SECOND ORDER ALGORITHM FOR SPARSELY CONNECTED NEURAL NETWORKS

by

PARASTOO KHEIRKHAH

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2016

To my father, mother, sister,

and my grandmas (**مادرجون و مامان مولوک**)

چنین گفت داننده دهقان پیر

که دانش بود مرد را دستگیر

هر ان مغز کو را خرد روشنست

ز دانش به گرد تنش جوشنست

حکیم ابوالقاسم فردوسی (۴۱۶ – ۳۲۹)

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advising professor Dr. Michael T. Manry for his support and the efforts in guiding me in my research. I would also thank him for the time he had spent for me and the knowledge he provided me from his courses and lab meetings.

I would also thank my thesis committee members Dr. W. Lee and Dr. Ionnis D. Schizas for reviewing my work and accepting my request to be in my committee members.

I would thank my lab-mates Rohit Rawat, Kanishka Tyagi, and Son Nguyen for their assistances and suggestions throughout this work.

I would like to express my special thanks to my father Siavash Kheirkhah for his sacrifice, support and encouragement; to my mother Mastaneh Chilan for her patience, endless love and support; to my dearest sister Parisa Kheirkhah for giving me motivations, love, and encouragement; to my dearest cousin Manousha Khairkhah for her support, love, and her belief in me.

Also, I would thank all my aunts specially Azita Chilan, my cousins, and my friends specially Annahita Habibi, Babak Ebrahimi, and Mr. Faize who always wished to see my success but life and destiny did not let me fulfill my promise to him.

July 29, 2016

۱ مرداد ۱۳۹۵

ABSTRACT

SECOND ORDER ALGORITHM FOR SPARSELY

CONNECTED NEURAL NETWORKS

Parastoo Kheirkhah, M.S.

The University of Texas at Arlington, 2016

Supervising Professor: Michael T. Manry

A systematic two-step batch approach for constructing a sparsely connected neural network is presented. Unlike other sparse neural networks, the proposed paradigm uses orthogonal least squares (OLS) to train the network. OLS based pruning is proposed to induce sparsity in the network. Based on the usefulness of the basic functions in the hidden units, the weights connecting the output to hidden units and output to input units are modified to form a sparsely connected neural network. The proposed hybrid training algorithm has been compared with the fully connected MLP and sparse softmax classifier that uses second order training algorithm. The simulation results show that the proposed algorithm has significant improvement in terms of convergence speed, network size, generalization and ease of training over fully connected MLP. Analysis of the proposed training algorithm on various linear and non-linear data files is carried out. The ability of the proposed algorithm is further substantiated by clearly differentiating two separate datasets when feed into the proposed algorithm. The experimental results are reported using 10-fold cross validation. Inducing sparsity into a fully connected neural network, pruning of the hidden units, Newton's method for optimization, and orthogonal least squares are the subject matter of the present work.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION

From an artificial intelligence point of view, machine learning is a subfield that recently has seen rapid growth. Today, with all the technical advances and booming growth of data than any other time in the human history, machine learning lies at the core of data science that will play a crucial role in the technical innovation for future. Recent machine learning applications have been far reaching and highly impactful. It's the heart of autonomous vehicle control [1], speech processing [2], natural language processing [3] and [4], computer vision [5], cancer prognosis and predictions [6], astronomy [7].

A neural network can be thought of as a complicated mathematical function that has various constants called weights and biases, which must be determined. Training a neural network is the process of finding a set of weight so that for a given set of inputs, the outputs produced are very close to some target values. Learning algorithms for training neural networks are generally divided into two types, supervised and unsupervised learning. In supervised learning, a training dataset or desired outputs of the neural network is provided for the purpose of training. On the other hand, unsupervised learning has to make sense of the inputs without outside help. The network is only provided with the input dataset and it has to find out some of the characterization of the data and learn to reflect these properties in its output.

Sparse modeling or representation is attracting an increasing attention in problems with very high dimension, often larger than the sample size. These problems are usually ill-posed, prone to over-fitting and, if no correcting action is taken, likely to yield poor models. Sparse methods can be a helpful solution to remedy these problems. Sparsity has also played a central role in the success of many machine learning algorithms and techniques such as matrix factorization [8] [9]. In sparse models, although only a small fraction of non-zero values remain in the representation, this is sufficient to achieve equivalent results to non-sparse models. So far,

numerous sparsity measurements have been developed [10]. Also, various methods have been proposed to achieve sparse solutions of machine learning problems as in [11] and [12].

With the rapid growth of the scale of data science, various forms of online and offline data, there is a need for efficient computational models. Sparsity is one promising solution that provides efficient input representation. On the other hand neural networks are immensely successful in parallelizing massive computation. By placing sparsity in the neural network models, we empower them to compute and process much more data. Sparse neural networks are much smaller in size and hence computationally very efficient and can deal with much larger data with higher dimensions. However, the challenge lies in making the data sparse. Efficient representation of data is a key component in its later stage of classification. Sparse coding [13] provides an efficient representation that is localized, oriented and receptive to band-pass fields similar to those found in primary visual cortex. Efficient sparse coding algorithms as in [12] have shown extremely promising results with the sparse auto-encoder network. In [12], a novel fast algorithm is proposed to solve L_1 and L_2 constrained least square problems. The algorithm learns large sparse codes and has been applied to natural images, speech and video. However, the training requires heuristically picked, difficult to tune, hyper-parameters and box constraints. In the present investigation, we propose an efficient algorithm that optimally designs the sparse neural network model.

Our design philosophy is to build small but powerful networks that have minimum human intervention to tune the hyper-parameters. We include sparsity into the family of multi-layer perceptron neural networks in which the parameters are found in an optimal method. Based on the simulation results on various benchmark and real life datasets, the proposed algorithm performs better than its counterparts. In this paper, we formulate an optimized sparse connected MLP structure that has optimal parameters and is relatively easy to implement.

Chapter 2

THE MULTILAYER PERCEPTRON

The multi-layer perceptron (MLP) is a feed forward artificial neural network that maps sets of input data onto a set of outputs and is widely used for regression and classification problems. The MLP is a modification of the linear perceptron. The MLP is used for regression and classification applications such as parameter estimation, document analysis and recognition, finance and manufacturing and data mining [14].

A conventional MLP consists of multiple layers of hidden units, with each layer connected to the next one. Except for the inputs, each node is a neuron (or processing element) with an activation function [15] and [16]. Figure 1 illustrates the structure of an MLP having an input layer, a hidden layer and an output layer.

A. Notation

We denote the number of hidden units by N_h, the number of outputs by M and the number of inputs by N. In order to handle hidden units thresholds and output units thresholds, the input vector $\boldsymbol{x}_p$ is augmented with an extra element $\boldsymbol{x}_p(\mathrm{N}+1)$, where $\boldsymbol{x}_p(\mathrm{N}+1)=1$. Here, the input vector is $\boldsymbol{x}_p \in \mathbb{R}^{\mathrm{N}+1}$, and the M-dimensional desired output vector is $\boldsymbol{t}_p$. The training dataset consists of $\{\boldsymbol{x}_p, \boldsymbol{t}_p\}$ pairs in which we denote a particular pattern with an index $p \in \{1, 2, \ldots, \mathrm{N}_v\}$. Figure 1-1 is the architecture of a MLP.

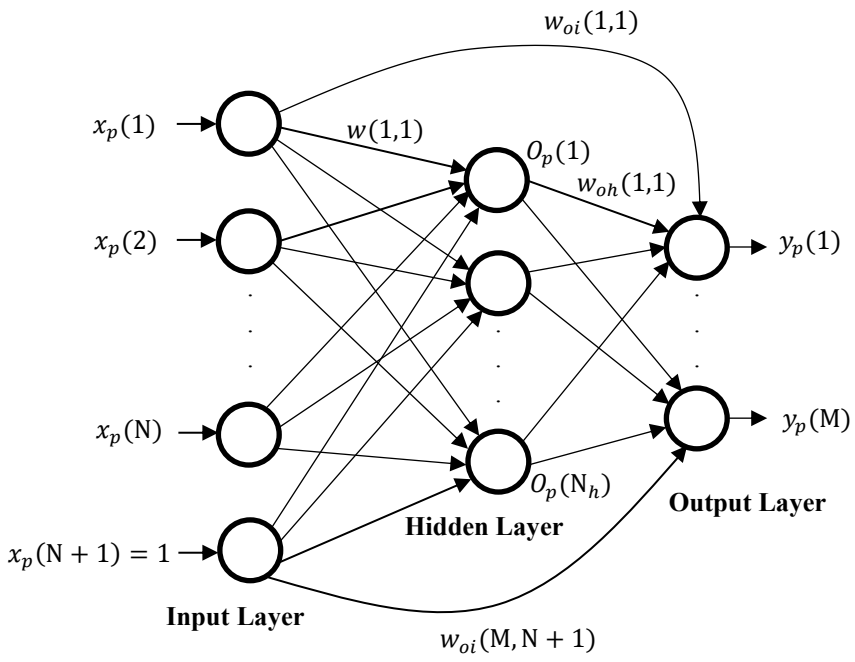


Figure 2-1: Illustration of a fully connected multilayer perceptron architecture

In the figure above, the bypass weight $w_{oi}(i,n)$ converts the n^{th} input to the i^{th} output coefficient. The output hidden weight $w_{oh}(i,k)$ converts the k^{th} hidden unit to the i^{th} output. The output weight matrix, $\mathbf{W_o}$, is augmented as $[\mathbf{W}_{oh} \vdots \mathbf{W}_{oi}]$. One commonly used activation function in the neural network is the sigmoid function. The MLP's net function for the p^{th} pattern and the k^{th} hidden unit is represented as

$$n_p(k) = \sum_{n=1}^{\mathrm{N}_h} w(k,n)x_p(n) \tag{2.1}$$

where $w(k,n)$ denotes the input weight connecting the k^{th} hidden unit to the n^{th} input. In the vector format equation (2.1) can be written as

$$\boldsymbol{n}_p = \mathbf{W} \cdot \boldsymbol{x}_p \tag{2.2}$$

in which $\mathbf{W}$ is an $\mathrm{N}_h$by $(\mathrm{N}+1)$ matrix and $\boldsymbol{n}_p$ is the N_h-dimensional net function vector.

The activation function output for the p^{th} pattern and the k^{th} hidden unitis denoted as $O_p(k)$ where $O_p(k) = f(n_p(k))$ and $f(\cdot)$ denotes the hidden layer's activation function

$$f(net) = \frac{1}{1 + e^{-net}} \tag{2.3}$$

The i^{th}output for the p^{th} patternin the M-dimensional output vector, $\boldsymbol{y}_p$, is

$$y_p(i) = \sum_{n=1}^{\mathrm{N}+1} w_{oi}(i,n) \cdot x_p(n) + \sum_{k=1}^{\mathrm{N}_h} w_{oh}(i,k) \cdot O_p(k) \tag{2.4}$$

which can be written in vectorized form as

$$\boldsymbol{y}_p = \mathbf{W}_{oi} \cdot \boldsymbol{x}_p + \mathbf{W}_{oh} \cdot \boldsymbol{O}_p \tag{2.5}$$

The functional link net (FLN) consists of input and output layers, but no hidden layers. An input layer has enhanced input values which are created by various functional links with original input values [17]. In the FLN [18] notation it is possible to write equation (2.5) as

$$y_p(i) = \sum_{n=1}^{\mathrm{N}_u} w_o(i,n) X_p(n) \tag{2.6}$$

where $\mathrm{N}_u = \mathrm{N} + 1 + \mathrm{N}_h$ is the number of basis functions and $X_p(n)$ is the n^{th} element of the basis vector $\boldsymbol{X}_p$. Equation (2.7) can be written in matrix format as

$$\boldsymbol{y}_p = \mathbf{W}_o \cdot \boldsymbol{X}_p \tag{2.7}$$

where

$$\boldsymbol{X}_p = [\boldsymbol{x}_p{}^{\mathrm{T}} : \boldsymbol{O}_p{}^{\mathrm{T}}]^{\mathrm{T}} \tag{2.8}$$

In Figure 2-2, the FLN model of the MLP is depicted.

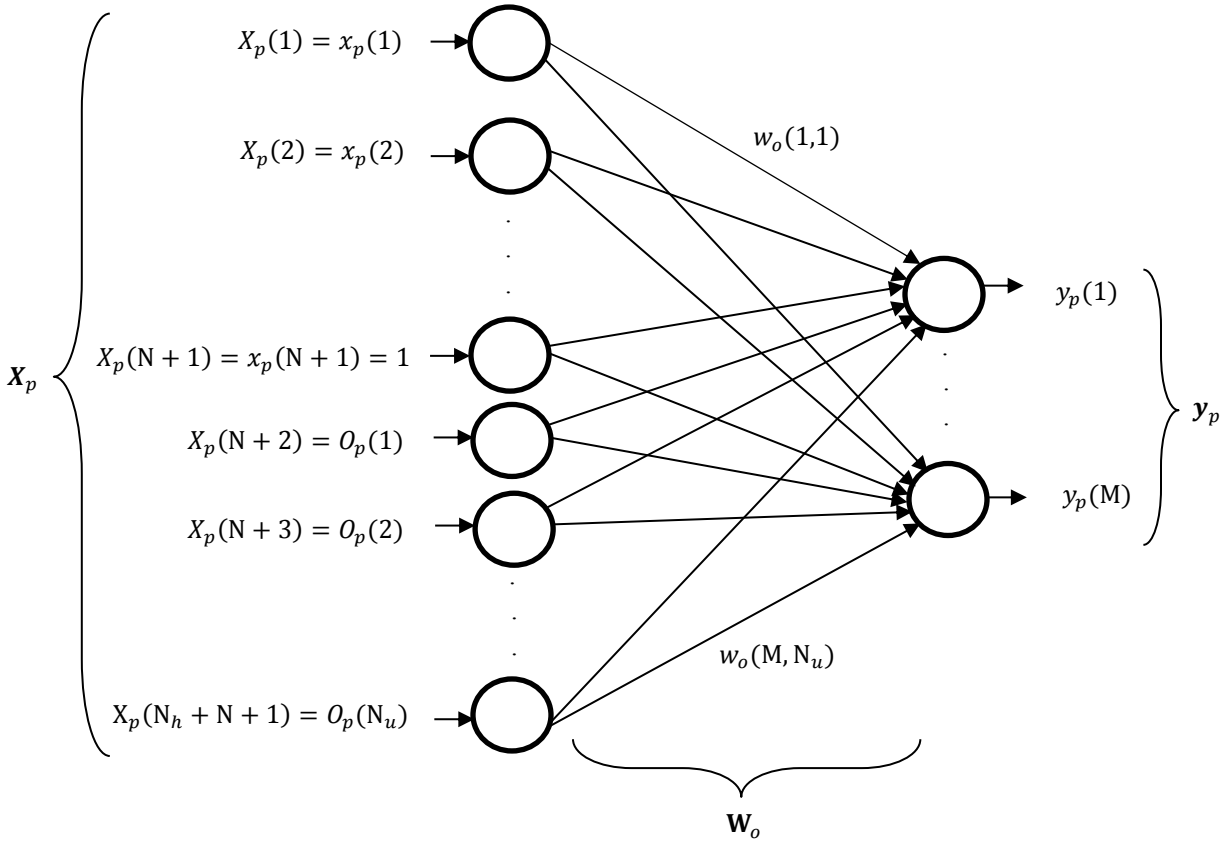


Figure 2-2: FLN model of MLP

B. The MLP training

MLP utilizes first order learning techniques such as back-propagation (BP) and conjugate gradient (CG), and second order techniques related to Newton's method and Levenberg-Marquardt (LM) for training the network.

The problem of training a MLP can be transformed into the minimization of a well-defined cost function. The most commonly used and the 'traditional' cost function is the mean square error, MSE, and it is denoted as E for convenience in this work. The mapping error for the i^{th} output unit is defined as

$$\mathbf{E}_o(i) = \frac{1}{\mathrm{N}_v}\sum_{p=1}^{\mathrm{N}_v}\left(t_p(i) - y_p(i)\right)^2 \tag{2.9}$$

where $t_p(i)$ is the i^{th} desired output in the M-dimensional desired output vector $\boldsymbol{t}_p \in \mathbb{R}^{\mathrm{M}\times 1}$. The MSE over a training dataset is called the training error and is given by

$$\mathrm{E} = \sum_{i=1}^{\mathrm{M}} \mathbf{E}_o(i) = \frac{1}{N_v} \sum_{p=1}^{\mathrm{N}_v} \sum_{i=1}^{\mathrm{M}} \left(t_p(i) - y_p(i)\right)^2 \tag{2.10}$$

1) Steepest descent and conjugate gradient

In steepest descent all the network weights are updated in the direction of negative gradients of the error function [19]

$$\mathbf{w} = \mathrm{vec}(\mathbf{W}, \mathbf{W}_{oi}, \mathbf{W}_{oh}) \tag{2.11}$$

$$\mathbf{g} \equiv \frac{-\partial \mathrm{E}}{\partial \mathbf{w}} \tag{2.12}$$

where $\mathbf{g}$ is the negative gradient with respect to E. The weights are updated by

$$\mathbf{w} \leftarrow \mathbf{w} + z.\mathbf{g} \tag{2.13}$$

in which z is the scalar learning factor containing the information about the step size to be taken in the direction of $\mathbf{g}$.

Conjugate gradient method is the most popular iterative method for solving large systems of linear equations. Use of the CG is motivated by the desire to accelerate the typically slow rate of convergence experienced with the method of steepest descent, while avoiding the computational requirements associated with the evaluation, storage, and inversion of the Hessian in Newton's method. CG can be considered first order as the error function is not quadratic.CG is effective in minimization of the quadratic functions.

In the conjugate gradient method, the weights are not directly updated using the gradient vector; instead they are updated using a direction vector $\mathbf{p}$ in each iteration. For the k^{th} iteration the vector $\mathbf{p}$ is

$$\mathbf{p}_k = \text{vec}(\mathbf{p}, \mathbf{p}_{oi}, \mathbf{p}_{oh}) \tag{2.14}$$

and the weights are updated as

$$\mathbf{w} \leftarrow \mathbf{w} + z.\mathbf{p}_k \tag{2.15}$$

2) Output weight optimization – backpropagation (OWO-BP)

The most popular first-order learning algorithm known as back propagation algorithm (BP) is relatively simple and it can handle problems with basically an unlimited number of patterns. It is the standard algorithm for training supervised feed-forward neural nets. However, the BP algorithm is slow. Many improvements have been made to speed up the BP algorithm and some of them, such as momentum and adaptive learning constant work relatively well [20]. But as long as first-order algorithms are used, improvements are not dramatic.

OWO is a technique to calculate output weights including hidden output weights and bypass weights after the input weight matrix is determined in some fashion. With the linearity property of the output units, as in most cases, the output weights are found by solving a set of linear equations.

The OWO-BP technique iteratively solves linear equations for the output weights and uses back-propagation with full batching to change input weights. So, one option to train an MLP would be to use two stage training.

As the output units have linear activation functions, the OWO procedure for finding the output weights can be realized by solving linear equations that result when gradients of E with respect to the output weights are set to zero ($\frac{\partial \mathrm{E}}{\partial \mathbf{W}_o} = 0$) leading to a set of linear equations as

$$c(i,m) = \sum_{n=1}^{\mathrm{N}_u} w_o(i,n) r(n,m) \tag{2.16}$$

$$\mathbf{C} = \mathbf{R} \cdot \mathbf{W}_o^{\mathrm{T}} \tag{2.17}$$

where $\mathbf{C}$ is M by N_u cross-correlation matrix and $\mathbf{R}$ is N_u by N_u auto-correlation matrix defined as

$$\mathbf{C} = \frac{1}{N_v}\sum_{p=1}^{N_v} \boldsymbol{X}_p \boldsymbol{t}_p{}^{T} \tag{2.18}$$

$$\mathbf{R} = \frac{1}{N_v}\sum_{p=1}^{N_v} \boldsymbol{X}_p \boldsymbol{X}_p{}^{T} \tag{2.19}$$

Equation (2.17) can be best solved through using orthogonal least squares [8] method. Finding the output weight matrix in this fashion is called output weight optimization (OWO) and it has the advantage of being fast due to linearity property of the equations and it avoids local minima. Therefore, in a given iteration of OWO-BP, we repeat the following steps:

I. Performing OWO: solve the system of linear equations in (2.17) and find the output weight matrix $\mathbf{W}_o$.

II. Find the negative Jacobian matrix in equation (2.12) for $\mathbf{W}$.

III. Performing BP: update the input weights $\mathbf{W}$ using equation (2.13) for the input weights.

and during either stages, the other weights are not updated.

