Vision*, pages 210–223. Springer, 2010.

[125] Michael C Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. 1989.

[126] Divya Murli, Shailesh Jami, Devika Jog, and Sreesha Nath. Credit card fraud detection using neural networks. *International Journal of Students' Research in Technology & Management*, 2(2):84–88, 2015.

[127] Pramod L Narasimha, Walter H Delashmit, Michael T Manry, Jiang Li, and Francisco Maldonado. An integrated growing-pruning method for feedforward network training. *Neurocomputing*, 71(13):2831–2847, 2008.

[128] Pramod L Narasimha, Michael T Manry, and Francisco Maldonado. Upper bound on pattern storage in feedforward networks. *Neurocomputing*, 71(16):3612–3616, 2008.

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

[130] Andrew Ng. Sparse autoencoder. *CS294A Lecture notes*, 72:1–19, 2011.

[131] Son Nguyen, Kanishka Tyagi, Parastoo Kheirkhah, and Michael Manry. Partially affine invariant back propagation. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 811–818. IEEE, 2016.

[132] Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: a survey. *Cognitive processing*, 12(4):319–340, 2011.

[133] Kamal Nigam, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using em. *Machine learning*, 39(2-3):103–134, 2000.

[134] Fredrik Olsson. A literature survey of active machine learning in the context of natural language processing. 2009.

[135] J Olvera, X Guan, and MT Manry. Theory of monomial networks. In *Proc. Symp. Implicit and Nonlinear Systems*, pages 96–101, 1992.

[136] Genevieve B Orr and Klaus-Robert Müller. *Neural networks: tricks of the trade*. Springer, 2003.

[137] Simon Osindero and Geoffrey E Hinton. Modeling image patches with a directed hierarchy of markov random fields. In *Advances in neural information processing systems*, pages 1121–1128, 2008.

[138] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[139] Lorien Y Pratt. *Comparing biases for minimal network construction with back-propagation*, volume 1. Morgan Kaufmann Pub, 1989.

[140] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes in C*, volume 2. Cambridge university press Cambridge, 1996.

[141] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th international conference on Machine learning*, pages 759–766. ACM, 2007.

[142] Ashok Rao and S Noushath. Subspace methods for face recognition. *Computer Science Review*, 4(1):1–17, 2010.

[143] Rohit Rawat, Jignesh K Patel, and Michael T Manry. Minimizing validation error with respect to network size and number of training epochs. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2013.

[144] Russell Reed. Pruning algorithms-a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[145] Michael D Richard and Richard P Lippmann. Neural network classifiers estimate bayesian a posteriori probabilities. *Neural computation*, 3(4):461–483, 1991.

[146] Martin Riedmiller and Heinrich Braun. Rprop-a fast adaptive learning algorithm. In *Proc. of ISCIS VII), Universitat*. Citeseer, 1992.

[147] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 833–840, 2011.

[148] Melvin Deloyd Robinson and Michael Thomas Manry. Two-stage second order training in feedforward neural networks. In *FLAIRS Conference*, 2013.

[149] Fabio Roli. Statistical and neural classifiers: an integrated approach to design (advances in pattern recognition series). by s. raudys. *Pattern Analysis & Applications*, 7(1):114–115, 2004.

[150] Peter J Rousseeuw and Annick M Leroy. *Robust regression and outlier detection*, volume 589. John Wiley & sons, 2005.

[151] D. B. Rubin. Iterative recursive least squares. In *Encyclopaedia of Statistical Sciences*, pages 272–275. Wiley, 1983.

[152] Dennis W Ruck, Steven K Rogers, Matthew Kabrisky, Mark E Oxley, and Bruce W Suter. The multilayer perceptron as an approximation to a bayes optimal discriminant function. *IEEE Transactions on Neural Networks*, 1(4):296–298, 1990.

[153] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[154] David E Rumelhart, James L McClelland, PDP Research Group, et al. *Parallel distributed processing*, volume 1. IEEE, 1988.

[155] Michael A Sartori and Panos J Antsaklis. A simple method to derive bounds on the size and to train multilayer neural networks. *IEEE Transactions on Neural Networks*, 2(4):467–471, 1991.

[156] Eric Saund. Dimensionality-reduction using connectionist networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(3):304–314, 1989.

[157] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[158] Bernhard Schölkopf, Koji Tsuda, and Jean-Philippe Vert. *Kernel methods in computational biology*. MIT press, 2004.

[159] Mark Schwabacher and Kai Goebel. A survey of artificial intelligence for prognostics. In *AAAI fall symposium*, pages 107–114, 2007.

[160] Holger Schwenk, Anthony Rousseau, and Mohammed Attik. Large, pruned or continuous space language models on a gpu for statistical machine translation. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pages 11–19. Association for Computational Linguistics, 2012.

[161] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *Interspeech*, pages 437–440, 2011.

[162] Jocelyn Sietsma and Robert JF Dow. Neural net pruning-why and how. In *IEEE International Conference on Neural Networks*, volume 1, pages 325–333. IEEE San Diego, 1988.

[163] Patrice Y Simard, David Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[164] DN Sonar and UV Kulkarni. Lung cancer classification. *International Journal of Computer Sciences and Engineering*, page 51, 2016.

[165] Xiaomu Song, Guoliang Fan, and Mahesh Rao. Automatic crp mapping using nonparametric machine learning approaches. *IEEE transactions on geoscience and remote sensing*, 43(4):888–897, 2005.

[166] Xiaoyuan Su and Khoshgoftaar M Taghi. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.

[167] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[168] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[169] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[170] Adi L Tarca, Vincent J Carey, Xue-wen Chen, Roberto Romero, and Sorin Drăghici. Machine learning and its applications to biology. *PLoS Comput Biol*, 3(6):e116, 2007.

[171] Kanishka Tyagi. Second order training algorithms for radial basis function neural networks. *Masters Thesis*, 2012.

[172] Vladimir Vapnik and Rauf Izmailov. V-matrix method of solving statistical inference problems. *Journal of Machine Learning Research*, 16:1683–1730, 2015.

[173] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[174] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

[175] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[176] Jason Weston, Samy Bengio, and Nicolas Usunier. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine learning*, 81(1):21–35, 2010.

[177] Jorg Wille. On the structure of the hessian matrix in feedforward networks and second derivative methods. In *International Conference on Neural Networks*, volume 3, pages 1851–1855. IEEE, 1997.

[178] Robert C Williamson and Uwe Helmke. Existence and uniqueness results for neural network approximations. *IEEE Transactions on Neural Networks*, 6(1):2–13, 1995.

[179] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[180] Hung-Chun Yau and Michael T Manry. Iterative improvement of a nearest neighbor classifier. *Neural Networks*, 4(4):517–524, 1991.

[181] Changhua Yu, Michael T Manry, Jiang Li, and Pramod Lakshmi Narasimha. An efficient hidden layer training method for the multilayer perceptron. *Neurocomputing*, 70(1):525–535, 2006.

[182] David Zhang, Wangmeng Zuo, and Feng Yue. A comparative study of palmprint recognition algorithms. *ACM Computing Surveys (CSUR)*, 44(1):2, 2012.

[183] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

[184] Will Y Zou, Andrew Y Ng, and Kai Yu. Unsupervised learning of visual invariance with temporal coherence. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 3, 2011.