3) Output weight optimization – hidden weight optimization (OWO-HWO)

Hidden weights can be updated by minimizing separate error functions for each hidden unit. The error functions measure the difference between the desired and the actual net function. By minimizing many simple error functions instead of one large one, it is assumed that the training speed and convergence can be improved. The desired net function can be approximated by the current net function plus a designed net change. For the k^{th} hidden unit and p^{th} pattern, the desired net function is constructed as

$$n_{pd}(k) \cong n_p(k) + z \cdot \delta_p(k) \tag{2.20}$$

where $n_{pd}(k)$ is the desired net function and $n_p(k)$is the actual net function for the k^{th} hidden unit, z is the learning factor and $\delta_p(k)$ is the delta function for the k^{th} hidden. In this algorithm, the hidden weights are to be updated as

$$w(k,n) \leftarrow w(k,n) + z \cdot e(k,n) \tag{2.21}$$

where $e(k,n)$ is the weight change and serves the same purpose as the negative gradient element, $\frac{-\partial \mathrm{E}}{\partial w(k,n)}$, in back-propagation method. The following equation can be used to solve for the changes in the hidden weights

$$n_p(k) + z \cdot \delta_p(k) \cong \sum_{n=1}^{\mathrm{N}+1} [w(k,n) + z \cdot e(k,n)] x_p(n) \tag{2.22}$$

deleting the current net function and eliminating the learning factor z from both sides we get

$$\delta_p(k) \cong \sum_{n=1}^{\mathrm{N}+1} e(k,n) x_p(n) \tag{2.23}$$

Defining the objective function for the k^{th} hidden unit as

$$\mathrm{E}_\delta(k) = \frac{1}{\mathrm{N}_v} \sum_{p=1}^{\mathrm{N}_v} \left(\delta_p(k) - \sum_{n=1}^{\mathrm{N}+1} e(k,n) x_p(n) \right)^2 \tag{2.24}$$

taking the gradient of the above error function with respect to the weight changes we get

$$g(m) \equiv \frac{-\partial \mathrm{E}_\delta(k)}{\partial e(k,m)} = -2 \left(c_\delta(k,m) - \sum_{n=1}^{\mathrm{N}+1} e(k,n) r(n,m) \right) \tag{2.25}$$

where

$$c_\delta(k,m) = \frac{1}{\mathrm{N}_v} \sum_{p=1}^{\mathrm{N}_v} \left(\delta_p(k) \cdot x_p(m) \right), r(n,m) = \frac{1}{\mathrm{N}_v} \sum_{p=1}^{\mathrm{N}_v} x_p(n) x_p(m) \tag{2.26}$$

equating equation (2.25) to zero ($\frac{-\partial \mathrm{E}_\delta(k)}{\partial e(k,m)} = 0$) we would have

$$\sum_{n=1}^{\mathrm{N}+1} e(k,n)\, r(n,m) = c_\delta(k,m) \tag{2.27}$$

or in matrix format

$$\mathbf{G}_{hwo} \cdot \mathbf{R}_{\boldsymbol{i}} = \mathbf{G} \tag{2.28}$$

where $\mathbf{G}_{hwo}$ is the matrix of hidden weight changes, $\mathbf{R}_{\boldsymbol{i}}$ is the $\mathrm{N}+1$ by $\mathrm{N}+1$ input auto-correlation matrix, and $\mathbf{G}$ is the negative gradient matrix. Similarly we can have

$$\mathbf{G}_{hwo} = \mathbf{G} \cdot \mathbf{R}_i^{-1} \tag{2.29}$$

Solving equation (2.29) for $\mathbf{G}_{hwo}$ and finding the hidden weight changes we can update the hidden weights [21] by

$$\mathbf{W} \leftarrow \mathbf{W} + z \cdot \mathbf{G}_{hwo} \tag{2.30}$$

C. Second order training

Second order methods have better convergence than first order methods; however, some care must be taken when employing them because the full network Hessian is inherently rank deficient that can create problems in training. Second order algorithms, such as Newton's algorithm [22] and Levenberg-Marquardt (LM) algorithm [23], use Hessian matrix to perform better estimations on both step sizes and directions, so that they can converge much faster than first order algorithms.

1) Newton's method

Newton's algorithm is the basis of a number of popular second order optimization algorithms including Levenberg-Marquardt [23] and BFGS. It is derived from a second order Taylor series approximation to the error function about the point $\boldsymbol{w}$ [22]. Using Newton's method in the MLP, we need to calculate the necessary first and second derivatives of E with respect to the weights. Newton's algorithm is an iterative method where in each iteration it

I. calculates the Newton's weight change vector, $\boldsymbol{e} = \Delta\mathbf{w}$ **,** and

II. updates the weights with this weight change vector.

The weight change vector is calculated by solving the linear equations of

$$\mathbf{H} \cdot \boldsymbol{e} = \mathbf{g} \tag{2.31}$$

where **g** is the vector of negative gradient and **H** is the Hessian of the objective function calculated with respect to all the weights in the network and has elements defined as

$$h(n,m) = \frac{\partial^2 \mathrm{E}}{\partial w(k,n)\partial w(l,m)} = \frac{2}{\mathrm{N}_v}\sum_{p=1}^{\mathrm{N}_v}\sum_{i=1}^{\mathrm{M}} \frac{\partial y_p(i)}{\partial w(k,n)} \cdot \frac{\partial y_p(i)}{\partial w(l,m)} \tag{2.32}$$

Equation (2.31) can be solved using conjugate gradient or orthogonal least squares and the weight changes can be found. The weights are then updated as

$$\mathbf{w} \leftarrow \mathbf{w} + \boldsymbol{e} \tag{2.33}$$

Second order algorithms related to Newton's method often have non-positive definite or singular Hessian matrices which may result in unstable training. In this case the Levenberg-Marquardt (LM) algorithm is used.

2) Levenberg–Marquardt (LM)

Levenberg–Marquardt algorithm is regarded as one of the most efficient algorithms for training small and medium sized patterns. It is a standard technique used to solve nonlinear least squares problems. The LM algorithm is very computationally intensive due to the large size of the Hessian matrix so its use in training large networks is often limited. This method is a compromise between the following two methods

- Gauss-Newton's method, which converges rapidly near a local or global minimum, but may also diverge;
- Gradient descent, which is assured of convergence through a proper selection of the step-size parameter, but converges slowly.

Generally in the gradient descent method, the sum of the squared errors is reduced by updating the parameters in the steepest-descent direction. And in the Gauss-Newton method, the

sum of the squared errors is reduced by assuming the least squares function is locally quadratic, and finding the minimum of the quadratic. The Levenberg-Marquardt method acts more like a gradient-descent method when the parameters are far from their optimal value, and acts more like the Gauss-Newton method when the parameters are close to their optimal value. The LM algorithm is a sub-optimal method which updates all weights as

$$\mathbf{w} \leftarrow \mathbf{w} + [\mathbf{H} + \lambda \cdot diag(\mathbf{H}).\mathbf{I}]^{-1} \cdot \boldsymbol{g} \tag{2.34}$$

where λ is a controlling factor which tunes LM either towards the first order or the second order methods, and $\mathbf{I}$ is the identity matrix. For training the network using LM we take the following steps [24]:

I. Present all patterns to the network and compute the mean square error (MSE).

II. Compute the Hessian and gradient matrices for all the weights.

III. Calculate the updated weights using equation (2.34).

IV. Re-compute the MSE by using the updated weights, if the new error is smaller than that computed in step (I) then reduce λ and go back to step (I); if the error is not reduced then increase λ.

V. The algorithm converges when the norm of the gradient is less than some pre-determined value, so the MSE has been reduced to a fixed error.

3) Affine invariance in MLP training

In training the MLP, using an objective function E and initial weights vector $\mathbf{w}$

$$\mathbf{w} = \mathbf{A} \cdot \mathbf{w}' \tag{2.35}$$

and the vector of weight changes $\boldsymbol{e}$

$$\boldsymbol{e} = \mathbf{A} \cdot \boldsymbol{e}' \tag{2.36}$$

where **A** is a constant and nonsingular matrix, then the MLP training algorithm has affine invariance for the initial weights vector if E satisfies

$$\mathrm{E}(\mathbf{w} + \boldsymbol{e}) = \mathrm{E}(\mathbf{A} \cdot (\mathbf{w}' + \boldsymbol{e}')) \tag{2.37}$$

An affine invariance algorithm will yield the same sequence of iteration for a countless infinite number of different initial weight vectors. Therefore, using an affine invariant training is the first step towards making MLP training insensitive to initial weights. Unfortunately, most training algorithm, including BP, CG, and LM, lack affine invariance [25].

D. Basic MLP properties

1) Minimum mean square error estimator

One of the properties of neural network is that it is a minimum mean square error estimator. To understand this from the statistical point of view, suppose $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{t} \in \mathbb{R}^m$ are random vectors, we seek to estimate $\boldsymbol{t}$ given $\boldsymbol{x}$; thus, we seek a function φ: $\mathbb{R}^m \to \mathbb{R}^n$, such that $\boldsymbol{y} = \varphi(\boldsymbol{x})$ is near $\boldsymbol{t}$. One common measure of nearness is the mean-square error and minimum mean-square estimator (MMSE) minimizes the MSE. As MSE is defined in our notations

$$\mathrm{MSE} = \frac{1}{\mathrm{N}_v} \sum_{p=1}^{\mathrm{N}_v} (\boldsymbol{t}_p - \boldsymbol{y}_p)^2 \tag{2.38}$$

then the MMSE is the conditional expectation of $\boldsymbol{x}$ given $\boldsymbol{t}$ as

$$\boldsymbol{y}_{mmse} = \varphi_{mmse}(\boldsymbol{x}) = \underset{\boldsymbol{y}}{\operatorname{argmin}}\, \mathrm{MSE} \tag{2.39}$$

In the MLP, since the training minimizes MSE with respect to the weights vector **w**, the MLP is effectively a minimum mean squared error approximation to the Bayes optimal discriminant function [26].

2) Universal approximation

Theorem: "*Let f be a non-constant, bounded and monotonically increasing function. Let $\boldsymbol{x}$ be the input with dimensionality of* $1 \times \mathrm{N}$. *If each of the input vectors are drawn from a specific*

distribution and $\varepsilon > 0$*, there exist a number,* M*, and real valued constants* $x(\mathrm{N}+1)$*,* $\mathbf{W}_{oh}$*, and* $\mathbf{W}$ *such that an output* $\boldsymbol{y}$ *with dimensionality of* $1 \times \mathrm{M}$ *can be defined as an approximate realization of the desired output* $\boldsymbol{t}$*"* [27], [28]

$$|y(i) - t(i)| < \varepsilon, \qquad for\ 0 < i \leq \mathrm{M} \tag{2.40}$$

Universal approximation theorem states that single hidden layer feed-forward networks can approximate any measurable function arbitrarily well regardless of the activation function, the dimension of the input space, and the input space environment [29]. The universal approximation theorem is directly applicable to multilayer perceptrons.

The universal approximation theorem is important from a theoretical viewpoint because it provides the necessary mathematical tool for the viability of feed-forward networks with a single hidden layer as a class of approximate solutions. Without such a theorem, we could conceivably be searching for a solution that cannot exist. However, the theorem is not constructive; since it does not actually specify how to determine a multilayer perceptron with the stated approximation properties. The universal approximation theorem assumes that the continuous function to be approximated is given and that a hidden layer of unlimited size is available for the approximation. Both of these assumptions are violated in most practical applications of multilayer perceptrons [20].

3) Bayes discriminant

A Bayes discriminant which minimizes the probability of error, P_e, can be expressed in any of the following three forms

$$f(x|i)\mathrm{P}(i) \tag{2.41}$$

$$\mathrm{g}(f(x|i)\mathrm{P}(i)) \tag{2.42}$$

$$\mathrm{P}_b(i|x) \tag{2.43}$$

where g(·) is either an increasing or decreasing function.

Theorem states that: "*When MLP classifiers are trained to minimize the mean-squared error, the MSE approaches a constant value plus the expected squared error between the classifier output and Bayes discriminant, as the number of training patterns approaches infinity*."

The Bayes posterior probability for an input vector $\boldsymbol{x}$ for the p^{th} training pattern, belonging to a class i within the output discriminant vector d_1 is given by

$$d_1\,(i) \;=\; \mathrm{P}_b(i|x_p) \tag{2.44}$$

4) Memorization

Memorization or over-fitting is one of the problems that mostly occurs during training a neural network due to having too many numbers of free parameters comparing to the number of data samples. In this case, the network memorizes the training samples rather than learning them so that it cannot generalize to the new and unseen samples. The number of patterns a neural network can memorize is called the information capacity. It is very important to find the upper bound on memorization in order to design an optimal neural network.

For a MLP, the known parameters are the number of inputs, the number of training patterns, and the number of outputs. Therefore, the only parameters which can be manipulated are the number of hidden units and the number of training iterations. The upper bound for the MLP, C_{MLP}, also called the storage capacity of the MLP is found as

$$C_{\mathrm{MLP}} \leq \frac{\mathrm{N}_w}{\mathrm{M}} \tag{2.45}$$

where N_w is the total number of weights in the network as in

$$\mathrm{N}_w = (\mathrm{N}+1)\mathrm{N}_h + \mathrm{N}_h\mathrm{M} + (\mathrm{N}+1)\mathrm{M} \tag{2.46}$$

In order to prevent memorization and increase generalization, the number of patterns of a dataset should satisfy the condition

$$\mathrm{N}_v \geq \frac{\mathrm{N}_w}{\mathrm{M}} \tag{2.47}$$

or in another word, it has to be greater than the C_{MLP}. From equation (2.47) and (2.46), the number of hidden units that should be chosen for a given MLP can be found as

$$\mathrm{N}_h \leq \frac{\mathrm{M}(\mathrm{N}_v - \mathrm{N} - 1)}{\mathrm{N} + 1 + \mathrm{M}} \tag{2.48}$$

This upper bound is independent of the activation function used and is valid for most of the feed forward neural networks, irrespective of the connectivity of the network [30].

Chapter 3

THE MLP ADVANCE TRAINING

In this chapter, we discuss a few advance MLP training algorithms.

A. OWO-MOLF

This is a two-step algorithm in which for every hidden unit an optimal learning factor z_k (learning factor for the k^{th} hidden unit) is calculated using the multiple optimal learning factor method (MOLF) [14], [31]. Generally, finding a vector of optimal learning factors $\boldsymbol{z}$ which has one element for each hidden unit increases the speed of learning and overall convergence. In this method the first and second partial derivative of the error function with respect to $z_k$is computed and the error function is minimized accordingly

$$w(k,n) \leftarrow w(k,n) + z_k \cdot \mathrm{g}(k,n). \tag{3.1}$$

The error function to be minimized is given by equation (2.10), and the predicted output $y_p(m)$ is given by

$$y_p(m) = \sum_{n=1}^{\mathrm{N+1}} w_{oi}(m,n)x_p(n) + \sum_{k=1}^{\mathrm{N}_v} w_{oh}(m,k) f\left(\sum_{i=1}^{\mathrm{N+1}} (w(k,i) + z_k \cdot \mathrm{g}(k,n))x_p(i)\right) \tag{3.2}$$

taking the first partial derivative of E with respect to z_k, we have

$$\mathrm{g}_{mlof}(j) = \frac{-\partial \mathrm{E}}{\partial z_j} = \frac{2}{N_v} \sum_{p=1}^{\mathrm{N}_v} \sum_{m=1}^{\mathrm{M}} \left[\bar{t}_p(m) - \sum_{k=1}^{\mathrm{N_h}} w_{oh}(m,k) O_p(z_k)\right] \cdot w_{oh}(m,j) O_p{}'(j) \Delta n_p(j) \tag{3.3}$$

where

$$\bar{t}_p(m) = \mathrm{t}_p(m) - \sum_{m=1}^{\mathrm{N+1}} w_{oh}(m,n)x_p(n) \tag{3.4}$$

$$\Delta n_p(j) = \sum_{m=1}^{\mathrm{N+1}} x_p(n) \cdot \mathrm{g}(j,n) \tag{3.5}$$

$$O_p(z_k) = f\left(\sum_{n=1}^{\mathrm{N}+1}(w(k,n) + z_k \cdot \mathrm{g}(k,n))x_p(n)\right) \tag{3.6}$$

using Gauss-Newton updates, the second partial derivative elements of the Hessian $\mathbf{H}_{molf}$ are derived as

$$h_{molf}(l,j) = \frac{\partial^2 \mathrm{E}}{\partial z_l \partial z_j} = \frac{2}{\mathrm{N}_v}\sum_{p=1}^{\mathrm{N}_v}\sum_{m=1}^{\mathrm{M}}\frac{\partial y_p(m)}{\partial z_l}\frac{\partial y_p(m)}{\partial z_j} \tag{3.7}$$

which is

$$\begin{aligned} h_{molf}(l,j) &\approx \frac{2}{\mathrm{N}_v}\sum_{m=1}^{\mathrm{M}} w_{oh}(m,j)w_{oh}(m,l)\sum_{p=1}^{\mathrm{N}_v}\Delta n_p(j)\Delta n_p(l){O'}_p(j){O'}_p(k) \\ &= \sum_{i=1}^{\mathrm{N}+1}\sum_{n=1}^{\mathrm{N}+1}\left[\frac{2}{N_v}u(l,j)\sum_{p=1}^{\mathrm{N}_v}x_p(i)x_p(n){O'}_p(j){O'}_p(l)\right]\mathrm{g}(l,i)\mathrm{g}(j,n) \end{aligned} \tag{3.8}$$

$$u(j,k) = \sum_{m=1}^{\mathrm{M}} w_{oh}(m,j)w_{oh}(m,k) \tag{3.9}$$

Given the negative gradient vector $\mathbf{g}_{molf} = [-\frac{\partial \mathrm{E}}{\partial z_1}, -\frac{\partial \mathrm{E}}{\partial z_2}, \cdots, -\frac{\partial \mathrm{E}}{\partial z_{\mathrm{N}_h}}]^{\mathrm{T}}$ and the Hessian $\mathbf{H}_{molf}$, the error functionE is minimized with respect to the vector $\mathbf{z}$ using Newton's method. The training algorithm steps for every epoch are:

I. Find the negative Jacobean matrix $\mathbf{G}$ and solve $\mathbf{G}_{hwo} = \mathbf{G} \cdot {\mathbf{R}_i}^{-1}$ for $\mathbf{G}_{\boldsymbol{hwo}}$.

II. Solve $\mathbf{H}_{molf} \cdot \boldsymbol{z} = \mathbf{g}_{\mathrm{molf}}$ for $\boldsymbol{z}$ and update the input weights as in equation (3.1)

III. Perform OWO for output weights.

B. Standard OLS pruning

The purpose of pruning the hidden units is to eliminate less useful hidden units that have no relevant information for estimating the outputs or are linearly dependent on inputs or on other hidden units. It is a method to avoid over-fitting and memorization, and to gain better generalization.

In [21], orthogonal least squares (OLS) [8] or Schmidt procedure has been used for pruning the neural network's hidden units. OLS is equivalent to QR decomposition [8]. In [21], using the Schmidt procedure, the orthonormal basis functions are found and the inputs and the hidden units are optimally ordered. In order to find the orthonormal basis function, $\boldsymbol{X'}$, an **A** matrix that converts the network's basis function to $\boldsymbol{X'}$ has to be found. Elements of **A** matrix is computed by given the ordered function, $o(m)$, for $m = 1$, $a_{11} = \frac{1}{\|x_{o(1)}\|} = \frac{1}{r(o(1),o(1))^{1/2}}$ in which $r(i,j)$ is the $i^{th}$and j^{th} element of auto-correlation matrix **R,** and then for $1 \leq m \leq \mathrm{N}_u$ calculate

$$c_i = \sum_{k=1}^{m} a_{mk} \cdot r(o(m), o(k)) \text{ for } 1 \leq i \leq m-1 \tag{3.10}$$

set $b_m = 1$ and get

$$b_k = -\sum_{i=k}^{m-1} c_i \cdot a_{ik} \quad \text{for } 1 \leq k \leq m-1 \tag{3.11}$$

get coefficients a_{mk} as

$$a_{mk} = \frac{b_k}{\left[r\big(o(m), o(m)\big) - \sum_{i=1}^{m-1} c_i^2\right]^{1/2}} \quad \text{for } 1 \leq k \leq \mathrm{m} \tag{3.12}$$

after finding all the coefficients of **A**, the orthonormal basis functions are found as

$$\boldsymbol{X'} = \mathbf{A} \cdot \boldsymbol{X} \tag{3.13}$$

The output weight matrix is in the normal system found by mapping back the orthonormal weights, $\mathbf{W'}$, as

$$\mathbf{W}_o = \mathbf{W'} \cdot \mathbf{A} \tag{3.14}$$

where

$$\mathbf{W'} = \mathbf{C}^{\mathrm{T}} \cdot \mathbf{A}^{\mathrm{T}} \tag{3.15}$$

in which **C** is the cross-correlation matrix in equation (2.18).

C. One-pass validation

One-pass validation refers to the fact that in each epoch, the algorithm requires a pass through the validation data for once. In [21] in order to stop the network's training, the author combined the optimally ordered neural network with the early stopping method. So, the validation error vs. the ordered basis functions is obtained for this purpose. Given the matrix **A** and the MLP network with the ordered basis functions, the validation error versus hidden units curve $\mathrm{E}_v(\mathrm{N}_h)$ from the validation data is generated. In order to get the validation error for all network size in a single pass through the data, first the linear network output is obtained and the corresponding error is calculated, then for $1 \leq m \leq \mathrm{N}_h$ the following two steps are performed for $1 \leq p \leq \mathrm{N}_v$ where w' is the orthonormal output weights obtained from previous section

$$y_p(i,m) = y_p(i,m-1) + w'(i,m+1+\mathrm{N}).X'_v(m+1+\mathrm{N}) \tag{3.16}$$

$$\mathrm{E}_v(m) \leftarrow \mathrm{E}_v(m) + \sum_{i=1}^{\mathrm{M}} \left[t_p(i) - y_p(i,m)\right]^2 \tag{3.17}$$

then these error values are normalized as

$$\mathrm{E}_v \leftarrow \frac{\mathrm{E}_v}{\mathrm{N}_v} \tag{3.18}$$

D. MOLF-Adapt

MOLF-Adapt algorithm is a batch training algorithm for the MLP that optimizes validation error with respect to two parameters. At the end of each training epoch, the method temporarily prunes the network and calculates the validation error versus hidden units curve in one pass through the validation data. Since pruning is done at end of each epoch and the best networks are saved, the validation error is optimized over the number of hidden units and the number of epochs simultaneously. In this algorithm OWO-MOLF is used for training and updating the network weights and OLS is used both for finding the unknowns and pruning the hidden units permanently during training as well. At the end of each OWO-MOLF epoch, the OLS is utilized to

optimally order basis functions and then one pass validation is performed to get the validation error vs. the hidden units curve. For each network size, if the validation error for that network size is lower than that from the previous epoch, the network weights are saved. The training algorithm steps for every epoch are:

I. Find the negative Jacobean matrix $\mathbf{G}$ and solve $\mathbf{G}_{hwo} = \mathbf{G} \cdot \mathbf{R}_i^{-1}$ for $\mathbf{G}_{\boldsymbol{hwo}}$.

II. Solve $\mathbf{H}_{molf} \cdot \boldsymbol{z} = \mathbf{g}_{\text{molf}}$ for $\boldsymbol{z}$ and update the input weights as in equation 3.1).

III. Perform OWO for output weights using OLS.

IV. Perform one-pass validation and find the optimum network size.

V. Prune the network if better network is generated through step IV.

Chapter 4

SPARSITY

Sparsity in artificial neural networks mostly refers to the sparse structure of the network; it can either relate to the sparse connectivity or sparse activity. It can involve pruning the network's weights or pruning the network's hidden units. Sparse connectivity removes the useless connections of the network while maintaining the networks performance [32]. Sparseness can be used as another way of avoiding overtraining and it is considered as a powerful regularizer. In general, over-fitting is thought to happen when the network has more degrees of freedom (the number of weights) than the number of the training samples-when there are not enough examples to constrain the network [33], and it results to have a poor generalization. Therefore, applying sparseness to the network's connections can decrease the number of free parameters in the network so that it can overcome over-fitting in applications that are prone to memorization. Sparse networks are mostly of smaller size in comparison to their fully connected ones and are more likely to give higher accuracy than non-sparse networks.

There exist models that incorporate sparse information processing with respect to sparse activity or sparse connectivity [11]. Sparse coding denotes the idea that signals can be approximated by superposition of very few elements of a large dictionary. The field of sparse coding has received a lot of attention since the evolving work of [11]**.** Optimal brain damage (OBD) method is also devoted to sparse connectivity of a network and it is not based on sparse coding. OBD uses a measure of 'saliency' in the objective function and determines the impact of each connection on the training error. Smaller saliency increases the possibility that removing the corresponding connection will have less impact on the overall training error [34]. Optimal brain surgeon (OBS) is an extension to OBD but it is not practical for large networks due to huge computation of the entire Hessian matrix [35]. Non-negative matrix factorization (NMF) is another technique that tries to find a suitable linear representation of non-negative data [36] [37].

One of the most useful properties of NMF is that it usually produces a sparse representation of data [9]. Yet, NMF has no control over the sparseness degree of the data representation. In non-negative matrix factorization with sparseness constraints (NMFSC), the author proposed a method in which extends the NMF idea and gives the option to control the degree of sparseness explicitly through introducing a measure of sparseness. In [9], $\mathrm{SP}(\boldsymbol{x})$ or the sparsity measurement is given for a d-dimensional vector $\boldsymbol{x}$ as

$$\mathrm{SP}(\boldsymbol{x}) = \frac{\sqrt{d} - \|\boldsymbol{x}\|_1 / \|\boldsymbol{x}\|_2}{\sqrt{d} - 1} \tag{4.1}$$

In [38] the theory of sparse multi-layer perceptron (SMLP) is discussed in which the outputs of one of the hidden layers are forced to be sparse by adding a sparse regularization term to the cross-entropy cost function where this term is a trade-off between the sparsity and the cross-entropy cost function.

In [11], a proposed method for SMLP has been proposed that implements sparse connectivity paradigm for a two layer MLP structure. In this method, the weight matrix of the hidden layer is restricted to be sparsely populated by only using the sparse connectivity constraints.

In this study, sparse connectivity for a two layer MLP has been achieved by using OLS algorithm and a sparsely populated output weight matrix is generated with a reasonable measurement of sparseness as in equation (4.1).

A. Prevention of over-training through regularization

Regularization is a penalized-based method which adds an extra term to the objective function in terms of the network weights to prevent memorization or over-training. There are two most commonly used regularization types, L_1 and L_2 regularization. In the L_2 regularization, the complexity penalty term is defined as the squared norm of all the free parameters or weights. This approach operates by forcing some of the network's weights to take a close-to-zero value, while

permitting other weights to retain their relatively large values [20]. The L_2 regularization is also commonly referred as weight-decay procedure. The L_1 regularization, on the other hand, employs the absolute values of all the free parameters as the complexity penalty term in the objective function. The L_1 regularization concentration is mostly on a very small set of highly important connections, while making the rest of the weights zero. It is worth mention that while using the L_1regularization in training, the weight vectors in the network tend to become sparse.

In [39], the L_1 regularization penalty on the linear output weights is used to build a sparse MLP with one hidden layer. Also, the author uses fast iterative shrinkage thresholding algorithm (FISTA) proximal optimization algorithms for finding output weights, and the hidden units are computed by unconstrained minimization.

As a conclusion, sparseness can implicitly be a solution to avoid memorization since it reduces the number of free parameters in a network and resulting in a better generalization.

B. Inducing sparsity

The concept of sparsity [10] is heavily used in many distinct areas such as antennas and propagation [40], face recognition [41], image processing [42], and medical imaging [43]. Also, sparsity has been utilized in advancement of many machine learning algorithms and techniques such as matrix factorization [44], compressed sensing [45], signal representation [46], support vector machines [47], sampling theory [48], and many more.

1) Feature selections on inputs: Feature selection refers to the process of selecting a subset of features that can retain most of the intrinsic information content of the original data [49]. Feature selections can help in reducing the possibility of overtraining [50]as well as computational time, improving prediction performance, and a better understanding of the data in machine learning or pattern recognition applications [51].

Feature selection technique should not be mistaken with dimension reduction techniques. The former mostly concentrates on selection of the information-rich features whereas the latter

concentrates on creation of a new combination of features by using a kind of transformation. The most commonly used dimension reduction methods are principle component analysis (PCA), and singular value decomposition (SVD). PCA is a standard technique that is widely used for linear dimensionality reduction in statistical pattern recognition and signal processing [20].

A popular way to minimize the information content in a feature vector or presentation is to make its components sparse. In sparse methods, the feature vector is forced to contain very few non-zero values while the rest of the values are zeros [52]. NMF, Olshausenand Field's sparse coding method, or the energy-based models are all sparse-overcomplete representations methods whose features are sparse in high-dimension [53]. Recently, several works have advocated the use of sparse overcomplete representations for images, in which the dimension of the feature vector is larger than the dimension of the input, but only a small number of components are non-zero for any one image [54]. In [52], an unsupervised method to produce sparse overcomplete representations based on the encoder-decoder paradigm namely sparse encoding symmetric machine is proposed.

2) Transformation: As discussed earlier, many feature selection methods have been using different transformation techniques to reduce the dimensionality of input vectors such as PCA, SVM, and NMF. Recently, efforts have been made to use neural networks for feature generation and selection, auto-associative networks, where during training the desired outputs are set as the inputs. In this method, the transformation from inputs to features is derived using a neural network that is optimized to minimize the reconstruction error. This technique has been used in creating sparse features in many algorithms. In [38], sparse features are derived at the internal hidden-layer outputs of a MLP-structured network which is trained to classify multiple classes by introducing a sparse regularization term to the cost function.

3) Independent component analysis: ICA is a higher-order de-correlation method that seeks to find a linear transformation of a non-Gaussian data in which its components are statistically as

independent as possible [55]. ICA maximizes the independency via minimizing the mutual information, MMI, or maximizing the non-Gaussianity. In [56], ICA has been used in an infomax network and it is shown that the resulting ICA filters have more sparsely distributed outputs on natural images. They show that these filters are similar to those of Olshausen and Field [13].

4) Pruning: In order to choose the appropriate network size, there are two kinds of algorithms which are often used: the growing algorithm and the pruning algorithm. The growing algorithms start from a small-sized network and gradually increase the number of network's units and layers until they meet the learning requirements, such as cascade correlation algorithm [33]. On the other side, the pruning algorithms involves three distinct processes to achieve the suitable network's size: at first, a very large network with many redundant connections and units is trained, in the second stage the redundant units and connections are removed based on some network's requirements and conditions, and in the last stage the network resumes training and updates the remaining weights. These steps will continue until the network's performance satisfies some principle conditions. There are various ways to implement pruning but mostly it can be put into two main categories: one is the sensitivity method which measures the sensitivity of the objective function in respect to elimination of an element of network. The other method is based on penalization which adds a new term to the objective function and rewards the network for finding an efficient solution [33].

As mentioned before, the most popular sensitivity based algorithms are optimal brain damage (OBD) [34] and optimal brain surgeon (OBS) [35]. The sensitivity based methods attempt to find the contribution of each weight or hidden unit in the network and then prunes the weights or hidden units that have the least effect on the objective function. Many other pruning algorithms have been proposed based on the theory of OBD and OBS later.

Another technique, Iterative Pruning (IP) algorithm [57], solves a system of linear equations using an efficient conjugate gradient algorithm namely conjugate gradient precondition

normal equation (CGPCNE) in its least squares. Improved iterative pruning (IIP) [58] algorithm which adopts dividing-block strategy uses generalized inverse matrix (GIM) algorithm instead of the CGPCNE in IP for solving the set of equations. [33] and [32] detail out a complete survey on pruning algorithms.

5) L_1 *regularization*: As described in previous paragraphs, regularization is a mathematical solution for preventing memorization or ill-posed problems by adding up a new term to the objective function. In general, regularization is formulated as

$$\mathrm{E}(\boldsymbol{w}) + \lambda \cdot \mathrm{R}(\boldsymbol{w}) \tag{4.2}$$

where E is mostly least square cost function, λ is the regularization parameter, $\mathrm{R}(\boldsymbol{w})$ is the complexity penalty, and $\boldsymbol{w}$ is the weight vector. In L_1 regularization, $\mathrm{R}(\boldsymbol{w})$ is formulated as

$$\mathrm{R}(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i| \tag{4.3}$$

where $\|\boldsymbol{w}\|_1$ is the $l_1 - norm$ of vector $\boldsymbol{w}$ or sum of the absolute value of coefficients of vector $\boldsymbol{w}$.

L_1 regularization is known as LASSO which was originally introduced in [59] for the least squares regression models. It was shown that minimizing the cost function with $l_1 - norm$ penalty encourages spasity, the coefficients of vector $\boldsymbol{w}$ will have many zero values. L_1 regularization is widely used in creating sparse feature in machine learning and also effectively recovering sparse signals in signal processing.

C. Need for further work

Current family of algorithms has mostly used L_1 regularization in order to infuse sparseness into the network's connectivity. Nonetheless, that makes it hard to program and therefore difficult to make intrinsic changes in the code as per required. Moreover, the hyper-parameters such as network's hidden units and learning factors are heuristic and generally require cross validation to determine a good value. Here in this work, we propose a new approach for pruning the output layer connectivity which benefits from having a closed form expression for

optimal learning factors, being completely free from any hand tuned parameters, having an optimized number of hidden units via using pruning, being an easily programmable algorithm for not utilizing the L_1 regularization well as employing the powerful OLS which makes the current investigation totally gradient free. The proposed algorithm has lower time complexity for small and mid-sized datasets but it is computationally efficient for any various kinds of datasets. In this work, no other packages have been used which makes it an absolute genuine study.

Chapter 5

PRUNING-BASED SPARSENESS

In this chapter, the review of the OLS-based pruning is discussed. In section A.3 on page 5-34, a new pruning method is proposed which is the goal of this thesis.

A. Review of OLS-based pruning

The goal of this method is to find an order function which leads to the minimum training error and minimum number of basis functions for training the network. The order function defines the structure of the basis functions in the network and the number of basis functions defines the minimum value necessary to build up the neural network. This method has the following two steps in achieving these goals.

First, use sequential forward selection to order basis functions so that each additional basis function causes the largest possible decrease on the training error. Second, pick the number of basis functions where the monotonically non-increasing validation error (E_v) versus basis functions gets its minimum value or in mathematical term $\mathrm{N}_u{}' = \underset{m}{argmin}\, \mathrm{E}_v(m)$ where $\mathrm{N}_u{}'$ is the new number of basis functions which is less than the original value of the number of basis functions denoted as N_u.

To simplify the algorithms we take three different cases for expanding the algorithm step by step and explain each one in details.

1) Pruning basis functions for single output case

Considering the case when only one output is available, the output weights will be a row vector and the output y will be a scalar

$$y = \boldsymbol{w}_o \cdot \boldsymbol{X} \tag{5.1}$$

where $\boldsymbol{w}_o \in \mathbb{R}^{\mathrm{N}_u}$ is a row vector and $\boldsymbol{X} \in \mathbb{R}^{\mathrm{N}_u}$ is the basis functions vector.

The training error which has to be minimized is defined as

$$\mathrm{E} = \frac{1}{\mathrm{N}_v} \sum_{p=1}^{\mathrm{N}_v} (t - y)^2 \tag{5.2}$$

the output y can be computed in the orthonormal system as

$$y = \boldsymbol{w}_o{}' \cdot \boldsymbol{X}_o{}' \tag{5.3}$$

where $\boldsymbol{w}_o{}'$ is the output weight vector in the orthonormal system

$$\boldsymbol{w}_o{}' = \boldsymbol{c}^{\mathrm{T}} \cdot \mathbf{A}^{\mathrm{T}} \tag{5.4}$$

and $\boldsymbol{X}_o{}'$ is the ordered orthonormal basis functions which is defined as

$$\boldsymbol{X}_o{}' = \mathbf{A} \cdot \boldsymbol{X}_o \tag{5.5}$$

$\boldsymbol{c} \in \mathbb{R}^{\mathrm{N}_u}$ is the ordered cross-correlation vector, $\mathbf{A} \in \mathbb{R}^{\mathrm{N}_u \times \mathrm{N}_u}$ is the orthonormal transfer matrix defined in the orthonormal system, and $\boldsymbol{X}_o$ is the ordered normal basis functions which for the n^{th} basis function is found as

$$X_o(n) = X(o(n)) \tag{5.6}$$

in which $o(n) \in \mathbb{R}^{\mathrm{N}_u}$ is the order function that has to be found.

The normal output weight vector can be found by mapping back $\boldsymbol{w}_o{}'$ to the normal system using the $\mathbf{A}$ matrix as

$$\boldsymbol{w}_o = \boldsymbol{w}_o{}' \cdot \mathbf{A} \tag{5.7}$$

Denoting the training error for the m^{th} basis function as E_m, the goal is to find the order function so that we would have

$$\mathrm{E}_m(o(m)) \equiv \min_{\boldsymbol{w}_o} \mathrm{EX}[(t - y_m)^2 \mid o(1), o(2), \cdots o(m-1)] \tag{5.8}$$

where

$$y_m = \sum_{n=1}^{m} w_o(o(n))X(o(n))$$
$$= \sum_{n=1}^{m}\left(\sum_{k=1}^{m} w_o{}'(n)a(n,k)\right)X\big(o(n)\big) = \sum_{n=1}^{m} w_o{}'(n)\, X_o{}'(n) \tag{5.9}$$

and EX is the expected value notation. Elements of **A** are found by computing the equations below

$$c_l = \sum_{k=1}^{m} a(m,k)\cdot r\big(o(m),o(k)\big) \quad \text{for } 1 \le l \le m-1 \tag{5.10}$$

$$b_k = \begin{cases} 1 & \text{for } k = m \\ -\sum_{l=k}^{m-1} c_l \cdot a(l,k) & \text{for } 1 \le k \le m-1 \end{cases} \tag{5.11}$$

$$a(m,k) = \frac{b_k}{\sqrt{r\big(o(m),o(m)\big) - \sum_{l=1}^{m-1} c_l^2}} \qquad \text{for } 1 \le k \le m \tag{5.12}$$

The first basis function $X_o(1) = X(o(1))$ is chosen in such a way that it satisfies the equation below

$$o(1) = \underset{k \in S_u}{argmin}\, \mathrm{E}_1(k) \tag{5.13}$$

where S_u is the set of unchosen basis functions. Updating $S_u = \{1, 2, \cdots, N_u\} - o(1)$, we can find the rest of the basis functions with the same procedure. For the n^{th} basis function we would have

$$o(n) = \underset{k \in S_u}{argmin}\, \mathrm{E}_n(k) \tag{5.14}$$

where $S_u = \{1, 2, \cdots, N_u\} - \{o(1), o(2), \cdots, o(n-1)\}$.

After finding the order function using all the basis functions, we want to find the number of basis functions which can lead to a decrease in the validation error. In order to achieve this goal, we need to generate the validation error versus basis functions in order of their importance. In other words, we should first find $\mathrm{E}_v\big(o(n)\big)$ for $1 \le n \le N_u$ by calculating y_m for the validation data and then calculate the corresponding validation error. Rewriting equation (5.9) for the validation data for the m^{th} basis function, the validation error can be obtained by

$$y_m = y_{m-1} + w_o{}'(m) \cdot X_{o-v}{}'(m) \tag{5.15}$$

$$\mathrm{E}_v(m) \leftarrow \mathrm{E}_v(m) + (t - y_m)^2 \tag{5.16}$$

After calculating the validation error for all the values of $m \in o(n)$ for $1 \le n \le N_u$, we are able to pick the new number of useful basis functions by finding

$$\mathrm{N}_u{}' = \underset{m}{argmin}\, \mathrm{E}_v(m) \tag{5.17}$$

where $\mathrm{N}_u{}' \le \mathrm{N}_u$ and $\mathrm{N}_u{}' \in \mathbb{R}$.

2) Pruning basis *functions* for *multi-output case*

When there is more than one output present in the dataset, each of the outputs will have an output weight vector and we would have an M-dimensional output vector where the number of outputs is M. Therefore, we would have $\mathbf{W}_o{}'$ and $\mathbf{W}_o \in \mathbb{R}^{\mathrm{M} \times \mathrm{N}_u}$ as the orthonormal and normal output weight matrices, and the output vector as $\boldsymbol{y} \in \mathbb{R}^{\mathrm{M}}$. The equations (5.8) and (5.9) are modified as

$$\mathrm{E}_m(o(m)) \equiv \underset{\mathbf{W}_o}{min}\, \mathrm{EX}[(\boldsymbol{t} - \boldsymbol{y}_m)^2 \mid o(1), o(2), \cdots o(m-1)] \tag{5.18}$$

and

$$\begin{aligned} \boldsymbol{y}_m = \sum_{i=1}^{\mathrm{M}} \left(\sum_{n=1}^{m} w_o(i, o(n)) X(o(n)) \right) &= \sum_{i=1}^{\mathrm{M}} \left(\sum_{n=1}^{m} \left(\sum_{k=1}^{m} w_o{}'(i,n) a(n,k) \right) X\big(o(n)\big) \right) \\ &= \sum_{i=1}^{\mathrm{M}} \left(\sum_{n=1}^{m} w_o{}'(i,n)\, X_o{}'(n) \right) \end{aligned} \tag{5.19}$$

the elements of **A** matrix are found by computing the equations (5.10), (5.11), and (5.12). As in part previous section, the first basis function $o(1)$ can be found using equation (5.13) and $o(n)$ for $2 \le n \le \mathrm{N}_u$ can be found using equation (5.14).

After finding the order function using all the basis functions, we want to find the number of useful basis functions using the validation error. We generate the validation error versus basis

functions in order of their importance. First we find $\mathrm{E}_v\big(o(n)\big)$ for $1 \le n \le \mathrm{N}_u$ by calculating $\boldsymbol{y}_m$ in equation (5.19) for the validation data, and then we calculate the validation error using (5.21). Rewriting equation (5.19) for the validation data for the m^{th} basis function, the validation error can be obtained for the m^{th} basis function

$$\boldsymbol{y}_m = \boldsymbol{y}_{m-1} + \sum_{i=1}^{\mathrm{M}} w_o{}'(i,m) X_{o-v}{}'(m) \tag{5.20}$$

$$\mathrm{E}_v(m) \leftarrow \mathrm{E}_v(m) + (\boldsymbol{t} - \boldsymbol{y}_m)^2 \tag{5.21}$$

where $m \in o(n)$ for $1 \le n \le \mathrm{N}_u$. Calculating the validation error, we are able to pick the new number of useful basis functions by finding

$$\mathrm{N}_u{}' = \underset{m}{argmin}\, \mathrm{E}_v(m) \tag{5.22}$$

where $\mathrm{N}_u{}' \le \mathrm{N}_u$ and $\mathrm{N}_u{}' \in \mathbb{R}$.

3) Pruning basis functions individually for multi-output case

In this section, a new pruning method is proposed that is the combination of the two previous pruning methods. In section A.2 and A.1 of this chapter, the order functions were row vectors. In section A.2, only one order function was used for pruning the basis functions in the multi-output case. In this section, we also prune the basis functions for a multi-output case but by introducing an individual order function for each output.

The output vector $\boldsymbol{y}$ in this method is computed in the orthonormal system using equation (5.23), where for each output an individual set of ordered orthonormal basis functions has to be found

$$\boldsymbol{y} = \sum_{i=1}^{\mathrm{M}} \mathbf{W}_o{}'(i) \cdot \boldsymbol{X}_{i,o}{}' \tag{5.23}$$

where $\mathbf{W}_o{}'(i)$ is the i^{th} row of $\mathbf{W}_o{}' \in \mathbb{R}^{\mathrm{M} \times \mathrm{N}_u}$ and $\boldsymbol{X}_{i,o}{}'$ is the ordered othornormal basis functions for the i^{th} output. The orthonormal output weight matrix is calculated from

$$\mathbf{W}_o{}' = \sum_{i=1}^{\mathrm{M}} \boldsymbol{c}_i{}^T \cdot \mathbf{A}_i{}^T \tag{5.24}$$

where $\boldsymbol{c}_i$ is the i^{th} column of the ordered cross-correlation matrix $\mathbf{C} \in \mathbb{R}^{\mathrm{N}_u \times \mathrm{M}}$, and $\mathbf{A}_i$ is the orthonormal transfer matrix for the i^{th} output. The ordered orthonormal basis functions for the i^{th} output are calculated as

$$\boldsymbol{X}_{i,o}{}' = \mathbf{A}_i \cdot \boldsymbol{X}_o \tag{5.25}$$

in which $\boldsymbol{X}_o$ is the ordered normal basis functions which for the n^{th} basis function and the i^{th} output is found as

$$X_o(n) = X(o_i(n)) \tag{5.26}$$

where $o_i(n) \in \mathbb{R}^{\mathrm{N}_u}$ is the i^{th} row of the order function, $\mathbf{o} = [o_1, o_2, \cdots, o_{\mathrm{M}}]^{\mathrm{T}}$. The order function is $\mathbf{o} \in \mathbb{R}^{\mathrm{M} \times \mathrm{N}_u}$.The normal output weight matrix can be found by mapping back $\mathbf{W}_o{}'$ to the normal system using the $\mathbf{A}_i$ matrices as

$$\mathbf{W}_o = \sum_{i=1}^{\mathrm{M}} \mathbf{W}_o{}'(i) \cdot \mathbf{A}_i \tag{5.27}$$

Denoting the training error for the m^{th} basis function as E_m, the goal is to find the order function so that we would have

$$\mathrm{E}_m(\mathbf{o}(m)) \equiv \min_{\mathbf{W}_o} \mathrm{EX}[(\boldsymbol{t} - \boldsymbol{y}_m)^2 \mid \mathbf{o}(1), \mathbf{o}(2), \cdots \mathbf{o}(m-1)] \tag{5.28}$$

where

$$\begin{aligned} \boldsymbol{y}_m &= \sum_{i=1}^{\mathrm{M}} \left(\sum_{n=1}^{m} w_o(i, o_i(n)) X(o_i(n)) \right) \\ &= \sum_{i=1}^{\mathrm{M}} \left(\sum_{n=1}^{m} \left(\sum_{k=1}^{m} w_o{}'(i,n) a_i(n,k) \right) X\big(o_i(n)\big) \right) \\ &= \sum_{i=1}^{\mathrm{M}} \left(\sum_{n=1}^{m} w_o{}'(i,n)\, X_{i,o}{}'(n) \right) \end{aligned} \tag{5.29}$$

in which a_is are elements of $\mathbf{A}_\mathrm{i}$ matrix for the i^{th} output that are found by computing the following equations

$$c_l = \sum_{k=1}^{m} a_i(m,k) \cdot r\big(o_i(m), o_i(k)\big) \quad \text{for } 1 \le l \le m-1 \tag{5.30}$$

$$b_k = \begin{cases} 1 & \text{for } k = m \\ -\sum_{l=k}^{m-1} c_l \cdot a_i(l,k) & \text{for } 1 \le k \le m-1 \end{cases} \tag{5.31}$$

$$a_i(m,k) = \frac{b_k}{\sqrt{r\big(o_i(m), o_i(m)\big) - \sum_{l=1}^{m-1} c_l^2}} \qquad \text{for } 1 \le k \le m \tag{5.32}$$

The first basis function $X_o(1) = X(\mathbf{o}(1))$ for all the outputs is chosen in such a way that

$$\mathbf{o}(1) = \underset{\boldsymbol{k} \in \boldsymbol{S}_u}{argmin}\, \mathrm{E}_1(\boldsymbol{k}) \tag{5.33}$$

where

$$\boldsymbol{S}_u = \{S_{u1}, S_{u2}, \cdots, S_{uM}\} = \{S_{ui}\} \text{ for } 1 \le i \le \mathrm{M} \tag{5.34}$$

contains the sets of unchosen basis functions for all the outputs. Updating the unchosen set as $\boldsymbol{S}_u = \{1, 2, \cdots, N_u\} - \mathbf{o}(1)$, we can find the rest of the basis functions with the same procedure. For the n^{th} basis function we would have

$$\mathbf{o}(n) = \underset{\boldsymbol{k} \in \boldsymbol{S}_u}{argmin}\, \mathrm{E}_n(\boldsymbol{k}) \tag{5.35}$$

where $\boldsymbol{S}_u = \{1, 2, \cdots, N_u\} - \{\mathbf{o}(1), \mathbf{o}(2), \cdots, \mathbf{o}(n-1)\}$.

After finding the order of basis functions individually for each output based on their importance, we need to find the number of useful basis functions using the validation error. In order to generate $\mathrm{E}_v\big(\mathbf{o}(n)\big)$ for $1 \le n \le \mathrm{N}_u$, first we need to calculate $\boldsymbol{y}_m$ where $m \in \mathbf{o}(n)$ for $1 \le n \le \mathrm{N}_u$ for the validation data and then calculate the corresponding validation error as in equation (5.37). Rewriting equation (5.29) for the m^{th} basis function using the validation data, the validation error can be obtained from

$$\boldsymbol{y}_m = \boldsymbol{y}_{m-1} + \sum_{i=1}^{\mathrm{M}} w_o{}'(i,m) X_{i,o-v}{}'(m) \tag{5.36}$$

$$\mathrm{E}_v(m) \leftarrow \mathrm{E}_v(m) + (\boldsymbol{t} - \boldsymbol{y}_m)^2 \tag{5.37}$$

Calculating the validation error for each output individually, we are able to pick the new number of useful basis functions by finding

$$\mathrm{N_u}{}'(i) = \underset{m_i}{\operatorname{argmin}}\, \mathrm{E}_v(i, m_i) \qquad \text{for } 1 \le i \le \mathrm{M} \text{ and } m_i \in o_i \tag{5.38}$$

where $\mathrm{N_u}{}'(i) \le \mathrm{N_u}$ and $\mathrm{N_u}{}' \in \mathbb{R}^{\mathrm{M}}$.

B. Experimental results

The proposed method has been verified using several datasets. The performance of the proposed pruning method is compared to the old pruning method, MOLF-Adapt, explained in A.2 which is published in [21].

The new pruning method is tested on 4 datasets, 2 regression data files and 2 classification data files. During training a median filter has been used to smooth the validation curve, and the training and testing of the network have been conducted by averaging over 10-fold datasets created from the original data files. The first two datasets are regression models and the next two datasets are classification models. The last experiment has been conducted on a combination of two different datasets to see the performance of the algorithm in disjoining the different data files.

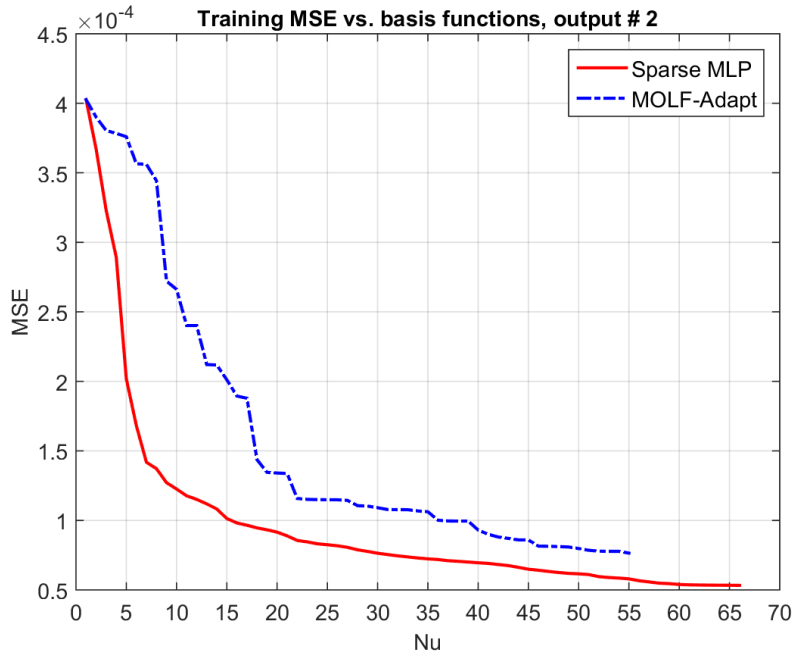
For each dataset, the training MSE of each output for the proposed pruning method is compared to [21] and a plot of their performance is depicted. In the end, a table of the average testing and training MSE for all the data files and a table of sparsity measurements is provided in section C.

The proposed pruning method that introduces sparsity to the structure of MLP is denoted as Sparse MLP in the graphs.

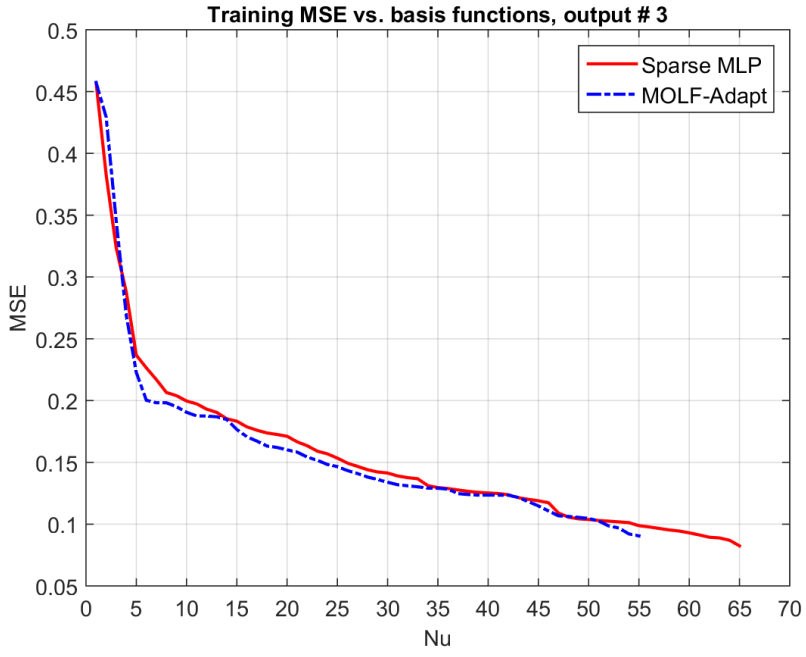
1) Twod dataset

Twod dataset has highly correlated inputs. It is consisted of 7 inputs and 8 outputs. We trained a neural network using the proposed algorithm with initial hidden units as 300. In Figure 5-1 thru Figure 5-4, the average training mean square error (MSE) versus basis functions is plotted for the proposed method and MOLF-Adapt.

Figure 5-1: Comparison of training MSE between the proposed method and MOLF-Adapt using twod dataset for output # 1

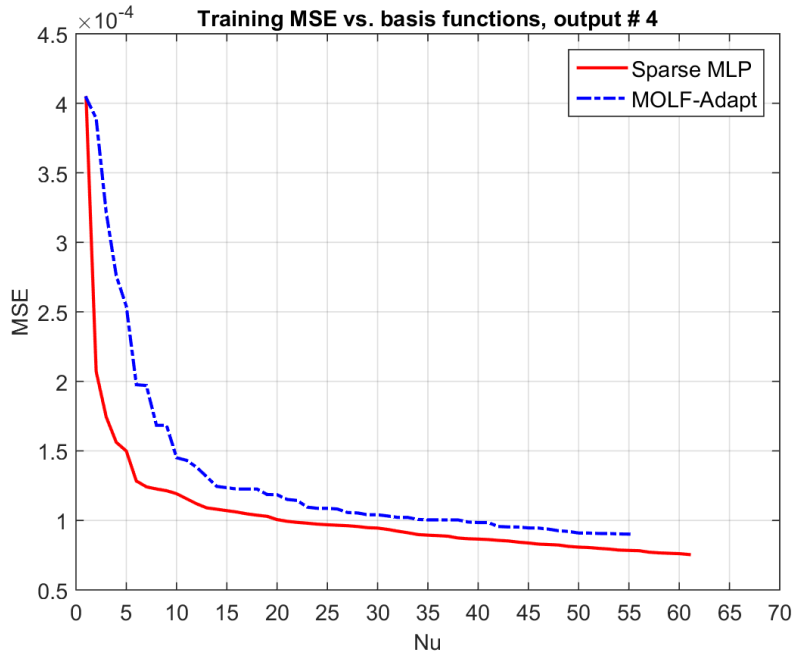


(a)

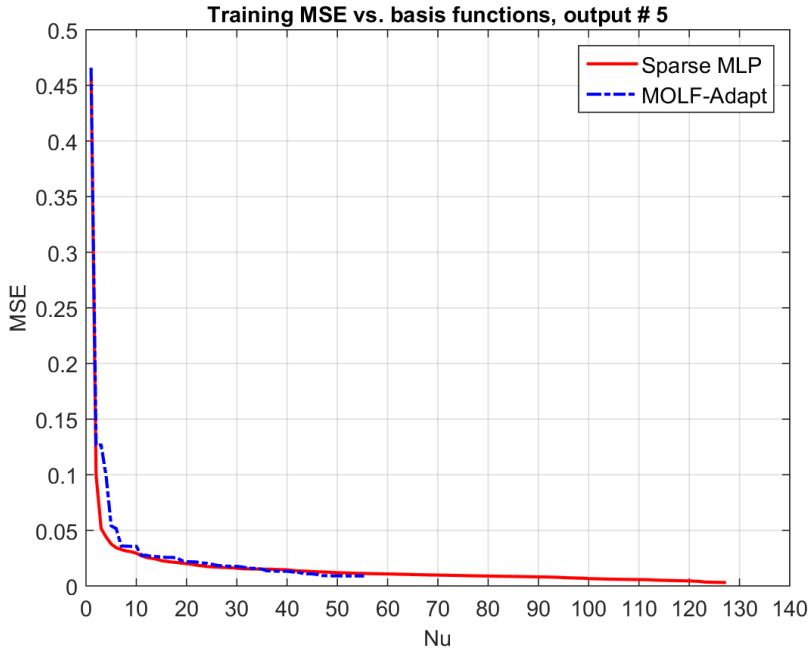


(b)

Figure 5-2: Comparison of training MSE between the proposed method and MOLF-Adapt using twod dataset for (a) output # 2 (b) output # 3

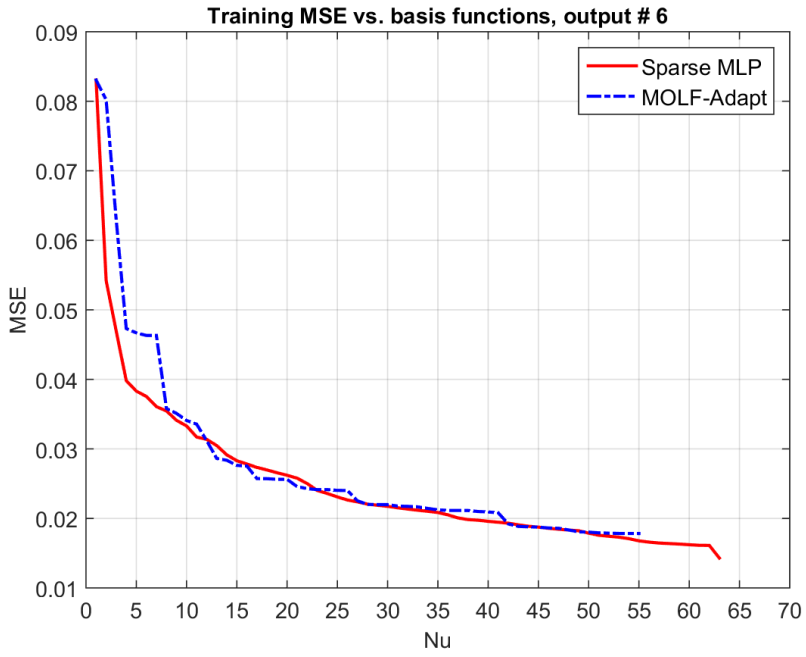


(a)

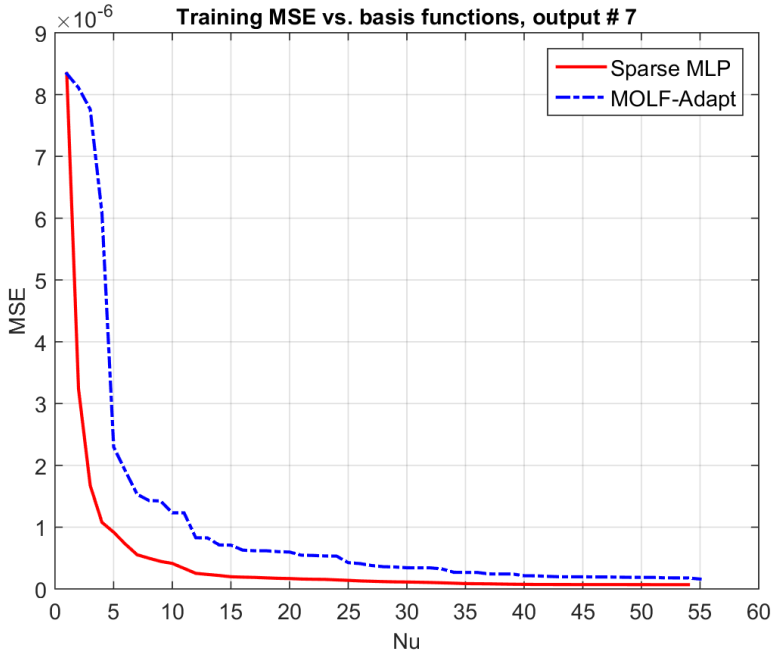


(b)

Figure 5-3: Comparison of training MSE between the proposed method and MOLF-Adapt using twod dataset for (a) output # 4 (b) output # 5



(a)



(b)

Figure 5-4: Comparison of training MSE between the proposed method and MOLF-Adapt using twod dataset for (a) output # 6 (b) output # 7

In these figures, the blue curves are the training MSE from [21] and the red curves are derived from the proposed algorithm. As it can be seen the red curves either have fewer values or approximately the same values as the blue curves. Therefore, the proposed algorithm trains a better network. Also, the number of ordered hidden units picked for the network is the same for all the outputs in the old algorithms whereas there are different values for each output in the new algorithm.

2) Oh7 dataset

In oh7 dataset, not only the input values but also one of the output values is highly correlated with the input values unlike twod dataset. We used the proposed algorithms for training a neural network with initial hidden unit as 300. In Figure 5-5 thru Figure 5-6, the average training mean square error (MSE) versus basis functions is plotted for the proposed method and the method in MOL-Adapt.

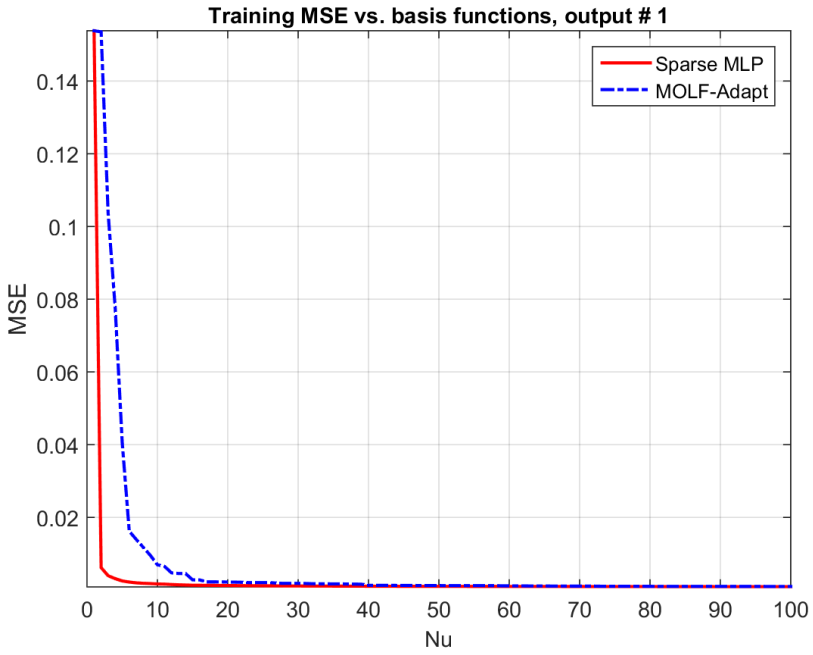
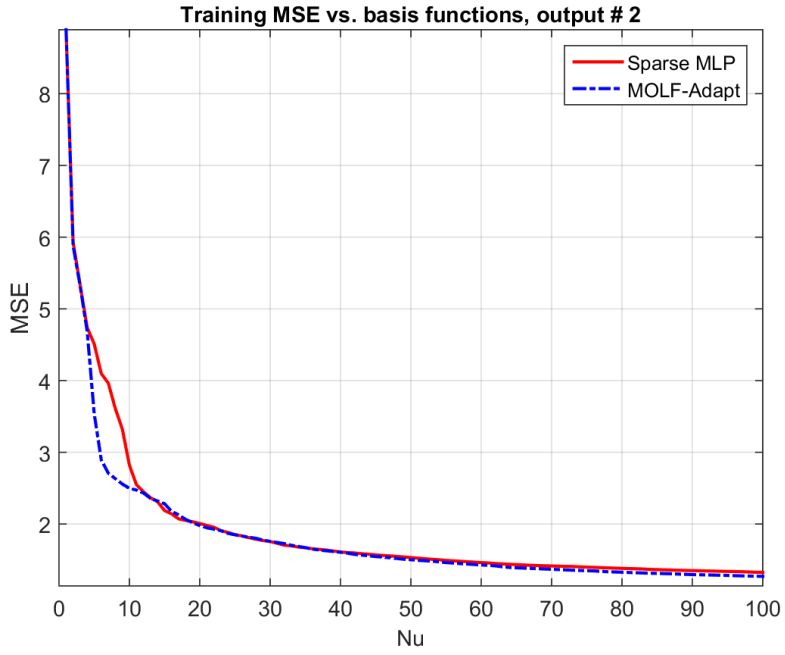
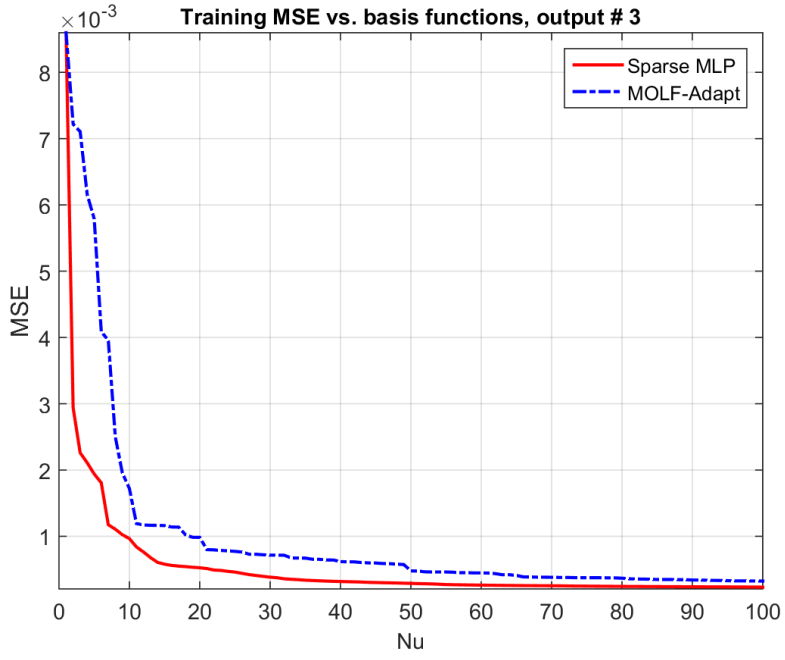


Figure 5-5: Comparison of training MSE between the proposed method and MOLF-Adapt using oh7 dataset for output # 1



(a)



(b)

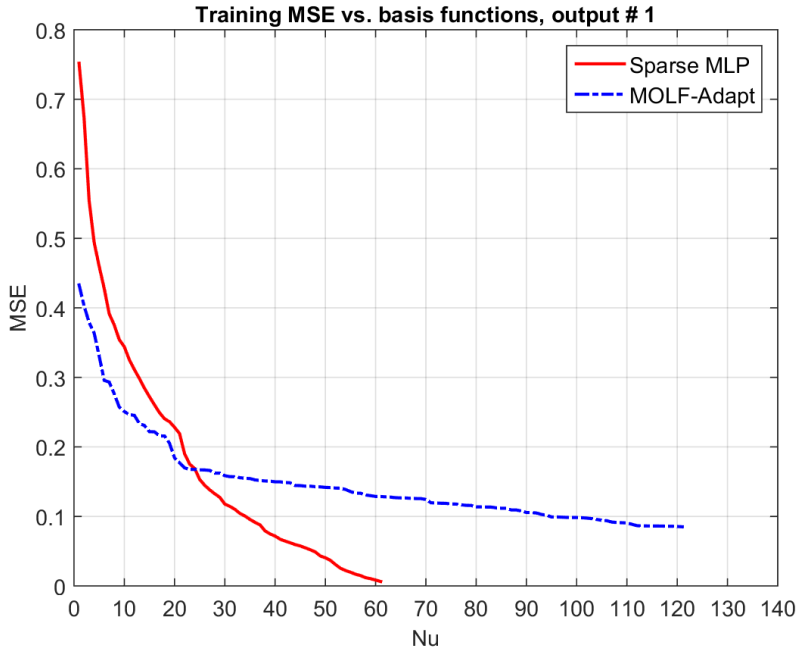
Figure 5-6: Comparison of training MSE between the proposed method and MOLF-Adapt using oh7 dataset for (a) output # 2 (b) output # 3

As mentioned, this data file is highly nonlinear. Therefore, the second output is mostly dominating the network's training. It is noticeable in the figures above, the two algorithms' performance is almost similar for the second output. Yet, the other two outputs' performance is highly improved using the proposed algorithm.

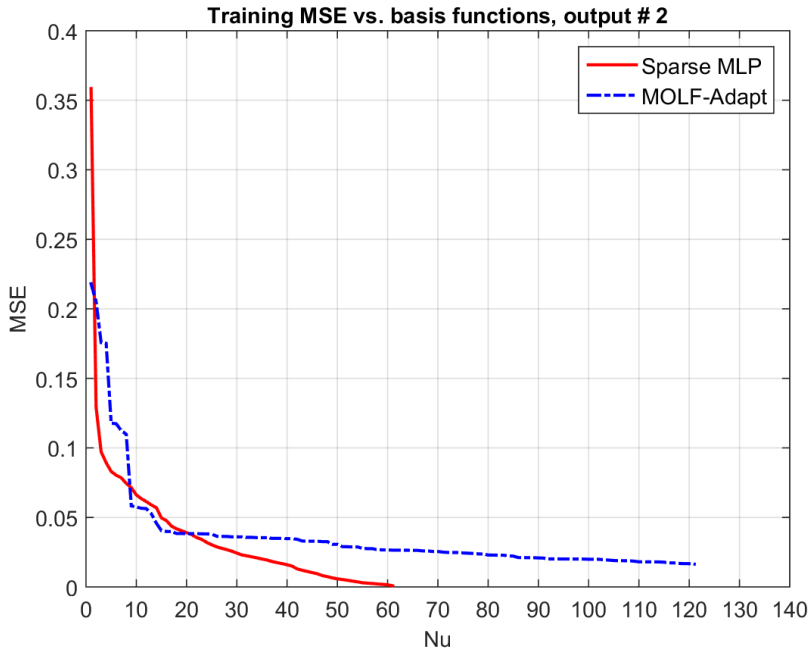
3) Gongtrn dataset

This dataset is a classification problem. Sparse network training for classification problems is different from training sparse network for regression problems. The difference arises from the fact that the error calculation is based on probability of error that is the number of misclassification to the correct classification. Therefore, although the training error increases or decreases, it does not necessarily mean the probability of error is decreasing. In the below figures, training MSEs for each class is depicted for the proposed method (Sparse MLP) and MOLF-Adapt.

For this experiment, we trained a neural network using the proposed algorithms with initial hidden unit as 300. In Figure 5-7 thru Figure 5-11, the average training MSE versus basis functions is drawn for the proposed method and MOLF-Adapt.

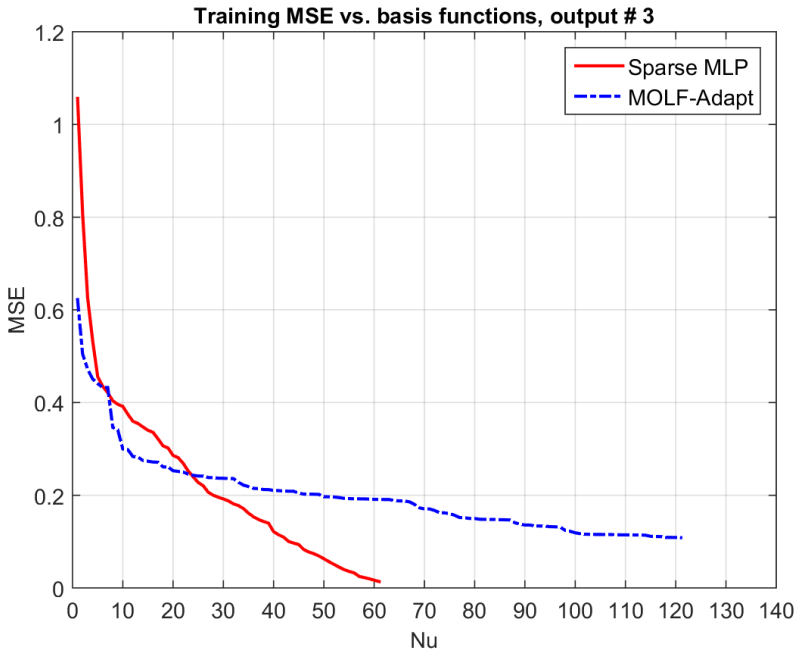


(a)

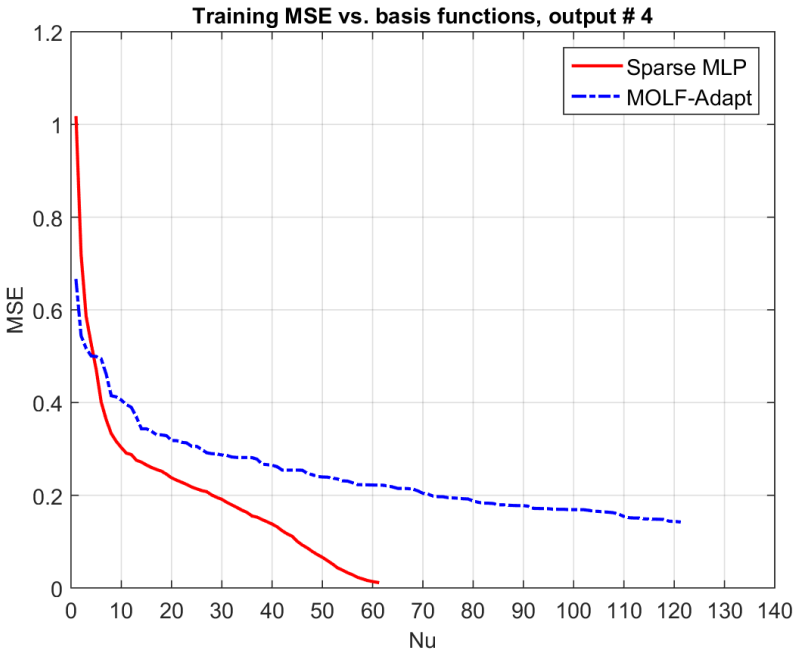


(b)

Figure 5-7: Comparison of training MSE between the proposed method and MOLF-Adapt using gongtrn dataset for (a) class # 1 (b) class # 2

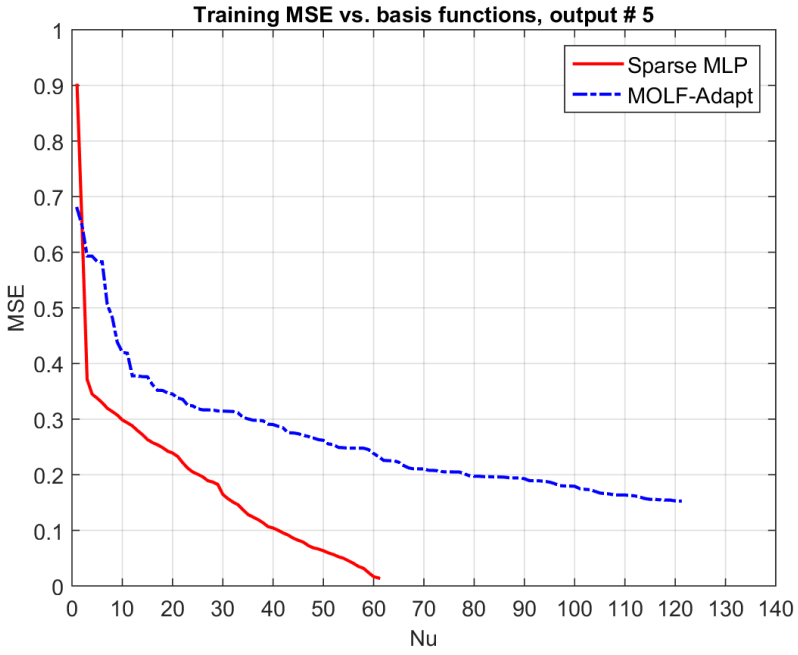


(a)

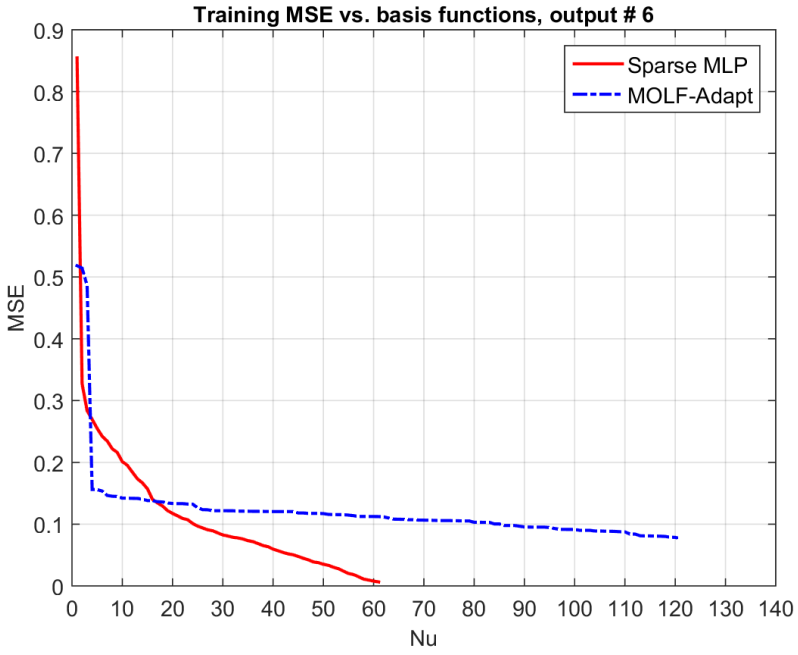


(b)

Figure 5-8: Comparison of training MSE between the proposed method and MOLF-Adapt using gongtrn dataset for (a) class # 3 (b) class # 4

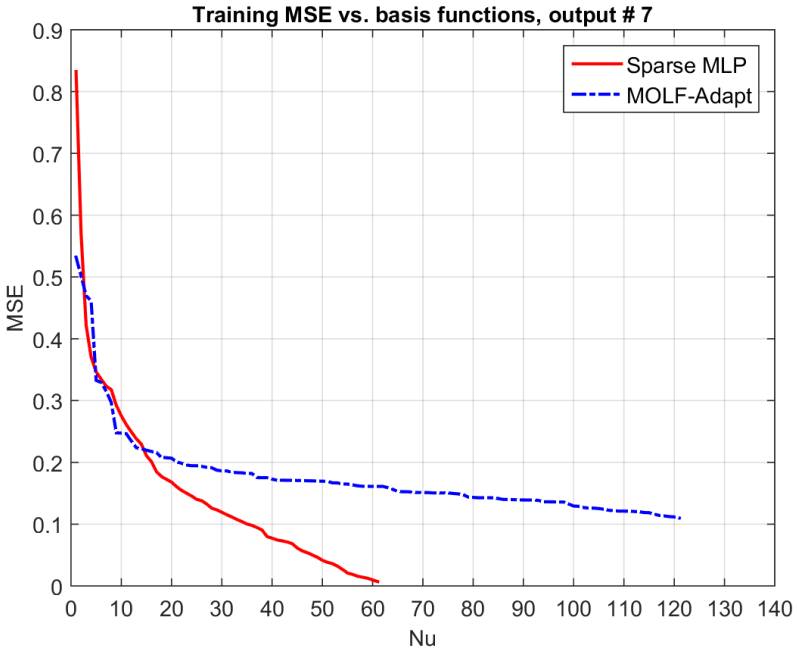


(a)

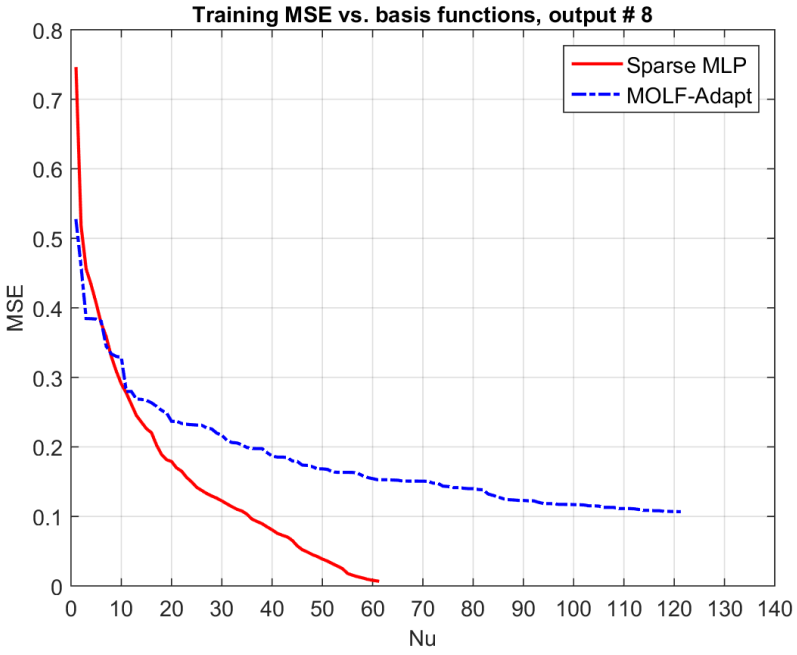


(b)

Figure 5-9: Comparison of training MSE between the proposed method and MOLF-Adapt using gongtrn dataset for (a) class # 5 (b) class # 6

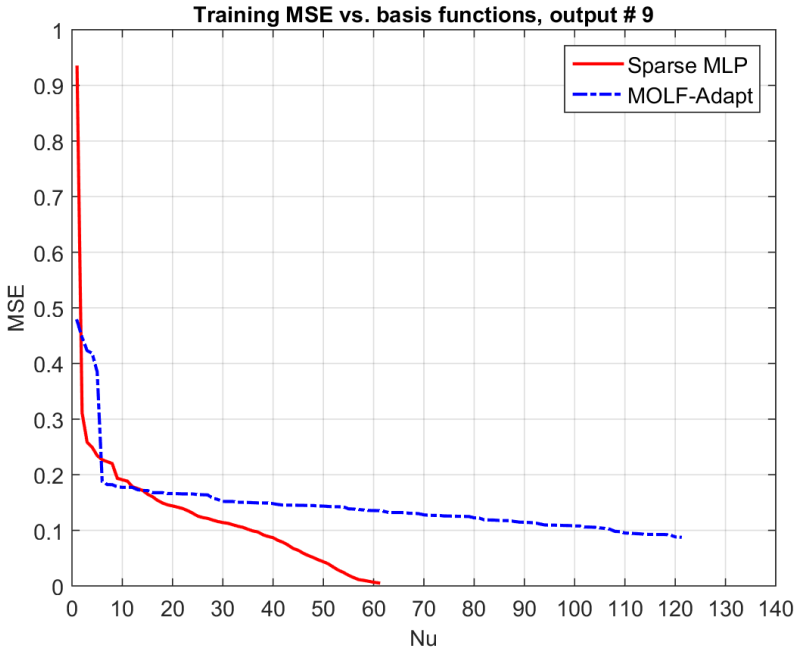


(a)

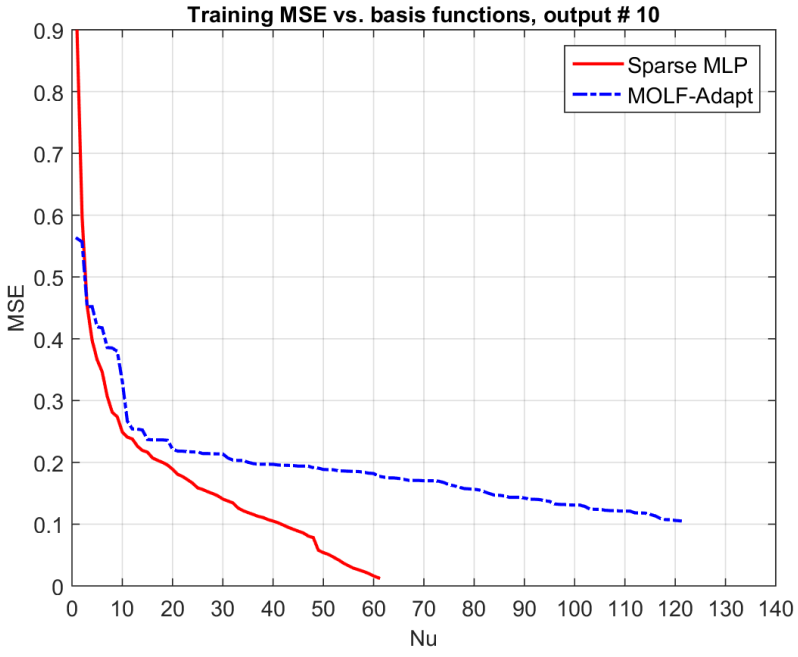


(b)

Figure 5-10: Comparison of training MSE between the proposed method and MOLF-Adapt using gongtrn dataset for (a) class # 7 (b) class # 8



(a)



(b)

Figure 5-11: Comparison of training MSE between the proposed method and MOLF-Adapt using gongtrn dataset for (a) class # 9 (b) class # 10

In the above figures, it can be seen that the red curves outperforms the blue curves, therefore, the new algorithm is giving a better trained network. The number of hidden units that is used for training the network using the proposed method is much less than MOLF-Adapt. The old algorithm fails to show a good performance compared to the proposed algorithm.

4) MNIST dataset

In this section, the comparison of the training MSE between the proposed method and MOLF-Adapt using the MNIST dataset is provided. The MNIST dataset is a classification problem consisting of 10 classes with 784 inputs. It is the collection of 60000 handwritten digits training patterns and 10000 testing patterns. In this experiment the initial number of hidden units are set to 800. In Figure 5-12 thru Figure 5-17, the average training mean square error (MSE) versus basis functions is depicted for the proposed method and MOLF-Adapt.

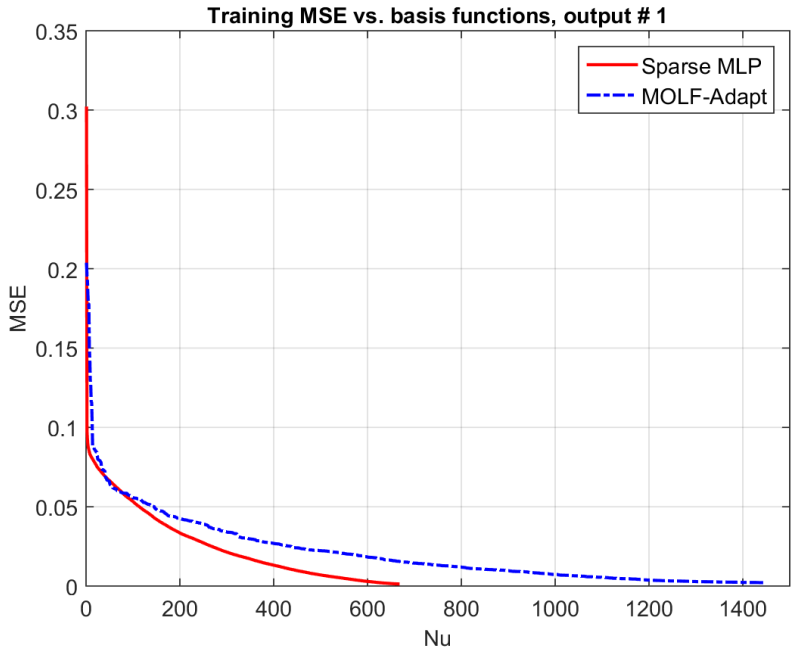
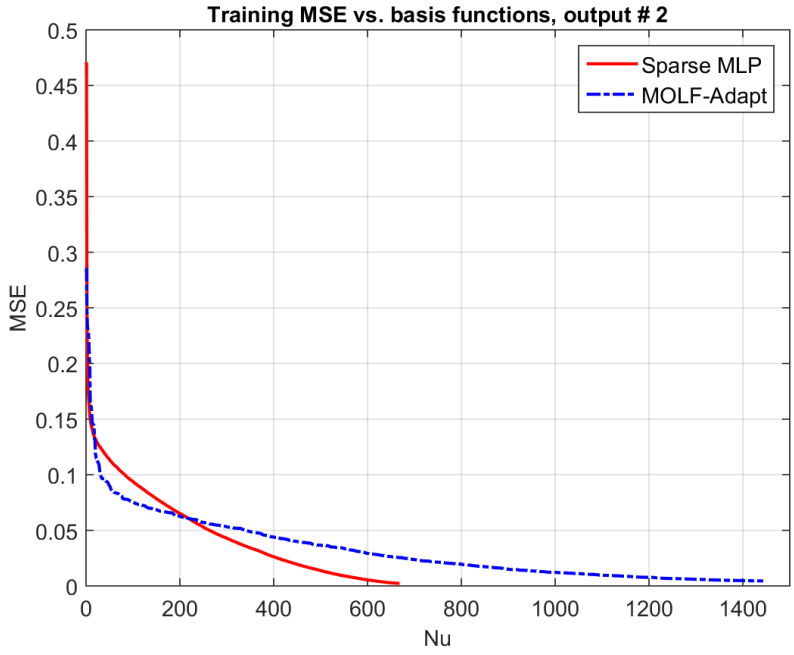
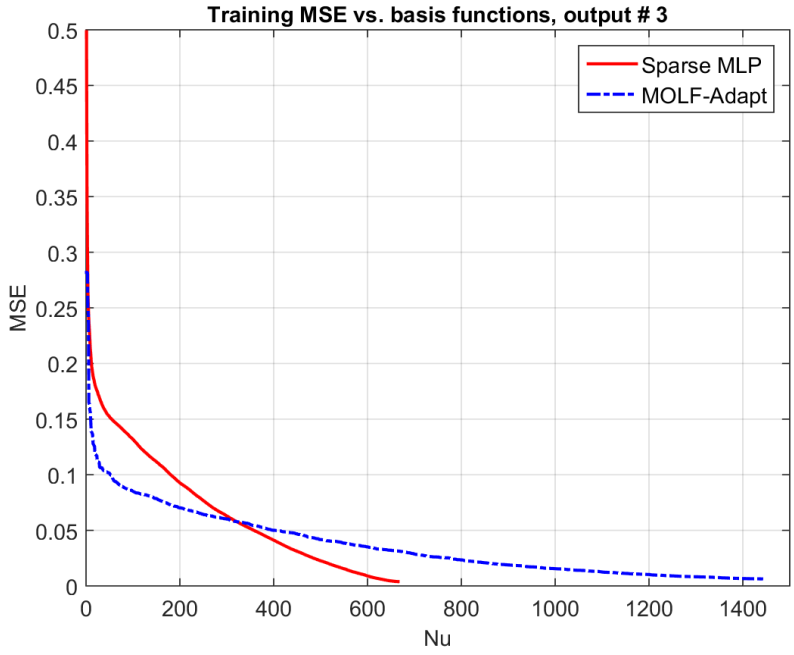


Figure 5-12: Comparison of training MSE between the proposed method and MOLF-Adapt using MNIST dataset for class # 1

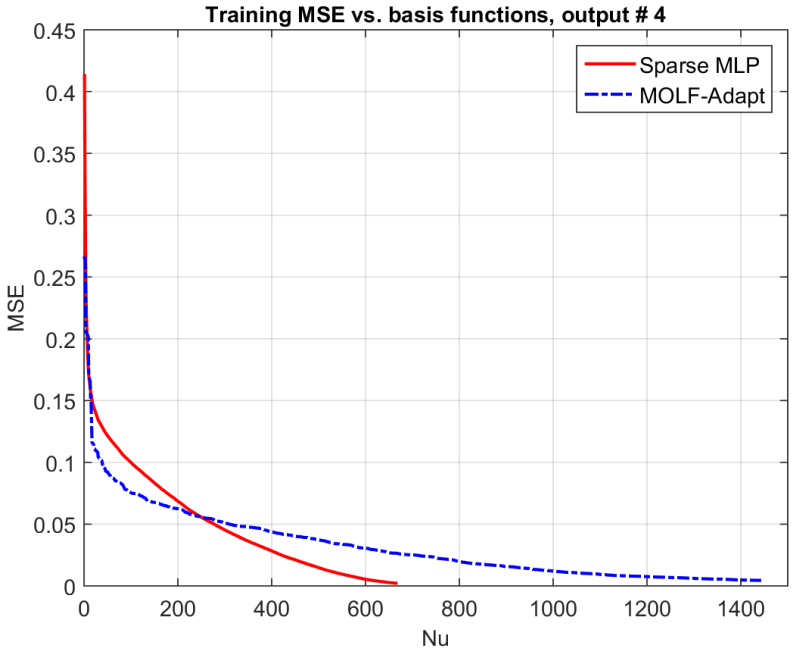


(a)

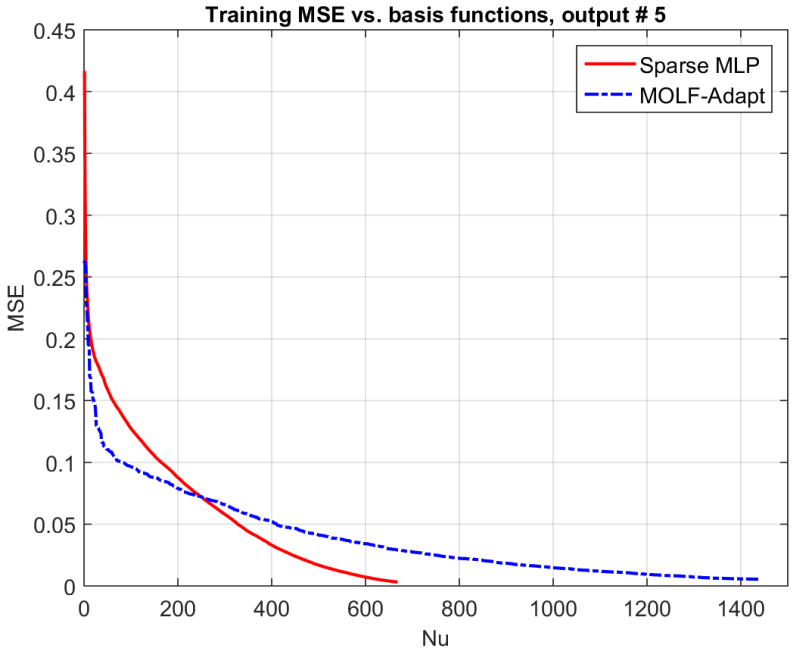


(b)

Figure 5-13: Comparison of training MSE between the proposed method and MOLF-Adapt using MNIST dataset for (a) class # 2 (b) class # 3

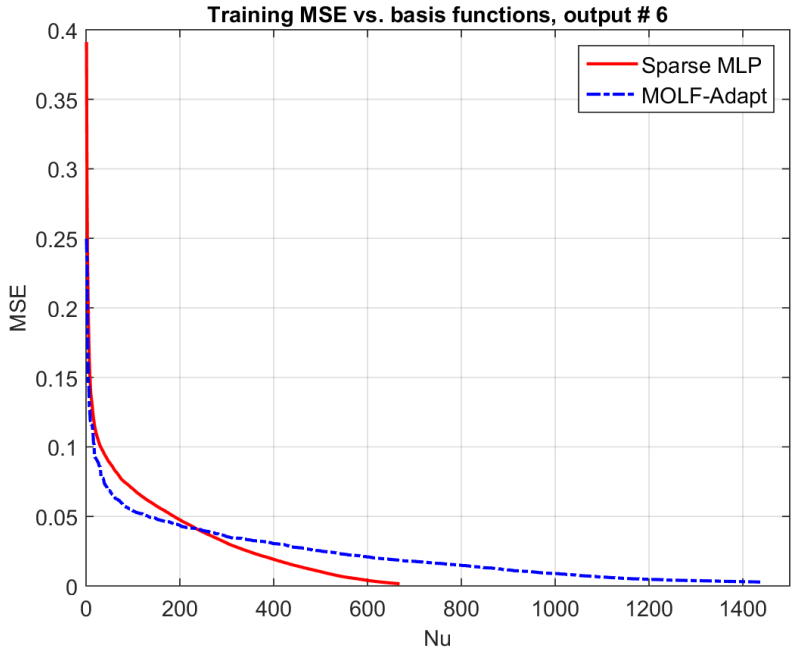


(a)

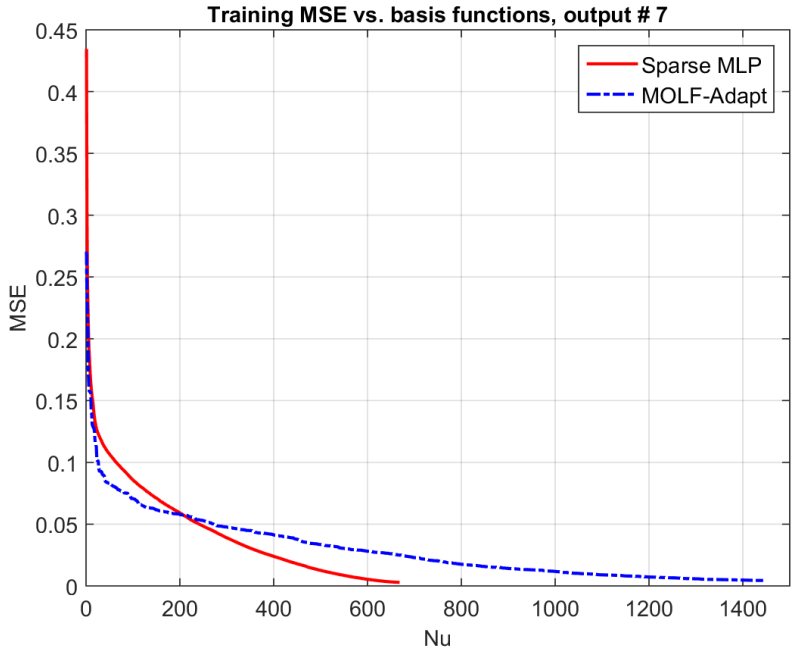


(b)

Figure 5-14: Comparison of training MSE between the proposed method and MOLF-Adapt using MNIST dataset for (a) class # 4 (b) class # 5

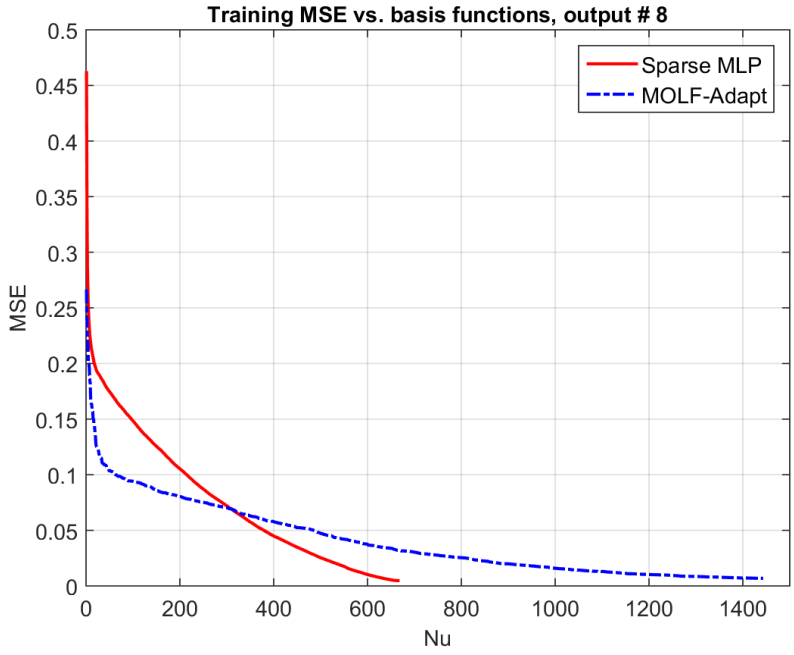


(a)

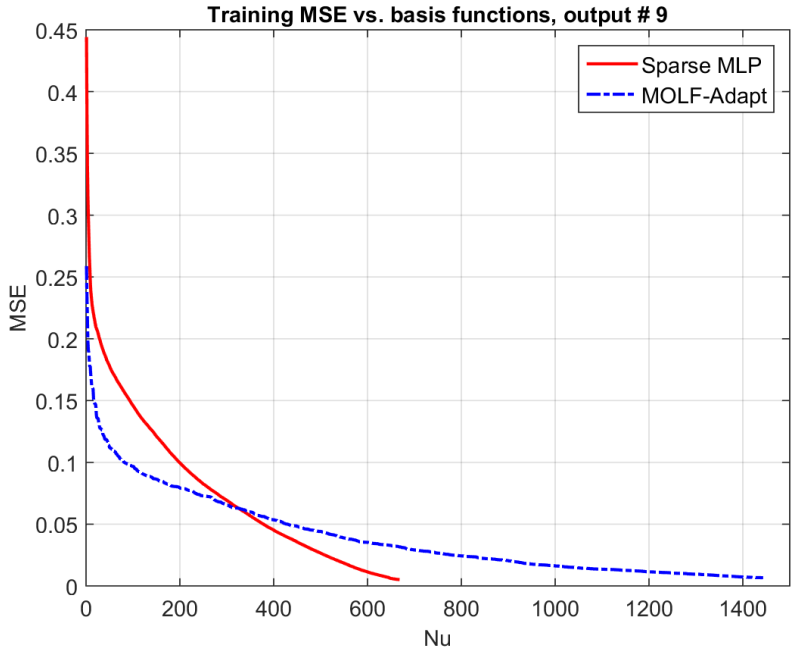


(b)

Figure 5-15: Comparison of training MSE between the proposed method and MOLF-Adapt using MNIST dataset for (a) class # 6 (b) class # 7



(a)



(b)

Figure 5-16: Comparison of training MSE between the proposed method and MOLF-Adapt using MNIST dataset for (a) class # 8 (b) class # 9

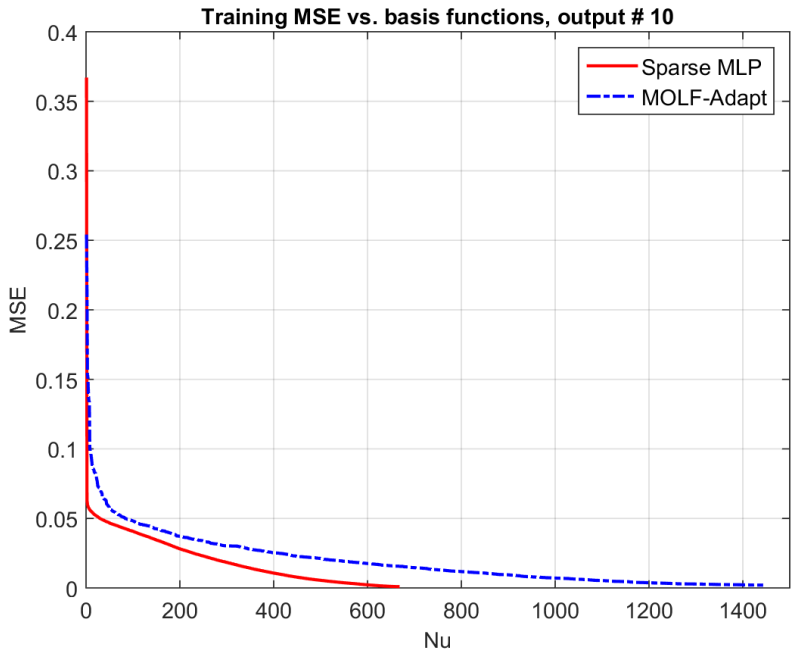


Figure 5-17: Comparison of training MSE between the proposed method and MOLF-Adapt using MNIST dataset for class # 10

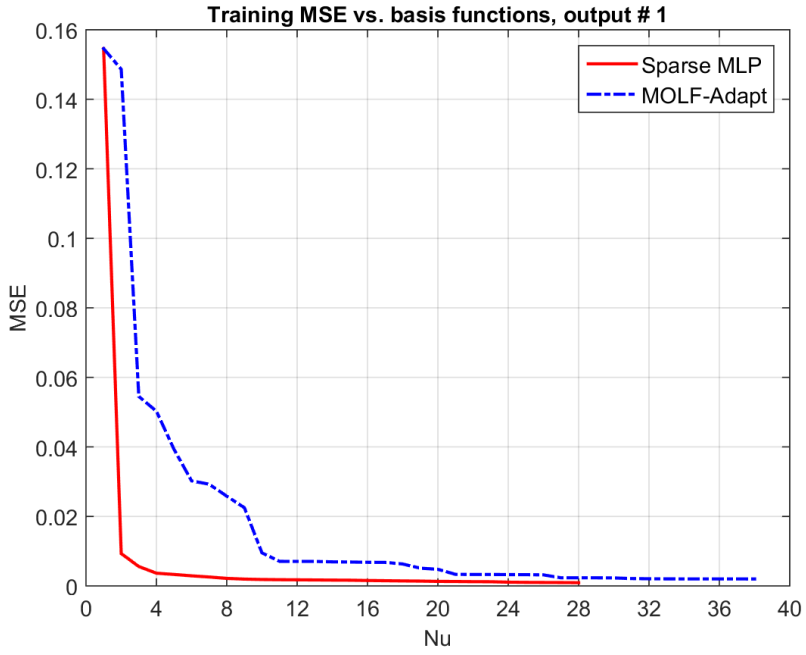
The above figures demonstrate that the proposed algorithm has less training mean square errors. Additionally, during training more number of hidden units are pruned in the Sparse MLP since the continuous red curve end is near 700 but the dotted blue curve end is near 1500.

C. Combined dataset

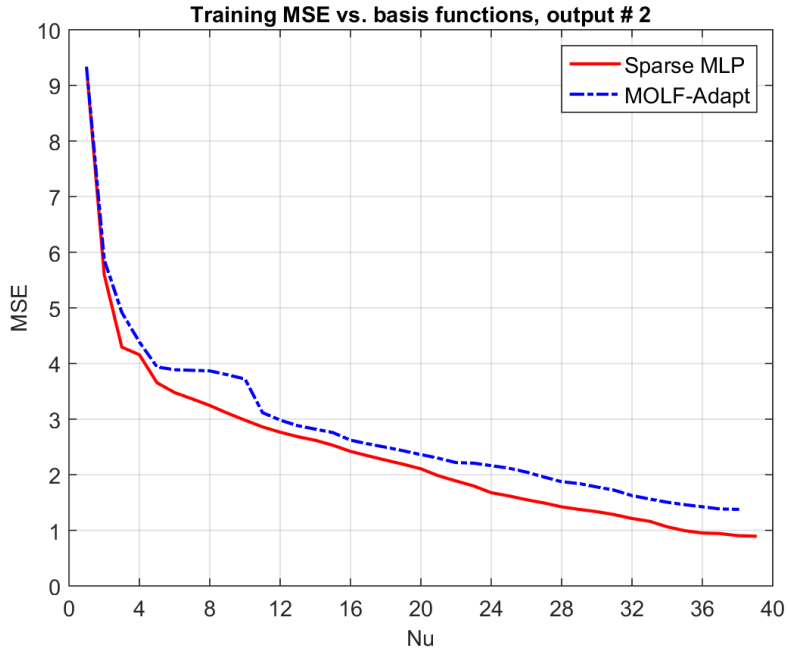
One of the intriguing results of the proposed algorithm is obtained from a dataset that is the combination of two disjoint datasets. By combining two distinct datasets that are statistically independent and creating a combined dataset, we trained an MLP with one hidden layer using the proposed method. This algorithm is capable of sparsing the network so that there will be no connectivity between the inputs and outputs of the two disjoint datasets. As we will show later, the proposed algorithm uses a different set of hidden units for training the overall network for each of the two datasets with very few shared hidden units. Since the proposed method introduces sparseness into the structure of the MLP or the connectivity of the network, this algorithm is able to distinguish the statistical relations between inputs and outputs of different datasets and it is capable of separating the network into different disconnected networks.

For the experiment, twod and oh7 datasets are combined together. A One-hidden layer MLP is trained using the proposed method and MOLF-Adapt. The results obtained from this comparison are very promising. Using sparse connectivity in the MLP structure, the network is capable of distinguishing between the two disjoint datasets and there will be no connectivity from the inputs of the first dataset to the outputs of the second dataset and vice versa.

We trained the proposed algorithms with initial hidden unit as 500. In Figure 5-18 thru Figure 5-22, the average training mean square error (MSE) versus basis functions is depicted for the proposed method (Sparse MLP) and MOLF-Adapt.

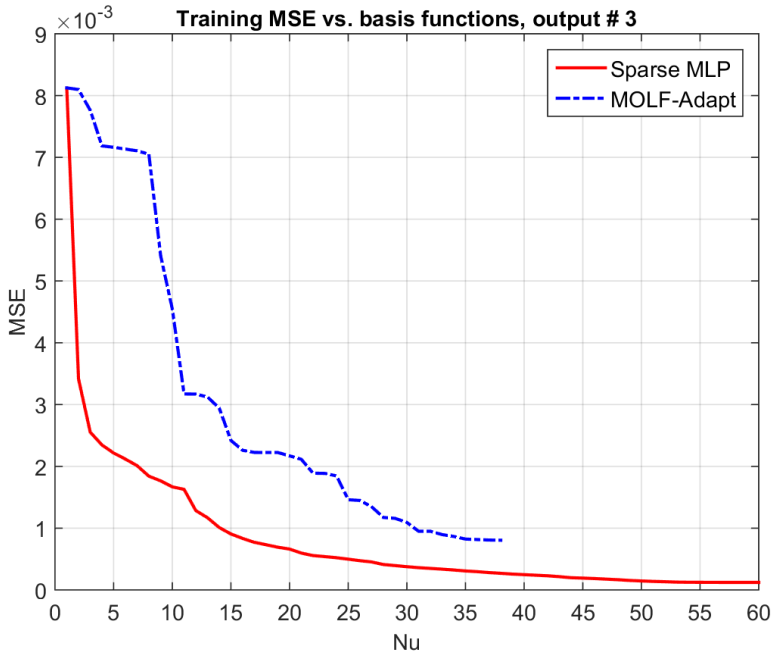


(a)

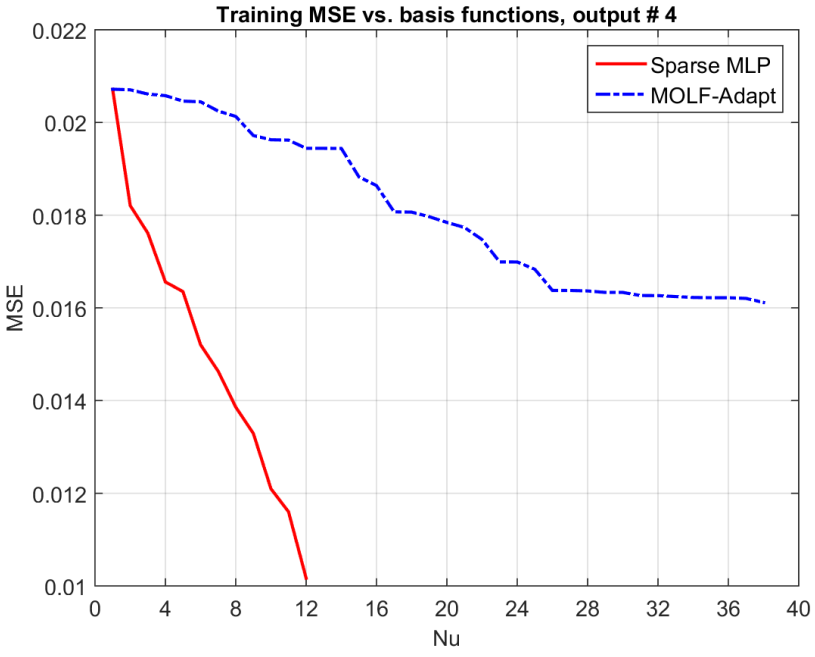


(b)

Figure 5-18: Comparison of training MSE between the proposed method and MOLF-Adapt using combined dataset for (a) class # 1 (b) class # 2

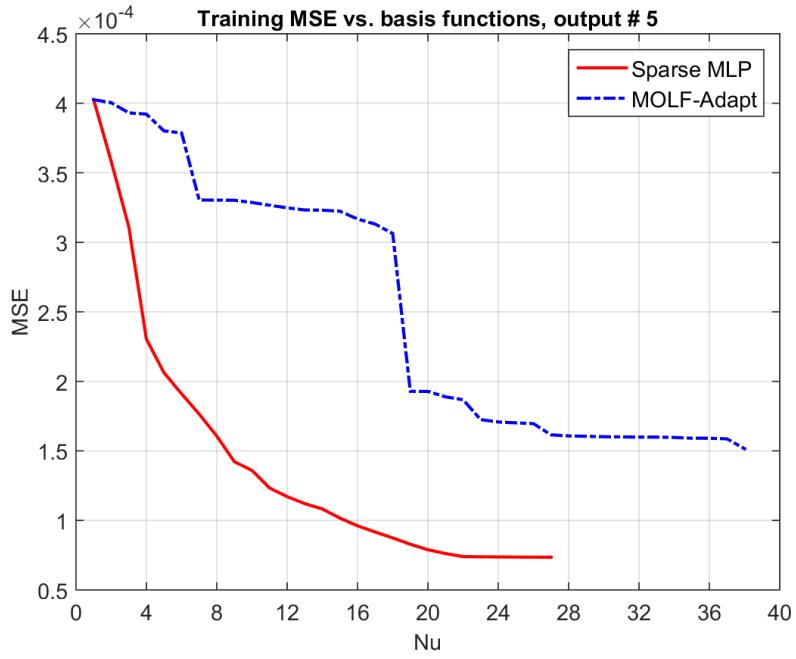


(a)

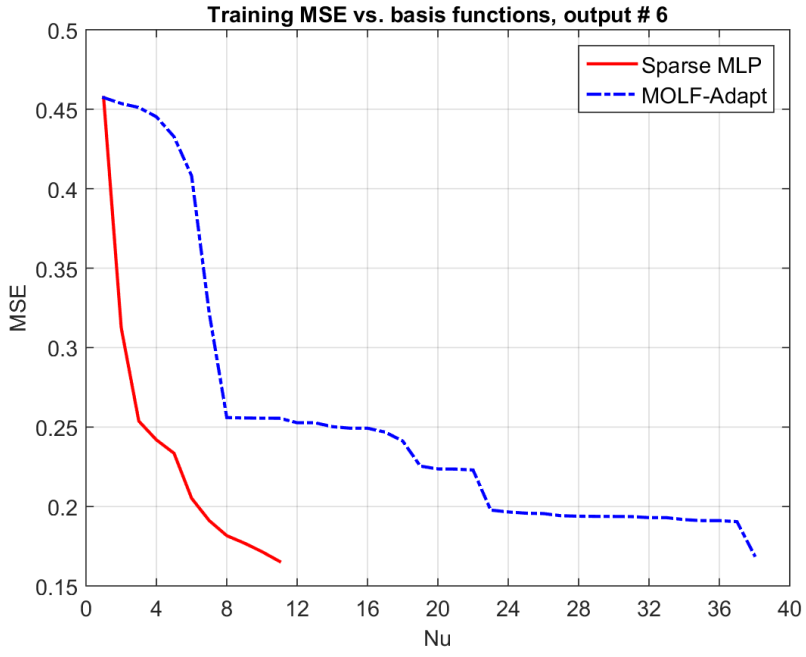


(b)

Figure 5-19: Comparison of training MSE between the proposed method and MOLF-Adapt using combined dataset for (a) class # 3 (b) class # 4

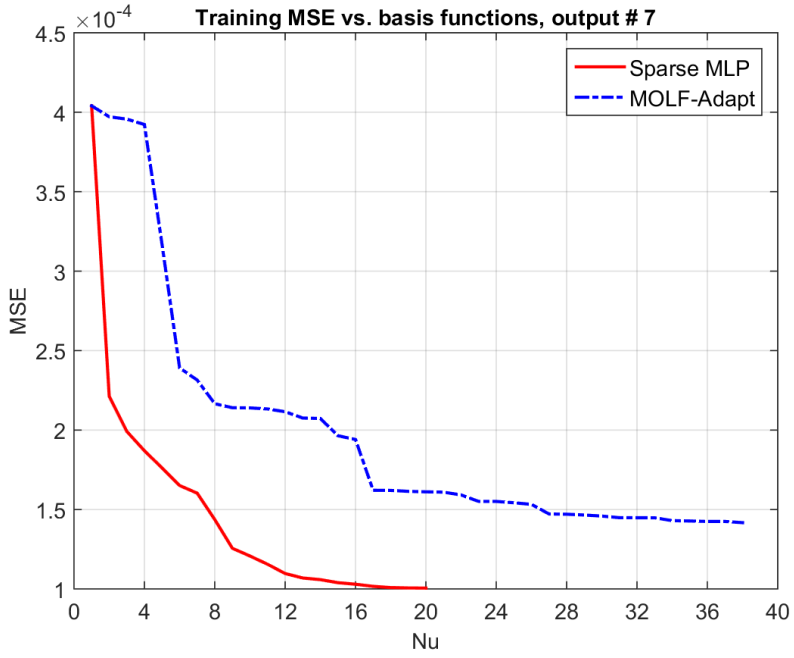


(a)

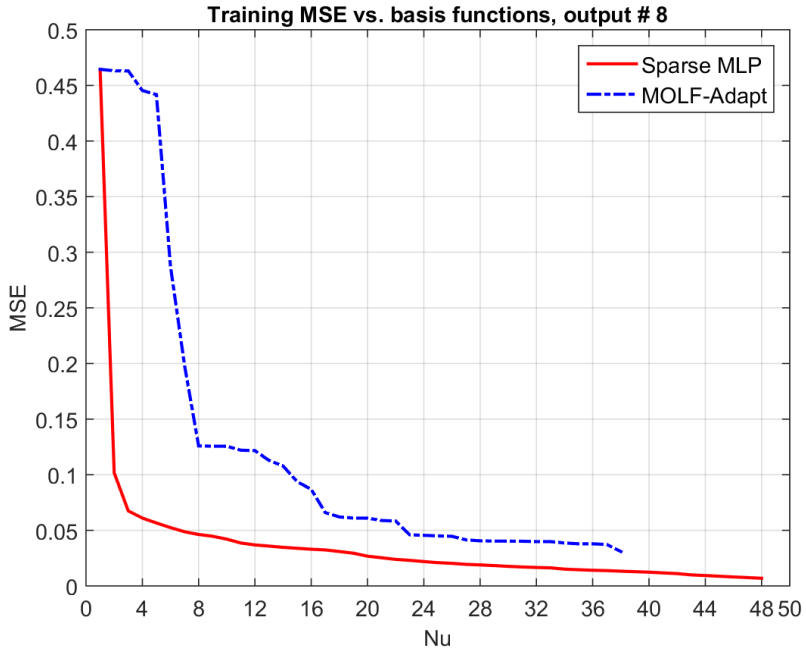


(b)

Figure 5-20: Comparison of training MSE between the proposed method and MOLF-Adapt using combined dataset for (a) class # 5 (b) class # 6

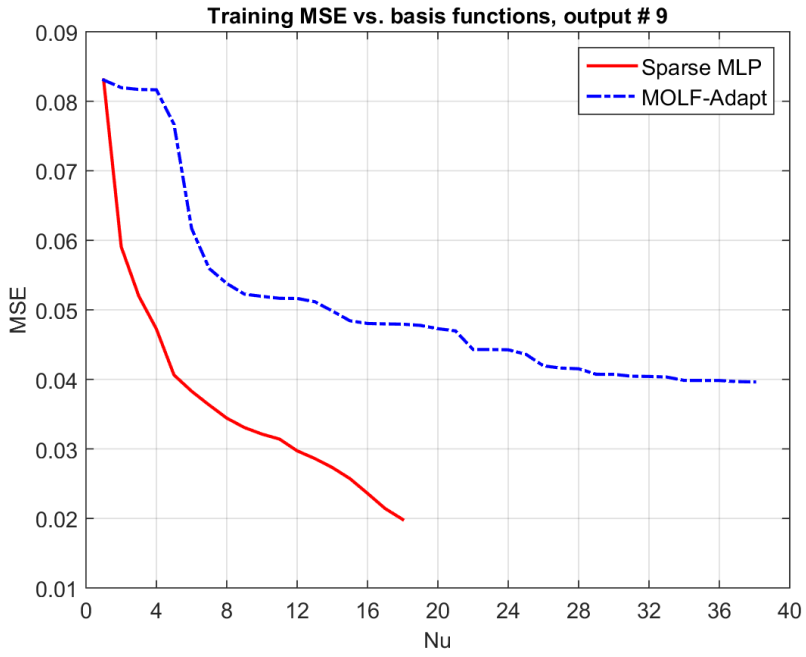


(a)

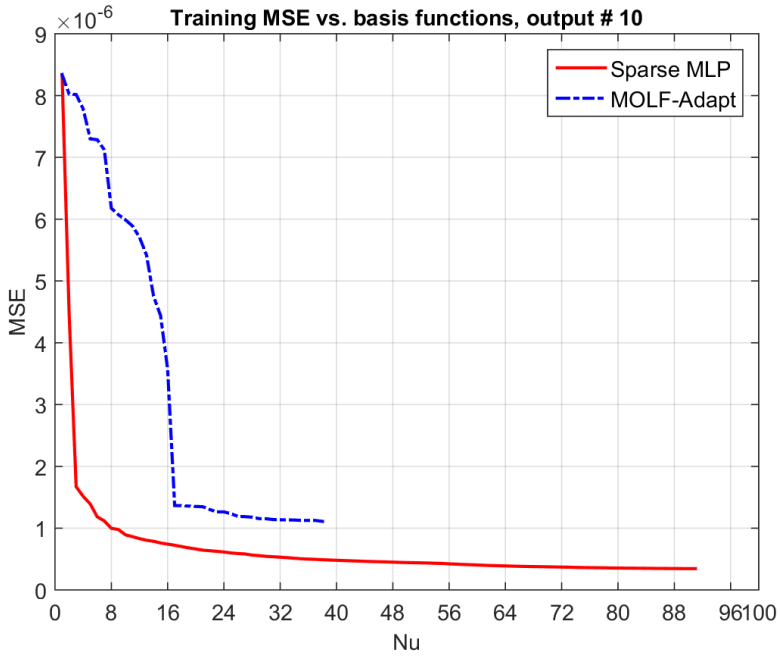


(b)

Figure 5-21: Comparison of training MSE between the proposed method and MOLF-Adapt using combined dataset for (a) class # 7 (b) class # 8



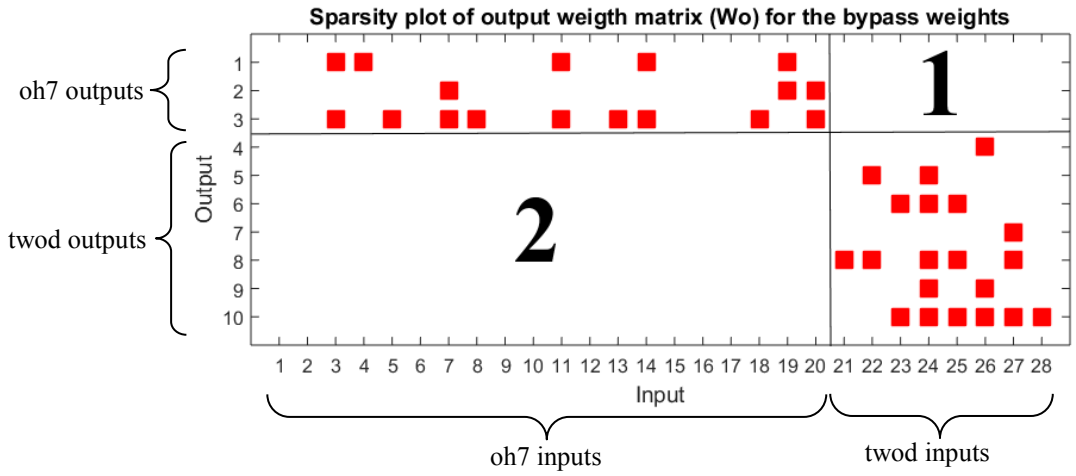
(a)



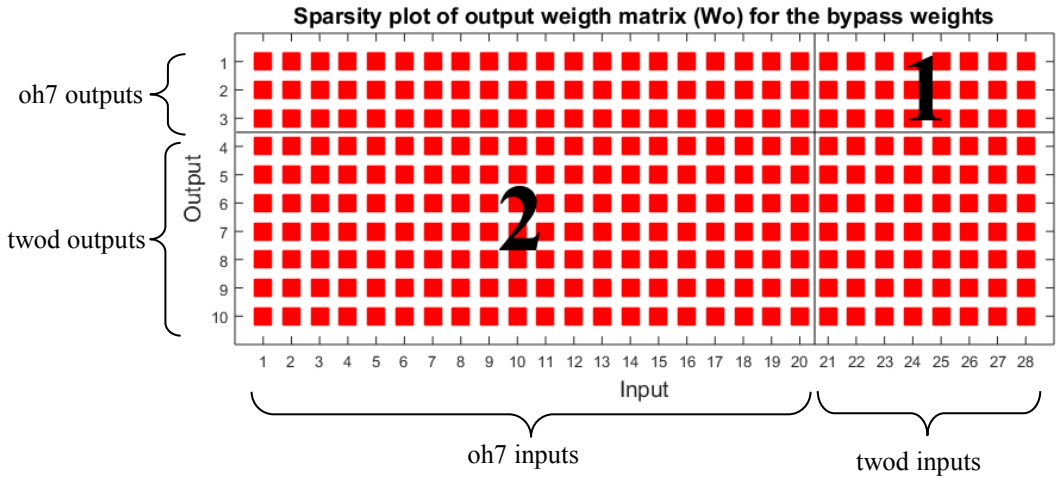
(b)

Figure 5-22: Comparison of training MSE between the proposed method and MOLF-Adapt using combined dataset for (a) class # 9 (b) class # 10

As it is can be seen in the above figures, the continuous red curves in all of the graphs are outperforming the dotted blue curves and the proposed algorithm has resulted in a better trained network. A plot of the output weight matrix can show the connectivity of the outputs and inputs of the combined dataset more clearly. The sparsity plot of the output matrix is depicted below for both algorithms.



(a)



(b)

Figure 5-23: Sparsity plot of the output weight matrix (Wo) of the combined dataset for the bypass weights using the (a) Sparse MLP (b) MOLF-Adapt

In the above figure, the red squares represent a non-zero element in the output weight matrix. Each row of the output weight matrix corresponds to one of the outputs in the network. As shown in Figure 5-23 (a) and (b), rows 1 to 3 correspond to the outputs of oh7 dataset and rows 4 to 10 correspond to the outputs of twod dataset. In Figure 5-23 (a), all the bypass weights connecting the inputs of the second dataset to the outputs of the first dataset are zero, but in Figure 5-23 (b) all the inputs of the second dataset are connected to the outputs of the fist dataset. The old algorithm is not capable of differentiating between the two datasets; therefore, all the bypass weights are considered useful in training of the network. For more clarity, these two regions are marked as 1 and 2. Region 1 corresponds to the bypass weights connecting the outputs of the first dataset (which is oh7) to the inputs of the second dataset (which is twod). Similarly, the region in which the outputs of twod dataset are connected to the inputs of oh7 dataset is marked as 2. It is noticeable that in Figure 5-23 (a) both of these regions are all zero, whereas in Figure 5-23 (b) both of these regions are non-zero.

By calculating the energy of each hidden unit, we are able to define how these hidden units are contributing in training of each dataset. As a result, 202 hidden units out of 243 remained hidden units, recalling that the initial number of hidden units was 500, are not shared between the two datasets. The total number of shared hidden units is 31. If we look at the energy of these shared hidden units for each of these datasets, we can see that most of these hidden units are actually contributing mostly to one of these datasets. Therefore, we can say although there are very few shared hidden units between the two disjoint datasets, they only contribute to one of these datasets. The table of the energy of the shared hidden units for each dataset is given in Table 5-1 on page 5-64.

This leads us to conclude that using the sparsity can strongly improve the training performance and it is capable of distinguishing between the disjoint datasets, whereas other algorithms fail to associate the same property.

Table 5-1: Comparison of the shared hidden units energy in combined dataset

hidden unit no.	oh7 dataset	twod dataset
1	0.003866	5.11e-08
2	0.003967	0.061937
3	3.94e-05	1.00e-06
40	1.462167	1.03e-05
42	0.034846	0.294724
73	0.049879	0.528596
85	0.003859	6.26e-08
93	0.145483	0.021955
99	0.000263	0.365217
103	0.002186	0.000109
108	107.8546	4.10e-07
109	41.76173	3.05e-06
116	0.006711	5.17e-06
123	138.1199	1.20e-06
143	0.031116	0.114367
149	13.42527	7.69e-07
157	41.0134	1.11e-06
163	0.002669	7.32e-07
166	8.919162	6.26e-05
182	0.013759	3.97e-05
188	8.339156	0.000947
197	0.000647	2.18e-05
204	223.8798	9.78e-06
210	0.005815	1.26e-06
213	2.45e-05	1.684683
216	7.62e-05	3.91e-07
229	0.011089	4.76e-07
230	0.006214	2.07e-08
240	0.001341	0.161292

In the above table, the first column is the hidden unit number in the network and this hidden unit is shared between twod and oh7 dataset, the second column is the energy of the shared hidden unit corresponding to oh7 dataset, and the third column is the energy of the shared hidden

unit corresponding to twod dataset. The bigger the energy value of a particular hidden unit for a dataset is, the more that hidden unit is contributing to that dataset. The bigger energies in each row are highlighted in this table for the ease of comparison. As it is noticeable in the table, there is a huge difference between the energy levels of the shared hidden units contributing to twod dataset and oh7 dataset. As a result, we can conclude that each of the shared hidden units are actually active in contributing to only one of these two datasets, and its contribution to the other dataset can be ignored.

D. Table of experimental results

In this section, the table of training MSE and testing MSE are provided. All the mean square errors are obtained using 10-fold training and testing, and averaged over all the 10 folds. The sparsity measurements are calculated for each of the datasets. The proposed algorithm is compared to the old algorithm (MOLF-Adapt) published in [21] and Andrew Ng sparse coding algorithm published in [50]. For the purpose of equal comparison, the number of epochs picked for the Ng's algorithm equals to the number of epochs where the best performance of the proposed algorithm is obtained. Additionally, different values for the parameters of Ng's algorithm are used to confirm that the proposed algorithm performs better in all the cases. The Ng's algorithm has the disadvantage of manually tuning up the parameters. And, it is highly sensitive to the initialization of these parameters.

In these tables, S_R is the row sparsity measurement computed using equation (4.1) in Chapter 4. The Ratio field equals to the ratio of the number of zero elements in the W_o to the total number of elements in the output weight matrix. The bigger these values are, the more the output weight matrix and the MLP structure are sparse. The best testing performances are highlighted. In Table 5-7 thru Table 5-11, the N_h value is the average number of hidden units for used for training each output in the network. In the proposed algorithm, each output is connected to a different set and number of hidden units. The accuracy calculated for the classification problem is the probability of correct classes.

Table 5-2: Comparison of training and testing MSE for twod dataset

Initial N_h		100		200		300		400	
Method	MSE	Train	Test	Train	Test	Train	Test	Train	Test
MOLF-Adapt		0.1202	0.1594	0.1120	0.1574	0.1117	0.1526	0.0820	0.1395
Sparse MLP with median filter		0.1166	0.1543	0.1016	0.1530	0.0975	0.1550	0.0918	0.1409
Sparse MLP without median filter		0.1197	0.1607	0.1050	0.1554	0.0952	0.1494	0.0936	0.1809
Ng Algorithm									
α=0.1, β=3, γ=3e-3		0.2277	0.2614	0.2377	0.2714	0.2088	0.2352	0.2010	0.2206
α=0.3, β=3, γ=3e-3		0.2306	0.2606	0.2328	0.2601	0.2454	0.2720	0.2562	0.2817
α=0.5, β=3, γ=3e-3		0.2893	0.3128	0.3001	0.3252	0.3104	0.3334	0.3171	0.3382
α=0.5, β=3, γ=3e-2		0.2832	0.3080	0.3076	0.3298	0.3162	0.3373	0.3184	0.3404

Table 5-3: Comparison of training and testing MSE for oh7 dataset

Initial N_h		100		200		300		400	
Method	MSE	Train	Test	Train	Test	Train	Test	Train	Test
MOLF-Adapt		1.2442	1.5613	1.2636	1.5494	1.2252	1.5478	1.2768	1.5643
Sparse MLP with median filter		1.2317	1.5649	1.2831	1.5607	1.2567	1.5419	1.2650	1.5468
Sparse MLP without median filter		1.2261	1.5711	1.2870	1.5467	1.2476	1.5487	1.2587	1.5453
Ng Algorithm									
α=0.1, β=3, γ=3e-3		2.5774	2.7143	2.2495	2.4579	2.0294	2.2498	1.9831	2.1915
α=0.3, β=3, γ=3e-3		2.0415	2.1478	1.8018	1.9513	1.7241	1.8876	1.7007	1.8689
α=0.5, β=3, γ=3e-3		2.4501	2.5649	2.1656	2.3545	2.2209	2.4721	2.1591	2.4304
α=0.5, β=3, γ=3e-2		2.5608	2.7840	2.5819	2.8067	2.6038	2.8287	2.6526	2.8721

Table 5-4: Comparison of training and testing MSE for gongtrn dataset

Initial N_h		100		200		300		400	
Method	Accuracy	Train	Test	Train	Test	Train	Test	Train	Test
MOLF-Adapt		95.6500	93.3000	97.1722	92.9500	97.1806	92.4500	96.8917	92.6167
Sparse MLP with median filter		94.5250	92.9167	95.4500	93.4000	96.5500	93.1167	96.0861	93.1333
Sparse MLP without median filter		94.4083	92.7833	95.7333	93.2500	96.0778	93.2333	96.4750	93.2500
Ng Algorithm									
α=0.1, β=3, γ=3e-3		88.3000	86.0533	89.3067	86.8000	89.1333	86.8533	89.0667	86.9267
α=0.3, β=3, γ=3e-3		91.4467	88.7867	92.6733	90.4533	93.1533	90.6933	93.2067	91.0200
α=0.5, β=3, γ=3e-3		91.7600	89.7800	92.7133	90.5267	92.8867	91.0133	92.5000	90.3667
α=0.5, β=3, γ=3e-2		89.8133	87.4533	90.3133	88.1400	90.7133	88.6067	89.8467	87.6400

Table 5-5: Comparison of training and testing MSE for MNIST dataset

Initial N_h		700		800		900		1000	
Method	Accuracy	Train	Test	Train	Test	Train	Test	Train	Test
MOLF-Adapt		98.8	94.83	98.6041	95.02	99.01	95.66	99.3	95.64
Sparse MLP with median filter		99.5	95.70	98.8312	95.72	99.8	95.77	98.8	96.01
Sparse MLP without median filter		99.4	95.75	99.2166	95.67	99.4	95.76	99.02	96.10
Ng Algorithm									
α=0.1, β=3, γ=3e-3		92.4565	92.7506	93.8880	93.9796	92.8095	92.9925	93.3586	93.6004
α=0.3, β=3, γ=3e-3		93.0971	93.0906	92.5218	92.6787	91.7309	92.2081	91.8943	92.2342
α=0.5, β=3, γ=3e-3		92.2473	92.4042	91.1426	91.4825	92.5872	92.6460	91.9531	91.8943
α=0.5, β=3, γ=3e-2		87.8284	88.4167	89.7176	90.0967	87.9396	88.4102	90.5085	90.4758

Table 5-6: Comparison of training and testing MSE for combined dataset

Initial N_h		300		400		500		600	
Method	MSE	Train	Test	Train	Test	Train	Test	Train	Test
MOLF-Adapt		1.7073	2.4413	1.7226	2.5361	1.8022	2.4521	1.6550	2.3627
Sparse MLP with median filter		1.7145	2.4755	1.7609	2.4313	1.7130	2.4615	1.7505	2.4625
Sparse MLP without median filter		1.7161	2.4800	1.7249	2.4092	1.7203	2.4460	1.7483	2.4401
Ng Algorithm									
α=0.1, β=3, γ=3e-3		2.2332	3.1410	1.9522	3.1917	1.8487	3.1543	1.8417	3.0572
α=0.3, β=3, γ=3e-3		2.0274	3.0690	1.8351	3.0312	1.7480	3.0823	1.6775	2.9765
α=0.5, β=3, γ=3e-3		2.3651	3.2949	2.2202	3.4578	2.1102	3.2250	2.1217	3.3572
α=0.5, β=3, γ=3e-2		2.4176	3.2336	2.4345	3.1999	2.4432	3.2106	2.5483	3.2676

Table 5-7: Comparison of the sparsity measurements for twod dataset

Initial N_h		100			200			300			400		
Method	Sparsity	S_R	Ratio	N_h	S_R	Ratio	N_h	S_R	Ratio	N_h	S_R	Ratio	N_h
MOLF-Adapt		0.4637	0.355	55	0.4133	0.544	82	0.4583	0.7090	78	0.4511	0.7490	91
Sparse MLP with median filter		0.6659	0.5602	40	0.7132	0.647	62	0.7668	0.7489	70	0.8031	0.8285	64
Sparse MLP without median filter		0.6816	0.5872	39	0.7162	0.673	61	0.7654	0.7377	74	0.8073	0.8247	65
Ng Algorithm													
α=0.1, β=3, γ=3e-3		0.2871	-	100	0.2599	-	200	0.2574	-	300	0.2522	-	400
α=0.3, β=3, γ=3e-3		0.2404	-	100	0.2576	-	200	0.2741	-	300	0.2774	-	400
α=0.5, β=3, γ=3e-3		0.3163	-	100	0.3383	-	200	0.3301	-	300	0.3442	-	400
α=0.5, β=3, γ=3e-2		0.3118	-	100	0.3396	-	200	0.3286	-	300	0.2655	-	400

Table 5-8: Comparison of the sparsity measurements for oh7 dataset

Initial N_h		100			200			300			400		
Method	Sparsity	S_R	Ratio	N_h	S_R	Ratio	N_h	S_R	Ratio	N_h	S_R	Ratio	N_h
MOLF-Adapt		0.5547	0.1120	68	0.5480	0.3960	100	0.5511	0.4177	154	0.5476	0.4838	186
Sparse MLP with median filter		0.6306	0.2740	93	0.6333	0.3603	177	0.6366	0.3474	265	0.6086	0.3175	370
Sparse MLP without median filter		0.6281	0.2786	92	0.6280	0.3533	177	0.6277	0.3394	257	0.6003	0.3173	372
Ng Algorithm													
α=0.1, β=3, γ=3e-3		0.3059	-	100	0.2698	-	200	0.2399	-	300	0.2219	-	400
α=0.3, β=3, γ=3e-3		0.2827	-	100	0.2442	-	200	0.2358	-	300	0.2324	-	400
α=0.5, β=3, γ=3e-3		0.2483	-	100	0.2443	-	200	0.2439	-	300	0.2265	-	400
α=0.5, β=3, γ=3e-2		0.2381	-	100	0.2561	-	200	0.2626	-	300	0.2708	-	400

Table 5-9: Comparison of the sparsity measurements for gongtrn dataset

Initial N_h		100			200			300			400		
Method	Sparsity	S_R	Ratio	N_h	S_R	Ratio	N_h	S_R	Ratio	N_h	S_R	Ratio	N_h
MOLF-Adapt		0.4040	0.4010	43	0.3427	0.3965	104	0.3440	0.4867	137	0.3354	0.5415	167
Sparse MLP with median filter		0.6312	0.6650	28	0.6069	0.6682	57	0.6291	0.7174	79	0.6412	0.7225	83
Sparse MLP without median filter		0.6537	0.6889	28	0.5787	0.6336	64	0.6239	0.7019	80	0.6573	0.7513	86
Ng Algorithm													
α=0.1, β=3, γ=3e-3		0.2708	-	100	0.2242	-	200	0.2092	-	300	0.2072	-	400
α=0.3, β=3, γ=3e-3		0.2101	-	100	0.2003	-	200	0.1980	-	300	0.1955	-	400
α=0.5, β=3, γ=3e-3		0.2168	-	100	0.2051	-	200	0.1905	-	300	0.1903	-	400
α=0.5, β=3, γ=3e-2		0.2160	-	100	0.1950	-	200	0.1903	-	300	0.1899	-	400

Table 5-10: Comparison of the sparsity measurements for MNIST dataset

Initial N_h		700			800			900			1000		
Method	Sparsity	S_R	Ratio	N_h	S_R	Ratio	N_h	S_R	Ratio	N_h	S_R	Ratio	N_h
MOLF-Adapt		0.9172	02408	601	0.4831	0.0425	774	0.4754	0.2082	784	0.9418	0.1037	996
Sparse MLP with median filter		0.7094	0.5025	326	0.7141	0.4876	453	0.7674	0.5828	398	0.7037	0.5010	678
Sparse MLP without median filter		0.7341	0.5851	369	0.7181	0.5748	371	0.7729	0.5680	407	0.7251	0.5104	496
Ng Algorithm													
α=0.1, β=3, γ=3e-3		0.1564	-	700	0.1543	-	800	0.1537	-	900	0.1539	-	1000
α=0.3, β=3, γ=3e-3		0.1543	-	700	0.1545	-	800	0.1527	-	900	0.1536	-	1000
α=0.5, β=3, γ=3e-3		0.1577	-	700	0.1581	-	800	0.1581	-	900	0.1587	-	1000
α=0.5, β=3, γ=3e-2		0.1655	-	700	0.1650	-	800	0.1656	-	900	0.1653	-	1000

Table 5-11: Comparison of the sparsity measurements for combined dataset

Initial N_h		300			400			500			600		
Method	Sparsity	S_R	Ratio	N_h	S_R	Ratio	N_h	S_R	Ratio	N_h	S_R	Ratio	N_h
MOLF-Adapt		0.5366	0.6940	63	0.5425	0.7918	55	0.5849	0.8578	43	0.5566	0.8593	56
Sparse MLP with median filter		0.8038	0.8156	46	0.7066	0.5758	66	0.7948	0.7620	62	0.7198	0.6005	73
Sparse MLP without median filter		0.7338	0.6539	54	0.7533	0.6868	55	0.7275	0.5969	68	0.7610	0.6907	70
Ng Algorithm													
α=0.1, β=3, γ=3e-3		0.2799	-	300	0.2524	-	400	0.2395	-	500	0.2249	-	600
α=0.3, β=3, γ=3e-3		0.2534	-	300	0.2326	-	400	0.2261	-	500	0.2209	-	600
α=0.5, β=3, γ=3e-3		0.2467	-	300	0.2281	-	400	0.2203	-	500	0.2190	-	600
α=0.5, β=3, γ=3e-2		0.2380	-	300	0.2262	-	400	0.2357	-	500	0.2403	-	600

The values of the above tables prove that the proposed algorithm in most of the cases has a better performance when using the testing dataset. The Ng's algorithm does not prune the hidden unit, therefore the ratio field is not calculated. Another important observation from the tables is that the proposed algorithm uses less number of hidden units in average comparing to the other two algorithms.

Chapter 6

CONCLUSION AND FUTURE WORK

In the present work, we have designed a sparse neural network that is capable of generating sparse models with low storage capacity. We introduce a new pruning method that prunes the hidden weights and hidden units simultaneously.

Experimental results in our investigation conclude that the sparse neural network performs better than the conventional MLP even if the number of weight connections is less in a sparse neural network. We also conclude that the proposed sparse neural network is able to differentiate between datasets that have completely different statistical properties, thereby making separate networks for each of the given datasets. The ability of the sparse neural network to differentiate different datasets is strikingly different from the conventional MLP.

On the other side, the training time to design the space neural network is high since the number of hidden units is more than a conventional MLP.

Appendix A

DESCRIPTION OF DATASETS USED FOR TRAINING AND TESTING

Twod dataset - Inversion of surface scattering parameters

This training file is used in the task of inverting the surface scattering parameters from an inhomogeneous layer above a homogeneous half space, where both interfaces are randomly rough. The parameters to be inverted are the effective permittivity of the surface, the normalized rms height, the normalized surface correlation length, the optical depth, and single scattering albedo of an inhomogeneous irregular layer above a homogeneous half space from back scattering measurements.

The training data file contains 1768 patterns, 8 inputs, and 7 outputs. The inputs consist of eight theoretical values of back scattering coefficient parameters at V and H polarization and four incident angles. The outputs were the corresponding values of permittivity, upper surface height, lower surface height, normalized upper surface correlation length, normalized lower surface correlation length, optical depth and single scattering albedo which had a joint uniform PDF. [60]

Oh7 dataset - Radar Scattering from Bare Soil Surfaces

This data set is given in [61]. The training set contains VV and HH polarization at L 30, 40 deg, C 10, 30, 40, 50, 60 deg, and X 30, 40, 50 deg along with the corresponding unknowns rms surface height, surface correlation length, and volumetric soil moisture content in g / cubic cm. The file has 20 inputs, 3 outputs and 10,453 training patterns.

Gongtrn dataset – Handwritten images

The raw data consists of images from hand printed numerals collected from 3000 people by the Internal Revenue Service. We randomly chose 300 characters from each class to generate 3000 character training data. Images are 32 by 24 binary matrices. An image scaling algorithm is used to remove size variation in characters. The feature set contains 16 elements. The 10 classes correspond to 10 Arabic numerals. [62]

MNIST dataset – Handwritten digits

The MNIST ("Modified National Institute of Standards and Technology") database of handwritten digits, has a training set of 60000 examples, and a test set of 10000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. This dataset is a classic within the Machine learning community and has been extensively studied. It has 784 inputs and 10 classes. [63]

REFERENCES

[1] S. Thayer, W. Whittaker, C. Baker, W. Burgard, D. Ferguson S. Thrun and D. Montemerlo, A. Morris, Z. Omohundro, C. Reverte, Whittaker W D. Hahnel, "Autonomous exploration and mapping of abandoned mines," *IEEE Robotics & Automation Magazine*, vol. 11, no. 4 , pp. 79 - 91, Dec. 2004.

[2] Xiao Li Li Deng, "Machine Learning Paradigms for Speech Recognition: An overview," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 21, no. 5, pp. 1060-1089, May 2013.

[3] Jason Weston Ronan Collobert, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 160-167.

[4] J Weston, L Bottou, M Karlen, K Kavukcuoglu, P Kuksa R Collobert, "Natural language processing (almost) from scratch," *The Journal of Machine Learning Research* , vol. 12, pp. 2493-2537, Feb 2011.

[5] C Wojek, B Schiele, P Perona P Dollar, "Pedestrian Detection: An Evaluation of the State of the Art," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, pp. 743 - 761, April 2012.

[6] DS Wishart JA Cruz, "Applications of machine learning in cancer prediction and prognosis," *Cancer informatics*, vol. 2, pp. 59-77, Jan. 2006.

[7] Nicholas M., and Robert J. Brunner Ball, "Data Mining and Machine Learning in Astronomy," *International Journal of Modern Physics D*, vol. 19, no. 07, pp. 1049-1106, July 2010.

[8] CF Van Loan GH Golub, *Matrix computations*.: JHU Press, 2012, vol. 3.

[9] Patrik O. Hoyer, "Non-negative Matrix Factorization with Sparseness Constraints," *The Journal of Machine Learning Research*, vol. 5, pp. 1457-1469, Dec. 2004.

[10] Niall Hurleyand Scott Rickard, "Comparing Measures of Sparsity," *IEEE TRANSACTIONS ON INFORMATION THEORY*, vol. 55 , no. 10, Oct. 2009.

[11] Markus Thom and Günther Palm, "Sparse Activity and Sparse Connectivity in Supervised Learning," *Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1091-1143, Jan. 2013.

[12] A Battle, R Raina, and AY Ng H Lee, "Efficient sparse coding algorithms," in *Advances in neural information processing systems*, 2006, pp. 801-808.

[13] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, June 1996.

[14] Sanjeev S. Malalur and Michael T. Manry, "Multiple optimal learning factors for feed-forward networks," *Proceedings of SPIE: Independent Component Analyses, Wavelets, Neural Networks, Biosystems, and Nanoengineering VIII*, vol. 7703, April 2010.

[15] David E., Geoffrey E. Hinton, and R. J. Williams. Rumelhart, "Learning Internal Representations by Error Propagation," *MIT Press*, vol. 1, 1986.

[16] Frank Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Wahington DC: Spartan Books, 1961.

[17] M. Klaseen and Y. -H. Pao, "The functional link net in structural pattern recognition," in *IEEE Region 10 Conference on Computer and Communication Systems*, 1990.

[18] Y. -H. Pao and Y. Takefuji, "Functional-link net computing: theory, system architecture, and

functionalities," *IEEE Computer Society*, vol. 25, no. 5.

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

[20] Simon Haykin, *Neural Networks and Learning Machines*, 3rd ed. USA: Pearson, 2009.

[21] Rohit Rawat, Jignesh K. Patel, and Michael T. Manry, "Minimizing Validation Error With Respect to Network Size and," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, Dallas, 2013, pp. 1 - 7.

[22] Melvin D. Robinson and Michael T. Manry, "Two-Stage Second Order Training in Feedforward Neural Networks," in *FLAIRS Conference*, 2013.

[23] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly Journal of Applied Mathematics II*, vol. 2, pp. 164–168, July 1944.

[24] Kanishka Tyagi, "Second Order training Algorithms for Radial basis," University of Texas at Arlington, Master's Thesis 2011.

[25] Son Nguyen, Kanishka Tyagi, Parastoo Kheirkhah, and Michael T. Manry, "Partially Affine Invariant Back Propagation," in *IJCNN*, Vancouver, 2016.

[26] S. K. Rogers, M. Kabrisky, M. E. Oxley and B. W. Suter D. W. Ruck, "The multilayer perceptron as an approximation to a Bayes optimal discriminant function," in *IEEE Transactions on Neural Networks*, vol. 1, 1990, pp. 296-298.

[27] Soumitro Swapan Auddy , "DISCRIMINANT PROCESSING IN MULTI-CLASS PATTERN RECOGNITION SYSTEMS," The University of Texas at Arlington, Master's Thesis 2013.

[28] Soumitro Sawpan Auddy, Kanishka Tyagi, and Michael T. Manry, "Discriminant Vector Tranformations in Neural Network Classifiers," in *IJCNN*, Vancouver, 2016.

[29] Kurt, Maxwell Stinchcombe, and Halbert White Hornik, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359-366, 1989.

[30] M. T. Manry and F. Maldonado P. L. Narasimha, "Upper Bound on Pattern Storage in Feedforward Networks," in *Neurocomputing*, vol. 71, 2008, pp. 3612-3616.

[31] Kanishka Tyagi, Xun Cai, and Michael T. Manry, "Fuzzy C-means clustering based construction and training for second order RBF network," in *Fuzzy Systems (FUZZ), 2011 IEEE International Conference on*, Taipei, 2011, pp. 248 - 255.

[32] T. Kathirvalavakumar M. Gethsiyal Augasta, "Pruning algorithms of neural networks — a comparative study," *Central European Journal of Computer Science*, vol. 3, no. 3, pp. 105-115, Sep. 2013.

[33] Russell Reed, "Pruning Algorithms-A Survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740-747, Sep. 1993.

[34] John S. Denker, Sara A. Solla Yann Le Cun, "Optimal brain damage," *Advances in neural information processing systems 2*, vol. 2, pp. 598-605, 1989.

[35] David G. Stork, and Gregory J. Wolff Babak Hassibi, "Optimal brain surgeon and general network pruning," in *IEEE International Conference on Neural Networks*, vol. 1, 1993, pp. 293 - 299.

[36] Daniel D. Lee and H. Sebastian Seung, "Algorithms for non-negative matrix factorization," in *Advances in neural information processing systems*, 2001, pp. 556-562.

[37] H. Sebastian Seung Daniel D. Lee, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788-791, 1999.

[38] Hynek Hermansky G. S. V. S. Sivaram, "Multilayer perceptron with sparse hidden outputs for phoneme recognition," in *Acoustics, Speech and Signal Processing (ICASSP),*

2011 IEEE International Conference on, 2011, pp. 5336 - 5339.

[39] D Díaz, JR Dorronsoro A Torres, "Sparse one hidden layer MLPs," in *European Symposium on Artificial Neural Networks, Computational Intelligenceand Machine Learning*, 2014.

[40] Yuan Xu, Francis X. Canning Robert J. Adams, "Sparse pseudo inverse of the discrete plane wave transform," *IEEE Transactions on Antennas and Propagation* , vol. 56, no. 2 , pp. 475 - 484, Feb. 2008.

[41] Allen Y. Yang, Arvind Ganesh, S. Shankar Sastry John Wright, "Robust Face Recognition via Sparse Representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 2 , pp. 210 - 227, April 2008.

[42] Julien Mairal, Michael Elad, and Guillermo Sapiro, "Sparse Representation for Color Image Restoration," *IEEE Transactions on Image Processing* , vol. 17, no. 1 , pp. 53 - 69, Jan. 2008.

[43] D Donoho, JM Pauly M Lustig, "Sparse MRI: The Application of Compressed Sensingfor Rapid MR Imaging," *Magnetic Resonance in Medicine*, vol. 58, no. 6, pp. 1182–1195, Dec. 2007.

[44] G Karypis, V Kumar A Gupta, "Highly scalable parallel algorithms for sparse matrix factorization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 5, pp. 502 - 520, May 1997.

[45] YC Eldar M Mishali, "Reduce and boost: Recovering arbitrary sets," *IEEE Transactions on Signal Processing*, vol. 56, no. 10, pp. 4692 - 4702, Sep. 2008.

[46] Ron Rubinstein, Michael Zibulevsky, and Michael Elad, "Double sparsity: Learning sparse dictionaries for sparse signal approximation," *IEEE Transactions on Signal*

Processing, vol. 58, no. 3, pp. 1553 - 1564, Nov. 209.

[47] Federico Girosi, "An Equivalence Between Sparse Approximation and Support Vector Machines," *Neural Computation*, vol. 10, no. 6, pp. 1455 - 1480, Aug. 1998.

[48] P.-L. Dragotti, M. Vetterli, P. Marziliano, and L. Coulot T. Blu, "Sparse sampling of signal innovations," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 31 - 40, March 2008.

[49] Richard O., Peter E. Hart, and David G. Stork Duda, *Pattern Classification*, 2nd ed.: John Wiley & Sons, 2012.

[50] Andrew Y. Ng, "Feature selection, L 1 vs. L 2 regularization, and rotational invariance," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 78.

[51] F. Sahin G. Chandrashekar, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16-28, Jan. 2014.

[52] Y-L Boureau, Y LeCun MA Ranzato, "Sparse feature learning for deep belief networks," *Advances in neural information processing systems*, vol. 20, pp. 1185-1192, 2008.

[53] Y-lan, Sumit Chopra, and Yann Lecun Boureau, "A Unified Energy-Based Framework for Unsupervised Learning," in *International Conference on Artificial Intelligence and Statistics*, 2007.

[54] Sumit Chopra, Yann LeCun Christopher Poultney, "Efficient learning of sparse representations with an energy-based model," *Advances in neural information processing systems*, pp. 1137-1144, 2006.

[55] E Oja A Hyvärinen, "Independent component analysis: algorithms and applications," *Neural networks*, vol. 13, no. 4-5, pp. 411–430, June 2000.

[56] Terrence J. Sejnowski Anthony J. Bell, "The “independent components” of natural scenes are edge filters," *Vision research*, vol. 37, no. 23, pp. 3327–3338, Dec. 1997.

[57] A. M. Fanelli , and M. Pelillo G. Castellano, "An iterative pruning algorithm for feedforward neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 519 - 531, May 1997.

[58] Fangju Ai, "A New Pruning Algorithm for Feedforward Neural Networks," in *Fourth International Workshop on Advanced Computational Intelligence (IWACI)*, Wuhan, Hubei, 2011, pp. 286 - 289.

[59] Robert Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267-288, 1996.

[60] A. K. Fung and M. T. Manry M. S. Dawson, "Surface parameter retrieval using fast learning neural networks.," in *Remote Sensing Reviews*, vol. 7, 1993, pp. 1-18.

[61] K. Sarabandi and F. T. Ulaby. Y. Oh, "An empirical model and an inversion technique for radar scattering from bare soil surfaces," in *IEEE Trans. on Geoscience and Remote Sensing*, vol. 2, 1992, pp. 370-381.

[62] H. C. Yau W. Gong and Michael T. Manry, "Non-Gaussian Feature Analyses Using a Neural Network," in *Progress in Neural Networks,* vol. 2, 1994, pp. 253-269.

[63] L. Bottou, Y. Bengio, and P. Haffner. Y. LeCun, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, 1998, pp. 2278-2324.

BIOGRAPHICAL INFORMATION

Parastoo Kheirkhah was born in Karaj, Tehran, Iran, in 1986. She received her Bachelor of Science degree in Electrical Engineering in 2010 from I.K. International University. From 2010 to 2013 she was an electrical engineer consultant at engineering consulting company Ivan Sepid Pars in Tehran. There she gained her first hands in industry. She worked on electrical engineering installation of buildings, lighting systems, and building protection systems. She is currently a graduate student in Image Processing and Neural Networks Lab, The University of Texas at Arlington, USA. During Summer 2016, she worked as an Grad Tech intern at Nokia Siemens Networks, Irving, Texas to develop validating tools for RF antennas and LTE networks.

Her specific research interests are neural networks, image processing and embedded systems.